TM-57

GENERAL-PURPOSE INVERSE KINEMATICS TRANSFORMATIONS FOR ROBOTIC MANIPULATORS

by

M. H. Ang, Jr. and V. D. Tourassis

PRODUCTION AUTOMATION PROJECT

November 1986
Technical Memorandum No. 57

PRODUCTION AUTOMATION PROJECT
College of Engineering & Applied Science
The University of Rochester
Rochester, New York 14627

TM-57

GENERAL-PURPOSE INVERSE KINEMATICS
TRANSFORMATIONS FOR ROBOTIC MANIPULATORS

by

M. H. Ang, Jr. and V. D. Tourassis

Submitted to the Journal of Robotic Systems

The work described in this report is supported by the Department of Electrical Engineering and by companies in the P.A.P.'s Industrial Associates Program. Any opinions, findings, conclusions, or recommendations expressed in this report are those of the authors and do not necessarily reflect the views of the industrial sponsors or the University of Rochester.
TABLE OF CONTENTS

1. Introduction ................................................................. 1
2. Robot Kinematics ............................................................ 2
3. Kinematics of Position ...................................................... 4
   3.1 Introduction ............................................................ 4
   3.2 Forward Recursion ..................................................... 4
   3.3 Inverse Recursion .................................................... 5
   3.4 Computational Requirements ......................................... 7
4. Kinematics of Velocity ..................................................... 9
   4.1 Introduction ............................................................ 9
   4.2 Forward Kinematics ................................................... 9
   4.3 Jacobian Transformations .......................................... 10
   4.4 Inverse Kinematics ................................................ 10
5. Kinematics of Acceleration .............................................. 12
   5.1 Introduction ............................................................ 12
   5.2 Forward Kinematics ................................................ 12
   5.3 Computational Requirements ..................................... 17
   5.4 Inverse Kinematics ................................................ 17
6. The Complete Kinematic Algorithm .................................. 18
7. Comparative Evaluation .................................................. 19
   8.1 Introduction ............................................................ 19
   8.2 Kinematics of Position .......................................... 19
   8.3 Kinematics of Velocity ............................................ 20
   8.4 Kinematics of Acceleration ..................................... 20
8. Conclusions ..................................................................... 23
Appendix A ......................................................................... 24
References .......................................................................... 29
ABSTRACT

Real-time robot control requires efficient inverse kinematics transformations to compute the temporal evolution of the joint coordinates from the motion of the end-effector. The development of a coherent, general-purpose framework, incorporating position, velocity and acceleration transformations, is the theme of this paper. In this framework, the computational requirements of a new inverse kinematic algorithm are delineated. The algorithm is applicable to serial (open-chain) manipulators with arbitrary axes of motion. Comparative evaluations of the computational cost of the algorithm demonstrate its efficacy and feasibility for real-time applications.
1. INTRODUCTION

Robotic manipulators are articulated chains of rigid bodies (links) which are connected serially by joints. The initial end of the chain (the base) is attached firmly to a support and the final end (the end-effector) is free to move to accomplish manipulation tasks. Each joint signifies a physical axis along (or around) which motion can occur. Robot joints are actuated independently and the joint motion produces the relative motion of the links.

Industrial robots are primarily positioning mechanisms. Each joint is equipped with a dedicated processor to control axial position. The robot control problem centers around the design of a stable and robust algorithm to coordinate joint motion and enable the robot to position and orient its end-effector accurately in the workspace. The robot control problem is complicated by the practical fact that the desired motion of the end-effector is frequently described in a cartesian coordinate frame while the servo control system requires the reference inputs be specified in joint coordinates.

The description of the end-effector motion in joint coordinates is the subject of robot kinematics. In robotics, we distinguish between forward and inverse kinematics [1]. Forward kinematics deals with the problem of determining the position of the end-effector from a given set of joint coordinates. General-purpose forward kinematic algorithms provide a unique solution to this problem and are straightforward to apply. In contrast, inverse kinematics addresses the problem of computing the temporal evolution of the joint coordinates from the motion of the end-effector. Inverse kinematics transformations are generally complex and depend upon the manipulator geometry. The lack of universal approaches to solve the inverse kinematics problem complicates task planning and has led to the development of special-purpose inverse kinematics algorithms for position and velocity transformations.

Recent developments in the area of nonlinear feedback robot control [2,3,4,5] call for inverse kinematics algorithms that incorporate acceleration transformations. Inverse kinematic transformations for acceleration have been developed for specific manipulators that employ spherical wrists [6,7,8]. There is a need, however, for (i) a coherent, general-purpose inverse kinematics framework to accommodate position, velocity, and acceleration transformations; and (ii) modular kinematics algorithms that exploit the synergism between kinematic transformations and robot dynamics computations.
The overriding constraint for all inverse kinematics transformations is computational efficiency. In many industrial applications, the desired motion of the end-effector is not known in advance. The transformation from the cartesian coordinates to the joint coordinates must then be computed on-line. This real-time kinematic transformation imposes a significant computational burden on the control computer [9] and creates the need for efficient kinematic algorithms and special-purpose controller architectures.

The objectives of this paper are to:

- develop an efficient inverse kinematics algorithm that incorporates position, velocity, and acceleration transformations in a general-purpose framework; and
- delineate the computational requirements of the algorithm for real-time robot control.

This paper is organized as follows. In Section 2, we review robot kinematics and the associated nomenclature. In Sections 3, 4, and 5, we present the kinematic transformations for position, velocity, and acceleration, respectively. Then, in Section 6, we summarize our algorithm and enumerate its computational requirements. In Section 7, we present a comparative evaluation of the computational cost of the algorithm. Finally, in Section 8, we highlight the contributions of the paper and identify areas for future research.

2. ROBOT KINEMATICS

Robotic manipulators are articulated, open kinematic chains of $N$ rigid bodies (links) which are connected serially by $N$ joints. The links are numbered consecutively from the base (link 0) to the final end (link $N$). The joints are the points of articulation between the links and are numbered so that joint $i$ connects links $(i - 1)$ and $i$. To describe the geometry of robot motion, we assign a cartesian coordinate frame $(x_i, y_i, z_i)$ to each link, where $i = 0, 1, 2, \ldots, N$, according to the Denavit-Hartenberg convention [10]. Accordingly, each link translates along or rotates around the $z$-axis of the coordinate frame assigned to the previous link. The first link moves with respect to the $z$-axis of the zero (or base) coordinate frame $(x_0, y_0, z_0)$ which serves as a reference frame. The origin of the inertial frame $(x_0, y_0, z_0)$ can be placed anywhere in the supporting base and the origin of the last coordinate frame $(x_N, y_N, z_N)$ is specified by the geometry of the end-effector.

We proceed to model the kinematics of robotic mechanism in terms of the joint coordinates. The joint coordinate $q_i$ is the angular displacement around $z_{i-1}$ if joint $i$ is rota-
tional, or the linear displacement along \( z_{i-1} \) if joint \( i \) is translational. The \( N \)-dimensional space defined by the joint coordinates \( (q_1, \ldots, q_N) \) is called the configuration space of the mechanism.

![Denavit-Hartenberg Convention for Link Frames](image)

Figure 2-1: The Denavit-Hartenberg Convention for Link Frames

In the Denavit-Hartenberg convention, the relative position and orientation of two adjacent link coordinate frames can be described by an ordered sequence of four elementary rotations and translations (Figure 2-1):

- Rotation by an angle \( \theta_i \) (in the right-handed sense) about the \( z_{i-1} \) axis, so that the \( x_{i-1} \) axis is parallel to the \( x_i \) axis.

- Translation by a distance of \( r_i \) along the positive direction of the \( z_{i-1} \) axis, to align the \( x_{i-1} \) axis with the \( x_i \) axis.

- Translation by a distance of \( d_i \) along the positive direction of the \( x_{i-1} = x_i \) axis, to coalesce the origins \( O_{i-1} \) and \( O_i \).

- Rotation by an angle \( \alpha_i \) (in the right-handed sense) about the \( x_{i-1} = x_i \) axis, to coalesce the two coordinate systems.

The \( i \)-th coordinate frame is therefore characterized by the four Denavit-Hartenberg
kinematic link parameters $\theta_i, r_i, d_i, \alpha_i$. If joint $i$ is rotational, then $q_i = \theta_i$, and $a_i, d_i$ and $r_i$ are constant geometric parameters; if joint $i$ is translational, then $q_i = r_i$, and $d_i, a_i$ and $\theta_i$ are constant geometric parameters. For both rotational and translational joints, $r_i$ and $\theta_i$ are the distance and angle between links $(i-1)$ and $i$; $d_i$ and $\alpha_i$ are the length and twist of link $i$.

The Denavit-Hartenberg parameters have been used extensively to define homogeneous transformations between link coordinate frames [11,12]. Homogeneous transformations are described by $(4 \times 4)$ matrices which can represent rotations and translations. Accordingly, position vectors are written as $(4 \times 1)$ columns with the addition of a scaling factor. Homogeneous transformations, however, include unnecessary operations (such as multiplications by 0 and 1) and are not computationally efficient. Throughout this paper, we use geometrical $(3 \times 1)$ vectors rather than the $(4 \times 1)$ vectors of the homogeneous transformation formalism.

3. KINEMATICS OF POSITION

3.1 Introduction

In this section, we present an efficient kinematic algorithm for computing the position and orientation of the end-effector from the joint coordinates. This recursive algorithm is essential in the development of the kinematics of velocity and acceleration. The algorithm can be implemented in two ways: forward (spatial recursion from the base to the end-effector) and inverse (spatial recursion from the end-effector to the base).

3.2 Forward Recursion

In the forward implementation, the objective is to describe the position and orientation of the end-effector frame in the base frame. The inputs are the joint coordinates $q_1, q_2, \ldots, q_N$ and the geometric (Denavit-Hartenberg) parameters of each link. The outputs are the axes of the $N$-th frame $(x_N, y_N, z_N)$ and the position $p_N$ of the origin $O_N$ described in the base frame $(x_0, y_0, z_0)$.

The forward implementation consists of a set of recursive formulae which describe the position and orientation of each link frame in the base frame. In the forward implementation, all vectors are expressed in the base frame. The recursive formulae are based upon the four elementary operations performed on the $(i-1)$-th coordinate frame to coalesce it with the $i$-th frame (Section 2).
We rotate by $\theta_i$ about $z_{i-1}$ to align the $x_{i-1}$ axis with the $x_i$ axis. Let $(x'_{i-1}, y'_{i-1}, z'_{i-1})$ be the intermediate frame that results from this rotation (Figure 2.1). Then:

$$x'_{i-1} = x_i = x_{i-1} \cos \theta_i + y_{i-1} \sin \theta_i$$ (3.1)

$$y'_{i-1} = y_{i-1} \cos \theta_i - x_{i-1} \sin \theta_i$$ (3.2)

$$z'_{i-1} = z_{i-1}.$$ (3.3)

We translate the origin $O_{i-1}$ by $r_i$ along the $z'_{i-1} = z_{i-1}$ axis and by $d_i$ along $x'_{i-1} = x_i$. Thus,

$$p_i = p_{i-1} + r_i z_{i-1} + d_i x_i.$$ (3.4)

Finally, we rotate by $\alpha_i$ about $x'_{i-1} = x_i$ to align $z'_{i-1}$ with $z_i$. Therefore,

$$z_i = z'_{i-1} \cos \alpha_i - y'_{i-1} \sin \alpha_i.$$ (3.5)

$$y_i = z'_{i-1} \sin \alpha_i + y'_{i-1} \cos \alpha_i.$$ (3.6)

Substituting equations (3.2) and (3.3) into (3.5) yields

$$z_i = z_{i-1} \cos \alpha_i + (x_{i-1} \sin \theta_i - y_{i-1} \cos \theta_i) \sin \alpha_i.$$ (3.7)

Similarly, substituting equations (3.2) and (3.3) into (3.6) yields

$$y_i = z_{i-1} \sin \alpha_i - (x_{i-1} \sin \theta_i - y_{i-1} \cos \theta_i) \cos \alpha_i.$$ (3.8)

The $y_i$ axis, however, can be computed more efficiently by completing the right-handed frame:

$$y_i = z_i \times x_i.$$ (3.9)

A program logic for the forward recursions (along with the associated initial conditions) is provided in Table 3.1.

3.3 Inverse Recursion

Occasionally, the robot task is described in end-effector coordinates. It is then necessary to define the position and orientation of the base frame in the end-effector frame (inverse implementation). The inputs are the joint coordinates $q_1, q_2, \ldots, q_N$ and the geometric (Denavit-Hartenberg) parameters of each link. The outputs are the axes of the
base frame \((x_0, y_0, z_0)\) and the position \(p_0\) of the origin \(O_0\) described in the end-effector frame \((x_N, y_N, z_N)\).

The inverse implementation consists of a set of recursive formulae which describe the position and orientation of each link frame in the end-effector frame. In the inverse implementation, all vectors are expressed in the end-effector frame. The recursive formulae are based upon four elementary operations performed on the \(i\)-th coordinate frame to coalesce it with the \((i-1)\)-th coordinate frame; these operations mirror the elementary transformations highlighted in Section 2.

We rotate by \((-\alpha_i)\) about \(x_i\) to align the \(z_i\) axis with the \(z_{i-1}\) axis. Let \((x'_{i-1}, y'_{i-1}, z'_{i-1})\) be the intermediate frame that results from this rotation (Figure 2-1). Then:

\[
x'_{i-1} = x_i
\]
\[
y'_{i-1} = y_i \cos \alpha_i - z_i \sin \alpha_i
\]
\[
z'_{i-1} = z_{i-1} = y_i \sin \alpha_i + z_i \cos \alpha_i.
\]

We translate the origin \(O_i\) by \((-d_i)\) along the \(x'_{i-1} = x_i\) axis and by \((-r_i)\) along \(z'_{i-1} = z_{i-1}\). Thus,

\[
p_{i-1} = p_i - d_i x_i - r_i z_{i-1}
\]

Finally, we rotate by \((-\theta_i)\) about \(z'_{i-1} = z_{i-1}\) to align \(x'_{i-1}\) with \(x_{i-1}\). Therefore,

\[
x_{i-1} = x'_{i-1} \cos \theta_i - y'_{i-1} \sin \theta_i
\]
\[
y_{i-1} = x'_{i-1} \sin \theta_i + y'_{i-1} \cos \theta_i.
\]

Substituting equations (3.10) and (3.11) into (3.14) yields

\[
x_{i-1} = x_i \cos \theta_i + (z_i \sin \alpha_i - y_i \cos \alpha_i) \sin \theta_i
\]

Similarly, substituting equations (3.10) and (3.11) into (3.15) yields

\[
y_{i-1} = x_i \sin \theta_i + (y_i \cos \alpha_i - z_i \sin \alpha_i) \cos \theta_i.
\]

The \(y_{i-1}\) axis, however, can be computed more efficiently by completing the right-handed frame:

\[
y_{i-1} = z_{i-1} \times x_{i-1}
\]
A program logic for the inverse recursions (along with the associated initial conditions) is provided in Table 3-2.

3.4 Computational Requirements

We enumerate the multiplications \((m)\), additions \((a)\) required by the forward recursions in Table 3-1. (We elaborate upon transcendental function calls in Section 7.) The computational cost of each forward recursion is evaluated as follows.

- **Orientation**: \([23m + 12a]\)
- **Position**: \([6m + 6a]\)
- **Total**: \([29m + 18a]\)

The first recursion, however, requires only 6 multiplications and the second recursion requires only 27 multiplications and 16 additions, as can be seen from the initial conditions in Table 3-1. Therefore, the total computational cost for an \(N\)-joint manipulator is \([29N - 25)m + (18N - 20)a]\).

The inverse recursions (in Table 3-2) exhibit identical computational requirements.
1. INITIALIZE $x_0 = (1 \ 0 \ 0)^T$; $y_0 = (0 \ 1 \ 0)^T$; $z_0 = (0 \ 0 \ 1)^T$; $p_0 = 0$.

2. $i \leftarrow 1$.

3. IF $i > N$, GO TO STEP 11.

4. READ $\theta_i, r_i, d_i, \alpha_i$.

5. $x_i = x_{i-1} \cos \theta_i + y_{i-1} \sin \theta_i$.

6. $z_i = z_{i-1} \cos \alpha_i + x_{i-1} \sin \theta_i \sin \alpha_i - y_{i-1} \cos \theta_i \sin \alpha_i$.

7. $y_i = z_i \times x_i$.

8. $p_i = p_{i-1} + r_i z_{i-1} + d_i x_i$.

9. $i \leftarrow i + 1$.

10. GO TO STEP 3.

11. END

Table 3-1: Program Logic for the Forward Recursions

1. INITIALIZE $x_N = (1 \ 0 \ 0)^T$; $y_N = (0 \ 1 \ 0)^T$; $z_N = (0 \ 0 \ 1)^T$; $p_N = 0$.

2. $i \leftarrow N$.

3. IF $i < 1$, GO TO STEP 11.

4. READ $\theta_i, r_i, d_i, \alpha_i$.

5. $x_{i-1} = x_i \cos \theta_i + z_i \sin \alpha_i \sin \theta_i - y_i \cos \alpha_i \sin \theta_i$.

6. $z_{i-1} = z_i \cos \alpha_i + y_i \sin \alpha_i$.

7. $y_{i-1} = z_{i-1} \times x_{i-1}$.

8. $p_{i-1} = p_i - d_i x_i - r_i z_{i-1}$.

9. $i \leftarrow i - 1$.

10. GO TO STEP 3.

11. END

Table 3-2: Program Logic for the Inverse Recursions
4. KINEMATICS OF VELOCITY

4.1 Introduction

In this section, we develop the transformations needed to relate the velocity of the end-effector to the joint velocities. As in the previous section, we distinguish between forward and inverse transformations.

4.2 Forward Kinematics

The objective of the forward transformation is to describe the velocity of the end-effector in a reference frame. The motion of the end-effector is the result of the coordinated motion of each robot joint. The end-effector velocity is thus a function of the joint velocities. The end-effector velocity, \( v_N \), is expressed as a \((6 \times 1)\) vector that incorporates the \((3 \times 1)\) translational velocity \( u_N \) and the \((3 \times 1)\) rotational velocity \( w_N \) vectors of the end-effector. All vectors are expressed in the same reference frame.

The total velocity of the end-effector during coordinated motion is the superposition of all the elementary velocities that represent single joint motion:

\[
v_N = \begin{pmatrix} u_N \\ w_N \end{pmatrix} = J(q)\dot{q}.
\]  
(4.1)

where \( J(q) = (J_1 J_2 \cdots J_N) \) is the \((6 \times N)\) Jacobian matrix and \( \dot{q} = (\dot{q}_1, \dot{q}_2, \cdots, \dot{q}_N)^T \) is the \((N \times 1)\) vector of joint velocities.

The \((6 \times 1)\) columns of the Jacobian are computed depending upon the type of the joint [9]. For a translational joint \( i \):

\[
J_i = \begin{pmatrix} z_{i-1} \\ 0 \end{pmatrix},
\]  
(4.2)

and for a rotational joint \( i \):

\[
J_i = \begin{pmatrix} z_{i-1} \times (p_N - p_{i-1}) \\ z_{i-1} \end{pmatrix}.
\]  
(4.3)

We observe that the computation of the Jacobian matrix, and of the velocity of the end-effector, requires knowledge of the \((z_0, z_1, \ldots, z_{N-1})\) axes and the position vectors \((p_0, p_1, \ldots, p_N)\) expressed in the reference coordinate frame. Once the orientation and position of each frame is known, the computation of the Jacobian matrix involves, at most, vector cross-products.
4.3 Jacobian Transformations

The choice of a reference frame for the computation of the Jacobian matrix depends upon the task description. Typically, the robot task is specified in the robot base frame and the velocity of the end-effector is defined in the 0-th coordinate frame: $\dot{v}_N = J\ddot{q}$. The computation of the Jacobian matrix $J$ requires knowledge of the $(z_0, z_1, \ldots, z_{N-1})$ axes and the position vectors $(p_0, p_1, \ldots, p_N)$ expressed in the robot base frame. These $(3 \times 1)$ vectors have been computed during the forward recursions of the kinematics of position in Section 3.2. The computational cost for a column of the Jacobian corresponding to a translational joint is zero. The computational cost for a column of the Jacobian corresponding to a rotational joint is $[6m + 6a]$. The first column of $J$ requires no computation, as can be seen from the initial conditions in Table 3-1. The total computational cost for the Jacobian of an $N$-joint robot with all rotational joints (worst case) is therefore $[(6N - 6)m + (6N - 6)a]$.

If the robot trajectory is described in the end-effector coordinates, the velocity of the end-effector is defined in the $N$-th coordinate frame: $v_N = NJ\ddot{q}$. It is then necessary to transform the Jacobian matrix $NJ$ to its counterpart $J$ in the base coordinate frame. This transformation involves the axes $(x_0, y_0, z_0)$ of the base coordinate frame expressed in the end-effector frame [9]:

$$
0J = \begin{pmatrix}
x_0^T & \vdots \\
y_0^T & \vdots & 0 \\
z_0^T & \vdots & \vdots \vdots \\
\vdots & \vdots & \vdots \vdots \\
\vdots & x_0^T & \vdots \\
0 & y_0^T & \vdots \\
\vdots & z_0^T
\end{pmatrix} \quad N\ J
$$

(4.4)

The computational cost of this additional transformation is $[(18N)m + (12N)a]$.

4.4 Inverse Kinematics

The objective of the inverse transformation is to compute the joint velocities required to achieve a desired end-effector velocity. The inverse transformation is the solution of the system of 6 linear equations in (4.1) for the $N$ unknowns $(\dot{q}_1, \dot{q}_2, \ldots, \dot{q}_N)$. Accordingly, we consider three cases.
i) $N = 6$

The solution of the system can be represented symbolically as:

$$
\dot{q} = J^{-1}v_N
$$

(4.5)

The elements of the Jacobian matrix are complex trigonometric functions of the joint coordinates, prohibiting symbolic inversion except for trivial manipulators [10]. At any time instant, however, the Jacobian is an array of 36 numbers. Since the inverse of the Jacobian matrix is not of interest, numerical techniques, such as Gaussian elimination, can be employed to solve the system of 6 linear equations in $N = 6$ unknowns in (4.1). The total computational cost for Gaussian elimination is $\frac{1}{2}(N^3 + 3N^2 - 4N + 6)m + \frac{1}{6}(2N^3 + 3N^2 - 5N + 6)a$, or, $[102m + 86a]$ for $N = 6$ [13].

The solution of the linear system in (4.1) is possible for joint configurations for which the Jacobian matrix is invertible (nonsingular). A singularity occurs at manipulator positions that correspond to physical boundaries of the robot workspace or configurations at which two or more joint axes align [14]. Singular positions, therefore, are manipulator dependent and can be avoided in the planning stage [15].

ii) $N < 6$

In this case we have a linear system with more equations than unknowns. Such a system is overdetermined and it may or may not have a solution. The solution, if it exists, can be represented symbolically as:

$$
\dot{q} = J_l J_N \iff J^T v_N = (J^T J)q
$$

(4.6)

where $J_l = (J^T J)^{-1}J^T$ is the $(N \times 6)$ left pseudo-inverse of the Jacobian matrix [16]. Numerical techniques, such as Gaussian elimination, can be employed to solve the system of $N$ linear equations in $N$ unknowns in (4.6). The solution of the linear system in (4.6) is possible for joint configurations for which the Jacobian matrix is of rank $N$. If no solution exists, the joint velocity vector in (4.6) minimizes the Euclidean norm $\|J\dot{q} - v_N\|$. The absence of a solution indicates that the manipulator cannot achieve the desired position and orientation of the end-effector.

iii) $N > 6$

In this case we have a linear system with more unknowns than equations. Such a system is underdetermined and it may have more than one solution. The solution can be
represented symbolically as:

\[ \ddot{q} = R J v_N \quad \iff \quad \ddot{q} = J^T (J J^T)^{-1} v_N \quad (4.7) \]

where \( R J = J^T (J J^T)^{-1} \) is the \((N \times 6)\) right pseudo-inverse of the Jacobian matrix \([16]\).

The solution of the linear system in (4.7) is possible for joint configurations for which the Jacobian matrix is of rank 6. If more than one solution exists, the joint velocity vector in (4.7) has the minimum Euclidean norm \( \|\ddot{q}\| \). This solution characterizes the manipulator configuration closest to the origin that can achieve the desired position and orientation of the end-effector.

5. KINEMATICS OF ACCELERATION

5.1 Introduction

In this section, we develop the transformations needed to relate the acceleration of the end-effector to the joint accelerations. The development parallels that of the previous section.

5.2 Forward Kinematics

The objective of the forward transformation is to describe the acceleration of the end-effector in a reference frame. The end-effector acceleration \( \ddot{v}_N \), is expressed as a \((6 \times 1)\) vector that incorporates the \((3 \times 1)\) translational acceleration \( \ddot{u}_N \) and the \((3 \times 1)\) rotational acceleration \( \ddot{w}_N \) vectors of the end-effector:

\[ \ddot{v}_N = \begin{pmatrix} \ddot{u}_N \\ \ddot{w}_N \end{pmatrix} \quad (5.1) \]

All vectors are expressed in the same reference frame.

The end-effector acceleration is obtained by differentiating the end-effector velocity in (4.1) with respect to time:

\[ \ddot{v}_N = J \ddot{q} + \dot{J} \dot{q} \quad (5.2) \]

where \( \dot{J} \) is the time-derivative of the \((6 \times N)\) Jacobian matrix

\[ \dot{J} = \frac{dJ}{dt} = (\dot{J}_1 \dot{J}_2 \ldots \dot{J}_N), \quad (5.3) \]

and \( \ddot{q} \) is the \((N \times 1)\) vector of joint accelerations

\[ \ddot{q} = \begin{pmatrix} \ddot{q}_1 \\ \ddot{q}_2 \\ \vdots \\ \ddot{q}_N \end{pmatrix}. \quad (5.4) \]
The \((6 \times 1)\) columns of the derivative of the Jacobian are computed by differentiating equations (4.2) and (4.3) with respect to time. For a translational joint \(i\):

\[
\mathbf{j}_i = \begin{pmatrix} \dot{z}_{i-1} \\ \dot{0} \end{pmatrix},
\]

(5.5)

and for a rotational joint \(i\):

\[
\mathbf{j}_i = \begin{pmatrix} \dot{z}_{i-1} \times (p_N - p_{i-1}) + z_{i-1} \times (\dot{p}_N - \dot{p}_{i-1}) \\ \dot{z}_{i-1} \end{pmatrix}.
\]

(5.6)

We observe that the computation of the time-derivative of the Jacobian matrix, and of the acceleration of the end-effector, requires knowledge of the \((z_0, z_1, \ldots, z_{N-1})\) axes and the position vectors \((p_0, p_1, \ldots, p_N)\) of the link coordinate frames, as well as their time-derivatives, expressed in the reference frame. We continue the development by considering two typical choices of reference frames.

If the reference frame is the base (0-th) frame, the \((3 \times 1)\) orientation and position vectors have been evaluated during the forward recursions of the kinematics of position (Section 3.2). Their time-derivatives, however, have to be computed separately by differentiating the forward recursion equations in Table 3-1. For a translational joint \(i\), \(q_i = r_i\):

\[
\dot{x}_i = \dot{x}_{i-1} \cos \theta_i + \dot{y}_{i-1} \sin \theta_i,
\]

(5.7)

\[
\dot{y}_i = \dot{z}_{i-1} \sin \alpha_i - (\dot{x}_{i-1} \sin \theta_i - \dot{y}_{i-1} \cos \theta_i) \cos \alpha_i
\]

(5.8)

\[
\dot{z}_i = \dot{z}_{i-1} \cos \alpha_i + (\dot{x}_{i-1} \sin \theta_i - \dot{y}_{i-1} \cos \theta_i) \sin \alpha_i
\]

(5.9)

\[
\dot{p}_i = \dot{p}_{i-1} + r_i \dot{z}_{i-1} + \dot{r}_i z_{i-1} + \dot{d}_i \dot{x}_i.
\]

(5.10)

For a rotational joint \(i\), \(q_i = \theta_i\):

\[
\dot{x}_i = a_{i-1} \cos \theta_i + b_{i-1} \sin \theta_i,
\]

(5.11)

\[
\dot{y}_i = \dot{z}_i \times x_i + z_i \times \dot{x}_i
\]

(5.12)

\[
\dot{z}_i = \dot{z}_{i-1} \cos \alpha_i + (a_{i-1} \sin \theta_i - b_{i-1} \cos \theta_i) \sin \alpha_i
\]

(5.13)

\[
\dot{p}_i = \dot{p}_{i-1} + r_i \dot{z}_{i-1} + \dot{d}_i \dot{x}_i
\]

(5.14)

where

\[
a_{i-1} = \dot{x}_{i-1} + y_{i-1} \dot{\theta}_i
\]

(5.15)
and,

\[ b_{i-1} = \dot{y}_{i-1} - x_{i-1}\dot{\theta}_i. \] (5.16)

A program logic for the forward transformations (along with the associated initial conditions) is provided in Table 5-1.

If the reference frame is the end-effector \((N-\text{th})\) frame, the \((3 \times 1)\) orientation and position vectors have been evaluated during the inverse recursions of the kinematics of position (Section 3.3). Their time-derivatives, therefore, have to be computed by differentiating the inverse recursion equations in Table 3-2. For a translational joint \(i\), \(q_i = r_i\):

\[ \dot{x}_{i-1} = \dot{x}_i \cos \theta_i + (\dot{z}_i \sin \alpha_i - \dot{y}_i \cos \alpha_i) \sin \theta_i \] (5.17)

\[ \dot{y}_{i-1} = \dot{x}_i \sin \theta_i + (\dot{y}_i \cos \alpha_i - \dot{z}_i \sin \alpha_i) \cos \theta_i \] (5.18)

\[ \dot{z}_{i-1} = \dot{y}_i \sin \alpha_i + \dot{z}_i \cos \alpha_i \] (5.19)

\[ \dot{p}_{i-1} = \dot{p}_i - r_i \dot{z}_{i-1} - \dot{r}_i z_{i-1} - d_i \dot{x}_i. \] (5.20)

For a rotational joint \(i\), \(q_i = \theta_i\):

\[ \dot{x}_{i-1} = [\dot{x}_i + (z_i \sin \alpha_i - y_i \cos \alpha_i)\dot{\theta}_i] \cos \theta_i + (\dot{z}_i \sin \alpha_i - \dot{y}_i \cos \alpha_i - x_i \dot{\theta}_i) \sin \theta_i \] (5.21)

\[ \dot{y}_{i-1} = \dot{z}_{i-1} \times x_{i-1} + z_{i-1} \times \dot{x}_{i-1} \] (5.22)

\[ \dot{z}_{i-1} = \dot{y}_i \sin \alpha_i + \dot{z}_i \cos \alpha_i \] (5.23)

\[ \dot{p}_{i-1} = \dot{p}_i - r_i \dot{z}_{i-1} - d_i \dot{x}_i. \] (5.24)

A program logic for the inverse transformations (along with the associated initial conditions) is provided in Table 5-2.

A more compact way of evaluating the end-effector acceleration in (5.2) involves the direct computation of the \((6 \times 1)\) vector \(\dot{J}\dot{q}\) [17,18]. The advantage of our approach is that it provides explicit information about the time-derivatives of the individual link coordinate frames.
1. INITIALIZE $\dot{x}_0 = \dot{y}_0 = \dot{z}_0 = \dot{p}_0 = 0$.
2. $i \leftarrow 1$.
3. IF $i > N$, GO TO STEP 8.
4. READ $\theta_i, r_i, d_i, \alpha_i$.
5. IF JOINT $i$ IS TRANSLATIONAL THEN:
   a). READ $z_{i-1}, \dot{r}_i$.
   b). $\dot{x}_i = \dot{x}_{i-1} \cos \theta_i + \dot{y}_{i-1} \sin \theta_i$.
   c). $\dot{y}_i = \dot{z}_{i-1} \sin \alpha_i - \dot{x}_{i-1} \sin \theta_i \cos \alpha_i + \dot{y}_{i-1} \cos \theta_i \cos \alpha_i$.
   d). $\dot{z}_i = \dot{z}_{i-1} \cos \alpha_i + \dot{x}_{i-1} \sin \theta_i \sin \alpha_i - \dot{y}_{i-1} \cos \theta_i \sin \alpha_i$.
   e). $\dot{p}_i = \dot{p}_{i-1} + r_i \dot{z}_{i-1} + \dot{r}_i z_{i-1} + d_i \dot{x}_i$.
IF JOINT $i$ IS ROTATIONAL:
   a). READ $x_{i-1}, y_{i-1}, x_i, z_i, \dot{\theta}_i$.
   b). $a_{i-1} = \dot{x}_{i-1} + y_{i-1} \dot{\theta}_i$.
   c). $b_{i-1} = \dot{y}_{i-1} - x_{i-1} \dot{\theta}_i$.
   d). $\dot{x}_i = a_{i-1} \cos \theta_i + b_{i-1} \sin \theta_i$.
   e). $\dot{z}_i = \dot{z}_{i-1} \cos \alpha_i + a_{i-1} \sin \theta_i \sin \alpha_i - b_{i-1} \cos \theta_i \sin \alpha_i$.
   f). $\dot{y}_i = \dot{z}_i \times x_i + z_i \times \dot{x}_i$.
   g). $\dot{p}_i = \dot{p}_{i-1} + r_i \dot{z}_{i-1} + d_i \dot{x}_i$.
6. $i \leftarrow i + 1$.
7. GO TO STEP 3.
8. END

Table 5-1: Program Logic for the Forward Recursions of Coordinate Frame Time-Derivatives
1. **INITIALIZE** \( \dot{x}_N = \dot{y}_N = \dot{z}_N = \dot{p}_N = 0 \).

2. \( i \leftarrow N \).

3. IF \( i < 1 \), GO TO STEP 8.

4. READ \( \theta_i, r_i, d_i, \alpha_i \).

5. IF JOINT \( i \) IS TRANSLATIONAL THEN:
   a). READ \( z_{i-1}, \dot{r}_i \).
   b). \( \dot{x}_{i-1} = \dot{x}_i \cos \theta_i + \dot{z}_i \sin \alpha_i \sin \theta_i - \dot{y}_i \cos \alpha_i \sin \theta_i \).
   c). \( \dot{y}_{i-1} = \dot{x}_i \sin \theta_i + \dot{y}_i \cos \alpha_i \cos \theta_i - \dot{z}_i \sin \alpha_i \cos \theta_i \).
   d). \( \dot{z}_{i-1} = \dot{y}_i \sin \alpha_i + \dot{z}_i \cos \alpha_i \).
   e). \( \dot{p}_{i-1} = \dot{p}_i - r_i \dot{z}_{i-1} - \dot{r}_i z_{i-1} - d_i \dot{x}_i \).

ELSE JOINT \( i \) IS ROTATIONAL:
   a). READ \( x_{i-1}, z_{i-1}, x_i, y_i, z_i, \dot{\theta}_i \).
   b). \( \dot{x}_{i-1} = [\dot{x}_i + (z_i \sin \alpha_i - y_i \cos \alpha_i) \dot{\theta}_i] \cos \theta_i + (\dot{z}_i \sin \alpha_i - \dot{y}_i \cos \alpha_i - x_i \dot{\theta}_i) \sin \theta_i \).
   c). \( \dot{z}_{i-1} = \dot{y}_i \sin \alpha_i + \dot{z}_i \cos \alpha_i \).
   d). \( \dot{y}_{i-1} = \dot{z}_{i-1} \times x_{i-1} + z_{i-1} \times \dot{x}_{i-1} \).
   e). \( \dot{p}_{i-1} = \dot{p}_i - r_i \dot{z}_{i-1} - d_i \dot{x}_i \).

6. \( i \leftarrow i - 1 \).

7. GO TO STEP 3.

8. END

---

**Table 5-2: Program Logic for the Inverse Recursions of Coordinate Frame Time-Derivatives**
5.3 Computational Requirements

The forward recursions for the time-derivatives of the coordinate frames have a computational cost of \(37m + 24a\) for a translational joint and \(41m + 30a\) for a rotational joint. Each column of the derivative of the Jacobian matrix has an additional cost of \(12m + 12a\) for a rotational joint since the differences \((p_N - p_{i-1})\) have already been computed in the evaluation of the Jacobian in (4.3). There is no additional computation required for a translational joint. The first column of the time-derivative of the Jacobian, \(\dot{J}\), requires no computation while the second column requires \(10m + 9a\) as can be seen from the initial conditions. The total computational cost for the time-derivative of the Jacobian of an \(N\)-joint robot with rotational joints (worst case) is therefore \((53N - 47)m + (42N - 45)a\).

The inverse recursions for the time-derivatives of the coordinate frames have a computational cost of \(37m + 24a\) for a translational joint and \(48m + 33a\) for a rotational joint. Each column of the derivative of the Jacobian matrix has an additional cost of \(12m + 9a\) for a rotational joint, since \(p_N = \dot{p}_N = 0\). There is no additional computation required for a translational joint. The \(N\)-th column of the time-derivative of the Jacobian, \(N\dot{J}\), requires no computation. The total computational cost for the time-derivative of the \(N\) Jacobian of an \(N\)-joint robot with rotational joints (worst case) is therefore \((60N - 54)m + (42N - 42)a\).

5.4 Inverse Kinematics

The objective of the inverse transformation is to compute the joint accelerations from the end-effector acceleration. The inverse transformation is the solution of the following system of 6 linear equations

\[
\dot{v}_N - \dot{\mathbf{J}}\ddot{\mathbf{q}} = \mathbf{J}\ddot{\mathbf{q}}
\]

(5.25)

for the \(N\) unknowns \((\ddot{q}_1, \ddot{q}_2, \ldots, \ddot{q}_N)\). We note that equations (4.1) and (5.25) represent linear systems with the same coefficient matrix, \(\mathbf{J}\), but different left-hand side vectors. The discussion, therefore, on the solution of the system (4.1) in Section 4.4 applies to the system in (5.25).
6. THE COMPLETE KINEMATIC ALGORITHM

In this section, we summarize the complete kinematic algorithm that incorporates position, velocity, and acceleration transformations. The algorithm requires the following information as input at each time-instant:

- temporal evolution of the desired trajectory (generated by the trajectory planner; includes the desired position, velocity, and acceleration of the end-effector);

- present robot configuration (provided by the joint sensors); and

- geometric link parameters (constant quantities included in the robot database)

There are four major computational blocks in the algorithm. The first block computes the position and orientation of each link coordinate frame with respect to the robot base frame (Table 3-1). This information is then processed by the second block to generate the Jacobian matrix in (4.1) and solve equation (4.1) for the joint velocities. The third block computes the time-rate of change of each link coordinate frame with respect to the base frame (Table 5-1). The fourth block processes this information to generate the time-derivative of the Jacobian matrix in (5.3) and solves equation (5.25) for the joint accelerations.

The computational requirements of the four major blocks have been delineated in Sections 3 through 5. Although our presentation assumes serial implementation of the algorithm, we note that the four major blocks share a lot of operations thus making the algorithm amenable to parallel computation [19].
7. COMPARATIVE EVALUATION

7.1 Introduction

In this section, we compare the computational cost of our algorithm with that of algorithms that have previously appeared in the robotics literature. The number of transcendental function calls needed in the four major blocks of our algorithm is $2N$ for an $N$-joint manipulator. Algorithms which have been presented in the robotics literature and mentioned in this paper also require $2N$ transcendental function calls. Thus our comparisons and evaluations are based on the number of multiplications and additions only. Our comparative evaluation proceeds along three areas: i) kinematics of position; ii) kinematics of velocity; and iii) kinematics of acceleration. We address each of these areas in turn.

7.2 Kinematics of Position

In Table 7-1, we compare our algorithm with the Denavit-Hartenberg method [10] and the Wang-Ravani recursive formulation [20]. The computational costs given in Table 7-1 are representative of manipulators with general geometries. The conventional Denavit-Hartenberg method computes the position and orientation of each link frame from the products of $(4 \times 4)$ homogeneous transformation matrices which relate consecutive link frames. The formulae in Table 7-1 take into consideration that the last row of any homogeneous transformation matrix is \((0 \ 0 \ 0 \ 1)\) [10]. The Wang-Ravani recursive formulation is based upon the vector form of the Rodrigues’ equation and appears to be the most efficient approach in the robotics literature [20]. Our algorithm is more efficient than both the aforementioned algorithms.

<table>
<thead>
<tr>
<th>$N$-Joint Manipulator</th>
<th>Position</th>
<th>Orientation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Denavit-Hartenberg [20]</td>
<td>$(11N - 9)m + (9N - 9)a$</td>
<td>$(31N - 27)m + (18N - 18)a$</td>
</tr>
<tr>
<td>Wang-Ravani [20]</td>
<td>$(6N - 4)m + (6N - 6)a$</td>
<td>$(24N - 20)m + (12N - 12)a$</td>
</tr>
<tr>
<td>Section 3.4</td>
<td>$(6N - 4)m + (6N - 6)a$</td>
<td>$(23N - 21)m + (12N - 14)a$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>6-Joint Manipulator</th>
<th>Position</th>
<th>Orientation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Denavit-Hartenberg [20]</td>
<td>$57m + 45a$</td>
<td>$159m + 90a$</td>
</tr>
<tr>
<td>Wang-Ravani [20]</td>
<td>$32m + 30a$</td>
<td>$124m + 60a$</td>
</tr>
<tr>
<td>Section 3.4</td>
<td>$32m + 30a$</td>
<td>$117m + 58a$</td>
</tr>
</tbody>
</table>

Table 7-1: Comparative Computational Cost of Evaluating the Kinematics of Position
7.3 Kinematics of Velocity

Comparisons involving the cost of computing the Jacobian with respect to the base frame, $^0\mathbf{J}$, and the Jacobian with respect to the end-effector frame, $^N\mathbf{J}$, are presented in Tables 7-2 and 7-3 respectively. The entries are based upon the review paper by Orin and Schrader [21] and the results of Wang and Ravani [20]. Manipulators with general geometries as well as manipulators with parallel or perpendicular axes of motion ($\alpha_i = 0^\circ$ or $90^\circ$) are considered. We note that in the computation of $^0\mathbf{J}$ for 6-joint manipulators ($\alpha_i = 0^\circ$ or $90^\circ$, Table 7-2), our algorithm requires 12 less multiplications but 31 more additions than Waldron's in [21]. For all other cases in Table 7-3, our algorithm requires the least amount of computation.

7.4 Kinematics of Acceleration

The general-purpose algorithm for the kinematics of acceleration is a novel contribution of this paper. It involves the computation of the time-derivatives of the link coordinate frames. To demonstrate the efficacy of our algorithm, we modified the conventional approach (based upon the Denavit-Hartenberg convention and the homogeneous transformation formalism [10]) to decrease its computational cost. The modified approach, presented in Appendix A, significantly reduces the redundancy of the homogeneous transformations. We highlight the computational requirements of both approaches in Tables 7-4 and 7-5 respectively.

In the modified Denavit-Hartenberg approach (Table 7-4) the time-derivatives of the coordinate axes are computed independently. Since the computation of the time-derivative of the Jacobian requires the evaluation of $\hat{\mathbf{z}}_i$ and $\hat{\mathbf{p}}_i$ only, the modified Denavit-Hartenberg method requires $[294m + 215a]$ for a 6-joint manipulator. Our algorithm, though necessitating the computation of $\hat{\mathbf{x}}_i$, $\hat{\mathbf{y}}_i$, $\hat{\mathbf{z}}_i$ and $\hat{\mathbf{p}}_i$, requires $[211m + 150a]$ for a 6-joint manipulator, and is computationally more efficient.
APPENDIX A

In this appendix we describe a conventional method of computing the time-derivatives of the link coordinate frames. The procedure is based upon the Denavit-Hartenberg convention (Section 2) and is a modification of the homogeneous transformation approach.

Homogeneous transformations between link coordinate frames are described by \((4 \times 4)\) matrices which can represent rotations and translations. A coordinate frame is assigned to each link using the Denavit-Hartenberg convention (Section 2). The \(i\)-th coordinate frame is obtained from the \((i - 1)\)-th frame by performing the four operations outlined in Section 2. These four operations can be expressed by the following homogeneous transformation matrices:

\[
A_i(q_i) = \begin{pmatrix}
\cos \theta_i & -\cos \alpha_i \sin \theta_i & \sin \alpha_i \sin \theta_i & d_i \cos \theta_i \\
\sin \theta_i & \cos \alpha_i \cos \theta_i & -\sin \alpha_i \cos \theta_i & d_i \sin \theta_i \\
0 & \sin \alpha_i & \cos \alpha_i & r_i \\
0 & 0 & 0 & 1
\end{pmatrix}.
\tag{A.1}
\]

In (A.1), the first three columns define the axes of the \(i\)-th frame in the \((i - 1)\)-th coordinate frame, while the fourth column is the position vector of the origin of frame \(i\) with respect to frame \((i - 1)\). The geometric parameters \(\theta_i, r_i, d_i\) and \(\alpha_i\) characterize link \(i\) and have been defined in Section 2.

Generalizing (A.1), the position and orientation of the \(i\)-th coordinate frame expressed in the base coordinate frame can be described by the forward kinematic transformation matrix

\[
T_i(q_1, q_2, \ldots, q_i) = A_1 A_2 \cdots A_i = \begin{pmatrix}
x_i & y_i & z_i & p_i \\
0 & 0 & 0 & 1
\end{pmatrix}.
\tag{A.2}
\]

The origin of the \(i\)-th coordinate frame is specified by the position vector \(p_i\) in the base coordinate frame. The unit vectors \(x_i, y_i,\) and \(z_i\) describe the coordinate axes of frame \(i\) in the base frame.

The time-derivative of the forward kinematic transformation matrix \(T_i\) in (A.2) is given by the total differential:

\[
\dot{T}_i = \frac{\partial T_i}{\partial q_1} \dot{q}_1 + \frac{\partial T_i}{\partial q_2} \dot{q}_2 + \cdots + \frac{\partial T_i}{\partial q_i} \dot{q}_i = \begin{pmatrix}
\dot{x}_i & \dot{y}_i & \dot{z}_i & \dot{p}_i \\
0 & 0 & 0 & 1
\end{pmatrix},
\tag{A.3}
\]

where \(i = 1, 2, \ldots, N\) for an \(N\)-joint robot. The partial derivative of \(T_i\) with respect to a joint variable \(q_j\) \((j \leq i)\) is

\[
\frac{\partial T_i}{\partial q_j} = A_1 A_2 \cdots A_{j-1} \frac{\partial A_j}{\partial q_j} A_{j+1} \cdots A_{i-1} A_i.
\tag{A.4}
\]
8. CONCLUSIONS

In this paper, we developed a coherent, general-purpose framework, incorporating position, velocity, and acceleration transformations. In this framework, we delineated the computational requirements of a novel inverse kinematics algorithm. We then compared the computational cost of our algorithm with the cost of general-purpose algorithms that have previously appeared in the robotics literature. The comparative evaluation demonstrated the efficacy and feasibility of our algorithm.

Future research efforts will focus upon:

- the customization of our algorithm for specific manipulators [7];
- the design of special-purpose architectures that exploit the parallelism inherent in our algorithm [19]; and
- the development of robot dynamics algorithms to exploit the synergism between kinematics transformations and robot dynamics computations [20].
<table>
<thead>
<tr>
<th>( N ) Joints</th>
<th>( N = 6 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \dot{x}_i )</td>
<td>( (1/2)(9N^2 - 5N)m + (3N^2 - 4N + 1)a )</td>
</tr>
<tr>
<td>( \dot{y}_i )</td>
<td>( (1/2)(9N^2 - 5N)m + (3N^2 - 4N + 1)a )</td>
</tr>
<tr>
<td>( \dot{z}_i )</td>
<td>( (1/2)(9N^2 - 5N)m + (3N^2 - 4N + 1)a )</td>
</tr>
<tr>
<td>( \dot{p}_i )</td>
<td>( (1/2)(9N^2 - 5N)m + (1/2)(9N^2 - 11N + 2)a )</td>
</tr>
<tr>
<td>Total</td>
<td>( (18N^2 - 10N)m + (1/2)(27N^2 - 35N + 8)a )</td>
</tr>
</tbody>
</table>

**Table 7-4: Computational Cost for Evaluating the Time-Derivatives of the Link Coordinate Frames (Appendix A)**

<table>
<thead>
<tr>
<th>( N ) Joints</th>
<th>( N = 6 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \dot{x}_i, \dot{y}_i, \dot{z}_i, \dot{p}_i )</td>
<td>( (41N - 33)m + (30N - 30)a )</td>
</tr>
</tbody>
</table>

**Table 7-5: Computational Cost for Evaluating the Time-Derivatives of the Link Coordinate Frames (Section 5.3)**
<table>
<thead>
<tr>
<th>Method</th>
<th>(N) Joints</th>
<th>(N = 6)</th>
</tr>
</thead>
<tbody>
<tr>
<td>General Manipulator Geometry</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sections 3.4 &amp; 4.3</td>
<td>((35N - 31)m + (24N - 26)a)</td>
<td>(179m + 118a)</td>
</tr>
<tr>
<td></td>
<td>(\alpha_i = 0^\circ \text{ or } 90^\circ)</td>
<td></td>
</tr>
<tr>
<td>Waldron [21]</td>
<td>((30N - 55)m + (15N - 38)a)</td>
<td>(125m + 52a)</td>
</tr>
<tr>
<td>Sections 3.4 &amp; 4.3</td>
<td>((24N - 31)m + (18N - 25)a)</td>
<td>(113m + 83a)</td>
</tr>
</tbody>
</table>

**Table 7-2**: Comparative Computational Cost of Evaluating the Jacobian with respect to the Base Frame \((^0J)\)

<table>
<thead>
<tr>
<th>Method</th>
<th>(N) Joints</th>
<th>(N = 6)</th>
</tr>
</thead>
<tbody>
<tr>
<td>General Manipulator Geometry</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Orin-Schrader [21]</td>
<td>((45N - 33)m + (27N - 24)a)</td>
<td>(237m + 138a)</td>
</tr>
<tr>
<td>Wang-Ravani [20]</td>
<td>((36N - 24)m + (21N - 18)a)</td>
<td>(192m + 108a)</td>
</tr>
<tr>
<td>Sections 3.4 &amp; 4.3</td>
<td>((35N - 29)m + (21N - 23)a)</td>
<td>(181m + 103a)</td>
</tr>
<tr>
<td></td>
<td>(\alpha_i = 0^\circ \text{ or } 90^\circ)</td>
<td></td>
</tr>
<tr>
<td>Vukobratovic-Potkonjak [21]</td>
<td>((10N^2 - 15N + 9)m + (N^2 + 7N - 2)a)</td>
<td>(279m + 76a)</td>
</tr>
<tr>
<td>Paul [21]</td>
<td>((30N - 18)m + (14N - 15)a)</td>
<td>(162m + 69a)</td>
</tr>
<tr>
<td>Orin/Schrader [21]</td>
<td>((30N - 25)m + (15N - 25)a)</td>
<td>(155m + 65a)</td>
</tr>
<tr>
<td>Wang-Ravani [20]</td>
<td>((24N - 36)m + (15N - 21)a)</td>
<td>(108m + 69a)</td>
</tr>
<tr>
<td>Sections 3.4 &amp; 4.3</td>
<td>((24N - 36)m + (15N - 25)a)</td>
<td>(108m + 65a)</td>
</tr>
</tbody>
</table>

**Table 7-3**: Comparative Computational Cost of Evaluating the Jacobian with respect to the End-Effector Frame \((^N J)\)
Bejczy [22] proved that the first partial derivative of the homogeneous transformation matrix $A_i$ in (A.1), with respect to the joint variable $q_i$, can be written as the matrix product:

$$\frac{\partial A_i}{\partial q_i} = Q_i A_i$$  \hspace{1cm} (A.5)

where the sparse matrix $Q_i$ is

$$Q_i = Q_R = \begin{pmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \text{ for a rotational joint } i \hspace{1cm} (A.6)$$

and

$$Q_i = Q_T = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} \text{ for a translational joint } i. \hspace{1cm} (A.7)$$

Therefore, we can express the partial derivative $\frac{\partial T_i}{\partial q_j}$ in (A.4) as a product of homogeneous transformation matrices and the $Q_i$ matrix

$$\frac{\partial T_i}{\partial q_j} = A_1 A_2 \cdots A_{j-1} Q_j A_j A_{j+1} \cdots A_{i-1} A_i.$$  \hspace{1cm} (A.8)

Equation (A.3) then becomes

$$\dot{T}_i = \sum_{j=1}^{i} C_j \dot{q}_j$$  \hspace{1cm} (A.9)

and

$$C_j = T_{j-1} Q_j A_j A_{j+1} \cdots A_i.$$  \hspace{1cm} (A.10)

We can modify equation (A.10) to obtain a computationally more efficient expression for $\dot{T}_i$. We observe that $T_i$ has already been computed for $i = 1, 2, \ldots, N$ in the forward kinematics of position (Section 3.2). We can write

$$T_{j-1} A_j A_{j+1} \cdots A_i = T_i \iff A_j A_{j+1} \cdots A_i = (T_{j-1})^{-1} T_i.$$  \hspace{1cm} (A.11)

Substituting equation (A.11) into (A.10) yields

$$C_j = T_{j-1} Q_j (T_{j-1})^{-1} T_i.$$  \hspace{1cm} (A.12)

For expository convenience, let

$$T_{j-1} = \begin{pmatrix} x_{j-1} & y_{j-1} & z_{j-1} & p_{j-1} \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} n_x & a_x & p_x \\ n_y & a_y & p_y \\ n_z & a_z & p_z \end{pmatrix}, \hspace{1cm} (A.13)$$
and \[ T_i = \begin{pmatrix} x_i & y_i & z_i & p_i \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} N_x & S_x & A_x & P_x \\ N_y & S_y & A_y & P_y \\ N_z & S_z & A_z & P_z \\ 0 & 0 & 0 & 1 \end{pmatrix}. \] (A.14)

Then, for a translational joint \( j \), we have

\[ C_j = \begin{pmatrix} 0 & 0 & 0 & a_x \\ 0 & 0 & 0 & a_y \\ 0 & 0 & 0 & a_z \\ 0 & 0 & 0 & 0 \end{pmatrix}. \] (A.15)

Similarly, for a rotational joint \( j \) we have the following equations for each column of \( C_j \)

**first column:** \[ C_j[1] = \begin{pmatrix} a_y N_z - a_z N_y \\ a_z N_x - a_x N_z \\ a_y N_z - a_z N_x \\ 0 \end{pmatrix} \] (A.16)

**second column:** \[ C_j[2] = \begin{pmatrix} a_y S_z - a_z S_y \\ a_z S_x - a_x S_z \\ a_y S_z - a_z S_y \\ 0 \end{pmatrix} \] (A.17)

**third column:** \[ C_j[3] = \begin{pmatrix} a_y A_z - a_z A_y \\ a_z A_x - a_x A_z \\ a_y A_z - a_z A_y \\ 0 \end{pmatrix} \] (A.18)

**fourth column:** \[ C_j[4] = \begin{pmatrix} a_y (P_z - p_z) - a_z (P_y - p_y) \\ a_z (P_x - p_x) - a_x (P_z - p_z) \\ a_y (P_z - p_z) - a_z (P_y - p_y) \\ 0 \end{pmatrix} \] (A.19)

The time-rate of change in orientation and position of each coordinate frame is thus obtained from equations (A.9) and (A.15) through (A.19):

\[ \dot{x}_i = \sum_{j=1}^{i} C_j[1] \dot{q}_j. \] (A.20)

\[ \dot{y}_i = \sum_{j=1}^{i} C_j[2] \dot{q}_j. \] (A.21)

\[ \dot{z}_i = \sum_{j=1}^{i} C_j[3] \dot{q}_j. \] (A.22)
\[ \dot{\mathbf{p}}_i = \sum_{j=1}^{i} C_{j[4]} \ddot{q}_j. \]  
(A.23)

The first term in these summations can be isolated to reduce computational cost:

\[ \dot{x}_i = \begin{pmatrix} -x_{i,y} \\ x_{i,x} \\ 0 \end{pmatrix} \dot{q}_1 + \sum_{j=2}^{i} C_{j[1]} \ddot{q}_j. \]  
(A.24)

\[ \dot{y}_i = \begin{pmatrix} -y_{i,y} \\ y_{i,x} \\ 0 \end{pmatrix} \dot{q}_1 + \sum_{j=2}^{i} C_{j[2]} \ddot{q}_j. \]  
(A.25)

\[ \dot{z}_i = \begin{pmatrix} -z_{i,y} \\ z_{i,x} \\ 0 \end{pmatrix} \dot{q}_1 + \sum_{j=2}^{i} C_{j[3]} \ddot{q}_j. \]  
(A.26)

\[ \dot{p}_i = \begin{pmatrix} -p_{i,y} \\ p_{i,x} \\ 0 \end{pmatrix} \dot{q}_1 + \sum_{j=2}^{i} C_{j[4]} \ddot{q}_j. \]  
(A.27)

In (A.24) through (A.27), the second subscript indicates the component of the vector along the \( x_0, y_0 \) or \( z_0 \) axes of the base frame.

Equations (A.16) through (A.18) indicate identical computational requirements for the time-derivatives of the orientation vectors \( \dot{x}_i, \dot{y}_i, \) and \( \dot{z}_i \). The cost for computing the time-derivative of each orientation vector is detailed below:

- **Forming the summand:** \( (i - 1)(9m + 3a) \)
- **Summation operation:** \( (i - 2)(3a) \)
- **First Term:** \( 2m \)
- **Final Sum:** \( 2a \)
- **TOTAL:** \( (9i - 7)m + (6i - 7)a \)

The computational cost for an \( N \)-joint manipulator is thus \( \frac{1}{2}(9N^2 - 5N)m + (3N^2 - 4N + 1)a \) or \( [147m + 85a] \) for \( N = 6 \).

A few more additions are required for the computation of the time-derivative of each position vector in (A.19). The computational cost for the time-derivative of each position vector is detailed below:
- Forming the summand: \((i - 1)(9m + 6a)\)
- Summation operation: \((i - 2)(3a)\)
- First Term: \(2m\)
- Final Sum: \(2a\)
- TOTAL: \((9i - 7)m + (9i - 10)a\)

Thus for an \(N\)-joint manipulator, \([\frac{1}{2}(9N^2 - 5N)m + \frac{1}{2}(9N^2 - 11N + 2)a]\) are required to compute \(\hat{p}_1, \hat{p}_2, \ldots, \hat{p}_N\), or \([147m + 130a]\) for \(N = 6\).
REFERENCES


30