

Coordinate Transformation: A Solution Algorithm for One Class of Robots

LORENZO SCIAVICČO AND BRUNO SICILIANO

Abstract—One of the most important features of an advanced control system for articulated robots is the capability of transforming the work space coordinates, which naturally characterize any robot task, into the corresponding joint coordinates, on which control actions are developed. For each task, the coordinate transformation problem consists in calculating one trajectory in the joint space which corresponds to the end effector trajectory, usually given in the Cartesian space. While simple kinematical structures allow for closed-form solutions, there is a class of robots for which this is not true. Typical articulated robot structures have three revolute joints at the end effector; the geometric parameters of these joints actually determine the spatial configuration of the last axes of motion. The large majority of today's nonredundant structures have three intersecting axes at the end effector, and closed form solutions do exist in this case. If the axes intersect two-by-two, as in some rather common arm design, an exact solution seems not to exist. A quite different solution algorithm is established, as compared to the trigonometric approach widely adopted so far, which yields solutions in case of two-by-two intersecting axes. The convergence of the algorithm along any trajectory is proved. Effectiveness of the proposed technique can be argued by the fact that it only makes use of direct kinematics, thus resulting in a contained computational burden. A robot prototype of the kind described above is taken as a reference in order to discuss digital implementation and develop numerical examples.

INTRODUCTION

A SERIAL-LINK manipulator consists of a sequence of rigid bodies, called links, connected by actuated joints. Such a structure forms a kinematic chain and may be characterized through the link parameters that define the mutual position of the two connective joint coordinates [1]. For each joint, three of these parameters are properly geometric and do not change while the arm moves; whereas the one which accomplishes the joint motion is referred to as the joint variable. The basis for all advanced manipulator control is the relationship between the Cartesian coordinates of the end effector, which naturally identify each task, and the joint coordinates of the manipulator. As a rule, the direct (joint-to-Cartesian space) relationship is unique; whereas the inverse (Cartesian-to-joint) is not. Actually, while there is only one end effector state for a given set of joint coordinates, there are a number of different joint configurations which all place the end effector in the same position and orientation. Normally only one solution corresponding to a given kinematic configuration is desired, rather than the entire set of solutions. Usually the solution is to be implemented in real time as it

constitutes the servo-loop reference; a minimal number of mathematical computations is then to be performed.

Typical six-degree-of-freedom (DOF) nonredundant kinematic structures have three revolute joints at the end effector; it is the geometric parameters of such joints that determine the spatial configuration of the terminal axes of motion. Most of today's structures have three intersecting revolute joint axes; the application of the well-known trigonometric method, first proposed in [2], allows for closed form solutions only for these simple structures [3]. In other cases, such as either two-by-two intersecting axes or nonconverging at all axes, an explicit solution seems not to be attainable in closed form. An iterative solution technique has been proposed in [4] for the class of two-by-two intersecting revolute joint axes, but involves an order of magnitude and more computation than a closed form solution, even if the trigonometric approach has still been used. The existence of an explicit solution to the kinematic equations for this class of manipulators is of a great importance in evaluating the robot's suitability for computer control.

The goal of this paper is to demonstrate how the general dynamic approach proposed in [12], and later in [14], can be opportunely adopted for those structures with two-by-two intersecting revolute joint axes at the end effector [6]. The convergence of the resultant algorithm is proved by means of the Lyapunov direct method. Effectiveness of the proposed solution technique mainly lies in the fact that it only makes use of direct kinematics of the manipulator and the extra number of mathematical operations required is small, resulting thus in a contained computational burden. This issue favors the use of this coordinate transformation algorithm along any trajectory assigned in the task space, since the solution sample rates can be the same as those of the servo loops. In addition, it will be shown how joint velocities can be directly generated at servo rate without any further computations; this is of a great deal of utility for those advanced control techniques, such as [7] and [8], which require reference joint velocities as well as joint variables. The same robot prototype as in [4] is taken as a reference in order to analyze the computational burden and develop a case study via numerical simulations.

KINEMATICS

A serial-link manipulator consists of a sequence of links connected by either revolute or prismatic joints. Each joint-link pair constitutes one degree of freedom. Hence

Manuscript received August 12, 1985; revised March 30, 1986. This work was supported by the Ministero della Pubblica Istruzione.

The authors are with the Università di Napoli, Dipartimento di Informatica e Sistemistica, Via Claudio 21, 80125 Napoli, Italy.
IEEE Log Number 8608968.

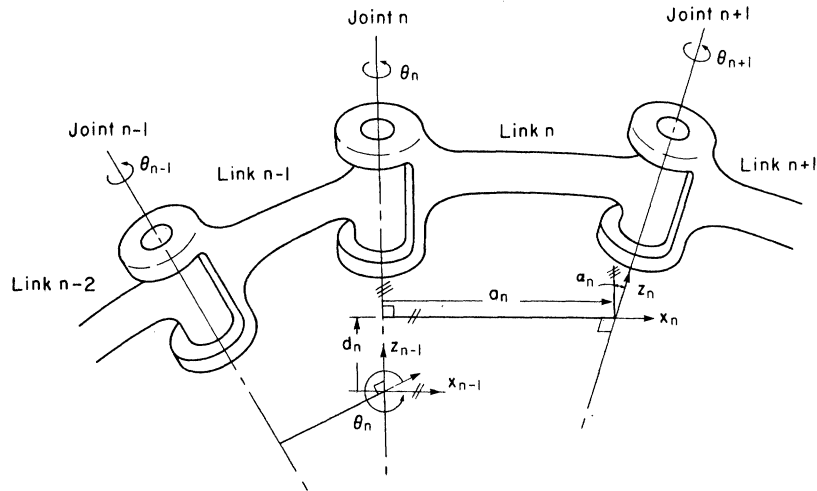


Fig. 1. Kinematic parameters.

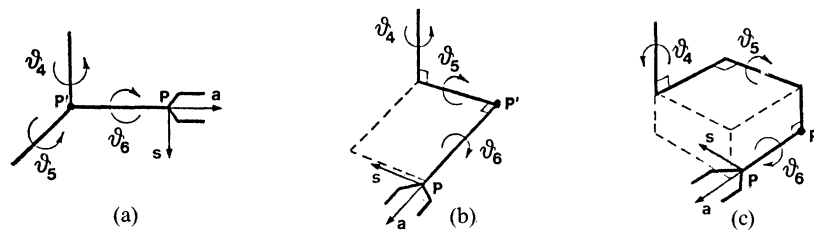


Fig. 2. The last three revolute axes at the end effector. (a) Concurrent, (b) Two-by-two intersecting, (c) Nonconverging.

for a N -degree-of-freedom manipulator, there will be conventionally N joint-link pairs. Adopting the notations given in [1], an orthonormal Cartesian frame (x_n, y_n, z_n) can be established for each link at its joint axis, where $n = 1, 2, \dots, N$, (see Fig. 1) plus the base coordinate frame which is usually taken as the reference one. The relationship between such frames can be expressed through four kinematic parameters: the link parameters (a_n, α_n) determine the structure of the link, the joint parameters (d_n, θ_n) the relative position of neighboring links. A link L_n is then characterized by its length a_n , a common normal distance between the axes of the two joints related to the link, and by the twist angle α_n between the two joint axes. Each joint axis has two normals connected to it, one for each link. The relative position of links L_{n-1} and L_n is defined by d_n , the distance between the normals along the joint n axis, and θ_n , the angle between the normals measured in a plane normal to the joint axis. For a revolute joint, d_n , a_n and α_n are the geometric parameters and remain constant while the joint rotates, while θ_n is the joint variable that accomplishes the rotation. For a prismatic joint, θ_n , a_n , and α_n are the geometric parameters, while d_n is the joint variable which performs the translation.

In order to identify the position and orientation of the end effector, six degrees-of-freedom are needed; typical kinematically nonredundant structures have three revolute joints $(\theta_4, \theta_5, \theta_6)$, whereas the first three joints can be either all revolute, such as the Puma 560 arm [9], or two revolute and one prismatic, such as the Stanford arm [10]. Hence, in the following these joint variables will be denoted (q_1, q_2, q_3) .

As far as the three revolute joint axes at the end effector are concerned, the three basic configurations are illustrated in Fig. 2. The case a) is of particular interest since it becomes possible to decouple the position of the end effector from its orientation, which may be useful for control purposes. The cases b) and c) may also occur in practical robot designs. All such structures can be conveniently identified through the following constraints on the geometric parameters of the last three joints. More specifically, the lengths a_n and the distances d_n are respectively in the following three cases:

- a) concurrent Axes: $a_4 = a_5 = d_5 = 0$ (fig. 2(a));
- b) two-by-two Intersecting Axes: $a_4 = a_5 = 0$, $d_5 \neq 0$ (Fig. 2(b)); and
- c) nonconverging Axes: $a_4 \neq 0$, $a_5 \neq 0$ (Fig. 2(c)).

As a consequence, the knowledge of the geometric parameters of a kinematical structure allows for its classification in terms of the three structures illustrated above.

On the other hand, a robot task is naturally specified in terms of end-effector Cartesian coordinates $(p_x, p_y, p_z, \alpha, \beta, \gamma)$ with respect to the base frame; p_i are the components of the end effector position vector $\mathbf{p}$; α, β, γ are the Euler angles which define its orientation (roll, pitch, and yaw angles can be adopted as well). The orientation, however, can be conveniently described through a unit approach vector $\mathbf{a}$, a unit sliding vector $\mathbf{s}$ and a unit normal vector $\mathbf{n}$, see Fig. 3. The orientation frame $(\mathbf{s}, \mathbf{a}, \mathbf{n})$ defined with reference to the base frame of the manipulator, can be easily determined starting from the Euler angles [11]. Such frame will be referred to in the following

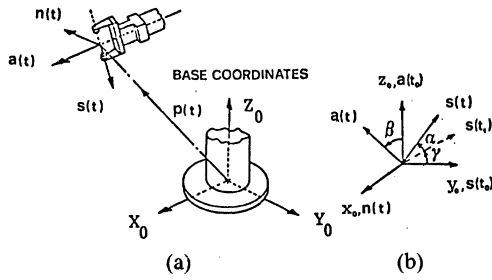


Fig. 3. (a) Orientation unit vectors, (b) Euler angles.

since it allows for a unique definition of the orientation in terms of direct relationship with the joint variables. Under these assumptions, for any robot kinematical structure with known geometric parameters, the direct kinematics can be written as

$$p = f_p(q), \quad s = f_s(q), \quad a = f_a(q) \quad (1)$$

where q is the (6×1) vector of joint coordinates, and f_p, f_s, f_a are nonlinear vector functions which are always unique; $n = f_n(q)$ is redundant since it can be determined as the vector product $s \times a$.

With reference to Fig. 2, the end effector position vector p and the approach unit vector a are always independent of the last rotation θ_6 . Hence as

$$p' = p - d_6 a, \quad (2)$$

p' can be assumed as position vector. Furthermore, depending upon the particular structure, it follows that such position vector p' is only dependent on a) the first three joint variables (q_1, q_2, q_3); b) the first four joint variables (q_1, q_2, q_3, θ_4); and c) the first five joint variables ($q_1, q_2, q_3, \theta_4, \theta_5$), respectively, in the three cases of Fig. 2. The first case easily allows the proper definition of a "wrist" whose position depends on the first three DOF, and a "hand" whose orientation depends on all DOF's. In the second case, which is of interest in this paper, four DOF's are disposable to position the vector p' , since also θ_4 concurs to its determination. It is then possible to position p' by ∞^1 values of q_1, q_2, q_3, θ_4 , unless one constraint is added in order to obtain a unique solution. In the last case there are 5 DOF's at disposal which involve ∞^2 values of $q_1, q_2, q_3, \theta_4, \theta_5$ to position p' , except when two constraints are introduced.

SOLUTION ALGORITHM

The inverse kinematic problem is reconceived as a dynamical one in order to achieve a solution algorithm which only involves the computation of direct kinematics (1), [12], [14]. With reference to the kinematic notations previously introduced, a typical robot task in the Cartesian space can be assigned through the vectors $(\hat{p}, \hat{s}, \hat{a})$. Let $\hat{q}$ denote a solution of (1) relative to these vectors, and q the algorithm current state variables. Accounting for the kinematic structures of Fig. 2 leads to decompose the coordinate transformation problem into two stages; in particular, by working back from point P through the structure, it is convenient to partition the problem at that

point P' , dependent on a reduced number of joint variables, which can be still expressed in terms of the Cartesian coordinates of the assigned task. Correspondingly the vector q can be partitioned as

$$q^T = (q_p^T | q_h^T) \quad (3)$$

where $q_p \in \mathcal{R}^3$, $q_h \in \mathcal{R}^3$ in case of concurrent axes, $q_p \in \mathcal{R}^4$, $q_h \in \mathcal{R}^2$ if the axes intersect two-by-two, and $q_p \in \mathcal{R}^5$, $q_h \in \mathcal{R}$, if the axes do not converge at all. This approach better copes with the actual kinematic structure and proves very useful as regards robot control. The case of concurrent axes has been widely treated in [5]. The aim of the work now is to establish a two-stage convergent algorithm to solve (1) in case of two-by-two intersecting axes at the end effector. For such structures direct kinematics (1) becomes

$$p' = f_p(q_p), \quad s = f_s(q), \quad a = f_a(q_p, \theta_5). \quad (4)$$

First Stage

Since the position vector p' is determined through four joint variables, a constraint must be introduced in order to achieve a unique solution for q_p . To this purpose, once the task has been assigned, i.e. $(\hat{p}, \hat{s}, \hat{a})$, the first stage of the algorithm must guarantee not only that p' coincides with $\hat{p}$ but also that the fifth link is oriented in such a way to form an angle with the sixth link which coincides with the twist angle between the two links, α_5 , so as to be sure that the actual configuration is a feasible one as regards its constant geometric parameters. Such a constraint can be kinematically expressed by

$$\hat{a}^T z_4 = \cos \alpha_5 \quad (5)$$

where $\hat{a} \equiv \hat{z}_5$ is given in the task space, and z_4 must be determined through q_1, q_2, q_3, θ_4 in order to satisfy the above constraint (5).

Setting out a dynamic algorithm allows the definition of the following errors:

$$e_p = \hat{p}' - f_p(q_p) \\ e_{z_4} = \cos \alpha_5 - \hat{a}^T f_{z_4}(q_p), \quad (6)$$

where $f_{z_4}(q_p)$ is the direct kinematic function which relates q_p to z_4 , and $q_p^T = (q_1 \ q_2 \ q_3 \ \theta_4)$.

In order to assure the convergence of the state variables q_p to the desired ones $\hat{q}_p$, error dynamics is involved, i.e. in compact form

$$\begin{bmatrix} \dot{e}_p \\ \dot{e}_{z_4} \end{bmatrix} = \begin{bmatrix} \dot{\hat{p}}' \\ -\hat{a}^T \dot{f}_{z_4} \end{bmatrix} - \begin{bmatrix} J_p \\ \hat{a}^T J_{z_4} \end{bmatrix} \dot{q}_p \quad (7)$$

where J_p is the (3×4) Jacobian matrix $\partial f_p / \partial q_p$ and similarly J_{z_4} is the (3×4) Jacobian matrix $\partial f_{z_4} / \partial q_p$. The point then is to relate $\dot{q}_p$ to e_p, e_{z_4} so as to guarantee that such errors go asymptotically to zero, and consequently $q_p \rightarrow \hat{q}_p$. Let

$$V_p = \frac{1}{2} (e_p^T e_p + e_{z_4}^2) \quad (8)$$

be a positive Lyapunov function associated with the above

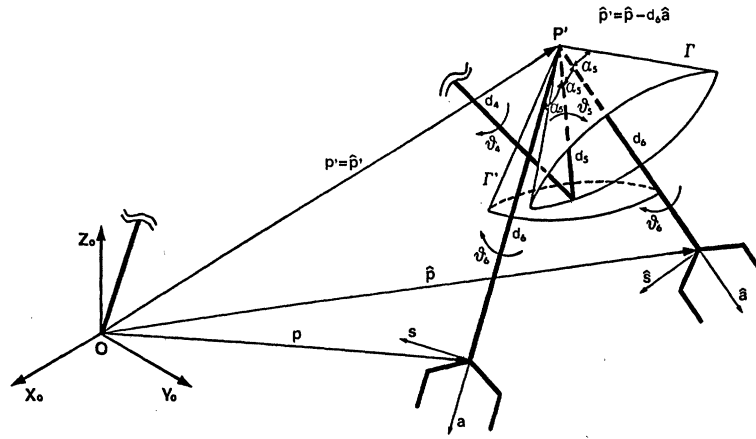


Fig. 4. Geometric task requirements for the first stage of the algorithm.

errors. Differentiating with respect to time and accounting for (7) yield

$$\dot{V}_p = (e_p^T e_{z_4}) \begin{bmatrix} \dot{p} \\ -\dot{a}^T f_{z_4} \end{bmatrix} - (e_p^T e_{z_4}) \begin{bmatrix} J_p \\ \hat{a}^T J_{z_4} \end{bmatrix} \dot{q}_p. \quad (9)$$

It is to underline that the matrix premultiplied to $\dot{q}_p$ in (9) has rank four almost everywhere, since four DOF's are needed to position P' and lay the fifth link on a surface. To be more specific (Fig. 4), the first stage of the algorithm must assure not only that $p' = \hat{p}'$ but also that the fifth link, individuated by d_5 , must lie on the cone Γ of axis $\hat{a}$, angle α_5 and vertex P' ; in this way, however, a and $\hat{a}$ do not have necessarily the same direction, since a is only guaranteed to lie on the cone Γ' of axis z_4 , same angle α_5 , and vertex P' so as evidenced in Fig. 4. Nevertheless, in the second stage of the algorithm θ_5 will provide to align a with $\hat{a}$, and θ_6 , which does not modify a , will take s over $\hat{s}$.

So far a first suitable choice for $\dot{q}_p$ is

$$\dot{q}_p = \gamma_p \operatorname{sgn} (J_p^T e_p + J_{z_4}^T \hat{a} e_{z_4}), \quad \gamma_p > (\|\dot{p}\|_{\max} + \|\dot{\hat{a}}\|_{\max}) \left(\lambda \begin{bmatrix} J_p \\ \hat{a}^T J_{z_4} \end{bmatrix} \right)^{-1} \quad (10)$$

which assures that $\dot{V}_p$ be negative definite; $\operatorname{sgn} w \triangleq (\operatorname{sgn} w_1 \cdots \operatorname{sgn} w_r)$, $w \in \mathcal{R}^r$, $\lambda(A)$ denotes the minimum eigenvalue of matrix A .

Provided that $e_p(t=0) = 0$, $e_{z_4}(t=0) = 0$, (10) always guarantees null tracking position errors but naturally introduces, in the neighborhood of $e_p = 0$, $e_{z_4} = 0$, an equivalent gain which tends to ∞ . This issue leads to the generation of a $\dot{q}_p$ rich in harmonics whose effect on q_p , however, is cut off by the filtering nature of the integrators which generate q_p (see also Fig. 5).

On the other hand, if the kinematical structure of the manipulator is not so complex as to be able to evaluate on-line the inverse of the matrix premultiplied to $\dot{q}_p$ in a closed form, without excessive time expenditure, a rather different choice is

$$\dot{q}_p = \begin{bmatrix} J_p \\ \hat{a}^T J_{z_4} \end{bmatrix}^{-1} \begin{bmatrix} \dot{p} + M_p e_p \\ -\dot{\hat{a}}^T f_{z_4} + m_{z_4} e_{z_4} \end{bmatrix} \quad (11)$$

which reduces $\dot{V}_p$ to a negative quadratic form; the positive definite matrix M_p and the positive scalar m_{z_4} affect the convergence rate and are at designer's disposal. The choice (11), though it involves more computations, presents the inherent advantage of providing continuous joint velocities, as compared with the choice (10).

Second Stage

In order to align a with $\hat{a}$ and s with $\hat{s}$, the second stage of the algorithm must be able to determine θ_5 and θ_6 . Since both s and a are involved, it is natural to define the following errors:

$$\begin{aligned} e_s &= \hat{s} - f_s(q) \\ e_a &= \hat{a} - f_a(q_p, \theta_5) \end{aligned} \quad (12)$$

where $\hat{s}$, $\hat{a}$ are assigned in the task space and s , a are those in (4). Progressing as for the first stage gives error dynamics

$$\begin{bmatrix} \dot{e}_s \\ \dot{e}_a \end{bmatrix} = \begin{bmatrix} \dot{\hat{s}} \\ \dot{\hat{a}} \end{bmatrix} - \begin{bmatrix} J_{sp} \\ J_{ap} \end{bmatrix} \dot{q}_p - \begin{bmatrix} J_s \\ J_a \end{bmatrix} \dot{q}_h, \quad q_h^T = (\theta_5 \theta_6) \quad (13)$$

where J_{sp} and J_{ap} are the (3×4) Jacobian matrices $\partial f_s / \partial q_p$ and $\partial f_a / \partial q_p$ respectively, and similarly J_s and J_a are the (3×2) Jacobian matrices $\partial f_s / \partial q_h$ and $\partial f_a / \partial q_h$ respectively. Thus $\dot{q}_h$ must be related to e_s , e_a so as to assure that such errors go asymptotically to zero ($s = \hat{s}$, $a = \hat{a}$, and obviously $n = \hat{n}$) and consequently $q_h \rightarrow \hat{q}_h$, where $\hat{q}_h$ are the desired joint variable solutions, together with $\hat{q}_p$, of (4). To this end let

$$V_h = \frac{1}{2} (e_s^T e_s + e_a^T e_a) \quad (14)$$

be a positive definite Lyapunov function associated with the above errors. Differentiating with respect to time and accounting for (13) give

$$\begin{aligned} \dot{V}_h &= -s^T \dot{\hat{s}} - a^T \dot{\hat{a}} - (\hat{s}^T J_{sp} + \hat{a}^T J_{ap}) \dot{q}_p \\ &\quad - (\hat{s}^T J_s + \hat{a}^T J_a) \dot{q}_h. \end{aligned} \quad (15)$$

At this extent one must recall the following kinematic

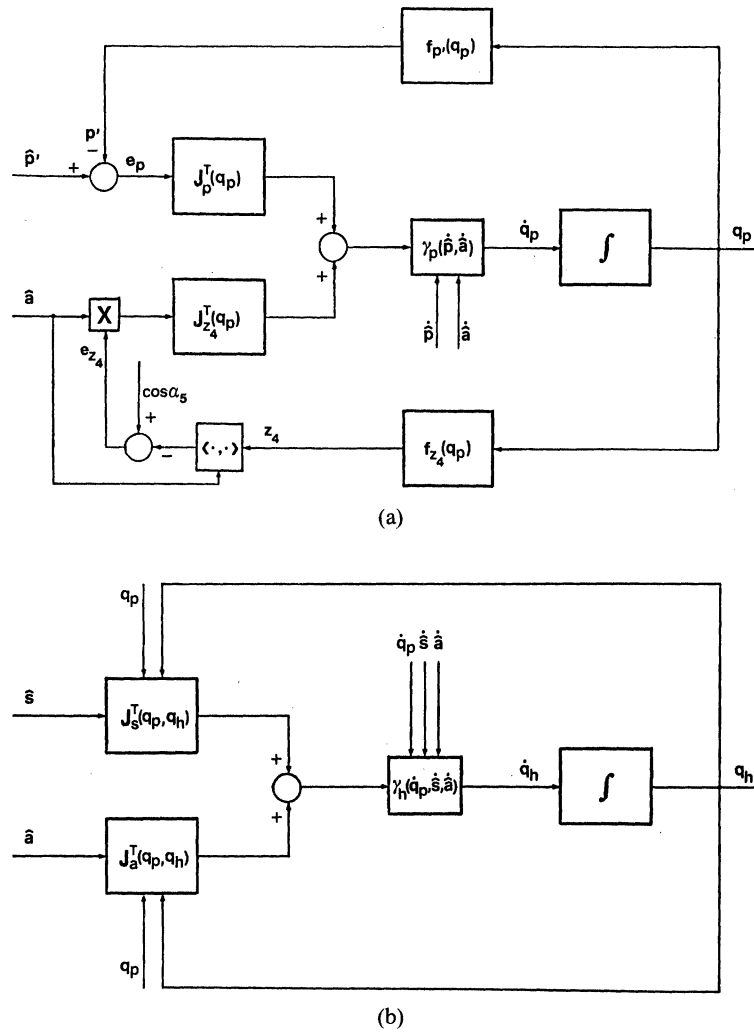


Fig. 5. Coordinate transformation scheme. (a) First stage, (b) Second stage.

properties concerning with the unit vectors s , a [5]:

$$1) \quad \text{rank}(J_s) = \text{rank}(J_a) = 2 \quad \forall q \quad (16)$$

$$2) \quad \mathcal{N}(J_s) = \text{span}(s), \quad \mathcal{N}(J_a) = \text{span}(a) \quad \forall q \quad (17)$$

$$3) \quad \text{Given } x, y \in \mathcal{R}^3, \quad J_s^T x + J_a^T y = 0 \\ \text{if } x \in \text{span}(s), y \in \text{span}(a) \quad (18)$$

where $\mathcal{N}(A)$ denotes the null space of matrix A .

By accounting for such properties and observing that other orientation singularities of (18) are of no interest for a convergent algorithm, in [5] it has been already proved that a suitable choice for $\dot{q}_h$ results

$$\dot{q}_h = \gamma_h \text{sgn}(J_s^T \dot{s} + J_a^T \dot{a}), \\ \gamma_h > (\|\dot{s}\|_{\max} + \|\dot{a}\|_{\max} + \|\dot{q}_p\|_{\max}) \\ \times [\Lambda(J_{sp}^T J_{sp} + J_{ap}^T J_{ap})]^{1/2} [\lambda(J_s^T J_s + J_a^T J_a)]^{-1/2} \quad (19)$$

where $\Lambda(A)$ denotes the maximum eigenvalue of matrix A . It is to underline that (19) suffers from the same problems concerned with sgn type laws as in (10).

Remarks

In sum, realizing the above two stage algorithm always assures the state variables q converge to $\hat{q}$, thus performing the coordinate transformation required. It must be emphasized that the two stage partitioning of the principle solution proposed in [12] and [14], according to the particular kinematic structure, contributes to obtain reduced order Jacobian matrices to compute. Since the number of computations is easily seen to be contained, the application of such an algorithm along a prespecified trajectory in the task space looks attractive; from the implementation standpoint the solution sample rates can be conveniently increased up to the same values of joint servo sample rates. A digital implementation by means of a single dedicated microprocessor system has been realized in the laboratory, although for the case of concurrent axes; it is fully described in [13].

Nevertheless, as the algorithm provides at each step a solution, which is adjacent to the preceding one, uniqueness of the solution is automatically assured. Starting with the same initial conditions $q(0) = \hat{q}(0)$, moreover, avoids any problem of indeterminacy concerned with $\cos \alpha_5 = \cos(-\alpha_5)$ and the orientation singularities of (18), [5].

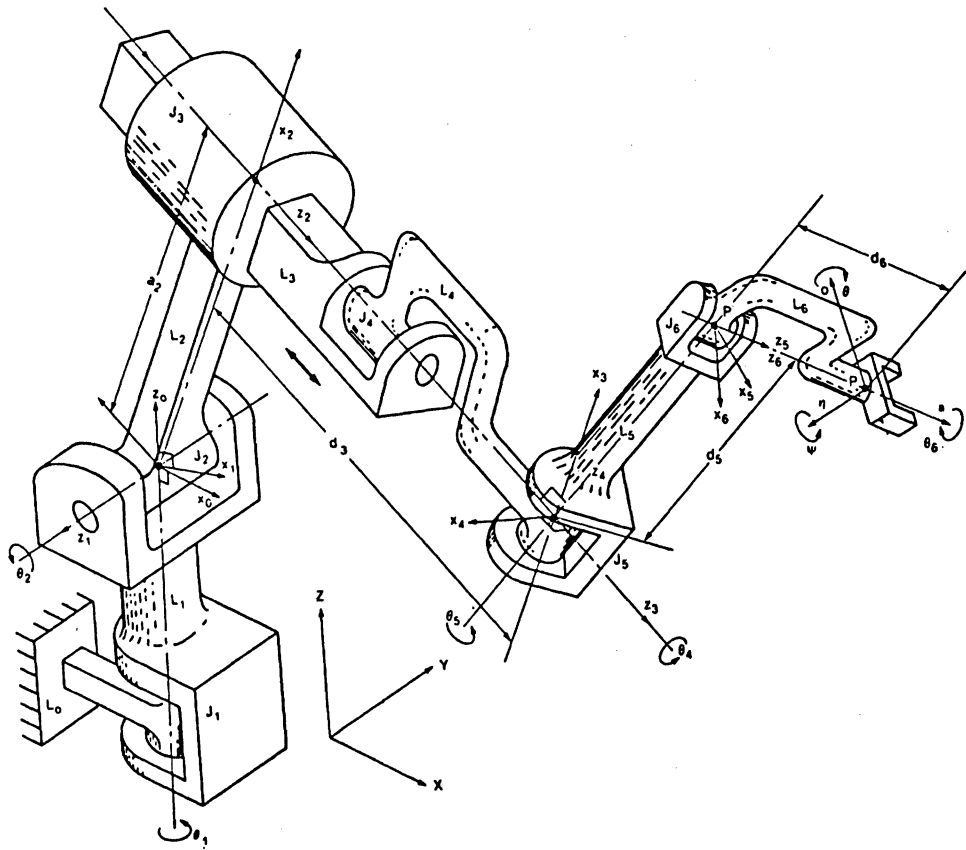


Fig. 6. Prototype arm reported in [4].

It must be emphasized also that the algorithm can directly generate joint velocities $\dot{q}$ corresponding to the assigned task space velocities $\dot{p}, \dot{s}, \dot{a}$, so as required by advanced control techniques, such as [7], [8]. To this goal, in order to avoid joint velocities rich in harmonics, as previously illustrated, one must give up to obtain null tracking errors, but reasonably small ones, and replace the sgn type laws in (10) and (19) respectively by the proportional type laws

$$\dot{q}_p = \gamma_p (J_p^T e_p + J_{z_4}^T \hat{a} e_{z_4}) \quad (20)$$

$$\dot{q}_h = \gamma_h (J_s^T \hat{s} + J_a^T \hat{a}). \quad (21)$$

With such choices, whose block diagram schemes are shown in Figs. 5(a) and (b), respectively, $\dot{V}_p$ and $\dot{V}_h$ are negative only outside a region in the proper error spaces which contain the origin, thus assuring limited tracking errors. The maximum tracking errors will depend directly on task velocities and inversely on feedback gains; it must be underscored that the steady-state ($\dot{p} = 0, \dot{s} = 0, \dot{a} = 0$) errors are identically zero.

It is these laws (20), (21) which will be adopted in the case study.

A CASE STUDY

The robot prototype of Fig. 6, [4], has been selected as a reference to develop a case study for the coordinate transformation algorithm set forth in this paper, as compared to the iterative procedure proposed in [4].

Digital Implementation

Kinematics of the manipulator of Fig. 6, as regards the direct functions which are to be evaluated at each step of the algorithm just presented, is fully reformulated in Appendix.

The iterative procedure proposed in [4] is said to converge, only from the experimental standpoint, within four to five iterations under the assumptions to admit average orientation errors below 0.1° and to be at a sufficient distance from a point of degeneracy for the kinematical structure. Adopting such solution requires eight transcendental function calls, 41 floating-point multiplications, 24 additions, three square roots, along with seven two-argument arctangent function calls, for each iteration. Hence the average number of mathematical computations required makes this technique impractical for performing, at the same servo control rate, the coordinate transformation along a trajectory given in the task space, unless interpolation between a certain number of via points, which is obtained at a lower solution rate, is provided. Nevertheless, since the number of iterations cannot be *a priori* fixed, the worst-case solution period must be chosen as regards control purposes; the farther the via points in the joint space are each other, however, the larger the errors with respect to the exact joint variables are. If joint velocities are needed too, an even more conspicuous number of mathematical computations are likely to be involved.

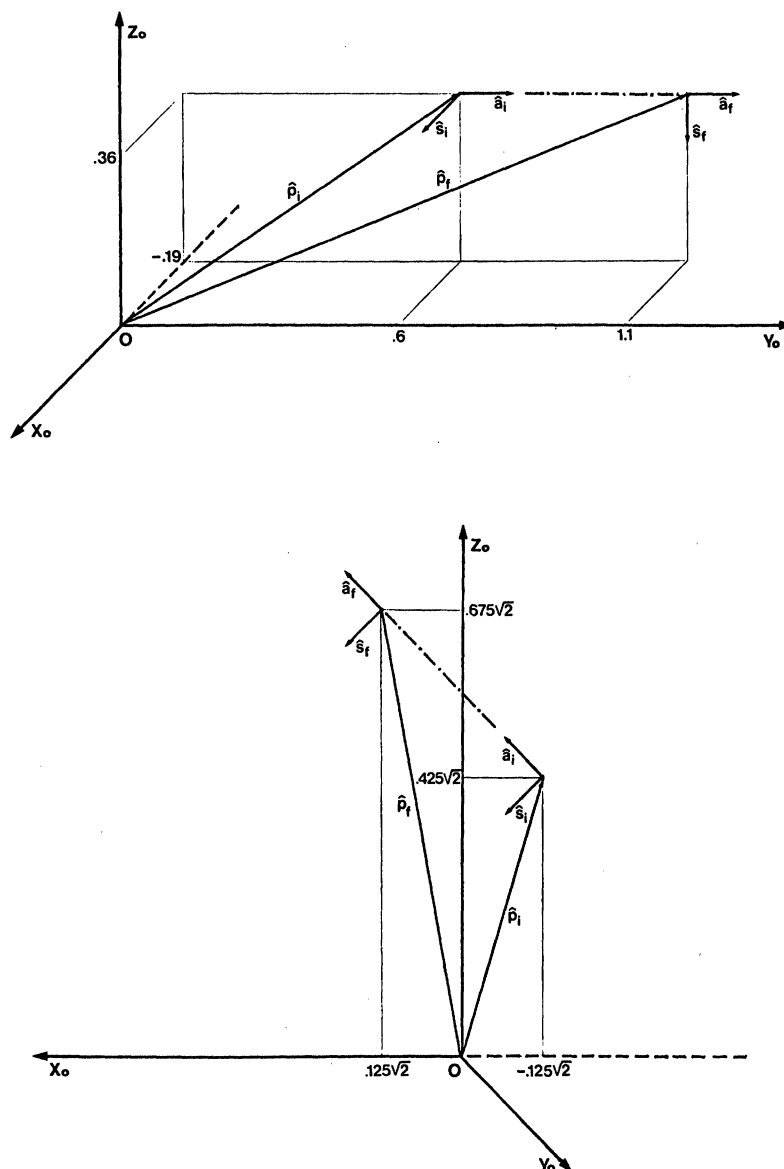


Fig. 7. Task trajectories to be tracked.

On the other hand, the algorithm described in this paper overtakes most of the above drawbacks: a moderate number of computations are required (ten transcendental function calls, 112 floating-point multiplications, and 58 additions); no problem of solution nonuniqueness arises since the algorithm starts with the same initial conditions on the state variables q and progresses with continuity along a trajectory; and tracking errors are very small while steady-state errors are practically zero, as will be shown later.

A further remark is to be made about the task trajectory planning. If the desired trajectory passes in the proximity of a singularity point, the algorithm obviously involves large tracking errors around such point, even if it converges at a steady-state; this is not surprising since the actual robot trajectory involves very high velocities. The attempt of increasing the feedback gains in (20) and (21) so as to counteract the terms related to the above velocities, indeed, can present some difficulties, as far as the digital implementation of the algorithm is concerned, unless the trajectory is conveniently slowed down.

On the other hand, if the planned trajectory crosses a point of singularity, the algorithm of (20) and (21) always works, since it does not require any function inversion (Jacobian, etc.).

It is also remarkable that the tracking results illustrated in the following numerical examples are obtained with one iteration of the algorithm so as to save computation time; in case of trajectories which pass by a singularity point, a sufficient number of iterations would allow lower tracking errors.

Last but not least, besides joint coordinates, even joint velocities are directly generated by the algorithm, without requiring any further computation, that is a good optional for tracking control.

Numerical Examples

In order to show the effectiveness of the proposed algorithm, two numerical examples have been simulated for the robot prototype of Fig. 6. The desired trajectories to track in the Cartesian space are illustrated in Fig. 7.

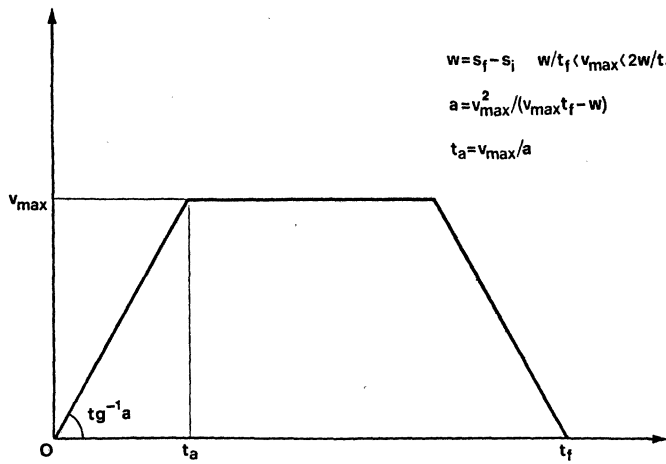
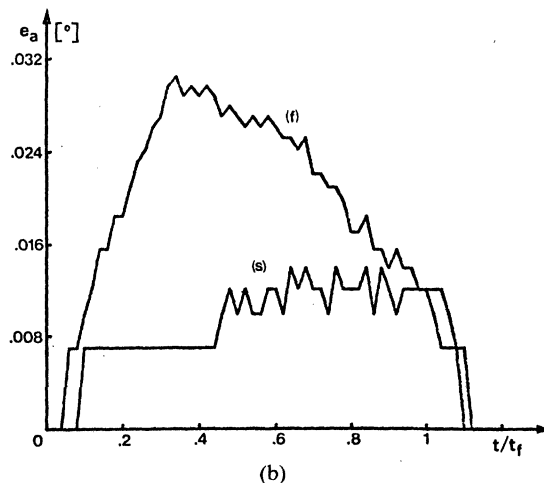
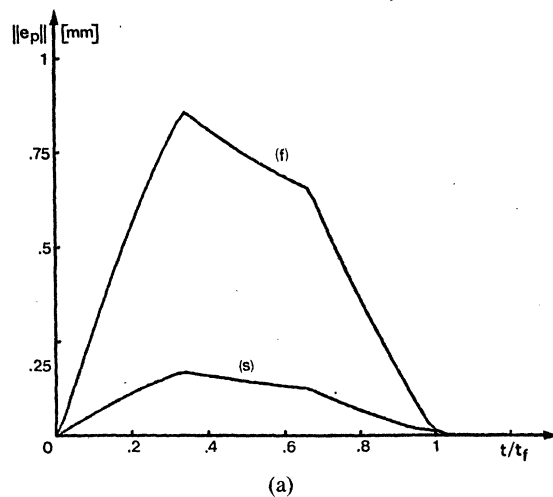
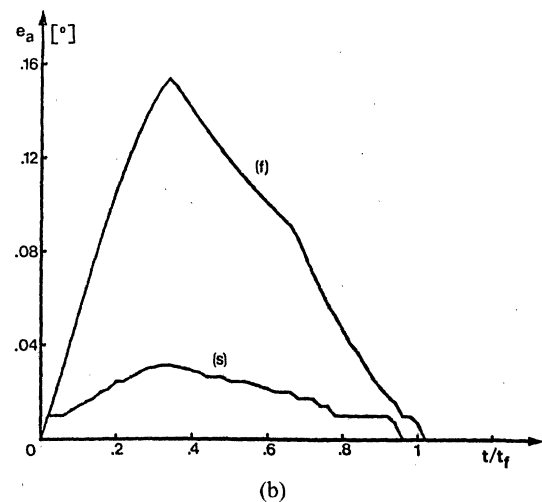
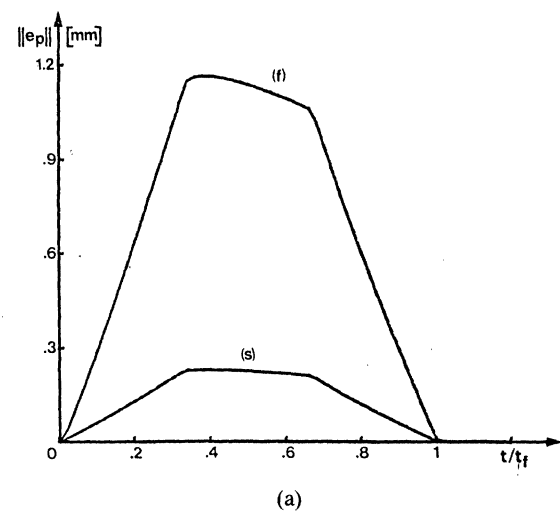


Fig. 8. Trapezoidal velocity profile in the task space.

 TABLE I
 SELECTED PARAMETERS FOR THE VELOCITY PROFILE OF FIG. 8

Trajectory	t_f [s]	$\ \dot{\mathbf{p}}\ _{\max}$ [m/s]	$\dot{\alpha}_{\max}$ [°/s]	$\dot{\beta}_{\max}$ [°/s]	$\dot{\gamma}_{\max}$ [°/s]
1	1	0.75	0	135	0
	5	0.15	0	27	0
2	1	0.75	0	0	0
	5	0.15	0	0	0


 Fig. 9. Tracking errors for trajectory n (f = fast, s = slow). (a) Position. (b) Orientation.

 Fig. 10. Tracking errors for trajectory n (f = fast, s = slow). (a) Position. (b) Orientation.

Trapezoidal velocity profiles (Fig. 8) have been imposed both for the position vector and for the Euler angles of orientation of the end effector. Two values for the maximum velocities $v_{\max}$ and time intervals t_f of Fig. 8 have been chosen for each trajectory, as reported in Table I, in order to investigate the influence of velocity on tracking accuracy; the slower trajectories are typical of those tasks where the robot is required to work, whereas the faster ones refer to material handling tasks. As it can be easily recognized through the kinematics reported in Appendix, the second trajectory crosses a singular point characterized by $(\dot{p}'_x = \dot{p}'_y = 0)$.

The algorithm which is to be implemented in practice leads to a discrete coordinate transformation problem in lieu of the continuous one so far discussed; this obviously involves tracking errors which are larger than those expected. In order to test tracking accuracy in the worst case, the proportional type solutions (20), (21) have been adopted, since finite tracking errors are already involved in the continuous case. The inherent advantage with this choice, however, is that also joint velocities are directly generated. The computational burden required has suggested a solution sample period of 2 ms, on condition that

all computations are performed by a single dedicated microprocessor with a floating-point unit, [13]. The gains in (20), (21) at designer's disposal have been set up at: $\gamma_p = \gamma_h = 500$.

Figs. 9(a) and 10(a) show tracking position errors $\|e_p\| = \|\hat{p} - p\|$, while in Figs. 9(b) and 10(b) reported are the larger tracking orientation errors between $e_s = \cos^{-1}(s^T \hat{s})$ and $e_a = \cos^{-1}(a^T \hat{a})$. Since the same initial conditions occur or $q(0) = \hat{q}(0)$, tracking errors maintain in each case very small, whereas at steady-state they vanish in virtue of the closed-loop structure of the developed coordinate transformation algorithm.

For the second trajectory, moreover, simulation results show that no problem arises at the singularity, so as it was expected, and tracking accuracy is satisfactory along the whole trajectory.

It is apparent, finally, that tracking errors increase as velocities increase; this result is acceptable in practice, since higher tracking performance is expected for slow assembly and processing tasks than in fast-material-handling tasks.

CONCLUSION

This paper has presented a solution algorithm for the inverse kinematic problem for robotic manipulators whose three end effector revolute joint axes intersect two-by-two. By working back from the end effector terminal point through the intermediate links, the algorithm has been partitioned at an opportune point whose position, dependent on a reduced number of joint variables, can be expressed in terms of the Cartesian position and orientation coordinates of the required task. In this way the first stage provides the joint coordinates which determine the position of the above point along with a feasible direction of the fifth link, while the second stage yields the remaining joint coordinates which align the orientation unit vectors. It must be underlined that such technique can also be applied to the case of nonconverging axes, see [6] for further details, still by working back and accounting for adequate mechanical and geometrical constraints.

The occurrence of singular solutions can be prevented by starting with initial conditions on joint variables congruent with the initial Cartesian position and orientation of the assigned task trajectories; as the numerical algorithm implemented works with continuity along the trajectory, adjacent solutions in the joint space are assured. The computational burden is contained, allowing for solution sample rates equal to those of joint servos, thus avoiding interpolation.

In short, the algorithm presented seems to be preferable to iterative-trigonometric techniques in terms of computation time, occurrence of singularities and need for joint velocities.

Future developments are devoted to extend this dynamical approach for the coordinate transformation to kinematically redundant manipulators which seem to show potential advantages over current robot designs.

TABLE II
LINK PARAMETERS FOR THE ARM OF FIG. 6

Link	Joint Variable	Twist Angle α [°]	Length a [m]	Distance Between Normals d [m]
1	θ_1	90	0	0
2	θ_2	90	.36	0
3	d_3	0	0	d_3
4	θ_4	90	0	0
5	θ_5	-90	0	0.19
6	θ_6	0	0	0.1

APPENDIX

Fig. 6 shows the six degree of freedom robotic manipulator prototype reported in [4]; its link parameters are evidenced in Table II. By using the short notations $\sin \theta_i = S_i$ and $\cos \theta_i = C_i$, (4) becomes

$$\begin{aligned}
 p' &= \begin{bmatrix} p'_x \\ p'_y \\ p'_z \end{bmatrix} = \begin{bmatrix} C_1(d_5 C_2 S_4 + a_2 C_2 + S_2 d_3) - S_1 d_5 C_4 \\ S_1(d_5 C_2 S_4 + a_2 C_2 + S_2 d_3) + C_1 d_5 C_4 \\ d_5 S_2 S_4 + a_2 S_2 - C_2 d_3 \end{bmatrix} \\
 s &= \begin{bmatrix} s_x \\ s_y \\ s_z \end{bmatrix} = \begin{bmatrix} C_1[-C_2(C_4 C_5 S_6 + S_4 C_6) - S_2 S_5 S_6] \\ -S_1(S_4 C_5 S_6 - C_4 C_6) \\ S_1[-C_2(C_4 C_5 S_6 + S_4 C_6) - S_2 S_5 S_6] \\ + C_1(S_4 C_5 S_6 - C_4 C_6); \\ -S_2(C_4 C_5 S_6 + S_4 C_6) \\ -C_2 S_5 S_6; \end{bmatrix} \\
 a &= \begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix} = \begin{bmatrix} C_1(-C_2 C_4 S_5 + S_2 C_5) - S_1 S_4 S_5 \\ S_1(-C_2 C_4 S_5 + S_2 C_5) + C_1 S_4 S_5 \\ -S_2 C_4 S_5 - C_2 C_5 \end{bmatrix} \quad (A1)
 \end{aligned}$$

along with $p = p' + d_6 a$. The unit vectors z_4 in (5) and y_4 , which is needed for J_p and J_{z_4} in (7), are

$$\begin{aligned}
 y_4 &= \begin{bmatrix} C_1 S_2 \\ S_1 S_2 \\ -C_2 \end{bmatrix} \\
 z_4 &= \begin{bmatrix} C_1 C_2 S_4 - S_1 C_4 \\ S_1 C_2 S_4 + C_1 C_4 \\ S_2 S_4 \end{bmatrix}. \quad (A2)
 \end{aligned}$$

Differentiating p' , z_4 with respect to q_p , and s , a with respect to q_p and q_h yields the Jacobian matrices, see (7) and (12) respectively, which can be written in compact form as

$$\begin{aligned}
 J_p &= [z_0 \times p' \mid t \times p' \mid y_4 \mid d_5 x_4] \\
 J_{z_4} &= [z_0 \times z_4 \mid -S_4 y_4 \mid 0 \mid x_4] \quad (A3)
 \end{aligned}$$

where $O^T = (0 \ 0 \ 0)$, $z_0^T = (0 \ 0 \ 1)$, $t^T = (S_1 \ -C_1 \ 0)$ and $x_4 = y_4 \times z_4$, and

$$\begin{aligned}
 J_{sp} &= [z_0 \times s \mid t \times s \mid O \mid y_4 \times s] \\
 J_s &= [-S_6 a \mid -n] \\
 J_{ap} &= [z_0 \times a \mid t \times a \mid O \mid y_4 \times a] \\
 J_a &= [-C_6 n + S_6 s \mid O] \quad (A4)
 \end{aligned}$$

where $n = s \times a$.

It must be underlined that some handy simplifications have been carried out in deriving the Jacobians in order to utilize the vectors in (A1) and (A2) which are to be evaluated at each step, as regards the digital implementation of the algorithm.

REFERENCES

- [1] J. Denavit and R. S. Hartenberg, "A kinematic notation for lower-pair mechanisms based on matrices," *ASME J. Appl. Mech.*, vol. 22, pp. 215-221, June 1955.
- [2] R. P. Paul, B. Shimano, and G. E. Mayer, "Kinematic control equations for simple manipulators," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-11, no. 6, pp. 449-455, June 1981.
- [3] D. L. Pieper, "The kinematics of manipulators under computer control," Stanford Artificial Intelligence Project, Stanford CA, memo. AIM-72, Oct. 1968.
- [4] V. I. Lumelsky "Iterative coordinate transformation procedure for one class of robots," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-14, no. 3, pp. 500-505, May/June 1984.
- [5] A. Balestrino, G. De Maria, L. Sciavicco and B. Siciliano, "A novel approach to coordinate transformation for robotic manipulators," *IEEE Trans. Syst., Man, Cybern.*, to be published.
- [6] G. De Maria, L. Sciavicco, and B. Siciliano, "A general solution algorithm to coordinate transformation for robotic manipulators," in *Proc. Int. Conf. Advanced Robotics*, Sept. 1985.
- [7] A. Balestrino, G. De Maria, and L. Sciavicco, "An adaptive model following control for robotic manipulators," *ASME J. Dyn. Syst., Meas. Contr.*, vol. 105, no. 3, pp. 143-151, Sept. 1983.
- [8] J. Y. S. Luh, M. W. Walker, and R. P. C. Paul, "Resolved acceleration control of mechanical manipulators," *IEEE Trans. Automat. Contr.*, vol. AC-25, no. 3, pp. 468-474, June 1980.
- [9] C. S. G. Lee, "Robot arm kinematics, dynamics, and control," *IEEE Computer*, vol. 15, no. 12, pp. 62-80, Oct. 1982.
- [10] V. D. Scheinman, "Design of a computer manipulator," Stanford Artificial Intelligence Laboratory, Stanford University, Stanford CA, memo. AIM-92, 1969.
- [11] J. Y. S. Luh, "An anatomy of industrial robots and their controls," *IEEE Trans. Automat. Contr.*, vol. AC-28, no. 2, pp. 133-153, Feb. 1983.
- [12] A. Balestrino, G. De Maria and L. Sciavicco, "Robust control of robotic manipulators," in *Proc. 9th IFAC World Congress*, July 1984.
- [13] G. De Maria and R. Marino, "A discrete algorithm for solving the inverse kinematic problem of robotic manipulators," in *Proc. Int. Conf. Advanced Robotics*, Sept. 1985.
- [14] W. A. Wolovich and H. Elliott, "A computational technique for inverse kinematics," in *Proc. 23rd IEEE Conf. Dec. Contr.*, Dec. 1984.