Funda, J. and Paul, R.P. "A comparison of transforms and quaternions in robotics." Proceedings of the IEEE Conference on Robotics and Automation (ICRA). 1998. pp 886-891. doi: 10.1109/ROBOT.1988.12172

A Comparison of Transforms and Quaternions in Robotics

Janez Funda Richard P. Paul

University of Pennsylvania, Philadelphia

Abstract

Three-dimensional (3-D) modeling of rotations and translations in robot kinematics is most commonly performed using homogeneous transforms. In this paper, an alternate approach, employing quaternion/vector pairs as spatial operators, is discussed and analyzed. Sequential and parallel algorithms for some of the common spatial operations are given for both homogeneous transforms and quaternion/vector pairs and compared in terms of computational efficiency. The two approaches are shown to be essentially equivalent in the absence of the need for frequent renormalization of rotational operators.

1 Introduction

A variety of different mathematical tools for expressing spatial relationships have been developed and successfully applied in robotics, computer vision, graphics, and other engineering disciplines. Most notable among the approaches to this problem is the *vectorial technique*. Although very elegant in expressing translational information, vector calculus does not lend itself naturally to representing 3-D rotations. In this system, spatial transformations have traditionally been expressed as a set of four vectors, arranged in the form of a 4×4 matrix termed a *homogeneous transform*.[9]

This paper discusses an alternate mathematical model of spatial transformations, where 3-D rotations are represented via *quaternions* and translations are modeled using ordinary vectors (*i.e.*, transformational operators are expressed as quaternion/vector pairs). At a first glance, quaternions appear as four-dimensional complex numbers, but their true significance lies in the fact that they subsume virtually all the properties of real and complex numbers with the exception of commutativity of multiplication. Moreover, quaternions can be viewed as rotational operators and can, due to their simplicity and conciseness, be used efficiently to model 3-D rotations.

The intent of this paper is to compare quaternion/vector pairs and homogeneous transforms in terms of their computational properties. Both sequential and parallel implementations of some of the most frequently encountered operations involving spatial transformations are discussed and compared. Finally, an outline of a solution to inverse kinematics for the Puma robot arm is given, employing quaternions and trigonometry.

2 The Quaternion

A quaternion is a mathematical object of the form

$$q = s + ix + jy + kz \tag{1}$$

where $s, x, y, z \in \mathbf{R}$, and i, j, k are mutually orthogonal imaginary units, whose composition rule can be stated concisely as follows

$$i^2 = j^2 = k^2 = ijk = -1 \tag{2}$$

Thus, algebraically, the set of quaternions comprises a four-dimensional vector space over $\mathbf{R}$ with basis $1, i, j$ and $k = ij$.[5]

Quaternions were originally discovered by Sir William R. Hamilton in early 1840's while investigating the properties of vector quotients. However, Hamilton soon observed that within the scope of 3-D geometry quaternions also arise as products of vectors, powers of vectors, and sums of scalars and vectors. [3][4]

For instance, multiplication of two 3-D vectors using Eq.(2) as the defining relationship between the imaginary units, gives [8]

$$\vec{v}_1 \bullet \vec{v}_2 = -(\vec{v}_1 \cdot \vec{v}_2) + (\vec{v}_1 \times \vec{v}_2) \tag{3}$$

which is of the form $s + \vec{v}$, where $s \in \mathbf{R}$ and $\vec{v} \in \mathbf{R} \times \mathbf{R} \times \mathbf{R}$ (*i.e.*, $\vec{v}$ is a vector), and thus constitutes a quaternion as defined by Eq.(1). For our purposes, it will be convenient to treat a quaternion as a sum of a scalar (s) and a 3-D vector ($\langle x, y, z \rangle$), which also seems to be the most natural interpretation of Eq.(1). Observe that vectors can thus be represented as quaternions with null scalar parts. We will abbreviate the notation of Eq.(1) as follows

$$q = [s, \langle x, y, z \rangle] = [s, \vec{v}] \tag{4}$$

Two quaternions $q_1 = [s_1, \vec{v}_1]$ and $q_2 = [s_2, \vec{v}_2]$ are added simply by adding their components, *i.e.*,

$$q = q_1 + q_2 = [s_1 + s_2, \vec{v}_1 + \vec{v}_2] \tag{5}$$

Employing the composition rule of Eq.(2), along with the result of Eq.(3), we find that the general multiplication rule for quaternions is

$$\begin{aligned} [s_1, \vec{v}_1] * [s_2, \vec{v}_2] &= s_1 s_2 + s_1 \vec{v}_2 + s_2 \vec{v}_1 + \vec{v}_1 \bullet \vec{v}_2 \\ &= [(s_1 s_2 - \vec{v}_1 \cdot \vec{v}_2), (s_1 \vec{v}_2 + s_2 \vec{v}_1 + \vec{v}_1 \times \vec{v}_2)] \end{aligned} \tag{6}$$

Note that due to the presence of a cross product term in the vector part of Eq.(6), quaternion multiplication is in general *not commutative*.

The *conjugate* ($\bar{q}$), *norm* ($N(q)$) and *inverse* (q^{-1}) of a quaternion $q = [s, \vec{v}]$ are defined in a straightforward fashion by the following equations:

$$\bar{q} = [s, -\vec{v}] \tag{7}$$

$$N(q) = \|q\| = \sqrt{q * \bar{q}} = \sqrt{s^2 + |\vec{v}|^2} \tag{8}$$

$$q^{-1} = \frac{1}{q} = \frac{1}{q} * \frac{\bar{q}}{\bar{q}} = \frac{\bar{q}}{N(q)^2} \tag{9}$$

If $N(q) = 1$, then the quaternion q is refered to as a *unit quaternion*. Note that for unit quaternions $\bar{q} = q^{-1}$.

In order to characterize quaternions algebraically, let $\mathbf{Q}$ denote the set of all possible quaternions. It can be easily shown that the set $\mathbf{Q}$ together with the binary operations of quaternion addition ($+$) and multiplication ($*$), as defined by Eqs.(5,6), comprises a *noncommutative division ring with identity and no zero divisors.*[1] [10][2] Moreover, notice that $\mathbf{R}$ is a subring of $\mathbf{Q}$ and that scaling a quaternion by a real value is a commutative operation. Hence, $\mathbf{Q}$ is a *real division algebra* of dimension 4 (recall that $\mathbf{Q}$ can also be regarded as a 4-dimensional vector space over $\mathbf{R}$) and is, in fact, the *largest* finite real division algebra. A famous theorem in the theory of algebraic structures states

[1] Observe that the structure $(\mathbf{Q}, *)$, *i.e.*, the set of quaternions under multiplication as defined by Eq.(6), constitutes a *noncommutative group*.

that "any finite-dimensional division algebra is isomorphic to one of **R**, **C**, or **Q**".[5] Quaternions, thus, comprise a fundamental algebraic structure.

So far we have restricted ourselves to representing quaternions as sums of scalars and vectors, in accordance with Eq.(1). However, quaternions can be equivalently expressed in trigonometric or exponential form. Let $q = [s, \vec{v}] = s + \vec{v}$ be an arbitrary quaternion and let

$$r = N(q) = \sqrt{s^2 + |\vec{v}|^2}, \qquad \phi = \arctan\frac{|\vec{v}|}{s}, \qquad \hat{k} = \frac{\vec{v}}{|\vec{v}|} \tag{10}$$

Then, the quaternion $q = [s, \vec{v}]$ can be written as [3][2]

$$q = r\,(\cos\phi + \hat{k}\sin\phi) \tag{11}$$

Furthermore, let $\hat{v}$ be an arbitrary unit vector and observe that Eq.(3) yields

$$\hat{v} * \hat{v} = \hat{v}^2 = -1 \tag{12}$$

Since $\hat{v}^2 = -1$ for any unit vector $\hat{v}$, we can extend Euler's formula for complex numbers (*i.e.*, $\cos\theta + i\sin\theta = e^{i\theta}$) to 4 dimensions and write [1][10]

$$\cos\theta + \hat{k}\sin\theta = e^{\hat{k}\theta} \tag{13}$$

Now, combining Eq.(11) and Eq.(13), we obtain an important and rather elegant result linking algebraic, trigonometric, and exponential representations of quaternions.

$$q = [s, \vec{v}] = r\,(\cos\phi + \hat{k}\sin\phi) = re^{\hat{k}\phi} \tag{14}$$

where $r, \hat{k}$, and ϕ are given by Eq.(10).

3 Rotational Operators

3.1 General Rotations using Homogeneous Transforms

Traditionally rotations in 3-space have been modeled by rotational matrices (embedded in homogeneous transforms). The general rotational matrix $\mathcal{M}$, representing a rotation by an angle θ about an arbitrary spatial axis $\hat{k} = \langle k_x, k_y, k_z\rangle$, is [9]

$$\mathcal{M} = \mathrm{Rot}(\hat{k}, \theta) =$$

$$\begin{bmatrix} k_xk_xV_\theta + C_\theta & k_yk_xV_\theta - k_zS_\theta & k_xk_xV_\theta + k_yS_\theta & 0 \\ k_xk_yV_\theta + k_zS_\theta & k_yk_yV_\theta + C_\theta & k_xk_yV_\theta - k_xS_\theta & 0 \\ k_xk_xV_\theta - k_yS_\theta & k_yk_xV_\theta + k_xS_\theta & k_xk_xV_\theta + C_\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{15}$$

where $S_\theta = \sin\theta$, $C_\theta = \cos\theta$, and $V_\theta = \mathrm{vers}\,\theta = 1 - \cos\theta$. Thus, to rotate a vector $\vec{v}$ through an angle θ about some axis $\hat{k}$, we have $\vec{v}' = \mathcal{M}\vec{v}$, where $\vec{v}'$ is the rotated vector.

3.2 Rotations using Quaternions

To motivate an investigation of quaternions in the context of 3-D rotations, observe that 3-D rotations form a noncommutative group under rotational composition. The noncommuting nature of the structure is due to the fact that rotations confound axes and so, in general, $\mathrm{Rot}(\hat{k}_1, \theta_1)\mathrm{Rot}(\hat{k}_2, \theta_2) \neq \mathrm{Rot}(\hat{k}_2, \theta_2)\mathrm{Rot}(\hat{k}_1, \theta_1)$. Recall that quaternions likewise comprise a noncommutative group under multiplication. In fact, as we shall see shortly, the two groups are closely related.

Let $\vec{a}$ and $\vec{b}$ be two 3-D vectors separated by an angle θ, such that $|\vec{a}| = |\vec{b}|$, and suppose that we want to rotate $\vec{a}$ into $\vec{b}$ about some axis $\hat{k}$, emanating from the origin. If the rotational axis $\hat{k}$ happens to be perpendicular to both $\vec{a}$ and $\vec{b}$, then it is easy to show [8][2] that

$$\vec{b} = (\cos\theta + \hat{k}\sin\theta)\vec{a} \tag{16}$$

Note that the parenthesized term in Eq.(16) corresponds to a unit quaternion. Thus, the unit quaternion $q = (\cos\theta + \hat{k}\sin\theta)$ represents the *orthogonal* rotation $\mathrm{Rot}(\hat{k}, \theta)$ where the rotational axis is normal to the plane of rotation.

In general, however, the axis of rotation $\hat{k}$ may bear an arbitrary relation to the plane of rotation. Thus, a more general result for quaternion rotations is needed. Again, let $\vec{a} = \langle a_x, a_y, a_z\rangle$ and $\vec{b} = \langle b_x, b_y, b_z\rangle$ be vectors as before and let $q = (\cos\frac{\theta}{2} + \hat{k}\sin\frac{\theta}{2})$ be a unit quaternion. Then, the general *non-orthogonal* rotation of $\vec{a}$ into $\vec{b}$ through an angle θ about an arbitrary axis $\hat{k}$ is given by the equation

$$\vec{b} = q * \vec{a} * q^{-1} \tag{17}$$

To see this, let us first derive the general expression for the product of the form $q * \vec{v} * q^{-1}$. Let $q = [s, \vec{u}]$ and recall that $\vec{v}$ can be written as $\vec{v} = [0, \vec{v}]$. Then, the expression becomes a product of three quaternions and straightforward expansion gives

$$q * \vec{v} * q^{-1} = [0, 2(\vec{u}\cdot\vec{v})\vec{u} + (s^2 - \vec{u}\cdot\vec{u}) + 2s(\vec{u}\times\vec{v})] \tag{18}$$

Now, substituting $s = \cos\frac{\theta}{2}$, $\vec{u} = \hat{k}\sin\frac{\theta}{2}$, and $\vec{v} = \langle a_x, a_y, a_z\rangle$ and employing the trigonometric equalities $\sin^2\frac{\theta}{2} = \frac{1}{2}(1 - \cos\theta)$, $\cos^2\frac{\theta}{2} = \frac{1}{2}(1 + \cos\theta)$, and $\sin\frac{\theta}{2}\cos\frac{\theta}{2} = \frac{1}{2}\sin\theta$, we obtain [2]

$$\begin{aligned} b_x &= (k_xk_xV_\theta + C_\theta)a_x + (k_yk_xV_\theta - k_zS_\theta)a_y + (k_xk_xV_\theta + k_yS_\theta)a_z \\ b_y &= (k_xk_yV_\theta + k_zS_\theta)a_x + (k_yk_yV_\theta + C_\theta)a_y + (k_xk_yV_\theta - k_xS_\theta)a_z \\ b_z &= (k_xk_xV_\theta - k_yS_\theta)a_x + (k_yk_xV_\theta + k_xS_\theta)a_y + (k_xk_xV_\theta + C_\theta)a_z \end{aligned}$$

which is precisely the vector that we would have obtained by premultiplying the vector $\vec{a}$ by the rotational matrix of Eq.(15), *i.e.*, [2]

$$\vec{b} = \mathcal{M}\vec{a} = (\cos\frac{\theta}{2} + \hat{k}\sin\frac{\theta}{2}) * \vec{a} * (\cos\frac{\theta}{2} - \hat{k}\sin\frac{\theta}{2}) \tag{19}$$

Thus, the unit quaternion pair (q, q^{-1}) with $q = (\cos\frac{\theta}{2} + \hat{k}\sin\frac{\theta}{2})$ represents the general rotation through an angle θ about the spatial axis $\hat{k}$. Note, however, that only unit quaternions are used as rotational operators — a quaternion q with $N(q) \neq 1$ will produce a vector which is oriented correctly, but whose magnitude does not equal that of the original vector.

Finally observe that Eq.(18) gives the general expression for 3-D rotations of vectors via quaternions. The form in which the equation is stated, however, although convenient for the above derivation, is not computationally optimal. By observing that for a unit quaternion $q = [s, \vec{u}]$ we have $s^2 - \vec{u}\cdot\vec{u} = 1 - 2(\vec{u}\cdot\vec{u})$ and $2(\vec{u}\cdot\vec{v})\vec{u} - 2(\vec{u}\cdot\vec{u})\vec{v} = 2\vec{u}\times(\vec{u}\times\vec{v})$, we can rewrite Eq.(18) as [7]

$$q * \vec{v} * q^{-1} = [0, \vec{v} + 2s(\vec{u}\times\vec{v}) + 2\vec{u}\times(\vec{u}\times\vec{v})] \tag{20}$$

Expressed in this form, rotation of a vector via a quaternion requires only 15 multiplies and 12 adds.[3]

4 Computational Comparison of Transforms and Quaternion/Vector Pairs

4.1 Internal Representation

For the purposes of the forthcoming discussion, let us assume that a quaternion $q = [s, \langle x, y, z\rangle]$ is stored in a straightforward fashion as a pair $(s, \vec{v})$, thus requiring storage for 4 floating point values.[4] A transform $\mathcal{T} = [\hat{n}\ \hat{o}\ \hat{a}\ \vec{p}]$, however, contains redundant information and its representation can thus be compacted. In particular, since the rotational part of $\mathcal{T}$ consists of 3 mutually orthogonal vectors $\hat{n}$, $\hat{o}$, and $\hat{a}$, we can eliminate one of them from the representation and recover it, when needed, as the cross product of the remaining two. Usually, $\hat{n}$ is omitted (note that $\hat{n} = \hat{o}\times\hat{a}$) and only the triple $(\hat{o}, \hat{a}, \vec{p})$ is explicitly stored. Note that this constitutes a mixed blessing —

[2] Alternatively, Eq.(17) can be verified by showing that it is equivalent to the Rodrigues's formula, *i.e.*, $q * \vec{v} * q^{-1} = C_\theta\vec{v} + S_\theta(\hat{k}\times\vec{v}) + (1 - C_\theta)(\hat{k}\cdot\vec{v})\hat{k}$. [6]

[3] Note that multiplications by two can be performed by simply incrementing the exponents of the floating point numbers, and have thus been neglected in the above cost analysis.

[4] The quaternion $q = [s, \langle x, y, z\rangle] = r\,(\cos\theta + \hat{k}\sin\theta)$ could also be stored as a triple (ψ, φ, θ), where θ is the angle of the associated rotation, and ψ and φ are Euler angles specifying the direction of the rotational axis $\hat{k}$. Note, however, that this representation becomes singular when $\psi = 0$.

on the one hand, less memory is being used for transform storage and fewer values need to be computed when evaluating transforms; on the other hand, the missing vector $\hat{n}$ needs to be recovered whenever a computation involves a transform. Overall, however, the machine cycles saved not computing the destination vector $\hat{n}$ outweigh the cost of restoring the normal vector of the original transform.

Since the speed of arithmetic operations in conventional computers tends to exceed that of operand fetching (main memory references), quaternion/vector pairs have an advantage over homogeneous transforms as fewer values need to be brought from main memory.

4.2 Sequential Implementations of Spatial Operations

4.2.1 Spatial Transformations

Homogeneous Transforms: In general, a homogeneous transform is any 4×4 matrix over reals and can represent translation, rotation, stretching, perspective transformation or any combination thereof. In robotics we are mainly interested in non-deforming spatial transformations, *i.e.*, rotations and translations without affecting the scale or perspective of the transformed objects.[5] The general form of such a non-deforming homogeneous transform is [9]

$$\mathcal{T} = \begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = \left[\text{Rot}(\hat{k},\theta) \mid \text{Trans}(\vec{p})\right] \qquad (21)$$

where the upper left 3×3 submatrix $\mathcal{M} = [\hat{n}\ \hat{o}\ \hat{a}]$ corresponds to the rotational portion and the vector $\vec{p}$ to the translational portion of the transformation. The bottom row elements are refered to as the *scaling factors* and are included to yield a square matrix and thus facilitate inversion and composition of homogeneous transforms. Alternatively, we will often refer to the rotational matrix $\mathcal{M}$ as $\text{Rot}(\hat{k},\theta)$ (*i.e.*, a rotation through an angle θ about the axis $\hat{k}$) and to the translation vector as $\text{Trans}(\vec{p})$, yielding an abbreviated notation for a homogeneous transform (as shown in Eq.(21)).

Let $\mathcal{T}$ be a homogeneous transform. Then, for an arbitrary vector $\vec{v}$,

$$\begin{aligned} \mathcal{T}\vec{v} &= \left[\text{Rot}(\hat{k},\theta) \mid \text{Trans}(\vec{p})\right][\vec{v}] \\ &= \left[\text{Rot}(\hat{k},\theta)\vec{v}\right] + [\vec{p}] \end{aligned} \qquad (22)$$

i.e., the transform $\mathcal{T}$ rotates the given vector $\vec{v}$ and adds the displacement vector $\vec{p}$ to the rotated vector $\vec{v}' = \text{Rot}(\hat{k},\theta)\vec{v}$. A brief analysis shows that the operation of Eq.(22) requires 15 multiplies and 12 adds, where $6*, 3+$ are needed to recover $\hat{n}$ of the given transform, and $9*, 9+$ are required to compute the components of the rotated vector.

Quaternion/Vector Pairs: Clearly, the above operation can be simulated using quaternions by simply rotating $\vec{v}$ through the quaternion operator (q, q^{-1}), where $q = (\cos\frac{\theta}{2} + \hat{k}\sin\frac{\theta}{2})$ and adding $\vec{p}$ to the resulting rotated vector $\vec{v}'$, *i.e.*,

$$\vec{v}'' = q * \vec{v} * q^{-1} + \vec{p} \qquad (23)$$

Now, recall that the expression $q*\vec{v}*q^{-1}$ (Eq.(18)) can be evaluated using $15*, 12+$. With the additional 3 adds to add on $\vec{p}$, the cost of a quaternion spatial transformation (Eq.(23)) is 15 multiplies and 15 adds. Note that no normalization is performed on the incoming quaternion.

4.2.2 Composition of Transformations

Homogeneous Transforms: Let $\mathcal{T} = [\hat{n}\ \hat{o}\ \hat{a}\ \vec{p}]$ and $\mathcal{T}' = [\hat{n}'\ \hat{o}'\ \hat{a}'\ \vec{p}']$ be homogeneous transforms and consider their product $\mathcal{T}\mathcal{T}'$. We have,

$$\mathcal{T}\mathcal{T}' = \begin{bmatrix} \vec{x}\cdot\hat{n}' & \vec{x}\cdot\hat{o}' & \vec{x}\cdot\hat{a}' & \vec{x}\cdot\vec{p}' + p_x \\ \vec{y}\cdot\hat{n}' & \vec{y}\cdot\hat{o}' & \vec{y}\cdot\hat{a}' & \vec{y}\cdot\vec{p}' + p_y \\ \vec{z}\cdot\hat{n}' & \vec{z}\cdot\hat{o}' & \vec{z}\cdot\hat{a}' & \vec{z}\cdot\vec{p}' + p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (24)$$

[5] Throughout the remainder of this paper we will refer to the non-deforming pure rotational/translational homogeneous transforms simply as homogeneous transforms.

where $\vec{x} = \langle n_x, o_x, a_x\rangle$, $\vec{y} = \langle n_y, o_y, a_y\rangle$, and $\vec{z} = \langle n_z, o_z, a_z\rangle$. Therefore, in order to compute $\mathcal{T}\mathcal{T}'$, we need to

1. recover $\hat{n}$ of the original transform $\mathcal{T}$ $(6*, 3+)$
2. form vectors $\vec{x}$, $\vec{y}$, and $\vec{z}$
3. compute new $\hat{o}$, $\hat{a}$, and $\vec{p}$ $(27*, 21+)$

Note that $\hat{n}'$ need not be recovered as the $\hat{n}$ vector of the resulting composition is not stored and thus need not be computed. The cost of composing two homogeneous transforms without normalizing the result, then, is 33 multiplies and 24 adds.

Quaternion/Vector Pairs: Now, letting $\mathcal{T} = [\text{Rot}(\hat{k},\theta) \mid \text{Trans}(\vec{p})]$ and $\mathcal{T}' = [\text{Rot}(\hat{k}',\theta') \mid \text{Trans}(\vec{p}')]$, we can rewrite Eq.(24) as follows

$$\mathcal{T}\mathcal{T}' = [\text{Rot}(\hat{k},\theta)\text{Rot}(\hat{k}',\theta') \mid \text{Trans}(\text{Rot}(\hat{k},\theta)\vec{p}' + \vec{p})] \qquad (25)$$

i.e., the net effect of the composition $\mathcal{T}\mathcal{T}'$ is the composite rotation $\text{Rot}(\hat{k},\theta)\text{Rot}(\hat{k}',\theta')$, followed by the translation $\text{Rot}(\hat{k},\theta)\vec{p}' + \vec{p}$. Now, given two quaternion/vector pairs $(q,\vec{v})$ and $(q',\vec{v}')$, representing the spatial transformations $\mathcal{T}$ and $\mathcal{T}'$, respectively, we can compose the two transformations as follows

$$(q,\vec{v}) * (q',\vec{v}') = (q * q', q * \vec{v}' * q^{-1} + \vec{v}) \qquad (26)$$

It can be easily verified that $16*, 12+$ are needed for quaternion multiplication (Eq.(6)). Also, as we saw in Section 4.2.1, the translational part of Eq.(26) requires $15*, 15+$, bringing the total cost of composing quaternion spatial transformations to 31 multiplies and 27 adds.

The fact that $\text{Rot}(\hat{k},\theta) = (e^{\hat{k}\frac{\theta}{2}}, e^{-\hat{k}\frac{\theta}{2}})$ (Eq.(14)) may seem to suggest a faster approach to composing rotations; namely, multiplying rotational operators in exponential form and adding the exponents. However, it can be shown [10][2] that

$$\forall\, p, q \in \mathbf{Q} : e^p * e^q = e^{p+q} \text{ iff } p * q = q * p \qquad (27)$$

and so, in general, rotations can not be composed by simply adding the exponents of their exponential representations.

4.2.3 Inverse of a Spatial Transformation

Homogeneous Transforms: Given a homogeneous transform $\mathcal{T} = [\text{Rot}(\hat{k},\theta) \mid \text{Trans}(\vec{p})] = [\hat{n}\ \hat{o}\ \hat{a}\ \vec{p}]$, its inverse $\mathcal{T}^{-1}$ is defined as follows [9]

$$\mathcal{T}^{-1} = \begin{bmatrix} n_x & n_y & n_z & -\vec{p}\cdot\hat{n} \\ o_x & o_y & o_z & -\vec{p}\cdot\hat{o} \\ a_x & a_y & a_z & -\vec{p}\cdot\hat{a} \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (28)$$

Note that the rotational matrix of the inverse transformation $\mathcal{T}^{-1}$ is simply the transpose of the rotational matrix of the original transformation. This is due to the orthogonality of the matrix $[\hat{n}\ \hat{o}\ \hat{a}]$. Postmultiplying the inverted (transposed) rotational matrix by the translational column vector $\vec{p}$, we have

$$\begin{bmatrix} n_x & n_y & n_z \\ o_x & o_y & o_z \\ a_x & a_y & a_z \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} = \begin{bmatrix} \vec{p}\cdot\hat{n} \\ \vec{p}\cdot\hat{o} \\ \vec{p}\cdot\hat{a} \end{bmatrix} \qquad (29)$$

and so the inverse transformation $\mathcal{T}^{-1}$ can be written as

$$\mathcal{T}^{-1} = [\text{Rot}(\hat{k},-\theta) \mid \text{Trans}(-\text{Rot}(\hat{k},-\theta)\vec{p})] \qquad (30)$$

In terms of computational expense, the homogeneous transform inverse requires

1. one cross product to recover $\hat{n}$ $(6*, 3+)$
2. three dot products to compute the new value of $\vec{p}$ $(9*, 6+)$
3. some pointer rearrangements to accomodate the transpose operation

for a total of 15 multiplications and 9 additions.

Quaternion/Vector Pairs: To derive an equivalent expression for the inverse of a quaternion/vector pair, let $(q,\vec{v})$ be the quaternion/vector pair corresponding to the transformation $\mathcal{T}$. Then it follows from Eq.(30) that

$$\mathcal{T}^{-1} = (q, \vec{v})^{-1} = (q^{-1}, -q^{-1} * \vec{v} * q) \qquad (31)$$

Therefore, the cost of performing a quaternion/vector pair inverse is the sum of the costs of computing a quaternion inverse and a quaternion rotation. Since the computation of a (unit) quaternion inverse only requires a change in sign of its vector part, the total cost essentially reduces to the 15 multiplies and 12 adds needed for the quaternion rotation.

4.2.4 Normalization of Rotational Operators

Homogeneous Transforms: As repeated calculations are performed on rotational operators (rotational matrices or quaternions), they may degenerate slightly, *i.e.*, become non-orthogonal or of non-unit magnitude. Thus there is a need to occasionally renormalize both rotational matrices and quaternions.

Recall that rotational matrices are embedded in homogeneous transforms as their upper left 3×3 submatrices, *i.e.*, $\mathcal{T} = [\mathcal{M}\ \vec{p}]$ (where the scaling factors have been ignored). Now, in order for a rotational matrix $\mathcal{M} = [\vec{n}\ \vec{o}\ \vec{a}]$ to represent a non-deforming 3-D rotation, we must have

$$\vec{n} = \vec{o} \times \vec{a}, \qquad \vec{o} = \vec{a} \times \vec{n}, \qquad \text{and} \qquad |\vec{n}| = |\vec{o}| = |\vec{a}| = 1 \qquad (32)$$

i.e., the vectors $\vec{n}, \vec{o}$, and $\vec{a}$ must be mutually orthogonal and of unit magnitude.

Let $\mathcal{T} = [\vec{n}\ \vec{o}\ \vec{a}\ \vec{p}]$ be an arbitrary homogeneous transform, whose rotational matrix $\mathcal{M} = [\vec{n}\ \vec{o}\ \vec{a}]$ is to be normalized. In accordance with the representational convention of Section 4.1, only vectors $\vec{o}$ and $\vec{a}$ of $\mathcal{M}$ are available to us. There are several reasonable approaches to the renormalization problem. We propose the following procedure:

1. assume the orientation of the $\vec{o}$-$\vec{a}$ plane correct and reconstruct $\vec{n}$ as $\vec{o} \times \vec{a}$ $(6*, 3+)$
2. obtain $\vec{a}'$, a first correction to $\vec{a}$, as $\vec{n} \times \vec{o}$ $(6*, 3+)$
3. compute $\vec{a}''$, the final correction to $\vec{a}$, as the vector sum of $\vec{a}$ and $\vec{a}'$ $(3+)$
4. obtain $\vec{o}'$, the new approximation for $\vec{o}$, as $\vec{a}'' \times \vec{n}$ $(6*, 3+)$
5. scale the new axes into unit vectors, *i.e.*, let $\hat{o} = \frac{\vec{o}'}{|\vec{o}'|}, \hat{a} = \frac{\vec{a}''}{|\vec{a}''|}$ $(12*, 4+, 2\surd)$

Note that the assumption in step 1 above may well not be quite accurate. However, as $\vec{n}$ is absent from the representation of $\mathcal{T}$ (and hence $\mathcal{M}$), we have no corrective information whatsoever and we are forced to assume that the orientation of the $\vec{o}$-$\vec{a}$ plane is correct. Also, observe that the directional corrections to the vectors $\vec{o}$ and $\vec{a}$ (steps 2 through 4) distribute the error between the two vectors (again, nothing but a reasonable guess). Finally, in step 5, we compute the unit vectors along the computed axes (note that $\hat{n}$ need not be computed as it is not explicitly stored). Adding up the computational costs listed at each step, we see that the procedure requires 30 multiplies, 16 adds, and 2 square-roots.

Quaternion/Vector Pairs: Contrast this with the quaternion normalization procedure suggested by Eq.(8), *i.e.*,

$$\hat{q} = \frac{q}{||q||} = \frac{q}{\sqrt{s^2 + |\vec{v}|^2}} \qquad (33)$$

where $\hat{q}$ denotes the normalized equivalent of q. It is easy to see that the cost of normalizing a quaternion rotational operator is 8 multiplies, 3 adds, and 1 square-root, which is clearly substantially cheaper than normalizing a rotational matrix.

Sections 4.2.1, 4.2.2 and 4.2.3 compared the computational costs of performing spatial transformations, compositions of transformational operators and inverse transformations using homogeneous transforms *vs* using quaternion/vector pairs. We saw that for the case where the rotational operators are not normalized neither before nor after the computations, the two approaches are essentially equivalent. In view of the discussion above, however, it is apparent that if the cost of normalization is included in the above operations, the computational balance tips strongly in favor of performing the operations using quaternions. Refer to the table in Section 4.4 for the exact costs of the normalized versions of the operations.

4.3 Parallel Implementations of Spatial Operations

The computational complexities of the parallel algorithms proposed in the following sections are analyzed in terms of the number of independent processors that they require, as well as the number of CPU cycles consumed during execution. A CPU cycle is defined as the time needed (by the hardware) to execute a floating point addition or multiplication, and may or may not correspond to the actual harware CPU cycle time on a given machine.

4.3.1 Spatial Transformations

Homogeneous Transforms: Ideally (*i.e.*, given a sufficient number of independent processors), the general spatial transformation using homogeneous transforms (Eq.(22)) can be performed in 4 CPU cycles:

1. - form the 6 cross-product terms of $\hat{o} \times \hat{a}$
   - compute $A_{13} = a_x v_z, A_{23} = a_y v_z, A_{33} = a_z v_z$ [9]
2. - compute n_x, n_y, n_z
   - compute $A_{12} = o_x v_y, A_{22} = o_y v_y, A_{32} = o_z v_y$
   - compute $A'_1 = A_{13} + p_x, A'_2 = A_{23} + p_y, A'_3 = A_{33} + p_z$ [9]
3. - compute $A_{11} = n_x v_x, A_{21} = n_y v_x, A_{31} = n_z v_x$
   - compute $A''_1 = A_{12} + A'_1, A''_2 = A_{22} + A'_2, A''_3 = A_{32} + A'_3$ [6]
4. - compute $v''_x = A_{11} + A'_1, v''_y = A_{12} + A'_2, v''_z = A_{13} + A'_3$ [3]

Procedure 1: Parallel implementation of a spatial transformation using homogeneous transforms

where $\vec{v}'' = \mathcal{T}\vec{v}$, *i.e.*, $\vec{v}''$ corresponds to the final transformed vector. Note that the number of independent processors needed to execute each of the steps of Procedure 1 in a single CPU cycle is indicated (in brackets) following the description of the step. Hence, 9 processors will suffice to execute the above procedure in optimal time.

Quaternion/Vector Pairs: Combining Eq.(20) and Eq.(23), we can write the general expression for a spatial transformation (using quaternion/vector pairs) as follows

$$\vec{v}'' = \vec{v} + 2s(\vec{u} \times \vec{v}) + 2\vec{u} \times (\vec{u} \times \vec{v}) + \vec{p} \qquad (34)$$

where $q = [s, \vec{u}]$. Thus, given a 9-processor computing machine, the above computation can be carried out in 6 CPU cycles as follows:

1. - form the 6 cross-product terms of $\vec{u} \times \vec{v}$ [6]
2. - compute $(\vec{u} \times \vec{v})_x, (\vec{u} \times \vec{v})_y, (\vec{u} \times \vec{v})_z$
   - compute $2\vec{u}, 2s$ [7]
3. - compute the 6 cross-product terms of $\mathcal{A} = 2\vec{u} \times (\vec{u} \times \vec{v})$
   - compute $\mathcal{B} = 2s(\vec{u} \times \vec{v})$ [9]
4. - compute $\mathcal{A}_x, \mathcal{A}_y, \mathcal{A}_z$
   - compute $\mathcal{C} = \vec{v} + \mathcal{B}$ [6]
5. - compute $\mathcal{D} = \mathcal{C} + \mathcal{A}$ [3]
6. - compute $\vec{v}'' = \mathcal{D} + \vec{p}$ [3]

Procedure 2: Parallel implementation of a spatial transformation using quaternion/vector pairs

4.3.2 Composition of Transformations

Homogeneous Transforms: To implement the computation specified by Eq.(24) efficiently, we can overlap the computation of $\hat{n}$ with the formation of the product terms not involving n_x, n_y, and n_z. Procedure 3 below implements this strategy, using 24 fully parallel processors and 4 CPU cycles.[6] Note that the final composite matrix is labeled $\mathcal{D}$.

1. - form the 6 cross-product terms of $\hat{o} \times \hat{a}$ [24]
   - compute the 18 dot-product terms not involving n_x, n_y, n_z
2. - compute n_x, n_y, n_z [12]
   - compute $\mathcal{A}_1 = o_x o'_y + a_x o'_x, \mathcal{A}_2 = o_y o'_y + a_y o'_x, \mathcal{A}_3 = o_z o'_y + a_z o'_x$
   - compute $\mathcal{B}_1 = o_x a'_y + a_x a'_x, \mathcal{B}_2 = o_y a'_y + a_y a'_x, \mathcal{B}_3 = o_z a'_y + a_z a'_x$
   - compute $\mathcal{C}_1 = o_x p'_y + a_x p'_x, \mathcal{C}_2 = o_y p'_y + a_y p'_x, \mathcal{C}_3 = o_z p'_y + a_z p'_x$
3. - compute the 9 dot-product terms involving n_x, n_y, n_z
   - compute $\mathcal{C}'_1 = \mathcal{C}_1 + p_x, \mathcal{C}'_2 = \mathcal{C}_2 + p_y, \mathcal{C}'_3 = \mathcal{C}_3 + p_z$ [12]
4. - compute $\mathcal{D}_{12} = \mathcal{A}_1 + n_x o'_x, \mathcal{D}_{22} = \mathcal{A}_2 + n_y o'_x, \mathcal{D}_{32} = \mathcal{A}_3 + n_z o'_x$
   - compute $\mathcal{D}_{13} = \mathcal{B}_1 + n_x a'_x, \mathcal{D}_{23} = \mathcal{B}_2 + n_y a'_x, \mathcal{D}_{33} = \mathcal{B}_3 + n_z a'_x$

[6] The reader may want to expand the dot products of Eq.(24) to expose the motivation for the approach of Procedure 3.

- compute $\mathcal{D}_{14} = C_1' + n_x p_x', \mathcal{D}_{24} = C_2' + n_y p_x', \mathcal{D}_{34} = C_3' + n_z p_x'$ [9]

Procedure 3: Parallel implementation of composition of spatial transformations using homogeneous transforms

Quaternion/Vector Pairs: The procedure for computing transformational composition using quaternion/vector pairs is given by Eq.(26). Observe that the translational portion of the resulting composition corresponds precisely to the expression for a general spatial transformation (Eq.(23)). Recall from Section 4.3.1 that a parallel implementation of Eq.(23) requires 6 CPU cycles and 9 CPUs. Now, the rotational portion of the resulting composition in Eq.(26) can be computed in 3 CPU cycles, using a total of 16 parallel CPUs as follows (see Eq.(6)):

1. - form the 6 cross-product terms of $\vec{u} \times \vec{u}'$
   - compute $ss', u_x u_x', u_y u_y', u_z u_z', s\vec{u}', s'\vec{u}$ [16]
2. - compute $\mathcal{A} = ss' - u_x u_x', \mathcal{B} = u_y u_y' + u_z u_z'$
   - compute $(\vec{u} \times \vec{u}')_x, (\vec{u} \times \vec{u}')_y, (\vec{u} \times \vec{u}')_z, C = s\vec{u}' + s'\vec{u}$ [8]
3. - compute $s'' = \mathcal{A} - \mathcal{B}, \vec{u}'' = C + (\vec{u} \times \vec{u}')$ [4]

Procedure 4: Parallel implementation of quaternion composition

where $q = [s, \vec{u}]$, $q' = [s', \vec{u}']$ are the quaternions being composed and $q'' = q * q' = [s'', \vec{u}'']$ is the resulting composition.

Now, overlapping the computations of the rotational (Procedure 4) and translational (Procedure 2) portions of the composition $(q'', \vec{v}'')$, we can easily arrange to carry out the complete procedure using at most 6 CPU cycles and 22 CPU's.

4.3.3 Inverse of a Spatial Transformation

Homogeneous Transforms: The operation of Eq.(28) can be performed on a parallel machine in 5 CPU cycles using 6 processors as follows

1. - form the 6 cross-product terms of $\hat{o} \times \hat{a}$ [6]
2. - compute n_x, n_y, n_z
   - compute $p_x o_x, p_y o_y, p_z o_z$ [6]
3. - compute $p_x n_x, p_y n_y, p_z n_z$
   - compute $p_x a_x, p_y a_y, p_z a_z$ [6]
4. - compute $\mathcal{A}_1 = p_x n_x + p_y n_y$
   - compute $\mathcal{A}_2 = p_x o_x + p_y o_y$
   - compute $\mathcal{A}_3 = p_x a_x + p_y a_y$ [3]
5. - compute $\mathcal{A}_1' = \mathcal{A}_1 + p_z n_z$
   - compute $\mathcal{A}_2' = \mathcal{A}_2 + p_z o_z$
   - compute $\mathcal{A}_3' = \mathcal{A}_3 + p_z a_z$ [3]

Procedure 5: Parallel implementation of the inverse of a spatial transformation using homogeneous transforms

Quaternion/Vector Pairs: As we saw in Section 4.2.3, the computation of a quaternion/vector pair inverse is dominated by the operation of quaternion rotation (Eq.(31)). A parallel procedure outlining the optimal sequence of computations needed to implement a quaternion rotation was given in Section 4.3.1 (Procedure 2, steps 1 – 5). The calculation of the quaternion inverse (*i.e.*, vector negation) can be easily incorporated into the procedure without increasing the amount of needed resources. Thus, given a quaternion/vector pair, the corresponding inverse transformation can be computed on a parallel machine in 5 CPU cycles using 9 processors.

4.3.4 Normalisation of Rotational Operators

Glancing back at the steps outlining the normalisation procedure for rotational matrices (Section 4.2.4) it is apparent that the procedure is very much sequential in nature, *i.e.*, each step requires all preceding steps to be completed before its own computations can proceed. Thus, relatively little parallelism can take place and, in fact, a brief analysis reveals that at least 6 processors and 12 CPU cycles are needed to carry out the computations. Thus, the CPU cycle cost of homogeneous transform normalisation by far dominates the cost of performing any one of the three operations discussed above (Procedures 1,3,5). Since the normalisation process must either precede (follow) the operation whose argument (result) is being normalised, the total number of CPU cycles needed for a normalising computation is the sum of the number of cycles needed for each of the two stages. The necessary and sufficient number of processors required for the entire computation, on the other hand, is the maximum of the processor requirements for the two stages. See the table in Section 4.4 for details.

The procedure for normalising quaternion rotational operators, as suggested by Eq.(33), is again not inherently parallel, but can, due to its simplicity, be carried out using only 4 processors and 5 CPU cycles. Again, since the normalisation and the actual computation must be executed serially, the CPU cycle cost of normalisation is added onto the cost of the operation. The table of Section 4.4 summarises the results.

4.4 Summary

Section 4 has been devoted exclusively to comparing the computational efficiency of homogeneous transforms and quaternion/vector pairs. We have looked at both sequential and parallel implementations of spatial transformations, their compositions and inverses. The following table provides a summary of the results.

Homogeneous Transforms *vs* Quaternion/Vector Pairs

Operation	Norm	Sequential Execution						Parallel Execution			
		HT			Q/V			HT		Q/V	
		*	+	$\surd$	*	+	$\surd$	CPUs	cycles	CPUs	cycles
Spatial Trans	–	15	12	0	15	15	0	9	4	9	6
Composition	–	33	24	0	31	27	0	24	4	22	6
Inverse	–	15	9	0	15	12	0	6	5	9	5
Spatial Trans	$\surd$	45	28	2	23	18	1	9	16	9	11
Composition	$\surd$	63	40	2	39	30	1	24	16	22	11
Inverse	$\surd$	45	25	2	23	15	1	6	17	9	10

The symbols and abbreviations used in the above table have the following interpretations: HT = Homogeneous Transform, Q/V = Quaternion/Vector pair, a checkmark ($\surd$) in the Norm column indicates a normalised operation, whereas a dash (–) denotes absence of normalisation.

5 Inverse Kinematics for the Puma Robot Arm

The inverse kinematic problem for the Puma can be stated as

$$q_1 * q_2 * q_3 * q_4 * q_5 * q_6 = Q \tag{35}$$

where $q_i = (\cos\frac{\theta_i}{2} + \hat{k}_i \sin\frac{\theta_i}{2})$, $\hat{k}_1 = \hat{k}_4 = \hat{k}_6 = \langle 0, 0, 1 \rangle$, $\hat{k}_2 = \hat{k}_3 = \hat{k}_5 = \langle 0, 1, 0 \rangle$, Q is the given orientation of the wrist, and θ_i, $1 \leq i \leq 6$, are the desired angular displacements. To avoid complicated quaternion product terms, we can approach the problem by solving for $\theta_1, \theta_2, \theta_3$

5 Inverse Kinematics for Puma 560

The inverse kinematic problem for the Puma can be stated as the problem of retrieving the joint angles θ_i, $1 \le i \le 6$, from the equation

$$\begin{aligned}(\mathrm{R}_w, \mathrm{T}_w) &= \mathrm{T}(z,a_1)\,\mathrm{R}(z,\theta_1)\,\mathrm{T}(y,a_2)\,\mathrm{R}(y,\theta_2)\,\mathrm{T}(z,a_3)\,\mathrm{R}(y,\theta_3) \\ &\quad \mathrm{T}(x,a_4)\,\mathrm{T}(z,a_5)\,\mathrm{R}(z,\theta_4)\,\mathrm{R}(y,\theta_5)\,\mathrm{R}(z,\theta_6) \qquad (35)\end{aligned}$$

Here T and R denote translations along and rotations about the specified coordinate axes, a_i are known translational offsets, and $(\mathrm{R}_w, \mathrm{T}_w)$ represents the known orientation and displacement of the robot's wrist with respect to the base coordinate frame. The first three angles θ_1, θ_2, and θ_3 can be recovered from the spatial position of the wrist (T_w) using geometric or analytical techniques. In the context of our investigation of quaternions, their derivation is not as interesting as the solution for the orientational angles θ_4, θ_5, and θ_6, and will thus be omitted from this presentation. The interested reader is refered to [2] for a complete solution.[7]

Let $\mathrm{R}_w = [w, \langle a, b, c\rangle]$ and assume that $\theta_1, \theta_2, \theta_3$ are known. Then, we can write the orientational equation for the wrist as follows

$$\begin{aligned}q_4 * q_5 * q_6 &= q_3^{-1} * q_2^{-1} * q_1^{-1} * \mathrm{R}_w \\ &= [s, \langle x, y, z\rangle] \qquad (36)\end{aligned}$$

where q_i are the quaternions corresponding to the rotations in Eq.(35). Evaluating the right-hand side of Eq.(36) gives

$$\begin{aligned}s &= C(\theta_2' + \theta_3')[C(\theta_1')w + S(\theta_1')c] + S(\theta_2' + \theta_3')[C(\theta_1')b - S(\theta_1')a] \\ x &= C(\theta_2' + \theta_3')[C(\theta_1')a + S(\theta_1')b] - S(\theta_2' + \theta_3')[C(\theta_1')c - S(\theta_1')w] \\ y &= C(\theta_2' + \theta_3')[C(\theta_1')b - S(\theta_1')a] - S(\theta_2' + \theta_3')[C(\theta_1')w + S(\theta_1')c] \\ z &= C(\theta_2' + \theta_3')[C(\theta_1')c - S(\theta_1')w] + S(\theta_2' + \theta_3')[C(\theta_1')a + S(\theta_1')b]\end{aligned}$$

with $\theta_i' = \frac{\theta_i}{2}$, and S and C denoting sin and cos, respectively. Expanding $q_4 * q_5 * q_6$ (see [2]) and equating it to $[s, \langle x, y, z\rangle]$ yields

$$\begin{aligned}s &= \cos(\theta_5')\cos(\theta_4' + \theta_6') & x &= \sin(\theta_5')\sin(\theta_6' - \theta_4') \\ y &= \sin(\theta_5')\cos(\theta_6' - \theta_4') & z &= \cos(\theta_5')\sin(\theta_4' + \theta_6')\end{aligned}$$

Notice that $s^2 + z^2 = \cos^2(\theta_5')$ and $x^2 + y^2 = \sin^2(\theta_5')$. Since $-180° \le \theta_5 < 180°$, we have $-90° \le \theta_5' < 90°$, and so $\cos(\theta_5') = +\sqrt{s^2 + z^2}$ and $\sin(\theta_5') = \pm\sqrt{x^2 + y^2}$. Therefore,

$$\theta_5 = 2\arctan\frac{\pm\sqrt{x^2 + y^2}}{+\sqrt{s^2 + z^2}} \qquad (37)$$

where the positive and negative signs of the numerator correspond to the wrist configurations with $\theta_5 > 0°$ and $\theta_5 < 0°$, respectively.

Assuming for the moment that θ_5 has a nonsingular value (*i.e.*, $\theta_5 \neq 0°, 180°$), both $\dot{S}_5$ and $\dot{C}_5$ are nonzero and we can cancel them from the ratios x/y and z/s, respectively, to obtain

$$\frac{x}{y} = \tan(\theta_6' - \theta_4') \qquad \frac{z}{s} = \tan(\theta_4' + \theta_6') \qquad (38)$$

and therefore

$$\theta_4 = \arctan\frac{z}{s} - \arctan\frac{x}{y} \qquad \theta_6 = \arctan\frac{z}{s} + \arctan\frac{x}{y} \qquad (39)$$

Observe that as θ_5 approaches 0°, $\sin(\theta_5')$ goes to 0, and the term $\arctan(x/y)$ becomes undefined. However, the *sum* of θ_4 and θ_6 remains well defined, as expected, since the condition $\theta_5 = 0°$ corresponds to the axes of the joints 4 and 6 being *similarly coaxial*. Similarly, as θ_5 approaches 180°, $\cos(\theta_5')$ approaches 0, and the term $\arctan(z/s)$ becomes progressively less well defined. However, the *difference* of the two angles $\theta_6 - \theta_4$ is preserved, corresponding to the fact that the axes of joints 4 and 6 are *oppositely coaxial* when $\theta_5 = 180°$. These considerations are consistent with the physical reality of the Euler wrist, which loses a degree of freedom as it approaches a singular configuration.

The following table summarizes the cost of the above procedure.[8]

[7] The development in this section is based in part on the unpublished work of Russel Taylor.

Inverse Kinematics for Puma 560				
	Cost			
Component	*	+	$\sqrt{}$	trig
θ_1	2	2	1	2
θ_2	5	4	1	4
θ_3	2	4	0	4
$[s, \langle x, y, z\rangle]$	16	8	0	4
θ_4	0	1	0	2
θ_5	4	2	2	1
θ_6	0	1	0	0
Total	29	22	4	17

6 Discussion

The main purpose of this paper was to compare the computational efficiency of some of the common spatial operations using homogeneous transforms on the one hand, and quaternion/vector pairs on the other. The table of Section 4.4 summarizes the main results, indicating that the two approaches are virtually equivalent for the case of *non-normalizing sequential* procedures, but also that the *non-normalizing* vectorial algorithms parallelize slightly better than their algebraic counterparts. If the cost of normalizing the rotational operator is included in the total cost, however, the quaternion/vector approach yields much more efficient implementations on both single and multi-processor systems. Clearly, the non-normalizing procedures (where the rotational operators are never normalized) and their normalized versions as defined above (where the rotational operators are normalized at every call) correspond to the two extremes. In practice, the need for normalization will be intermediate to the above two cases, depending mostly on the presence of chains of products, which are likely to denormalize rotational operators. Note, however, that the cost of testing for whether or not a rotational matrix needs renormalization (Eq.(32)) accounts for approximately $\frac{2}{3}$ of the cost of the normalizing process itself. Consequently, it is uncertain if normalizing homogeneous transforms "by need" will result in a more efficient overall performance.

In summary, the differences in computational efficiency between the two approaches are not sufficiently significant to warrant a particular choice on that basis alone. Instead, the deciding factor may be the nature of a particular computation or the available hardware.

[8] The above cost analysis assumes that inverse trigonometric functions are supplied with two separate arguments (rather than their quotient) to be able to determine the correct quadrant of the desired angle. Furthermore, any subexpressions involving the constant offsets a_i are assumed to have been precomputed off-line. Finally, since multiplications by two correspond to simply incrementing the exponents of the floating point numbers, their contribution to the overall cost has been ignored.

References

[1] Otto F. Fischer. *Five mathematical Structural Models in Natural Philosophy with Technical Physical Quaternions*, Chapters I & II. Axion Institute, N. Kunsvagen 8, Lindigo, Stockholm, 1957.

[2] Janez Funda. *Quaternions and Homogeneous Transforms in Robotics.* Technical Report MS-CIS-88-06, University of Pennsylvania, 1988.

[3] Sir William R. Hamilton. *Elements of Quaternions.* Volumes I & II, Chelsea Publishing Company, New York, NY, 1869.

[4] Sir William R. Hamilton. *Lectures on Quaternions.* Whittaker & Co., Ave-Maria Lane, London, 1853.

[5] I. N. Hernstein. *Topics in Algebra*, Chapter 7. John Wiley & Sons, Inc., 1975.

[6] Berthold Horn. Closed-form solution of absolute orientation using quaternions. *Journal of the optical society of America*, 4, April 1987.

[7] Berthold Horn. New notation for serial kinematic chains. Unpublished paper.

[8] Michael McCarthy. Complex rotational operators. Class notes.

[9] Richard P. Paul. *Robot Manipulators.* MIT Press, 1981.

[10] E. Pervin and J. Webb. *Quaternions in Vision and Robotics.* Technical Report, Carnegie-Mellon University, 1980.