Real Time Inverse Kinematics for General 6R Manipulators

Dinesh Manocha[1]
John F. Canny[2]

Computer Science Division
Department of Electrical Engineering and Computer Science
University of California at Berkeley
Berkeley, CA 94720

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 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, 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. 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 techniques 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 numerical accuracy of these operations is well understood and for most cases we are able to compute accurate solutions using double precision arithmetic. 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.

[1]Supported by IBM Graduate Fellowship, David and Lucile Packard Fellowship and National Science Foundation Presidential Young Investigator Award (# IRI-8958577).

[2]Supported in part by David and Lucile Packard Fellowship and National Science Foundation Presidential Young Investigator Award (# IRI-8958577).

1 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 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 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 from exists.

In the absence of a closed form solution, [WM91] claim that the problem of inverse kinematics for a general $6R$ 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.

All commercial robots with six revolute joints are designed with simple kinematic model such that their inverse kinematics can be expressed as a closed form solution. In real world applications the positioning accuracy of these manipulators depends on the kinematic model used to describe the robot geometry in a parametric form. However, manufacturing errors in machining and assembly of manipulators lead to discrepancies between the design parameters and physical structure. This mismatch is especially prevalent in manipulators with revolute joints in which small manufacturing errors produce significant errors between the actual and predicted positions and orientations of the end effector. The typical approach involves identification of the individual kinematic parameters and incorporating them into manipulator's controller to improve positional accuracy. The former process of identification is called the *arm signature identification* [SSN86, WL84]. Given the accurate kinematic parameters, a number of methods have been proposed to calibrate and compensate for the kinematic errors in robot manipulators [Hay83, VW87]. However, a practical solution for the inverse kinematics of general manipulators, especially 6, will eliminate the need for any algorithms for calibration and compensation of kinematic errors.

The inverse kinematics problem for six revolute joints has been studied for at least two decades. The earliest systematic attempt on this problem appears to have been by Pieper [Pie68]. Pieper developed closed form solutions for the case where the three consecutive axes

are concurrent. For $6R$ manipulators of general geometry, Pieper used a naive elimination strategy, which indicated an upper bound of $64,000$. The first major accomplishment on the general version problem was obtained by [RRS73], where an upper bound of 32 was given to the number of solutions. All these bounds on the number of solutions apply when the actual number is finite. The proof in [RRS73] is based on arguments from synthetic geometry and is non-constructive. The first constructive solution to the problem was given by [AA79]. In particular, [AA79] expressed the solution in the form of 12×12 determinant, whose entries were quartic polynomials in the tangent of the half-angle of one of the joint variables. [DC80] 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 [TM85]. 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*). They tried different configurations and found only 16 solutions (sometimes complex) for various $6R$ manipulators of different geometries. As a result, they 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 [Pri86]. In particular, [Pri86] showed that the remaining 16 solutions to the 32 degree polynomial in [DC80] have purely imaginary parts. Finally, [LL88a, LL88b] gave the exact solution in lower dimensions by reducing the problem to a 16 degree polynomial. Moreover, [RR89] used dialytic elimination and properties of the ideal generated by the multivariate equations to derive a 16 degree polynomial in the tangent of the half–angle of a joint variable. Complementing these results, [MD89] 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 [TM85, WM91]. According to [WM91], direct application of lower dimensional methods, like the one presented in [RR89], require hundreds of digits of precision for portions of the computation and therefore make it impractical for implementation on current workstations. However, algorithms based on continuation methods are rather slow. The best known algorithm takes about 10 seconds on an average of CPU time on an IBM $370 - 3090$ using double precision arithmetic [WM91], 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 general $6R$ 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 algebraic results presented in [RR89]. 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. In particular, we obtain a 24×24 matrix, which therefore has 24 eigenvalues. 8 of these eigenvalues are fixed constants and the 16 other eigenvalues correspond to the tangent of the half–angle of a joint variable. Furthermore, the eigenvectors

corresponding to these 16 eigenvalues are used to compute the rest of the joint variables. 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*[3] and fast implementations are available [GL89, ABB+92]. This is in contrast with expanding a symbolic determinant to compute a degree 16 polynomial and thereby, computing its roots. The latter method is relatively slower and the problem of computing roots of such polynomials can be ill–conditioned [Wil59, Wil63]. The numerical stability of the operations used in our algorithm is well understood. As a result, we are able to come up with tight bounds on the accuracy of the solution. For almost all instances of the problem we are able to compute accurate solutions using 64 bit IEEE floating point arithmetic [Gol91]. 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 6.2, 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 [RR89]. In Section 6.3, we present results from linear algebra and numerical analysis, which are being used in the algorithm. The algorithm has been presented in Section 6.4 and we discuss its accuracy, implementation and performance in Section 6.5. In Section 6.6 we extend the algorithm to six jointed manipulators consisting of revolute and prismatic joints. We also highlight the applications of this approach to redundant manipulators.

2 Inverse Kinematics

2.1 Problem Formulation

We use Denavit–Hartenberg formalism, [DH55], 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 [SV89, TM85]. The 4×4 transformation matrix relating $i+1$ coordinate system to i coordinate system is [SV89]:

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

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

where

$$s_i = \sin\theta_i, \quad c_i = \cos\theta_i, \quad \theta_i \text{ is the } i\text{th joint rotation angle,}$$

$$\mu_i = \sin\alpha_i, \quad \lambda_i = \cos\alpha_i, \quad \alpha_i \text{ is the twist angle between the axes of joints } i \text{ and } i+1,$$

$$a_i \text{ is the length of link } i+1,$$

$$d_i \text{ is the offset distance at joint } i.$$

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:

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

The problem of inverse kinematics corresponds to computing the joint angles, $\theta_1, \theta_2, \theta_3$, θ_4, θ_5 and θ_6 such that

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

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 $\mathbf{A}_{hand}$ 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.

2.2 Raghavan and Roth Solution

In this section, we briefly describe the lower dimensional approach described by Raghavan and Roth [RR89]. 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

$$\mathbf{A}_3\mathbf{A}_4\mathbf{A}_5 = \mathbf{A}_2^{-1}\mathbf{A}_1^{-1}\mathbf{A}_{hand}\mathbf{A}_6^{-1}. \tag{3}$$

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. 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:

$$EQ1: \quad c_3f_1 + s_3f_2 = c_2h_1 + s_2h_2 - a_2$$

$$EQ2: \quad s_3f_1 - c_3f_2 = -\lambda_2(s_2h_1 - c_2h_2) + \mu_2(h_3 - d_2)$$

$$
\begin{aligned}
EQ3: &\quad f_3 = \mu_2(s_2 h_1 - c_2 h_2) + \lambda_2(h_3 - d_2) \\
EQ4: &\quad c_3 r_1 + s_3 r_2 = c_2 n_1 + s_2 n_2 \\
EQ5: &\quad s_3 r_1 - c_3 r_3 = -\lambda_2(s_2 n_1 - c_2 n_2) + \mu_2 n_3 \\
EQ6: &\quad r_3 = \mu_3(s_2 n_1 - c_2 n_2) + \lambda_2 n_3, \qquad (4)
\end{aligned}
$$

where

$$
\begin{aligned}
f_1 &= c_4 g_1 + s_4 g_2 + a_3 \\
f_2 &= -\lambda_3(s_4 g_1 - c_4 g_2) + \lambda_3 g_3 \\
f_3 &= \mu_3(s_4 g_1 - c_4 g_2) + \lambda_3 g_3 + d_3
\end{aligned}
$$

$$
\begin{aligned}
r_1 &= c_4 m_1 + s_4 m_2 \\
r_2 &= -\lambda_3(s_4 m_1 - c_4 m_2) + \mu_3 m_3 \\
r_3 &= \mu_3(s_4 m_1 - c_4 m_2) + \lambda_3 m_3
\end{aligned}
$$

$$
\begin{aligned}
g_1 &= c_5 a_5 + a_4 \\
g_2 &= -s_5 \lambda_4 a_5 + \mu_4 d_5 \\
g_3 &= s_5 \mu_4 a_5 + \lambda_4 d_5 + d_4
\end{aligned}
$$

$$
\begin{aligned}
m_1 &= s_5 \mu_5 \\
m_2 &= c_5 \lambda_4 \mu_5 + \mu_4 \lambda_5 \\
m_3 &= -c_5 \mu_4 \mu_5 + \lambda_4 \lambda_5
\end{aligned}
$$

$$
\begin{aligned}
h_1 &= c_1 p + s_1 q - a_1 \\
h_2 &= -\lambda_1(s_1 p - c_1 q) + \mu_1(r - d_1) \\
h_3 &= \mu_1(s_1 p - c_1 q) + \lambda_1(r - d_1)
\end{aligned}
$$

$$
\begin{aligned}
n_1 &= c_1 u + s_1 v \\
n_2 &= -\lambda_1(s_1 u - c_1 v) + \mu_1 w \\
n_3 &= \mu_1(s_1 u - c_1 v) + \lambda_1 w
\end{aligned}
$$

$$
\begin{aligned}
p &= -l_x a_6 - (m_x \mu_6 + n_x \lambda_6) d_6 + q_x \\
q &= -l_y a_6 - (m_y \mu_6 + n_y \lambda_6) d_6 + q_y \\
r &= -l_z a_6 - (m_z \mu_6 + n_z \lambda_6) d_6 + q_z \\
u &= m_x \mu_6 + n_x \lambda_6 \\
v &= m_y \mu_6 + n_y \lambda_6 \\
w &= m_z \mu_6 + n_z . \lambda_6
\end{aligned}
$$

Let

$$\mathbf{h} = \begin{pmatrix} h_1 \\ h_2 \\ h_3 \end{pmatrix}, \quad \mathbf{f} = \begin{pmatrix} f_1 \\ f_2 \\ f_3 \end{pmatrix}, \quad \mathbf{n} = \begin{pmatrix} n_1 \\ n_2 \\ n_3 \end{pmatrix}, \quad \mathbf{r} = \begin{pmatrix} r_1 \\ r_2 \\ r_3 \end{pmatrix}$$

and the equations, EQ1–EQ6 can be rearranged to obtain 6 equations, $p_1, p_2, p_3, l_1, l_2, l_3$:

$$\mathbf{p} = \begin{pmatrix} p_1 \\ p_2 \\ p_3 \end{pmatrix} = \begin{pmatrix} c_2 & s_2 & 0 \\ s_2 & -c_2 & 0 \\ 0 & 0 & 1 \end{pmatrix} \mathbf{h} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & -\lambda_2 & \mu_2 \\ 0 & \mu_2 & \lambda_2 \end{pmatrix} \begin{pmatrix} c_3 & s_3 & 0 \\ s_3 & -c_3 & 0 \\ 0 & 0 & 1 \end{pmatrix} \mathbf{f} + \begin{pmatrix} a_2 \\ 0 \\ d_2 \end{pmatrix}$$

$$\mathbf{l} = \begin{pmatrix} l_1 \\ l_2 \\ l_3 \end{pmatrix} = \begin{pmatrix} c_2 & s_2 & 0 \\ s_2 & -c_2 & 0 \\ 0 & 0 & 1 \end{pmatrix} \mathbf{n} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & -\lambda_2 & \mu_2 \\ 0 & \mu_2 & \lambda_2 \end{pmatrix} \begin{pmatrix} c_3 & s_3 & 0 \\ s_3 & -c_3 & 0 \\ 0 & 0 & 1 \end{pmatrix} \mathbf{r}$$

It follows that 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. In a similar fashion 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 $\mathbf{p}$ and $\mathbf{l}$ as 3×1 vectors. According to [RR89], the left and right hand sides of the following equations have same power products as the left and right hand sides of p_i and l_i:

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

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

$$(\mathbf{Q}) \begin{pmatrix} s_1s_2 \\ s_1c_2 \\ c_1s_2 \\ c_1c_2 \\ s_1 \\ c_1 \\ s_2 \\ c_2 \end{pmatrix} = (\mathbf{P}) \begin{pmatrix} s_4s_5 \\ s_4c_5 \\ c_4s_5 \\ c_4c_5 \\ s_4 \\ c_4 \\ s_5 \\ c_5 \\ 1 \end{pmatrix}, \tag{6}$$

where $\mathbf{Q}$ 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). $\mathbf{P}$ 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, [RR89] obtain the relation:

$$(\mathbf{\Sigma}) \begin{pmatrix} s_4 s_5 \\ s_4 c_5 \\ c_4 s_5 \\ c_4 c_5 \\ s_4 \\ c_4 \\ s_5 \\ c_5 \\ 1 \end{pmatrix} = 0, \tag{7}$$

where $\mathbf{\Sigma}^4$ is 6×9 matrix, whose entries are linear combinations of s_3, c_3 and 1. Given (7), substitute

$$s_3 = \frac{2x_3}{1+x_3^2},\ c_3 = \frac{1-x_3^2}{1+x_3^2},\ s_4 = \frac{2x_4}{1+x_4^2},\ c_4 = \frac{1-x_4^2}{1+x_4^2},\ s_5 = \frac{2x_5}{1+x_5^2},\ c_5 = \frac{1-x_5^2}{1+x_5^2},$$

where $x_3 = tan(\frac{\theta_3}{2})$, $x_4 = tan(\frac{\theta_4}{2})$, $x_5 = tan(\frac{\theta_5}{2})$. After the substitution, multiply each equation by $(1+x_3^2)$, $(1+x_4^2)$ and $(1+x_5^2)$ to clear out the denominators and (7) can, therefore, be expressed as:

$$(\mathbf{\Sigma}^{'}) \begin{pmatrix} x_4^2 x_5^2 \\ x_4^2 x_5 \\ x_4^2 \\ x_4 x_5^2 \\ x_4 x_5 \\ x_4 \\ x_5^2 \\ x_5 \\ 1 \end{pmatrix} = 0, \tag{8}$$

where $(\mathbf{\Sigma}^{'})$ is 6×9 matrix, whose entries are quadratic polynomial in x_3. The system given above is not a square system and to convert it into a square system [RR89] use dialytic elimination. In particular, the equation expressed in (8) are multiplied by x_4 to obtain a

[4]We may obtain more than 6 equations after elimination. However, we choose any 6 of them in this matrix. More details are given in Section 6.4.

square system of the form

$$\begin{pmatrix} \mathbf{\Sigma}' & \mathbf{0} \\ \mathbf{0} & \mathbf{\Sigma}' \end{pmatrix} \begin{pmatrix} x_4^3 x_5^2 \\ x_4^3 x_5 \\ x_4^3 \\ x_4^2 x_5^2 \\ x_4^2 x_5 \\ x_4^2 \\ x_4 x_5^2 \\ x_4 x_5 \\ x_4 \\ x_5^2 \\ x_5 \\ 1 \end{pmatrix}, \tag{9}$$

where $\mathbf{0}$ is a 3×3 null matrix. Let

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

and $\mathbf{\Sigma}''$ is a 12×12 matrix whose entries are quadratic polynomials in x_3. Therefore, its determinant is a polynomial of degree 24 in x_3. Let us represent that polynomial as $R(x_3)$.

Lemma 2.1 *$(1 + x_3^2)^4$ divides $R(x_3)$.*

Proof: [RR89].

As a result, the degree 16 polynomial,

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

is the input–output polynomial, whose roots are used to compute the joint angle θ_3. Raghavan and Roth expand the determinant and use a root solver for computing the values of θ_3. Given θ_3, they solve 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.

2.3 Numerical Problems in Raghavan and Roth Solution

There are many computations in the solution highlighted above, which can have problems due to floating point arithmetic. For example, many properties of the ideal generated by $p_1, p_2, p_3, l_1, l_2, l_3$ may not hold due to floating point computation. These properties are being utilized while deriving the equations (5). Furthermore, computing the determinant of Σ'' can introduce significant numerical errors such that $(1 + x_3^2)^4$ may not exactly divide

the determinant. Finally, the computation of real roots of polynomials of degree 16 can be ill conditioned [Wil59, Wil63]. As a result, 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 roots of the polynomial. Many such examples are highlighted in [Wil59, Wil63]. For example, consider the polynomial

$$P(x) = \Sigma_{k=0}^{20} a_k x^k = \prod_{k=1}^{20} (x - k/20).$$

A small perturbation of relative magnitude 10^{-9} in a_{19} can induce a displacement of order unity in the larger roots of $P(x)$.

It is for this reason that the algorithm presented in [RR89] requires hundreds of digits of precision for portions of computations [WM91]. Most current workstations provide us with a hardware implementation of double precision arithmetic. It is possible to simulate higher order precision in software. However, that has a significant impact on the speed and the resulting algorithm becomes too slow for practical applications.

3 Matrix Computations

Many of the matrix computations used in the algorithm for inverse kinematics have been reviewed in the appendix. These include eigenvalues and eigenvectors of matrices, singular value decomposition, condition numbers of matrices, eigenvalues and eigenvectors, cluster of eigenvalues. We make use of these computations in the inverse kinematics algorithm presented in the next section.

4 Algorithm

In this section we describe our algorithm in detail. The initial steps in our algorithm make use of the results presented in [RR89]. 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:** Treat the a_i's, d_i's, λ_i's, μ_i's and the entries of the right hand side matrix $\mathbf{A}_{hand}$ as symbolic constants. As a result, express the entries of the 14×9 matrix $\mathbf{P}$ and 14×8 matrix $\mathbf{Q}$, as shown in equation (6), as functions of these symbolic constants. It corresponds to symbolic elimination and is performed using the properties highlighted in [RR89]. 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. The MAPLE program used in symbolic preprocessing for $6R$ manipulators is highlighted in the appendix at the end of this paper.

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 $\mathbf{P}$ and $\mathbf{Q}$, as shown in (6).

3. **Numerical Conditioning:** Compute the rank of $\mathbf{Q}$ using SVD. If $\mathbf{Q}$ 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 [SV89].

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

6. **Reduction to Eigenvalue Problem:** Reduce the problem of computing roots of, determinant$(\mathbf{\Sigma}'') = 0$, 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. **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.

4.1 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 $\mathbf{A}_{hand}$ (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 expressions. A major simplification is obtained by using the identities

$$sin^2(\alpha_i) + cos^2(\alpha_i) = 1, \quad sin^2(\theta_i) + cos^2(\theta_i) = 1.$$

The *simplify* command in MAPLE can perform this simplification. The left and right hand side of the 8 equations, shown in (5), are computed separately. Furthermore, we treat p, q, r, u, v, w in the equations (4) as symbolic constants. As a result, we obtain expressions as functions of these symbolic constants as opposed to $l_x, l_y, l_z, m_x, m_y, m_z, n_x, n_y, n_z, q_x, q_y, q_z$. After computing the 14 equations, EQ1–EQ6 and 8 equations shown in (5), we collect the terms as functions of sines and cosines of the joint angles θ_1 and θ_2 for the left hand sides of the equations and of the joint angles θ_3, θ_4 and θ_5 for the right hand side of the equations. All the constant terms from the left hand side of the equations are moved to the right hand sides. The coefficients of the equations are used to compute the entries of the matrices $\mathbf{P}$ and $\mathbf{Q}$. As a result, we are able to express the entries of $\mathbf{P}$ and $\mathbf{Q}$ 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 $\mathbf{P}$, 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 $\mathbf{Q}$ 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:

$$(\mathbf{Q}_1) \begin{pmatrix} s_1 \\ c_1 \end{pmatrix} = (\mathbf{P}_1) \begin{pmatrix} s_4 s_5 \\ s_4 c_5 \\ c_4 s_5 \\ c_4 c_5 \\ s_4 \\ c_4 \\ s_5 \\ c_5 \end{pmatrix}, \tag{11}$$

$$(\mathbf{Q}_2) \begin{pmatrix} s_1 s_2 \\ s_1 c_2 \\ c_1 s_2 \\ c_1 c_2 \\ s_2 \\ c_2 \end{pmatrix} = (\mathbf{P}_2) \begin{pmatrix} s_4 s_5 \\ s_4 c_5 \\ c_4 s_5 \\ c_4 c_5 \\ s_4 \\ c_4 \\ s_5 \\ c_5 \\ 1 \end{pmatrix}, \tag{12}$$

where $\mathbf{Q}_1, \mathbf{Q}_2, \mathbf{P}_1, \mathbf{P}_2$ are $6 \times 2, 8 \times 6, 6 \times 9, 8 \times 9$ matrices, respectively. The details of this formulation are given in [RR89]. In particular, we break the set of the 14 equations into sets of 6 and 8 equations. $\mathbf{Q}_1$, $\mathbf{Q}_2$ are minors of $\mathbf{Q}$ and $\mathbf{P}_1$, $\mathbf{P}_2$ are minors of $\mathbf{P}$.

The symbolic complexity of the entries of $\mathbf{P}_1, \mathbf{P}_2, \mathbf{Q}_1, \mathbf{Q}_2$ 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 by collecting terms with common subexpressions increases the efficiency and numerical accuracy of subsequent computations.

4.2 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 $\mathbf{P}_1, \mathbf{P}_2, \mathbf{Q}_1, \mathbf{Q}_2$. These entries are functions of p, q, r, u, v, w. While substituting the numerical entries, accuracy problems can arise due to catastrophic cancellation [GL89]. This happens when the number of significant digits (16 in the case of double precision arithmetic) is not enough for the accuracy of the result. We have tried many examples and never noticed this problem in our set of examples. However, the numerical accuracy can be improved by using higher precision arithmetic, implemented in software. 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 compute the values of p, q, r, u, v, w and substitute them to compute the entries of $\mathbf{P}_1, \mathbf{P}_2, \mathbf{Q}_1, \mathbf{Q}_2$. Let the corresponding numerical matrices (obtained after substitution) be $\overline{\mathbf{P}_1}, \overline{\mathbf{P}_2}, \overline{\mathbf{Q}_1}, \overline{\mathbf{Q}_2}$.

We use SVD to compute the ranks of $\overline{\mathbf{Q}_1}$ and $\overline{\mathbf{Q}_2}$. 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 $\overline{\mathbf{Q}_1}$ be expressed as:

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

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

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

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

$$(\mathbf{Q}_1^{'}) \begin{pmatrix} s_1 \\ c_1 \end{pmatrix} = (\mathbf{P}_1) \begin{pmatrix} s_4 s_5 \\ s_4 c_5 \\ c_4 s_5 \\ c_4 c_5 \\ s_4 \\ c_4 \\ s_5 \\ c_5 \end{pmatrix}. \tag{13}$$

We perform Gaussian elimination with complete pivoting on $\overline{\mathbf{Q}_1}$ and corresponding row and column operations are carried on the elements of $\mathbf{P}$. Depending on the rank of $\overline{\mathbf{Q}_1}$, 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 $\mathbf{\Sigma}$ in (7).

In a similar fashion we compute the rank of $\overline{\mathbf{Q}_2}$, as represented in (12). In case either of the singular values is close to 0.0, we recompute the matrix $\overline{\mathbf{Q}_2}$ from the singular value decomposition of $\overline{\mathbf{Q}_2}$. Otherwise $\overline{\mathbf{Q}_2}^{'} = \overline{\mathbf{Q}_2}$. The modified matrix is used in eliminating θ_1, θ_2 from (12). Depending on the rank of $\overline{\mathbf{Q}_2}$, we may obtain anywhere from 2 to 8 equations after elimination. Each equation corresponds to a row of Σ in (7).

The matrix $\mathbf{\Sigma}$ is a $p \times 9$ matrix, where $6 \leq p \leq 14$. 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 $P\mathbf{\Sigma_1}$ and back substitutes the solution into equations corresponding to $\mathbf{\Sigma_2}$. As a result, we solve for the system of equations represented by $\mathbf{\Sigma}$.

Given the 6×9 matrix $\mathbf{\Sigma_1}$, 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, $\mathbf{\Sigma}^{''}$, whose entries are quadratic polynomials in x_3.

4.3 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 efficiently computing its eigenvalues.

Given the 12×12 matrix, $\mathbf{\Sigma}^{''}$, and each of its entries is a quadratic polynomial in x_3.

Our problem is to solve the system of linear equations

$$\mathbf{\Sigma}^{''}\mathbf{v} = \mathbf{\Sigma}^{''} \begin{pmatrix} x_4^3x_5^2 \\ x_4^3x_5 \\ x_4^3 \\ x_4^2x_5^2 \\ x_4^2x_5 \\ x_4^2 \\ x_4x_5^2 \\ x_4x_5 \\ x_4 \\ x_5^2 \\ x_5 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}. \tag{14}$$

We express the matrix as

$$\mathbf{\Sigma}^{''} = \mathbf{A}x_3^2 + \mathbf{B}x_3 + \mathbf{C}, \tag{15}$$

where $\mathbf{A}, \mathbf{B}$ and $\mathbf{C}$ are 12×12 matrices consisting of numerical entries. We compute the condition number of $\mathbf{A}$. 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 [ABB+92]. If the matrix is singular, its condition number is infinity. Let us consider the case, when the matrix $\mathbf{A}$ is well conditioned. We take the matrix equation, (15), and multiply it by A^{-1}. Let

$$\overline{\mathbf{\Sigma}}^{''} = \mathbf{I}x_3^2 + \mathbf{A^{-1}B}x_3 + \mathbf{A^{-1}C},$$

where $\mathbf{I}$ is a 12×12 identity matrix. In practice $\mathbf{A^{-1}B}$ and $\mathbf{A^{-1}C}$ are computed by linear equation solvers. Given $\overline{\mathbf{\Sigma}}^{''}$, we use Theorem 1.1 [GLR82] to construct a 24×24 matrix $\mathbf{M}$ of the form

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

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

$$\mathbf{V} = \begin{pmatrix} \mathbf{v} \\ x_3\mathbf{v} \end{pmatrix}, \tag{16}$$

where $\mathbf{v}$ is the vector corresponding to the variables in (14). Thus, the eigenvectors of $\mathbf{M}$ can be used to compute the roots of the equations in (14).

Lets consider the case, when the matrix $\mathbf{A}$ 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, $x_3 = tan(\frac{\theta_3}{2}) \approx \infty$. Therefore, $\mathbf{A}$ is nearly singular. We take the matrix equation, (15), and

reduce it to a generalized eigenvalue problem by constructing two matrices, $\mathbf{M}_1$ and $\mathbf{M}_2$

$$\mathbf{M}_1 = \begin{pmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{A} \end{pmatrix}, \mathbf{M}_2 = \begin{pmatrix} \mathbf{0} & \mathbf{I} \\ -\mathbf{C} & -\mathbf{B} \end{pmatrix},$$

where $\mathbf{0}, \mathbf{I}$ are 12×12 null and identity matrices, respectively. Furthermore, the roots of determinant$(\mathbf{\Sigma}'') = 0$, correspond exactly to the eigenvalues of the generalized eigenvalue problem $\mathbf{M}_1 - x_3\mathbf{M}_2$ [GLR82]. 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

$$x_3 = \frac{a\overline{x}_3 + b}{c\overline{x}_3 + d}, \tag{17}$$

where a, b, c, d are random numbers. As a result of this transformation, (15) transforms into

$$\mathbf{\Sigma}_1'' = (a^2\ \mathbf{A} + ac\ \mathbf{B} + c^2\ \mathbf{C})\overline{x}_3^2 + (2ab\ \mathbf{A} + (ad+bc)\ \mathbf{B} + 2cd\ \mathbf{C})\overline{x}_3 + (b^2\ \mathbf{A} + bd\ \mathbf{B} + d^2\ \mathbf{C}). \tag{18}$$

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

$$\begin{pmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{A} \end{pmatrix} - \lambda \begin{pmatrix} \mathbf{0} & \mathbf{I} \\ -\mathbf{C} & -\mathbf{B} \end{pmatrix}$$

is a singular pencil. Such cases are possible, if the manipulator has less than 16 solutions.

$\mathbf{A}, \mathbf{B}, \mathbf{C}$ may have common singular pencils. In the latter case, $\overline{\mathbf{A}}$ 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 $\overline{\mathbf{A}}$. The cost of estimating condition number is rather small as compared to computing the eigendecomposition of the matrix. If $\overline{\mathbf{A}}$ is well conditioned, solve for determinant$(\mathbf{\Sigma}_1'') = 0$ 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 $\overline{x}_3$.

5 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.

5.1 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 2.1, 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 $\mathbf{M}$ of multiplicity 4 each, where $\iota = \sqrt{-1}$. If we transform the variable x_3, as shown in (17), these eigenvalues are suitably modified.

We use the structure of $\mathbf{M}$ in the eigenvalue algorithm. In the double shift QR algorithm we chose the shift value for the first few iterations to correspond to ι and $-\iota$. For example, in the single shift algorithm highlighted in (20),

$$\mathbf{H} - s\mathbf{I} = \mathbf{UR},$$

the upper triangular matrix $\mathbf{R}$ is singular if s corresponds to an eigenvalue of a matrix. In other words at most four iterations of the double shift algorithm reduce the problem to computing the eigenvalues of 16×16 matrix. The 16 eigenvalues of the latter matrix correspond exactly to the 16 solutions of $Q(x_3)$ in (10).

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.

Given the real Schur form, (21), 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} in (21). 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 an upper triangular matrix.

5.2 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 $\alpha_m = \frac{\alpha_1 + \alpha_2 + \ldots + \alpha_k}{k}$. In many cases it turns out that $\mid \alpha - \alpha_m \mid$ 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 [BDM89].

5.3 Eigenvector computation

The eigenvector corresponding to a real eigenvalue is computed by solving a quasi–upper triangular system [GL89]. 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 [Wil65]. 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 $\mathbf{V}$, 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

$$\mathbf{v}_1 = \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \\ v_7 \\ v_8 \\ v_9 \\ v_{10} \\ v_{11} \\ v_{12} \end{pmatrix} = \begin{pmatrix} x_4^3x_5^2 \\ x_4^3x_5 \\ x_4^3 \\ x_4^2x_5^2 \\ x_4^2x_5 \\ x_4^2 \\ x_4x_5^2 \\ x_4x_5 \\ x_4 \\ x_5^2 \\ x_5 \\ 1 \end{pmatrix}. \tag{19}$$

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.

5.4 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 $\mathbf{\Sigma_1}$. We substitute these solutions into the equation corresponding to the matrix $\mathbf{\Sigma_2}$. 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 $\mathbf{\Sigma}$.

Given a solution of $\mathbf{\Sigma}$, solve for s_1, c_1, s_2, c_2 from $\mathbf{Q}_1^{'}$ and $\mathbf{Q}_2^{'}$, as shown in (13). These solutions are substituted into (3) to compute θ_6.

5.5 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 in a few iterations.

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 eigenvalue algorithm highlighted above is used as the initial guess for the Newton's method. At each step of the iteration, we evaluate the functions and compute the jacobian. The improved solution is computed by solving a linear system of equations. This process is repeated till the residual is below a certain threshold, ϵ.

The jacobian is almost singular for solutions close to higher multiplicity roots. Modified versions of Newton's method to handle such cases are highlighted in [DK80, Kel86].

5.6 Performance

We have applied our algorithm to many examples. In particular, we used it on 21 problem instances given in [WM91] 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.

Example 5.1 *Let us consider the manipulator presented in [WM91] along with a pose of the end effector. This is problem 6 in [WM91] and corresponds to a slight variation of the manipulator presented in [MD89]. For this configuration the problem of inverse kinematics has* 16 *real solutions. The robot parameters are given in Table 5.1.*

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

$$\mathbf{A}_{hand} = \begin{pmatrix} -0.760117 & -0.641689 & 0.102262 & -1.140165 \\ 0.133333 & 0.0 & 0.991071 & 0.0 \\ -0.635959 & 0.766965 & -0.085558 & 0.0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Number	*Link Length*	*Offset Distance*	*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*

Table 1: The Denavit Hartenberg Parameters of a $6R$ manipulator

After substitution into the symbolic matrices, we obtain

$$\overline{\mathbf{Q}}_1 = \begin{pmatrix} -1.140174 & -0.0 \\ 0.0910474 & -0.990920 \\ -0.0 & 0.684105 \\ -0.297276 & -0.027314 \\ 0.0 & 0.1127979 \\ -0.1101672 & -1.377377 \end{pmatrix},$$

$$\overline{\mathbf{Q}}_2 = \begin{pmatrix} 0.0 & 0.0 & 0.0 & -1.1401 & -0.0 & -0.30 \\ 0.0 & 0.0 & -1.1401 & -0.0 & -0.30 & 0.0 \\ 0.0 & 0.9909 & 0.0 & 0.091 & 0.0989 & 0.0 \\ 0.990 & 0.0 & 0.091 & 0.0 & 0.0 & -0.0989 \\ -0.0273 & -0.1128 & 0.2972 & -0.0 & 1.129 & 0.0 \\ -0.1127 & 0.0273 & -0.0 & -0.2972 & 0.0 & -1.1298 \\ 0.0 & 1.199 & 0.0676 & -0.12655 & 0.1375 & -0.0623 \\ 1.1990 & 0.0 & -0.1265 & -0.0676 & -0.0622 & -0.1375 \end{pmatrix}.$$

The entries of $\mathbf{P}_1$ *and* $\mathbf{P}_2$ *are functions of* $s3$ *and* $c3$. $\mathbf{P}_1$ *is an* 6×9 *matrix,*

$$\overline{\mathbf{P}}_1 = \begin{pmatrix} 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & .304\text{e-}4 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 3.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.9998 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & -.0034 & -.0034 & 0.0 & 0.0 & 0.0 & 0.0 & -0.0436 & 0.0 \\ 0.0 & -0.399 & -0.399 & 0.0 & 0.0 & 0.0 & 0.0 & -2.998 & 0.0 \end{pmatrix} c3+$$

$$\begin{pmatrix} 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.026 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0174 & 0.0174 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & -1.7452 & 0.0 \\ 0.0 & -0.999 & -0.999 & 0.0 & 0.0 & 0.0 & 0.0 & 6.091\text{e-}5 & 0.0 \\ 0.0 & 0.1097 & -0.0211 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0174 & 0.0 \end{pmatrix} s3+$$

$$\begin{pmatrix} 0.0 & 0.0 & 0.0 & 0.0 & 1.499 & 0.0 & 0.0 & 0.0 & 0.199 \\ 0.999 & 0.0 & 0.0 & -0.999 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.6 & 0.0 & 0.0 & 0.0 & 1.9 \\ 0.2 & 0.0 & 0.0 & -0.1999 & 0.0 & 0.0 & 1.5 & 0.0 & 0.1038 \\ 0.0174 & 0.0 & 0.0 & -0.0436 & 0.0 & 0.0 & 0.0 & 0.0 & -0.0296 \\ -1.289 & 0.0 & 0.0 & -3.208 & 0.0 & 0.0 & -0.599 & 0.0 & 0.6778 \end{pmatrix}.$$

Similarly $\mathbf{P}_2$ *is a* 8×9 *matrix*

$$\overline{\mathbf{P}}_2 = \begin{pmatrix} 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.5 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.9998 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & -0.0174 & 0.0 \\ 0.0 & 0.0261 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 3.4904\text{e-}3 & 0.0 \\ 0.0 & 0.1999 & 0.1999 & 0.0 & 0.0 & 0.0 & 0.0 & 1.49924 & 0.0 \\ 0.0 & 1.289 & -3.21 & 0.0 & 0.0 & 0.0 & 0.0 & 0.5999 & 0.0 \\ 0.0 & -0.01745 & \text{-}6.980\text{e-}3 & 0.0 & 0.0 & 0.0 & 0.0 & -0.1097 & 0.0 \end{pmatrix} c3+$$

$$\begin{pmatrix} 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & -1.499 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & -0.0174 & 0.0 \\ 0.0 & -0.9996 & 0 - 0.9998 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0199 & 0.2 & 0.0 & 0.0 & 0.0 & 0.0 & 1.4997 & 0.0 \\ 0.0 & -0.0436 & -0.0174 & 0.0 & 0.0 & 0.0 & 0.0 & \text{-}3.489\text{e-}3 & 0.0 \\ 0.0 & -0.01047 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & -0.0225 & 0.0 \\ 0.0 & 3.288 & 1.209 & 0.0 & 0.0 & 0.0 & 0.0 & -0.5996 & 0.0 \end{pmatrix} s3+$$

$$\begin{pmatrix} 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0261 & 0.0 & 0.0 & 0.0 & 3.49\text{e-}3 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0174 & 0.0 & 0.0 & -0.01745 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ -0.9998 & 0.0 & 0.0 & 0.09992 & 0.0 & 0.0 & 0.0 & \text{-}3.489\text{e-}3 & 0.0 \\ -0.40 & 0.0 & 0.0 & 0.3999 & 0.0 & 0.0 & -3.0 & 0.0 & 0.0 \\ -0.0225 & 0.0 & 0.0 & -0.1083 & 0.0 & 0.0 & -0.0105 & 0.0 & 0.0 \end{pmatrix}.$$

The matrices $\overline{\mathbf{Q}}_1$ *and* $\overline{\mathbf{Q}}_2$ *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* $\mathbf{\Sigma}$ *given as:*

$$\mathbf{\Sigma} = \begin{pmatrix} 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 3.0 & 0.0 & 2.102\text{e-}4 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 3.0 & 0.0 & 2.1027\text{e-}4 & 0.0 \\ 0.0 & 0.9998\text{e-}1 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & \text{-}8.395\text{e-}6 & 0.0 \\ 0.0 & \text{-}3.489\text{e-}3 & \text{-}3.4904\text{e-}3 & 0.0 & 0.0 & 0.0 & 0.0 & \text{-}4.358\text{e-}2 & 0.0 \\ 0.0 & -0.3996 & -0.3999 & 0.0 & 0.0 & 0.0 & 0.0 & -2.998 & 0.0 \\ 0.0 & 7.998\text{e-}2 & -4.42 & 0.0 & 0.0 & -0.3114 & 0.0 & 0.5999 & 0.0 \\ 0.0 & \text{-}1.744\text{e-}2 & \text{-}6.980\text{e-}3 & 0.0 & 0.0 & \text{-}8.903\text{e-}2 & 0.0 & \text{-}8.8644\text{e-}2 & 0.0 \end{pmatrix} c3+$$

Num.	*Eigenvalue*	*Condition Num.*
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 2: Eigenvalues and their condition numbers

$$\begin{pmatrix} 0.0 & 1.204\text{e-}2 & 1.204\text{e-}2 & 0.0 & 0.0 & 1.443\text{e-}3 & 0.0 & 0.0 & 0.0 \\ 0.0 & 1.204\text{e-}2 & 1.204\text{e-}2 & 0.0 & 0.0 & 1.4430\text{e-}3 & 0.0 & 0.0 & 0.0 \\ 0.0 & -4.809\text{e-}4 & -4.810\text{e-}4 & 0.0 & 0.0 & -6.883\text{e-}3 & 0.0 & -1.745\text{e-}2 & 0.0 \\ 0.0 & -0.9972 & -0.99786 & 0.0 & 0.0 & 2.379\text{e-}4 & 0.0 & 6.0917\text{e-}5 & 0.0 \\ 0.0 & 8.550\text{e-}2 & -4.537\text{e-}2 & 0.0 & 0.0 & -5.435\text{e-}3 & 0.0 & 1.744\text{e-}2 & 0.0 \\ 0.0 & -1.047\text{e-}2 & 0.0 & 0.0 & 0.0 & -8.902\text{e-}2 & 0.0 & -1.396\text{e-}3 & 0.0 \\ 0.0 & -2.0784 & 2.419 & 0.0 & 0.0 & 0.3113 & 0.0 & -0.5996 & 0.0 \end{pmatrix} s3+$$

$$\begin{pmatrix} 0.6902 & 0.0 & 0.0 & -0.6901 & 0.68268 & 0.0 & 0.0 & 0.0 & 1.911 \\ 0.1724e-1 & 0.0 & 0.0 & -0.1724 & -0.3943 & 0.0 & 1.5 & 0.0 & 5.123\text{e-}2 \\ 0.1312 & 0.0 & 0.0 & -0.1574 & 1.363\text{e-}2 & 0.0 & 0.0 & 0.0 & -2.786\text{e-}2 \\ -2.679 & 0.0 & 0.0 & -1.8185 & -0.3113 & 0.0 & -0.5999 & 0.0 & 0.6363 \\ -0.4 & 0.0 & 0.0 & 0.3999 & 1.553\text{e-}3 & 0.0 & -3.0 & 0.0 & -0.2074 \\ -4.363\text{e-}2 & 0.0 & 0.0 & -8.724\text{e-}2 & -5.435\text{e-}3 & 0.0 & -1.047\text{e-}2 & 0.0 & -6.008\text{e-}2 \end{pmatrix}.$$

$\mathbf{\Sigma}$ *is converted into a matrix polynomial using the transformation* $x_3 = tan(\frac{\theta_3}{2})$ *and obtaining the* 12×12 *matrix* $\mathbf{\Sigma}''$*, expressed as a matrix polynomial in* x_3*. The estimated condition number of the leading matrix is* 5000.0*. 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*

Thus, we see that all the 16 *eigenvalues are real. Furthermore, they are computed up to*

i	θ_1	θ_2	θ_3	θ_4	θ_5	θ_6
1	-96.284531	-6.273561	179.968858	38.485979	52.550848	-39.404721
2	-120.788383	172.334376	-179.072836	31.331984	-146.715199	142.820883
3	88.678475	-176.724688	-176.729058	-63.241883	157.196191	140.436648
4	113.843614	5.306382	-177.744286	-55.924163	-62.984868	-43.377340
5	-178.126206	108.191647	-147.733832	-5.693263	-164.674567	179.580633
6	168.321914	-103.892172	146.603790	-17.240912	-171.879220	98.164928
7	-12.942930	-105.096318	-114.975385	3.023449	7.416983	-79.421763
8	2.517222	108.075883	112.043149	-10.522960	0.005115	-0.109419
9	2.517222	108.075883	-67.956851	-169.477040	179.994885	179.890581
10	-12.942930	-105.096318	65.024615	176.976551	172.583017	100.577967
11	168.321914	-103.892172	-33.396210	-162.759088	-8.120780	-81.834797
12	-178.126206	108.191647	32.266168	-174.306737	-15.325433	-0.419367
13	88.678475	-176.724688	3.270942	-116.758117	22.803809	-39.563352
14	-96.284531	-6.273561	-0.031142	141.514021	127.449152	140.595279
15	-120.788383	172.334376	0.927164	148.668016	-33.284801	-37.179117
16	113.843614	5.306382	2.255714	-124.075837	-117.015132	136.622660

Table 3: The joint angles corresponding to the solutions

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* $\mathbf{v}_1$ *equal to the last* 12 *elements of* $\mathbf{V}$ *the eigenvector, as shown in (19). Analyzing the elements of* $\mathbf{v}_1$ *results in* $\mid x_4 \mid < 1$ *and* $\mid x_5 \mid < 1$. *Elements of maximum magnitude of* $\mathbf{v}_1$ *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* $s1, s2, c1, c2$ *by solving a system of linear equations. Finally, these values are plugged into the original equations,* $EQ1 - -EQ6$ *to compute* $s6$ *and* $c6$.

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 and computing the residuals on* $EQ1 - -EQ6$. *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 6.3.*

More examples highlighting configurations with higher multiplicity solutions are highlighted in [MC92].

6 General Serial Manipulators

The techniques presented above can be extended to any serial manipulators with a finite number of solutions. 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 [RR92]. Our algorithm can be extended to all such manipulators. For each class of manipulator, different symbolic computations are performed by taking into account different joint variables. The rest of the numerical steps are similar.

The real time algorithm has also been used for computer animation and physical based modeling. More details are given in [MC92].

7 Conclusion

In this paper we presented a real time 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 milliseconds 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.

References

[AA79] 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.

[ABB+92] 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.* SIAM, PHILADELPHIA, 1992.

[BDM89] 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).

[DC80] J. Duffy and C. Crane. A displacement analysis of the general spatial 7r mechanism. *Mechanisms and Machine Theory*, 15:153–169, 1980.

[Dem89] J. Demmel. LAPACK: A portable linear algebra library for supercomputers. In *Proceedings of the 1989 IEEE Control Systems Society Workshop on Computer-Aided Control System Design*, Tampa, FL, Dec 1989. IEEE.

[DH55] J. Denavit and R.S. Hartenberg. A kinematic notation for lower–pair mechanisms based upon matrices. *Journal of Applied Mechanics*, 77:215–221, 1955.

[DK80] D. W. Decker and C. T. Kelley. Newton's method at singular points I. *SIAM J. Num. Anal.*, 17:66–70, 1980.

[GBDM77] B.S. Garbow, J.M. Boyle, J. Dongarra, and C.B. Moler. *Matrix Eigensystem Routines – EISPACK Guide Extension*, volume 51 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1977.

[GL89] G.H. Golub and C.F. Van Loan. *Matrix Computations*. John Hopkins Press, Baltimore, 1989.

[GLR82] I. Gohberg, P. Lancaster, and L. Rodman. *Matrix Polynomials*. Academic Press, New York, 1982.

[Gol91] D. Goldberg. What every computer scientist should know about floating point arithmetic. *ACM Computing Surveys*, 23(1), 1991.

[Hay83] S.A. Hayati. Robot arm geometric link calibration. In *IEEE Control and Decision Conference*, pages 1477–1483, 1983.

[Kel86] C. T. Kelley. A Shamanskii-like acceleration scheme for nonlinear equations at singular roots. *Math. Comp.*, 47:609–623, 1986.

[LL88a] H.Y. Lee and C.G. Liang. Displacement analysis of the general spatial 7-link 7r mechanism. *Mechanisms and Machine Theory*, 23(3):219–226, 1988.

[LL88b] 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.

[MC92] D. Manocha and J.F. Canny. Real time inverse kinematics of general 6r manipulators. Technical Report ESRC 92-2, RAMP 92-1, Engineering System Research Center, University of California, Berkeley, 1992.

[MD89] 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.

[Pie68] D. Pieper. *The kinematics of manipulators under computer control*. PhD thesis, Stanford University, 1968.

[Pri86] E.J.F. Primrose. On the input–output equation of the general 7r–mechanism. *Mechanisms and Machine Theory*, 21:509–510, 1986.

[RR89] 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.

[RR92] M. Raghavan and B. Roth. Inverse kinematics of the general 6r manipulator and related linkages. *Transactions of ASME, Journal of Mechanical Design*, 1992. To appear.

[RRS73] 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.

[SSN86] H.W. Stone, A.C. Sanderson, and C.P. Neuman. Arm signature identification. In *IEEE Conference on Robotics and Automation*, pages 41–48, 1986.

[SV89] M.W. Spong and M. Vidyasagar. *Robot Dynamics and Control*. John Wiley and Sons, 1989.

[TM85] 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.

[VW87] W.K. Veitschegger and C. Wu. A method for calibrating and compensating robot kinematic errors. In *IEEE Conference on Robotics and Automation*, pages 39–43, 1987.

[Wil59] 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.

[Wil63] J.H. Wilkinson. *Rounding Errors in Algebraic Processes*. Prentice-Hall, Englewood Cliffs, New Jersey, 1963.

[Wil65] J.H. Wilkinson. *The algebraic eigenvalue problem*. Oxford University Press, Oxford, 1965.

[WL84] D.E. Whitney and C.A. Lozinski. Industrial robot calibration methods and results. In *Proceedings of the International Computers in Engineering Conference*, pages 92–100, 1984.

[WM91] 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.

8 Appendix

In this section we review the matrix computations used in our kinematics algorithm and also present a sample of the MAPLE program used in symbolic preprocessing for the $6R$ manipulator algorithm.

8.1 Matrix Computations

In this section we present techniques from linear algebra and numerical analysis. We also highlight the numerical accuracy of the problems and the algorithm used to solve those problems in terms of their *condition numbers*.

8.2 Hessenberg Matrix

A Hessenberg matrix is of the form

$$\mathbf{H} = \begin{pmatrix} h_{11} & h_{12} & h_{13} & \dots & h_{1n} \\ h_{21} & h_{22} & h_{23} & \dots & h_{2n} \\ 0 & h_{32} & h_{33} & \dots & h_{3n} \\ 0 & 0 & h_{43} & \dots & h_{4n} \\ \vdots & \vdots & \vdots & \dots & \vdots \\ 0 & \dots & 0 & h_{n,n-1} & h_{nn} \end{pmatrix}.$$

In other words it is like an upper triangular matrix, except all that the subdiagonal elements may be non-zero. Given a matrix $\mathbf{A}$, it can be converted into a Hessenberg matrix using similarity transformations of the form $\mathbf{QAQ^{-1}}$, where $\mathbf{Q}$ is an orthogonal matrix. $\mathbf{Q}$ is an orthogonal matrix if $\mathbf{QQ}^T = \mathbf{I}$.

8.3 QR Factorization

The QR factorization of an $m \times n$ matrix $\mathbf{A}$ is given by

$$\mathbf{A} = \mathbf{QR},$$

where $\mathbf{Q}$ is an $m \times m$ orthogonal matrix and $\mathbf{R}$ is an $m \times n$ upper triangular matrix. More details on its computations are given in [GL89].

8.4 Singular Value Decomposition

The singular value decomposition (SVD) is a powerful tool which gives us accurate information about matrix rank in the absence of round off errors. The rank of a matrix can also be computed by Gauss elimination. However, there arise many situations where near rank deficiency prevails. Rounding errors and fuzzy data make rank determination a non–trivial exercise. In these situations, the numerical rank is easily characterized in terms of the SVD.

Given $\mathbf{A}$, a $m \times n$ real matrix then there exist orthogonal matrices $\mathbf{U}$ and $\mathbf{V}$ such that

$$\mathbf{A} = \mathbf{U}\Sigma\mathbf{V}^T$$

where $\mathbf{U}$ is a $m \times n$ orthogonal matrix, $\mathbf{V}$ is $n \times n$ orthogonal matrix and $\mathbf{\Sigma}$ is a $n \times n$ diagonal matrix of the form

$$\mathbf{\Sigma} = \text{diag}(\sigma_1, \sigma_2, \ldots, \sigma_n).$$

Moreover, $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_n \geq 0$. The σ_i's are called the *singular values* and columns of $\mathbf{U}$ and $\mathbf{V}$, denoted as u_i's and v_j's, are known as the left and right singular vectors, respectively [GL89]. The relationship between the elements of $\mathbf{A}$, singular values and singular vectors can be expressed as:

$$\mathbf{A}_{ij} = \Sigma_{k=1}^{n} \sigma_k \mathbf{U}_{ik} \mathbf{V}_{jk},$$

where $\mathbf{A}_{ij}, \mathbf{U}_{ij}, \mathbf{V}_{ij}$ represent the element in the ith row and jth column of $\mathbf{A}, \mathbf{U}$ and $\mathbf{V}$, respectively.

The singular values give accurate information about the rank of the matrix. The matrix $\mathbf{A}$ has rank $k < n$, if $\sigma_{k+1} = 0,\ \sigma_{k+2} = 0, \ldots, \sigma_n = 0$. Furthermore, the smallest positive singular value gives us information about the closeness to a rank deficient matrix [GL89].

8.5 Eigenvalues and Eigenvectors

Given a $n \times n$ matrix $\mathbf{A}$, its eigenvalues and eigenvectors are the solutions to the equation

$$\mathbf{A}\mathbf{x} = \lambda\mathbf{x},$$

where λ is the eigenvalue and $\mathbf{x} \neq \mathbf{0}$ is the eigenvector. The eigenvalues of a matrix are the roots of its characteristic polynomial determinant$(\mathbf{A} - \lambda\mathbf{I}) = 0$. As a result, the eigenvalues of a diagonal matrix, upper triangular matrix or a lower triangular matrix correspond to the elements on its diagonal. Efficient algorithms for computing eigenvalues and eigenvectors are well known, [GL89], and their implementations are available as part of packages EISPACK, [GBDM77], and LAPACK [Dem89, ABB+92]. Most algorithms make use of the similarity transformations of the form $\mathbf{A}' = \mathbf{Q}\mathbf{A}\mathbf{Q}^{-1}$, where $\mathbf{Q}$ is any non–singular $n \times n$ matrix. This transformation has the characteristic that the eigenvalues of $\mathbf{A}$ and $\mathbf{A}'$ are identical. Furthermore, if $\mathbf{y}$ is an eigenvector of $\mathbf{A}'$, $\mathbf{Q}^{-1}\mathbf{y}$ is an eigenvector of $\mathbf{A}$. Standard algorithms for eigenvalue computations, like the QR algorithm, choose $\mathbf{Q}$ to be an orthogonal matrix, since similarity transformation by an orthogonal matrix is a numerically stable operation [GL89]. Given $\mathbf{A}$ the eigendecomposition algorithm converts it into a Hessenberg matrix using a sequence of similarity transformations by orthogonal matrices. That is,

$$\mathbf{H} = \mathbf{Q}^T\mathbf{A}\mathbf{Q},$$

where $\mathbf{Q}$ is an orthogonal matrix and $\mathbf{H}$ is an Hessenberg matrix. Given $\mathbf{H}$, the eigendecomposition algorithm proceeds by similarity transformations by orthogonal matrices. Each of these similarity transformation corresponds to a QR iteration of the form:

$$\mathbf{H} - s\mathbf{I} = \mathbf{U}\mathbf{R}, \tag{20}$$

where s is a scalar referred to as a shift, $\mathbf{U}$ is an orthogonal matrix and $\mathbf{R}$ is an upper triangular matrix. This step corresponds to QR factorization of the matrix $\mathbf{H} - s\mathbf{I}$. Given $\mathbf{U}$ and $\mathbf{R}$, the next step of the iteration computes a modified Hessenberg matrix given by

$$\mathbf{H} = \mathbf{RU} + s\mathbf{I}.$$

The shifts are chosen appropriately such that the matrix converges to its to its *real Schur decomposition* of the form [GL89, Wil65]:

$$\mathbf{QAQ^{-1}} = \begin{pmatrix} R_{11} & R_{12} & \dots & R_{1m} \\ 0 & R_{22} & \dots & R_{2m} \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & R_{mm} \end{pmatrix}, \tag{21}$$

where each R_{ii} is either a 1×1 matrix or a 2×2 matrix having complex conjugate eigenvalues. Given the real Schur decomposition, computing the eigenvalues is a trivial operation. Many a times a matrix has complex eigenvalues, the above algorithm is modified to double shift consisiting of a complex number and its conjugate. More details are given in [GL89]. We will use the QR algorithm with *double implicit shift strategy* to compute the real Schur decomposition. Given the matrix eigenvalues, real Schur decomposition and matrix $\mathbf{Q}$, computing eigenvectors corresponds to solving quasi triangular systems [GL89, Wil65]. The running time of these algorithms is $O(n^3)$. However, the constant in front of n^3 can be as high as 25 for computing all the eigenvalues and eigenvectors.

8.6 Generalized Eigenvalue Problem

Given $n \times n$ matrices, $\mathbf{A}$ and $\mathbf{B}$, the generalized eigenvalue problem corresponds to solving

$$\mathbf{Ax} = \lambda\mathbf{Bx}.$$

We represent this problem as eigenvalues of $\mathbf{A} - \lambda\mathbf{B}$. The vectors $\mathbf{x} \neq \mathbf{0}$ correspond to the eigenvectors of this equation. If $\mathbf{B}$ is non–singular and its condition number (defined in the next section) is low, the problem can be reduced to an eigenvalue problem by multiplying both sides of the equation by $\mathbf{B^{-1}}$ and thereby obtaining:

$$\mathbf{B^{-1}Ax} = \lambda\mathbf{x}.$$

However, $\mathbf{B}$ may have a high condition number and such a reduction can cause numerical problems. Algorithms for the generalized eigenvalue problems apply orthogonal transformations to $\mathbf{A}$ and $\mathbf{B}$. In particular, we use the QZ algorithm for computing the eigenvalues and eigenvectors for this problem [GL89]. Its running time is $O(n^3)$. However, the constant can be as high as 75. Generally, it is slower by a factor of 2.5 to 3 as compared to QR algorithm for computing eigenvalues and eigenvectors of a matrix.

8.7 Condition Numbers

The condition number of a problem measures the sensitivity of a solution to small changes in the input. A problem is *ill–conditioned* if its condition number is large, and *ill–posed* if its condition number is infinite. These condition numbers are used to bound errors in computed solutions of numerical problems. More details on condition numbers are given in [GL89, Wil65]. The implementations of these condition number computations are available as part of LAPACK [BDM89].

In our algorithm, we will be performing computations like linear equation solving and computing eigenvalues and eigenvectors of a matrix. Therefore, we will be concerned with the numerical accuracy of these operations.

8.8 Condition Number of a Square Matrix

The *condition number of a square matrix* corresponds to $\frac{\sigma_1(\mathbf{A})}{\sigma_n(\mathbf{A})}$, where σ_1 and σ_n are the largest and smallest singular values. This condition number is used in determining the accuracy of $\mathbf{A}^{-1}$ computation or solving linear systems of the form $\mathbf{Ax} = \mathbf{b}$. Computing the singular values takes $O(n^3)$ time, which is rather expensive. Good estimators of $O(n^2)$ complexity, once $\mathbf{Ax} = \mathbf{b}$ has been solved via Gaussian elimination, are available in LINPACK and LAPACK and we use them in our algorithm.

8.9 Condition Number of Simple Eigenvalues

Let λ be a simple[5] eigenvalue of the $n \times n$ matrix, $\mathbf{A}$, with unit right eigenvector $\mathbf{x}$ and unit left eigenvector $\mathbf{y}$. That is, $\mathbf{Ax} = \lambda\mathbf{x}$, $\mathbf{y}^T\mathbf{A} = \lambda\mathbf{y}^T$ and $\| \mathbf{x} \|_2 = \| \mathbf{y} \|_2 = 1$. Here $\| \mathbf{v} \|_2$ stands for the 2–norm of a vector. Let $\mathbf{P} = (\mathbf{x} \cdot \mathbf{y}^T)/(\mathbf{y}^T \cdot \mathbf{x})$ be the spectral projector. Therefore, $\| P \|_2 = \frac{1}{|\mathbf{y}^T\mathbf{x}|}$. Let $\mathbf{E}$ be a perturbation of $\mathbf{A}$, and $\epsilon_2 = \| \mathbf{E} \|_2$. Moreover, let λ' be the perturbed eigenvalue of $\mathbf{A} + \mathbf{E}$. Then

$$| \lambda' - \lambda | \leq \epsilon_2 \| \mathbf{P} \|_2 + O(\epsilon_2^2).$$

Thus, for sufficiently small perturbations in the matrix, the perturbation in the eigenvalues is a function of $\| \mathbf{P} \|_2$.

8.10 Condition Number of Clustered Eigenvalues

In many cases we are interested in computing the condition numbers of a cluster of eigenvalues. We use these condition numbers in determining the accuracy of eigenvalues with multiplicity greater than one. We represent the real Schur decomposition as

$$\mathbf{A}' = \begin{pmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ 0 & \mathbf{A}_{22} \end{pmatrix}$$

[5] A simple eigenvalue is an eigenvalue of multiplicity one.

and the eigenvalues of the $m \times m$ matrix $\mathbf{A}_{11}$ are exactly those we are interested in. In particular we are interested in bounding the perturbation in the average of the eigenvalues of the cluster, represented as $\overline{\lambda} = \text{trace}(\mathbf{A}_{11})/m$.

To compute the error bound, we define the spectral projector

$$\mathbf{P} = \begin{pmatrix} \mathbf{I}_m & \mathbf{R} \\ \mathbf{0} & \mathbf{0} \end{pmatrix},$$

where $\mathbf{R}$ satisfies the system of linear equations

$$\mathbf{A}_{11}\mathbf{R} - \mathbf{R}\mathbf{A}_{22} = \mathbf{A}_{12}.$$

Thus, $\| \mathbf{P} \|_2 = (1 + \| \mathbf{R} \|_2^2)^{1/2}$. Computing $\| \mathbf{P} \|_2$ is expensive and a cheaper overestimate is obtained as

$$\| \mathbf{P} \|' = (1 + \| \mathbf{R} \|_F^2)^{1/2}.$$

Let $\mathbf{E}$ be the perturbation of $\mathbf{A}$ and $\epsilon_2 = \| \mathbf{E} \|_2$. Let $\overline{\lambda}'$ be the average of the perturbed eigenvalues. Then

$$| \overline{\lambda} - \overline{\lambda}' | \leq \epsilon_2 \| \mathbf{P} \|_2 + O(\epsilon_2^2).$$

We substitute $\| \mathbf{P} \|'$ to obtain a slightly weaker bound on the perturbation in $\overline{\lambda}$ for sufficiently small ϵ_2. The average of a cluster is often much better conditioned than individual eigenvalues in the cluster.

8.11 Accuracy of Right Eigenvectors

As far as eigenvectors are concerned, bounds for their accuracy are given in detail in [Wil65, ABB+92]. However, we will not be computing these bounds to analyze the accuracy of our computation. The actual bounds tell us about the maximum error in any term of the eigenvector. We only assume that each term of the eigenvector has a similar bound on the absolute error. Thus, the terms of maximum magnitude have the smallest bound on their relative error.

8.12 Symbolic Preprocessing Program

```
# MAPLE program for symbolic preprocessing of inverse kinematics.
# Given a n-jointed robot manipulator, with any combination of joints
# (prismatic or revolute), and the variables defining the position
# of end-effector (six of them), this program eliminates five of the joint
# variables. In other words it is computing the sparse resultant of the
# given system of equations.

# In the following example the symbolic derivation of the entries of
# matrices P and Q for general 6R is presented. Each entry of P and
```

Q is a function of the robot manipulator parameters and the variables
representing the position of the end effector. A lot of properties
related to the geometry of the manipulator can be interpreted from
the linear algebra structure of P and Q.
We use the Denavit-Hartenberg notation.

with(linalg);
readlib(evalm);
readlib(write);

X1, ..., X6 are the joint angles. They are the variable, which will be
eliminated.

$C1 := cos(X1);\quad C2 := cos(X2);\quad C3 := cos(X3);$
$C4 := cos(X4);\quad C5 := cos(X5);\quad C6 := cos(X6);$
$S1 := sin(X1);\quad S2 := sin(X2);\quad S3 := sin(X3);$
$S4 := sin(X4);\quad S5 := sin(X5);\quad S6 := sin(X6);$

Y1, ..., Y6 are the twist angles.
$M1 := sin(Y1);\quad M2 := sin(Y2);\quad M3 := sin(Y3);$
$M4 := sin(Y4);\quad M5 := sin(Y5);\quad M6 := sin(Y6);$
$L1 := cos(Y1);\quad L2 := cos(Y2);\quad L3 := cos(Y3);$
$L4 := cos(Y4);\quad L5 := cos(Y5);\quad L6 := cos(Y6);$

lx,ly,lz,mx,my,mz,nx,ny,nz correspond to the entries of the orthogonal # matrix defining the orientation of the end-effector.
$u := mx * M6 + nx * L6;$
$v := my * M6 + ny * L6;$
$w := mz * M6 + nz * L6;$

$p := -lx * a6 - (mx * M6 + nx * L6) * d6 + qx;$
$q := -ly * a6 - (my * M6 + ny * L6) * d6 + qy;$
$r := -lz * a6 - (mz * M6 + nz * L6) * d6 + qz;$

$b1 := C1 * u + S1 * v;$
$b2 := -L1 * (S1 * u - C1 * v) + M1 * w;$
$b3 := M1 * (S1 * u - C1 * v) + L1 * w;$

$h1 := C1 * p + S1 * q - a1;$
$h2 := -L1 * (S1 * p - C1 * q) + M1 * (r - d1);$
$h3 := M1 * (S1 * p - C1 * q) + L1 * (r - d1);$

$m1 := S5 * M5;$
$m2 := C5 * L4 * M5 + M4 * L5;$
$m3 := -C5 * M4 * M5 + L4 * L5;$

$g1 := C5 * a5 + a4;$
$g2 := -S5 * L4 * a5 + M4 * d5;$
$g3 := S5 * M4 * a5 + L4 * d5 + d4;$

$r1 := C4 * m1 + S4 * m2;$
$r2 := -L3 * (S4 * m1 - C4 * m2) + M3 * m3;$
$r3 := M3 * (S4 * m1 - C4 * m2) + L3 * m3;$

$f1 := C4 * g1 + S4 * g2 + a3;$
$f2 := -L3 * (S4 * g1 - C4 * g2) + M3 * g3;$
$f3 := M3 * (S4 * g1 - C4 * g2) + L3 * g3 + d3;$

$LM := array([[C3, S3, 0], [S3, -C3, 0], [0, 0, 1]]);$
$RM := array([[C2, S2, 0], [S2, -C2, 0], [0, 0, 1]]);$
$CM := array([[1, 0, 0], [0, -L2, M2], [0, M2, L2]]);$
$F := array([f1, f2, f3]);$
$H := array([h1, h2, h3]);$
$R := array([r1, r2, r3]);$
$B := array([b1, b2, b3]);$
$C := array([a2, 0, d2]);$
$PL := multiply(RM, H);$
$PR := add(multiply(multiply(CM, LM), F), C);$
$LL := multiply(RM, B);$
$LR := multiply(multiply(CM, LM), R);$

These are the left and right hand sides of the six equations # defining the kinematics problem.
$Leq1 := PL[1];$ $Leq2 := PL[2];$ $Leq3 := PL[3];$
$Req1 := PR[1];$ $Req2 := PR[2];$ $Req3 := PR[3];$
$Leq4 := LL[1];$ $Leq5 := LL[2];$ $Leq6 := LL[3];$
$Req4 := LR[1];$ $Req5 := LR[2];$ $Req6 := LR[3];$

Using dot and cross products of 6 equations to derive the rest of # the 8 equations.
$temp := dotprod(PL, PL);$ $Leq7 := simplify(temp);$
$temp := dotprod(PR, PR);$ $Req7 := simplify(temp);$
$temp := dotprod(PL, LL);$ $Leq8 := simplify(temp);$
$temp := dotprod(PR, LR);$ $Req8 := simplify(temp);$
$vtemp := crossprod(PL, LL);$ $Leq9 := simplify(vtemp[1]);$

$Leq10 := simplify(vtemp[2]);\quad Leq11 := simplify(vtemp[3]);$
$vtemp1 := crossprod(PR, LR);\quad Req9 := simplify(vtemp1[1]);$
$Req10 := simplify(vtemp1[2]);\quad Req11 := simplify(vtemp1[3]);$
$temp1 := scalarmul(LL, Leq7);\quad temp2 := -2 * Leq8;$
$temp3 := scalarmul(PL, temp2);\quad vtemp2 := add(temp1, temp3);$
$Leq12 := simplify(vtemp2[1]);\quad Leq13 := simplify(vtemp2[2]);$
$Leq14 := simplify(vtemp2[3]);$

$temp1 := scalarmul(LR, Req7);\quad temp2 := -2 * Req8;$
$temp3 := scalarmul(PR, temp2);\quad vtemp3 := add(temp1, temp3);$
$Req12 := simplify(vtemp3[1]);\quad Req13 := simplify(vtemp3[2]);$
$Req14 := simplify(vtemp3[3]);$

#
Collecting various coefficients from the 14 equations.
$lmat := array(1..14, 1..9);$

The following set of calls is repeated for the right hand side of
each equation, Req1–Req14.
As a result the matrix coefficients are being computed.

$row := 1;\quad exp := expand(Req1);\quad exp := collect(exp, cos(X3));$
$temp := coeff(exp, cos(X3), 2);\quad exp := exp - temp * cos(X3) * cos(X3);$
$temp := coeff(exp, cos(X3), 3);$
$exp := exp - temp * cos(X3) * cos(X3) * cos(X3);$
$exp := collect(exp, cos(X4));\quad temp := coeff(exp, cos(X4), 2);$
$exp := exp - temp * cos(X4) * cos(X4);\quad temp := coeff(exp, cos(X4), 3);$
$exp := exp - temp * cos(X4) * cos(X4) * cos(X4);$
$exp := collect(exp, cos(X5));$
$temp := coeff(exp, cos(X5), 2);\quad exp := exp - temp * cos(X5) * cos(X5);$
$temp := coeff(exp, cos(X5), 3);$
$exp := exp - temp * cos(X5) * cos(X5) * cos(X5);$
$exp := collect(exp, sin(X3));\quad temp := coeff(exp, sin(X3), 2);$
$exp := exp - temp * sin(X3) * sin(X3);\quad temp := coeff(exp, sin(X3), 3);$
$exp := exp - temp * sin(X3) * sin(X3) * sin(X3);$
$exp := collect(exp, sin(X4));$
$temp := coeff(exp, sin(X4), 2);\quad exp := exp - temp * sin(X4) * sin(X4);$
$temp := coeff(exp, sin(X4), 3);$
$exp := exp - temp * sin(X4) * sin(X4) * sin(X4);$
$exp := collect(exp, sin(X5));\quad temp := coeff(exp, sin(X5), 2);$
$exp := exp - temp * sin(X5) * sin(X5);\quad temp := coeff(exp, sin(X5), 3);$
$exp := exp - temp * sin(X5) * sin(X5) * sin(X5);\quad exp := expand(exp);$

$exp := collect(exp, sin(X4));\quad exp1 := coeff(exp, sin(X4));$
$exp2 := expand(exp - exp1 * sin(X4));\quad exp1 := collect(exp1, sin(X5));$
$exp3 := coeff(exp1, sin(X5));\quad exp4 := expand(exp1 - exp3 * sin(X5));$
$exp4 := collect(exp4, cos(X5));\quad exp5 := coeff(exp4, cos(X5));$
$exp6 := expand(exp4 - exp5 * cos(X5));\quad exp2 := collect(exp2, cos(X4));$
$exp7 := coeff(exp2, cos(X4));\quad exp8 := expand(exp2 - exp7 * cos(X4));$
$exp7 := collect(exp7, sin(X5));\quad exp9 := coeff(exp7, sin(X5));$
$exp10 := expand(exp7 - exp9 * sin(X5));\quad exp10 := collect(exp10, cos(X5));$
$exp11 := coeff(exp10, cos(X5));\quad exp12 := expand(exp10 - exp11 * cos(X5));$
$exp8 := collect(exp8, cos(X5));\quad exp13 := coeff(exp8, cos(X5));$
$exp14 := expand(exp8 - exp13 * cos(X5));\quad exp14 := collect(exp14, sin(X5));$
$exp14 := coeff(exp14, sin(X5));$
$lmat[row, 1] := exp3;\quad lmat[row, 2] := exp5;\quad lmat[row, 3] := exp9;$
$lmat[row, 4] := exp11;\quad lmat[row, 5] := exp6;\quad lmat[row, 6] := exp12;$
$lmat[row, 7] := exp14;\quad lmat[row, 8] := exp13;$
$check := expand(Req1 - lmat[row, 1] * S4 * S5 - lmat[row, 2] * S4 * C5 -$
$lmat[row, 3] * C4 * S5 - lmat[row, 4] * C4 * C5 - lmat[row, 5] * S4 - lmat[row, 6] *$
$C4 - lmat[row, 7] * S5 - lmat[row, 8] * C5);\quad lmat[row, 9] := check;$

The following set of calls is repeated for the left hand side of
each equation, Leq1–Leq14.
As a result the matrix coefficients are being computed.

$rmat := array(1..14, 1..9);$

$row := 1;\quad exp := expand(Leq1);\quad exp := collect(exp, sin(X1));$
$exp1 := coeff(exp, sin(X1));\quad exp2 := expand(exp - exp1 * sin(X1));$
$exp1 := collect(exp1, sin(X2));\quad exp3 := coeff(exp1, sin(X2));$
$exp4 := expand(exp1 - exp3 * sin(X2));\quad exp4 := collect(exp4, cos(X2));$
$exp5 := coeff(exp4, cos(X2));\quad exp6 := expand(exp4 - exp5 * cos(X2));$
$exp2 := collect(exp2, cos(X1));\quad exp7 := coeff(exp2, cos(X1));$
$exp8 := expand(exp2 - exp7 * cos(X1));\quad exp7 := collect(exp7, sin(X2));$
$exp9 := coeff(exp7, sin(X2));\quad exp10 := expand(exp7 - exp9 * sin(X2));$
$exp10 := collect(exp10, cos(X2));\quad exp11 := coeff(exp10, cos(X2));$
$exp12 := expand(exp10 - exp11 * cos(X2));\quad exp8 := collect(exp8, cos(X2));$
$exp13 := coeff(exp8, cos(X2));\quad exp14 := expand(exp8 - exp13 * cos(X2));$
$exp14 := collect(exp14, sin(X2));\quad exp14 := coeff(exp14, sin(X2));$
$rmat[row, 1] := exp3;\quad rmat[row, 2] := exp5;\quad rmat[row, 3] := exp9;$
$rmat[row, 4] := exp11;\quad rmat[row, 5] := exp6;\quad rmat[row, 6] := exp12;$
$rmat[row, 7] := exp14;\quad rmat[row, 8] := exp13;$
$check := expand(Leq1 - rmat[row, 1] * S1 * S2 - rmat[row, 2] * S1 * C2 -$
$rmat[row, 3] * C1 * S2 - rmat[row, 4] * C1 * C2 - rmat[row, 5] * S1 - rmat[row, 6] * C1$

$-rmat[row,7]*S2-rmat[row,8]*C2);$
$rmat[row,9]:=check;$

We break up the linear system of 14 equations into two different sets
of equations.

$Rsys1:=array(1..8,1..8);$
$Rsys2:=array(1..6,1..8);$
$Lsys1:=array(1..8,1..9);$
$Lsys2:=array(1..6,1..9);$

$for\ i\ from\ 1\ by\ 1\ to\ 8\ do$
$for\ j\ from\ 1\ by\ 1\ to\ 8\ do$
$temp:=rmat[i,j];$
$Rsys1[i,j]:=temp;$
$temp:=lmat[i,j];$
$temp:=collect(temp,cos(X3));$
$temp:=collect(temp,sin(X3));$
$Lsys1[i,j]:=temp;$
$od;$
$temp:=lmat[i,9]-rmat[i,9];$
$temp:=collect(temp,cos(X3));$
$temp:=collect(temp,sin(X3));$
$Lsys1[i,9]:=temp;$
$od;$

$for\ i\ from\ 9\ by\ 1\ to\ 14\ do$
$for\ j\ from\ 1\ by\ 1\ to\ 8\ do$
$temp:=rmat[i,j];$
$Rsys2[i-8,j]:=temp;$
$temp:=lmat[i,j];$
$temp:=collect(temp,cos(X3));$
$temp:=collect(temp,sin(X3));$
$Lsys2[i-8,j]:=temp;$
$od;$
$temp:=lmat[i,9]-rmat[i,9];$
$temp:=collect(temp,cos(X3));$
$temp:=collect(temp,sin(X3));$
$Lsys2[i-8,9]:=temp;$
$od;$

$Rsim1:=array(1..6,1..2);$

$Rsim2 := array(1..8, 1..6);$
$Lsim1 := array(1..6, 1..9);$
$Lsim2 := array(1..8, 1..9);$

$Rsim2[1,1] := Rsys1[1,1];$ $Rsim2[1,2] := Rsys1[1,2];$ $Rsim2[1,3] := Rsys1[1,3];$
$Rsim2[1,4] := Rsys1[1,4];$ $Rsim2[1,5] := Rsys1[1,7];$ $Rsim2[1,6] := Rsys1[1,8];$
$Rsim2[2,1] := Rsys1[2,1];$ $Rsim2[2,2] := Rsys1[2,2];$ $Rsim2[2,3] := Rsys1[2,3];$
$Rsim2[2,4] := Rsys1[2,4];$ $Rsim2[2,5] := Rsys1[2,7];$ $Rsim2[2,6] := Rsys1[2,8];$
$Rsim2[3,1] := Rsys1[4,1];$ $Rsim2[3,2] := Rsys1[4,2];$ $Rsim2[3,3] := Rsys1[4,3];$
$Rsim2[3,4] := Rsys1[4,4];$ $Rsim2[3,5] := Rsys1[4,7];$ $Rsim2[3,6] := Rsys1[4,8];$
$Rsim2[4,1] := Rsys1[5,1];$ $Rsim2[4,2] := Rsys1[5,2];$ $Rsim2[4,3] := Rsys1[5,3];$
$Rsim2[4,4] := Rsys1[5,4];$ $Rsim2[4,5] := Rsys1[5,7];$ $Rsim2[4,6] := Rsys1[5,8];$
$Rsim2[5,1] := Rsys2[1,1];$ $Rsim2[5,2] := Rsys2[1,2];$ $Rsim2[5,3] := Rsys2[1,3];$
$Rsim2[5,4] := Rsys2[1,4];$ $Rsim2[5,5] := Rsys2[1,7];$ $Rsim2[5,6] := Rsys2[1,8];$
$Rsim2[6,1] := Rsys2[2,1];$ $Rsim2[6,2] := Rsys2[2,2];$ $Rsim2[6,3] := Rsys2[2,3];$
$Rsim2[6,4] := Rsys2[2,4];$ $Rsim2[6,5] := Rsys2[2,7];$ $Rsim2[6,6] := Rsys2[2,8];$
$Rsim2[7,1] := Rsys2[4,1];$ $Rsim2[7,2] := Rsys2[4,2];$ $Rsim2[7,3] := Rsys2[4,3];$
$Rsim2[7,4] := Rsys2[4,4];$ $Rsim2[7,5] := Rsys2[4,7];$ $Rsim2[7,6] := Rsys2[4,8];$
$Rsim2[8,1] := Rsys2[5,1];$ $Rsim2[8,2] := Rsys2[5,2];$ $Rsim2[8,3] := Rsys2[5,3];$
$Rsim2[8,4] := Rsys2[5,4];$ $Rsim2[8,5] := Rsys2[5,7];$ $Rsim2[8,6] := Rsys2[5,8];$

$for\ i\ from\ 1\ by\ 1\ to\ 9\ do$
$Lsim2[1,i] := Lsys1[1,i];$
$od;$
$for\ i\ from\ 1\ by\ 1\ to\ 9\ do$
$Lsim2[2,i] := Lsys1[2,i];$
$od;$
$for\ i\ from\ 1\ by\ 1\ to\ 9\ do$
$Lsim2[3,i] := Lsys1[4,i];$
$od;$
$for\ i\ from\ 1\ by\ 1\ to\ 9\ do$
$Lsim2[4,i] := Lsys1[5,i];$
$od;$
$for\ i\ from\ 1\ by\ 1\ to\ 9\ do$
$Lsim2[5,i] := Lsys2[1,i];$
$od;$
$for\ i\ from\ 1\ by\ 1\ to\ 9\ do$
$Lsim2[6,i] := Lsys2[2,i];$
$od;$
$for\ i\ from\ 1\ by\ 1\ to\ 9\ do$
$Lsim2[7,i] := Lsys2[4,i];$

od;
$for\ i\ from\ 1\ by\ 1\ to\ 9\ do$
$Lsim2[8,i] := Lsys2[5,i]$;
od;

$Rsim1[1,1] := Rsys1[3,5]$; $Rsim1[1,2] := Rsys1[3,6]$; $Rsim1[2,1] := Rsys1[6,5]$;
$Rsim1[2,2] := Rsys1[6,6]$; $Rsim1[3,1] := Rsys1[7,5]$; $Rsim1[3,2] := Rsys1[7,6]$;
$Rsim1[4,1] := Rsys1[8,5]$; $Rsim1[4,2] := Rsys1[8,6]$; $Rsim1[5,1] := Rsys2[3,5]$;
$Rsim1[5,2] := Rsys2[3,6]$; $Rsim1[6,1] := Rsys2[6,5]$; $Rsim1[6,2] := Rsys2[6,6]$;

$for\ i\ from\ 1\ by\ 1\ to\ 9\ do$
$Lsim1[1,i] := Lsys1[3,i]$;
od;
$for\ i\ from\ 1\ by\ 1\ to\ 9\ do$
$Lsim1[2,i] := Lsys1[6,i]$;
od;
$for\ i\ from\ 1\ by\ 1\ to\ 9\ do$
$Lsim1[3,i] := Lsys1[7,i]$;
od;
$for\ i\ from\ 1\ by\ 1\ to\ 9\ do$
$Lsim1[4,i] := Lsys1[8,i]$;
od;
$for\ i\ from\ 1\ by\ 1\ to\ 9\ do$
$Lsim1[5,i] := Lsys2[3,i]$;
od;
$for\ i\ from\ 1\ by\ 1\ to\ 9\ do$
$Lsim1[6,i] := Lsys2[6,i]$;
od;

The 6 X 2 matrix, Rsim1, corresponds exactly to Q_1, a minor of Q.
Each entry of Rsim1 is a function of constants like manipulator parameters
and the right hand side variables.

The 8 X 6 matrix, Rsim2, corresponds exactly to Q_2, a minor of Q.
Each entry of Rsim1 is a function of constants like manipulator parameters
and the right hand side variables.

The 6 X 9 matrix, Lsim1, corresponds exactly to P_1, a minor of P.
Each entry of Lsim1 is a linear function of S3 and C3. The coeffcients
are functions of the constants.

The 8 X 9 matrix, Lsim2, corresponds exactly to P_2, a minor of P.

```
# Each entry of Lsim2 is a linear function of S3 and C3. The coefficients
# are functions of the constants.
```