Unifying Robot Arm Control

TREVOR N. MUDGE, SENIOR MEMBER, IEEE, AND JERRY L. TURNEY

Abstract—**A unified approach to three stages of robot arm control is presented based on the Newton-Euler equations of motion. The stages unified are resolved motion, gross motion, and fine motion. Apart from the conceptual advantage, this unification can require less computation than if the three stages are computed separately. In particular, fewer computations are required for arms with no more than about a dozen joints (a number unlikely to be exceeded for most arms, at least in the near future). Computation times are estimated assuming the computing elements are fabricated from current (very large-scale integrated circuit) (VLSIC) technology. It is also shown how friction can be incorporated into the unified approach. The concept of "pseudo-force" is introduced to relate fine motion to the Newton-Euler equations.**

INTRODUCTION

IN WHAT FOLLOWS, the term "robot arm" or "arm" will refer to a chain of links open on one end and animated by "actuators" (a term which refers to the hydraulic valves or electric motors responsible for driving the arm) located at each joint between links. A robot arm is often referred to as a manipulator, but the term arm is used here for brevity. The last link in the open chain will be referred to as the "hand" here, although "end effector" is also common terminology. The first link of the chain which is generally immobile and fixed in space will be referred to as the "base" of the arm. Joints are assumed to be rotational in order to avoid unduly complicating the discussion. Prismatic joints can be dealt with without any conceptual changes. Fig. 1 illustrates a typical robot arm (Unimate's PUMA 600).

Controlling a robot arm is a complex task that can be conveniently divided into the following four stages.

1) Trajectory Planning: In this stage the arm's path through space is determined. Depending on the application, this may involve obstacle avoidance strategies.

2) Resolved Motion: In this stage the arm's trajectory is resolved into the component joint motions.

3) Gross Motion: In this stage torques and forces are derived that are required by the joint actuators to generate the joint motions computed in stage 2. Typically, these torques and forces form the basis of a control law that incorporates some type of negative feedback.

4) Fine Motion: In this stage torques and forces are derived that are required to generate the incremental joint motions necessary once the arm is close to its goal.

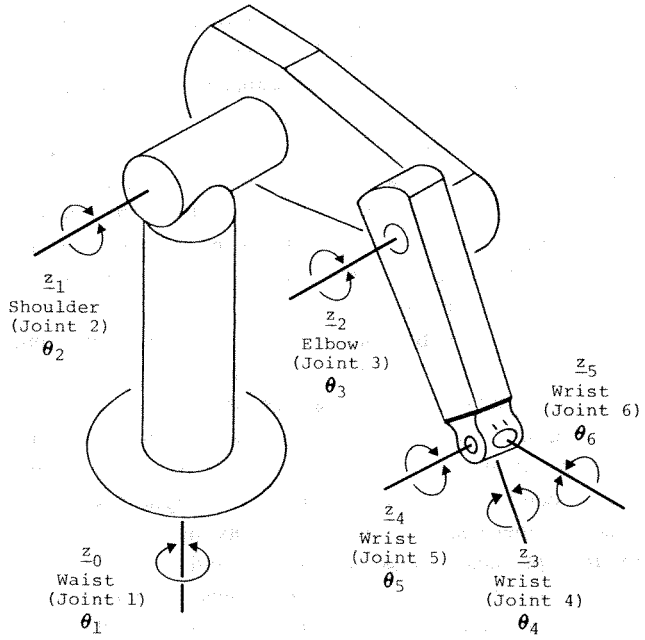


Fig. 1. PUMA 600 showing joint angles.

In this paper a unified solution to stages 2, 3, and 4 will be presented. Several algorithms, notably the resolved motion method of Whitney [1], the Newton-Euler (N-E) equations of motion first developed by Walker and Luh [2]-[6], and the static forces of Paul [7] are all merged into one unified algorithm. In addition, it is also shown how friction can be incorporated into this unified approach. The trajectory planning stage will not be covered in this paper. Representative schemes for trajectory planning are those of Paul [8] or the collision avoidance method of Perez and Wesley [9].

The purpose of control stages 2, 3, and 4 is to maintain the motion of the arm along the trajectory derived in stage 1 by applying corrective compensation through the actuators to adjust for any deviations of the arm from the desired arm trajectory. If a "perfect physical" model for the arm could be defined and if the model could be "solved" rapidly enough to output control signals at a rate compatible with the desired arm motion, there would be no need for feedback in the control strategy. A perfect physical model would, however, have to be one which accounted for, among other things, gravitational and inertial loading, friction, link flex, not to mention all possible external perturbances. Clearly, such a model requires an impossible amount of computation. Indeed in view of the need to include external perturbances, just defining the model is impossible.

Paper IUSD 82-41, approved by the Industrial Control Committee of the IEEE Industry Applications Society for presentation at the 1982 Industry Application Society Annual Meeting, San Francisco, CA, October 4-8, 1982. Manuscript released for publication January 27, 1984. This work was supported in part by the National Science Foundation under Grant ECS-8106954 and in part by the Robot Systems Division of the Center for Robotics and Integrated Manufacturing, University of Michigan, Ann Arbor.

The authors are with the Robot Systems Division, Center for Robotics and Integrated Engineering, College of Engineering, University of Michigan, 2514 East Engineering Building, Ann Arbor, MI 48109.

0093-9994/84/1100-1554$1.00

Since a complete model is impossible, the question is then, how sophisticated should the model be that is incorporated into the control strategy, and to what extent should the feedback compensate for errors in modeling. The appeal of most presently used control methods is that they ignore almost all the physical details of the arm[1] and still obtain reasonable control. However, rigid links are required, and lighter and more flexible links cannot be used because of the impossibility of determining how they would behave when most of the arm's physical properties are ignored. Another penalty one pays for such ignorance of detail is the need for large actuators to "force" the arm when it is least willing to move along the desired path, especially when large accelerations are required. A disadvantage of using large actuators to force the arm in its motion is made worse by the fact that employment of large actuators in the links near the hand result in an exponential increase in actuator size as one moves toward the base: the lower actuators must be able to move those in the upper links. This problem can be circumvented to some extent by placing the actuators in the base; however, the complexity of the drives required by this solution are a limitation. A further penalty for ignorance of detail is lack of speed. In order to maintain control of an arm by methods that ignore significant physical properties, it is necessary to operate at much slower rates than physically possible. Consequently, present control methods have resulted in the development of relatively slow robots that are unable to handle payloads of more than a few percent of their mass. For example, the PUMA 600 is limited to a payload of 5 lb and a tip speed of 1 m/s.

Limitations are also found in present solutions of the fine motion stage of arm control. For example, inserting a peg into a hole is a characteristic task required of the arm.Chamfering the peg to aid insertion is the most primitive approach to solving this problem. The remote compliance control technique [11], [12] is more sophisticated; it, in essence, places flex in the hand. It has had considerable success. However, to solve the general problem of fine motion during delicate operations of the hand, an active scheme is needed that explicitly models the arm in this stage of control [13].

In our opinion, one improvement to the present shortcomings of arm control is to include more of the physics of the arm in the model than is presently used. The physics of the arm is well understood [2]–[6], [14]. A more accurate model in the control loop offers greater versatility than the presently used control techniques as we shall show in the next section. One can determine the arm's gravitional loading, inertial loading, and friction, and can also model arm flex, allowing the use of much lighter and more efficient arm designs. However, the technique goes further. Inputs from hand sensors (devices which measure external forces and moments exerted on the hand by the external world) are easily incorporated into the model. This incorporation allows the model to adjust for the gravitational and inertial loading of the payload and to react to external forces and moments. As an offshoot of the preceding, during the fine motion stage of control, false or "pseudo-forces" and "pseudo-moments" can be artificially generated and summed into the sensor inputs from the hand. The arm will exert forces or moments to compensate for these pseudo-force and -moment inputs. Thus the arm can be made to exert any desired force and moment vectors by incorporating oppositely directed pseudo-force and -moment vectors into the hand inputs [6].

If a more accurate model of the arm is used, the amount of computation associated with controlling the arm can be a serious obstacle to meeting real-time constraints. However, we have shown that current VLSIC technology will allow the fabrication of cost-effective special-purpose processors that can overcome this difficulty. In particular, we have shown that if the control strategy is based on a model employing the N–E equations of motion, real-time constraints can easily be met assuming control stages 1 and 2 are computed off-line [10], [15]. The assumptions about stages 1 and 2 can be restrictive in those applications where the arm does not make repetitive motions because each new motion will, in general, require a new trajectory. Thus the stages of control will have to be completely recomputed for each motion. Unless stages 1 and 2 can be computed "on the fly," a set of precomputed solutions will have to be stored. If the number of possible trajectories is large, this approach is impractical. As will be shown, unifying stages 2–4 results in a need for fewer computations for arms with no more than about a dozen joints (a number unlikely to be exceeded for most arms, at least in the near future). This allows stage 2 to be calculated on the fly, thus reducing the restrictions on applications that do not make repetitive motions.

The next section introduces some necessary notation before explaining how stages 2–4 of robot arm control can be unified using the N–E equations. The final section adds some conclusions and comments about future work.

UNIFYING ROBOT ARM CONTROL

Notation

The paths along which the n joints of the arm of n links move during the arm's motion can be collected into a set of paths known as the arm trajectories. For arms with only rotary joints, such as the PUMA 600, the trajectories can be specified with respect the relative angles between the joints. These trajectories will be referred to as the relative joint angle (RJA) trajectories, and will be represented by a time-dependent vector of angles between joints $\boldsymbol{\vartheta}(t)$. For an n-jointed arm this vector will have n components. See Fig. 1 for an illustration of these angles for the PUMA arm. Discrete time points along the trajectories into which the trajectories may be divided for control purposes are called "set points," e.g., at time $t = T$ the set point is the vector of n values given by $\boldsymbol{\vartheta}(T)$. The first and second time derivatives of the RJA trajectories are the RJA velocities, $\dot{\boldsymbol{\vartheta}}(t)$ and RJA accelerations $\ddot{\boldsymbol{\vartheta}}(t)$, respectively.

Each link i of the arm has its own coordinate frame fixed in the link and referred to as the ith frame. The coordinate system of each link is located at the end farthest from the base (see

[1] For example, the PUMA 600 arm control is split into six independent subsystems (one per joint). No attempt is made to incorporate explicitly the effects of dynamic coupling between the joints. This and other effects must be compensated for by using negative feedback in the form of positional plus derivative control in each subsystem [10].

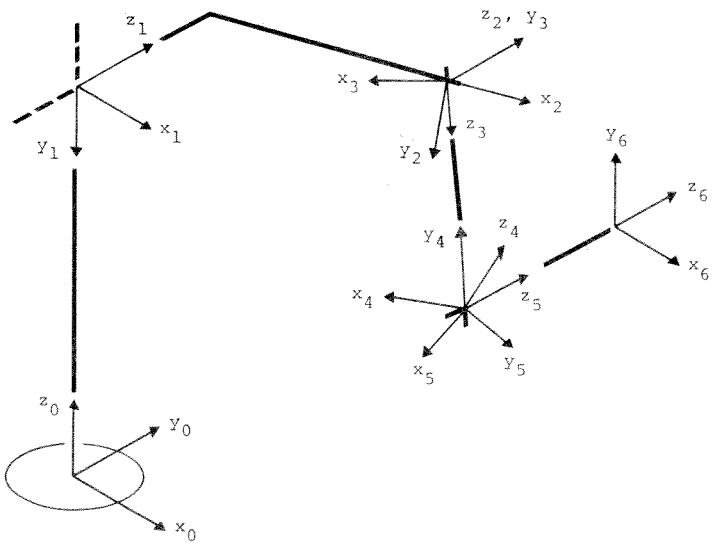


Fig. 2. Coordinated frames.

Fig. 2). A unit vector along the z axis of the ith frame and represented in the ith frame is denoted by z_i, similarly for unit vectors along the x and y axes. An alternative representation will sometimes be used in which the unit vector along the z axis of the ith frame is denoted by $(u_i)_z$, similarly for the x and y axes. By convention, joint i is at the $i-1$th origin and ϑ_i is taken about z_{i-1}.

Matrices and tensors are represented in upper case type, while vectors are in boldface type. Unless otherwise stated vectors will be treated as column vectors. Greek indices are used to denote components of a vector or matrix, and the "summation convention" is employed, i.e. repeated indices are assumed summed over all three coordinates. For example, the inner product between vectors $\boldsymbol{a}$ and $\boldsymbol{b}$ can be written in two ways: in matrix notation as $\boldsymbol{a}^t\boldsymbol{b}$ and in terms of the vector components using the summation convention $a_\gamma b_\gamma (= a_x b_x + a_y b_y + a_z b_z)$.

R_j^i represents a three by three rotation matrix which maps a vector from its representation in the ith link coordinate frame to its equivalent in the jth coordinate frame. Some well-known properties of rotation matrices represented in this notation are

$$(R_j^i)^t = (R_j^i)^{-1} = R_i^j. \tag{1}$$

A superscript t denotes a transpose of the matrix.

A rotation between coordinate frames i and j can be written as a chain product of rotations between successive frames:

$$R_j^i = R_j^{j+1} R_{j+1}^{j+2} \cdots R_{i+1}^i. \tag{2}$$

In general, with the inverse defined by (1) and R_i^i defined as the identity, one obtains the relation $R_j^k R_k^i = R_j^i$ for all integer values of k.

Vectors rotated into the base ($i = 0$) frame will be starred in order to shorten and clarify notation. For example, z_i^* is equivalent to $R_0^i z_i$. Matrices are represented in the base frame will be similarly starred.

Rotations operate on a vector product in the following fashion:

$$R_j^i(\boldsymbol{b}_i \times \boldsymbol{c}_i) = R_j^i \boldsymbol{b}_i \times R_j^i \boldsymbol{c}_i \tag{3}$$

where $\boldsymbol{b}_i$ and $\boldsymbol{c}_i$ are any vectors—a vector product transforms like a vector under rotation.

Finally, we define the Q^β matrices. The Q^β matrices have the property that $Q^x \boldsymbol{c}_i = \boldsymbol{x}_i \times \boldsymbol{c}_i$, $Q^y \boldsymbol{c}_i = \boldsymbol{y}_i \times \boldsymbol{c}_i$, and $Q^z \boldsymbol{c}_i = \boldsymbol{z}_i \times \boldsymbol{c}_i$. The action of a vector cross product is contained in these matrices. This notation is borrowed from quantum mechanics. The Q^β matrices are

$$Q^x = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} \quad Q^y = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ -1 & 0 & 0 \end{bmatrix}$$

$$Q^z = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{4}$$

The Newton–Euler Equations

The N–E equations represent a fairly detailed model of the dynamics of a robot arm. However, other formulations provide similar detail. A comparison was made between the N–E set of equations and several other arm formulations in [5], [6]. The N–E set was found to be equivalent but computationally much more efficient than any of the other formulations [7], [16], [17]. The equivalence of the N–E formulation to the other formulation is to be expected since all physical assumptions are the same. The N–E equations are listed in the form presented in [6]. This form is slightly more computationally efficient than the original form of Walker and Luh [2]:

$$\boldsymbol{\omega}_0 = 0, \qquad \boldsymbol{\alpha}_0 = 0, \qquad \boldsymbol{a}_0 = 9.8 \quad \text{m/s}^2 \tag{5}$$

$$\boldsymbol{\omega}_i = R_i^{i-1}(\boldsymbol{\omega}_{i-1} + \dot{\vartheta}_i \boldsymbol{z}_{i-1}) \tag{6}$$

$$\boldsymbol{\alpha}_i = R_i^{i-1}(\boldsymbol{\alpha}_{i-1} + \boldsymbol{\omega}_{i-1} \times \boldsymbol{z}_{i-1}\dot{\vartheta}_i + \ddot{\vartheta}_i \boldsymbol{z}_{i-1}) \tag{7}$$

$$A_i = (\alpha_i)_\beta Q^\beta + (\omega_i)_\beta Q^\beta (\omega_i)_\chi Q^\chi \tag{8}$$

$$a_i = A_i r_i + R_i^{i-1} a_{i-1} \tag{9}$$

$$\bar{a}_i = A_i(\bar{r}_i + r_i) + R_i^{i-1} a_{i-1} \tag{10}$$

$$f_i = m_i \bar{a}_i + R_i^{i+1} f_{i+1} \tag{11}$$

$$n_i = \mathrm{Tr}\ \{A_i K_i (Q^\gamma)^t\}(u_i)_\gamma + m(\bar{r}_i + r_i) \times \bar{a}_i + r_i \times R_i^{i+1} f_{i+1} + R_i^{i+1} n_{i+1} \tag{12}$$

$$\tau_i = (R_i^{i-1} z_{i-1})^t n_i \tag{13}$$

$$R_{i-1}{}^i = \begin{bmatrix} \cos\ \vartheta_i & -\cos\ \varphi_i \sin\ \vartheta_i & \sin\ \varphi_i \sin\ \vartheta_i \\ \sin\ \vartheta_i & \cos\ \varphi_i \cos\ \vartheta_i & -\sin\ \varphi_i \cos\ \vartheta_i \\ 0 & \sin\ \varphi_i & \cos\ \varphi_i \end{bmatrix} \tag{14}$$

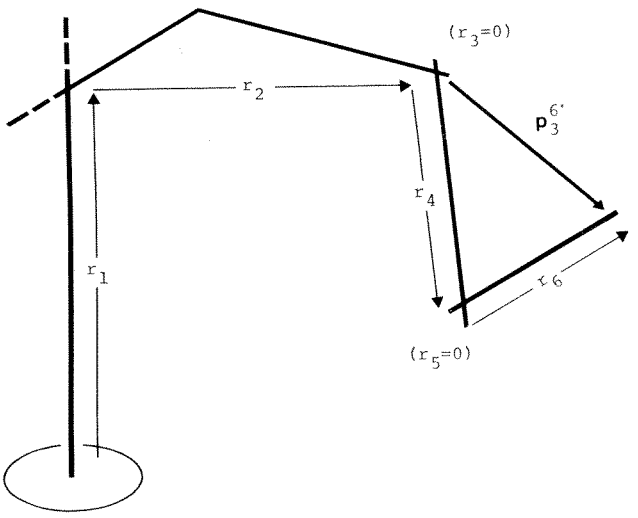


Fig. 3. r_i vectors and $p_3{}^{6*}$ vector.

$$K_q = \begin{bmatrix} \dfrac{-I_{qxx} + I_{qyy} + I_{qzz}}{2} & 0 & 0 \\ 0 & \dfrac{I_{qxx} - I_{qyy} + I_{qzz}}{2} & 0 \\ 0 & 0 & \dfrac{I_{qxx} + I_{qyy} - I_{qzz}}{2} \end{bmatrix} \tag{15}$$

Several ideas can be gleaned from these equations without becoming intimidated by their apparent complexity. We have assumed in presenting the foregoing equations that we are dealing with an arm having only rotary joints such as the PUMA of Fig. 1. The ϑ_i, $\dot{\vartheta}_i$, and $\ddot{\vartheta}_i$ are components of the RJA trajectories, the RJA velocities, and the RJA accelerations, respectively (see earlier definition), i.e., $\vartheta_i(t)$ is the trajectory for a single joint.

Equation (6) develops ω_i, the angular velocity of the ith joint as seen in its own frame. Notice that this equation is recursive in that it contains terms from the next lower joint which are rotated by rotation matrix R_i^{i-1} from the lower $i-1$th joint into the ith joint location. This recursion just reflects the simple fact that the angular velocities add together as one proceeds up the arm from the base to the hand. The same can also be said of the following equations: (7), which defines the angular acceleration α_i of the ith joint; (9), which defines the linear acceleration a_i of the ith joint; and (10), which defines the center of mass acceleration $\bar{a}_i$ of the ith link. They are all recursive and represent the kinematics of the arm; i.e., they define the geometrical motion of the arm rather than the actual forces and moments needed for such motion. The matrix A_i in (8) is an acceleration type matrix and is a combination of ω_i and α_i. It contains Coriolis and centrifugal accelerations. The term r_i is a distance vector from the ith joint to the $i+1$th joint represented in the ith frame (alternatively, from the $i-1$th origin to the ith origin, see Fig. 3).

Once the kinematics of the arm is calculated, the motion of the arm is defined, and the forces f_i and moments n_i needed to cause the required motion can then be determined. This is accomplished in (11) and (12) which represent the dynamics of the arm. These equations are recursive going down the arm. This reflects the fact that forces and moments felt by the $i+1$th link are passed down to the ith link. When $i = n$, indicating the uppermost, or hand, link, then f_n and n_n represent forces and moments acting on the hand. If the hand is equipped with hand, wrist, or joint sensors [18] the forces and moments acting on the hand can be determined and incorporated into the model by setting f_n and n_n to the values determined by the sensors. Notice how naturally forces and moments that the hand "feels" are incorporated into a model based on the N–E equations of motion.

The K_i matrix is an inertial matrix, and m_i is the mass of the ith link. The torque τ_i needed by ith actuator to achieve the desired motion is obtained by projecting the component of the moment n_i onto the ith actuator axis $R_i^{i-1} z_{i-1}$. Recall z_{i-1} is the axis of the ith actuator represented in the $i-1$th frame. The application of R_i^{i-1} rotates it into the ith frame.

In summary, the N–E equations provide a method for calculating the kinematics (angular and linear velocities and accelerations) of the arm using the RJA trajectories, the RJA velocities, and the RJA accelerations. The computation proceeds from the base to the hand. Once the kinematics have been determined, the N–E equations provide a method by which the dynmaics of the arm (forces and moments) can be computed. This part of the computation proceeds from the hand down. The forces and moments felt by the hand may be incorporated through all the joints down to the base. Fig. 4 illustrates how the computations proceed.

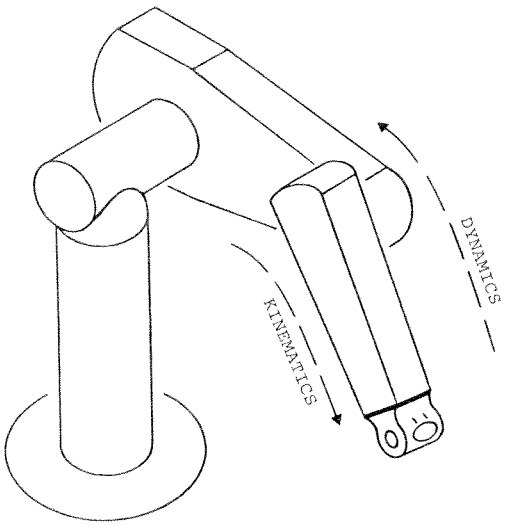


Fig. 4. Arm kinematics and dynamics.

Resolved Motion

Once the trajectory of the hand has been determined to avoid collisions and actuator limitations (trajectory planning stage), it becomes necessary to resolve this hand trajectory into the RJA trajectories in order to use the N–E equations, which are based on the RJA trajectories, to calculate the actuator torques needed to drive the arm. The term "resolved motion" was coined by Whitney when he originally proposed the technique [1]. Our approach determines the RJA accelerations $\ddot{\boldsymbol{\vartheta}}$ given the hand link's acceleration trajectory $\boldsymbol{a}_n$. Whitney originally proposed resolving velocities. Clearly, the two methods are equivalent provided appropriate initial conditions are given; however, resolving velocities does not fit in with the N–E equations as conveniently.

The first step in resolved motion is to determine how each infinitesimal joint motion (accelerations in our case) affects the infinitesimal motion of the hand. A linear mapping exists between the infinitesimal joint motion space and the infinitesimal hand motion space. This mapping is defined by a Jacobian. The Jacobian can be calculated explicitly or, as we shall show, can be "strobed" out of the N–E equations. The basic idea behind strobing results from the following observation: the ratios of infinitesimal hand accelerations to infinitesimal joint accelerations are the elements of the Jacobian matrix. Therefore, if the nonlinear components of the accelerations are deleted from the N–E equations to linearize them, and if one of the RJA accelerations is set to one and the rest are set to zero, the result will be a column of the Jacobian. (Once the equations have been linearized, we are no longer restricted to infinitesimal RJA accelerations and can use unit vectors.) Thus successive columns of the Jacobian can be strobed out of the N–E equations with unit RJA acceleration vectors. The resulting linear equations relating the linearized hand acceleration to the RJA accelerations can be solved to yield the RJA accelerations. Note that a unique solution can only be guaranteed if $n = 6$. Unique solutions for arms with more joints require further constraints.

First we will develop the linear relation between the hand's acceleration and the RJA accelerations using the N–E equations. Equations (6), (7), and (9) are given now in an equivalent but modified form for discussion:

$$\boldsymbol{\omega}_i = R_i^{i-1}(\boldsymbol{\omega}_{i-1} + \dot{\vartheta}_i \boldsymbol{z}_{i-1}) \tag{16}$$

$$\boldsymbol{\alpha}_i = R_i^{i-1}(\boldsymbol{\alpha}_{i-1} + \boldsymbol{\omega}_{i-1} \times \boldsymbol{z}_{i-1}\dot{\vartheta}_i + \ddot{\vartheta}_i \boldsymbol{z}_{i-1}) \tag{17}$$

$$\boldsymbol{a}_i = \boldsymbol{\omega}_i \times (\boldsymbol{\omega}_i \times \boldsymbol{r}_i) + \boldsymbol{\alpha}_i \times \boldsymbol{r}_i + R_i^{i-1}\boldsymbol{a}_{i-1}. \tag{18}$$

The hand linear and angular accelerations are specified in Cartesian coordinates in the base (zero frame) during the trajectory planning stage; however, in order to employ (16)–(18), these must be converted to the hand or nth frame. This conversion takes $18n$ multiplications and $12n$ additions for both the $\boldsymbol{a}_n$ and $\boldsymbol{\alpha}_n$ vector using a rotation on each vector at each link.

For convenience in the discussion that follows, the converted linear and angular hand acceleration vectors, $\boldsymbol{\alpha}_n$ and $\boldsymbol{a}_n$, are concatenated into six vector $\boldsymbol{ac}_n = \begin{bmatrix}\boldsymbol{\alpha}_n \\ \boldsymbol{a}_n\end{bmatrix}$. At any time along the hand's trajectory, one finds that the hand's angular and linear acceleration can be written in terms of linear and nonlinear contributions that are functions of $\dot{\boldsymbol{\vartheta}}$ and $\ddot{\boldsymbol{\vartheta}}$. The terms involving $\boldsymbol{\omega}_i$ in (17) and (18) represent nonlinear Coriolis and centrifugal accelerations. Omitting these terms, we obtain the linear relation

$$\boldsymbol{ac}_n{}^{\text{lin}} = D\ddot{\boldsymbol{\vartheta}} \tag{19}$$

where D is an n by n "distance" matrix. This is the linear relation between the hand's acceleration and the joint angular acceleration promised earlier. Now an input unit vector $\ddot{\boldsymbol{\vartheta}}$, where all components are zero except for one—for example,

$$\ddot{\boldsymbol{\vartheta}} = (\ddot{\vartheta}_1, \ddot{\vartheta}_2, \ddot{\vartheta}_3, \cdots, \ddot{\vartheta}_6)^t = (1, 0, 0, \cdots, 0)^t \tag{20}$$

can be used to strobe out columns of the D matrix from the linearized N–E equations.

We will now show that the D matrix, when represented in the base frame (D^*), is equivalent to Whitney's Jacobian J [1], confirming our assertion that the foregoing procedure is the same as resolved motion.

Theorem: $D^* = J$ where

$$J = \begin{bmatrix} \boldsymbol{z}_0^* & \boldsymbol{z}_1^* & \cdots & \boldsymbol{z}_{n-1}^* \\ \boldsymbol{z}_0^* \times \boldsymbol{p}_0{}^{n*} & \boldsymbol{z}_1^* \times \boldsymbol{p}_1{}^{n*} & \cdots & \boldsymbol{z}_{n-1}^* \times \boldsymbol{p}_{n-1}{}^{n*} \end{bmatrix}, \tag{21}$$

and

$$\boldsymbol{p}_{i-1}{}^{n*} = \boldsymbol{r}_i^* + \boldsymbol{r}_{i+1}^* + \cdots + \boldsymbol{r}_n^* \tag{22}$$

(see Fig. 3 for an illustration of $\boldsymbol{p}_3{}^{6*}$).

Proof: The linear part of $\boldsymbol{\alpha}_n{}^{\text{lin}}$ from (19) is

$$\boldsymbol{\alpha}_n{}^{\text{lin}} = R_n{}^{n-1}(\boldsymbol{z}_{n-1}\ddot{\vartheta}_n + \boldsymbol{\alpha}_{n-1}{}^{\text{lin}}) \tag{23}$$

which, when expanded by recursion, yields

$$\sum_{i=0}^{n} R_n{}^{i-1}\boldsymbol{z}_{i-1}\ddot{\vartheta}_i.$$

Applying a rotation into the base frame, $R_0{}^n$, results in

$$\boldsymbol{\alpha}_n{}^{\text{lin}*} = R_0{}^n\boldsymbol{\alpha}_n{}^{\text{lin}} = \sum_{i=0}^{n} R_0{}^{i-1}\boldsymbol{z}_{i-1}\ddot{\vartheta}_i = \sum_{i=0}^{n} \boldsymbol{z}_{i-1}{}^*\ddot{\vartheta}_i. \tag{24}$$

From (18) the linear part of ${}^{n}\boldsymbol{a}^{\text{lin}}$ using recursive expansion is

$$\begin{aligned}{}^{n}\boldsymbol{a}^{\text{lin}} &= {}^{n}\boldsymbol{\alpha}^{\text{lin}} \times \boldsymbol{r}_n + {}^{n-1}R_n({}^{n-1}\boldsymbol{\alpha}^{\text{lin}} \times \boldsymbol{r}_{n-1}) \\ &\quad + {}^{n-1}R_{n-1}{}^{n-2}R({}^{n-2}\boldsymbol{\alpha}^{\text{lin}} \times \boldsymbol{r}_{n-1}) + \cdots. \end{aligned} \tag{25}$$

Combining rotation matrices with (5) and distributing the rotation across the cross product with (7), one finds

$$\begin{aligned}{}^{n}\boldsymbol{a}^{\text{lin}} &= {}^{n}\boldsymbol{\alpha}^{\text{lin}} \times \boldsymbol{r}_n + {}^{n-1}R_n{}^{n-1}\boldsymbol{\alpha}^{\text{lin}} \times {}^{n-1}R_n{}^{n-1}\boldsymbol{r}_{n-1} \\ &\quad + {}^{n-2}R_n{}^{n-2}\boldsymbol{\alpha}^{\text{lin}} \times {}^{n-2}R_n{}^{n-2}\boldsymbol{r}_{n-2} + \cdots. \end{aligned}$$

Now expanding ${}^{n}\boldsymbol{\alpha}^{\text{lin}}$ using (23) and gathering terms of the same $\ddot{\vartheta}_i$ gives

$$\begin{aligned}{}^{n}\boldsymbol{a}^{\text{lin}} &= {}^{n-1}R_n{}^{n-1}\boldsymbol{z}_{n-1} \times \boldsymbol{r}_n\ddot{\vartheta}_n + {}^{n-2}R_n{}^{n-2}\boldsymbol{z}_{n-2} \\ &\quad \times (\boldsymbol{r}_n + {}^{n-1}R_n{}^{n-1}\boldsymbol{r}_{n-1})\ddot{\vartheta}_{n-1} \cdots \\ &= \sum_{i=0}^{n} ({}^{i-1}R_n{}^{i-1}\boldsymbol{z}_{i-1}\ddot{\vartheta}_i \times \sum_{m=i}^{n} {}^{m}R_n{}^{m}\boldsymbol{r}_m). \end{aligned}$$

Applying a rotation into the base frame, ${}^{0}R_n$, to both sides yields

$${}^{n}\boldsymbol{a}^{\text{lin}*} = {}^{0}R_n{}^{n}\boldsymbol{a}^{\text{lin}} = \sum_{i=0}^{n} \left({}^{0}R_n{}^{i-1}\boldsymbol{z}_{i-1} \times \sum_{m=i-1}^{n} {}^{0}R_n{}^{m}\boldsymbol{r}_m\ddot{\vartheta}_m \right).$$

From the foregoing definition of ${}^{i-1}\boldsymbol{p}_{i-1}{}^{n}*$, this is

$${}^{n}\boldsymbol{a}^{\text{lin}*} = \sum_{i=0}^{n} \boldsymbol{z}_{i-1}{}^{*} \times {}^{i-1}\boldsymbol{p}_{i-1}{}^{n}*\ddot{\vartheta}_i. \tag{26}$$

Combining (24) and (26) by concatenating the $\boldsymbol{\alpha}_n{}^{*}$ and $\boldsymbol{a}_n{}^{*}$ into $\boldsymbol{ac}_n{}^{*}$,

$$\begin{aligned}\boldsymbol{ac}_n{}^{*} &= \begin{bmatrix} \boldsymbol{\alpha}_n{}^{*} \\ \boldsymbol{a}_n{}^{*} \end{bmatrix} = \begin{bmatrix} \sum_{i=0}^{n} \boldsymbol{z}_{i-1}{}^{*}\ddot{\vartheta}_i \\ \sum_{i=0}^{n} \boldsymbol{z}_{i-1}{}^{*} \times \boldsymbol{p}_{i-1}{}^{*}\ddot{\vartheta}_i \end{bmatrix} \\ &= \begin{bmatrix} \boldsymbol{z}_0{}^{*} & \boldsymbol{z}_1{}^{*} \cdots \boldsymbol{z}_{n-1}{}^{*} \\ \boldsymbol{z}_0{}^{*} \times \boldsymbol{p}_0{}^{n}* & \boldsymbol{z}_{n-1}{}^{*} \times \boldsymbol{p}_1{}^{n}* \cdots \boldsymbol{z}_{n-1}{}^{*} \times \boldsymbol{p}_{n-1}{}^{n}* \end{bmatrix} \begin{bmatrix} \ddot{\vartheta}_1 \\ \cdot \\ \cdot \\ \cdot \\ \ddot{\vartheta}_n \end{bmatrix} \\ &= J\ddot{\boldsymbol{\vartheta}}. \end{aligned} \tag{27}$$

Rotating both sides of (19) into the base frame allows one to conclude that $D^{*} = J$. □

Once D has been determined through the strobing process, the RJA trajectory can be calculated. Equation (19), however, relates the RJA trajectory to the *linearized* hand accelerations. The precalculated hand accelerations can be linearized by subtracting out the nonlinear Coriolis and centrifugal terms. These nonlinear terms can be generated recursively using ϑ_i as inputs:

$${}^{i}\boldsymbol{\alpha}_i{}^{\text{nonlin}} = {}^{i-1}R_i({}^{i-1}\boldsymbol{\alpha}_{i-1}{}^{\text{nonlin}} + \boldsymbol{\omega}_{i-1} \times \boldsymbol{z}_{i-1}\ddot{\vartheta}_i) \tag{28}$$

$${}^{i}\boldsymbol{a}_i{}^{\text{nonlin}} = \boldsymbol{\omega}_i \times (\boldsymbol{\omega}_i \times \boldsymbol{r}_i) + {}^{i}\boldsymbol{\alpha}_i{}^{\text{nonlin}} \times \boldsymbol{r}_i + {}^{i-1}R_i{}^{i-1}\boldsymbol{a}_{i-1}{}^{\text{nonlin}}. \tag{29}$$

This vector can be subtracted from $\boldsymbol{ac}_n$ (the precalculated hand accelerations) to obtain a relation linear in $\ddot{\boldsymbol{\vartheta}}$:

$$\boldsymbol{ac}_n - \boldsymbol{ac}_n{}^{\text{nonlin}} = \boldsymbol{ac}_n{}^{\text{lin}} = D\ddot{\boldsymbol{\vartheta}}. \tag{30}$$

Now, using Gaussian elimination, one can solve for $\ddot{\boldsymbol{\vartheta}}$.

The resolved motion stage requires the following computations. Determining $\boldsymbol{ac}_n{}^{\text{nonlin}}$ takes $44n$ multiplications and 39 n additions. It takes 24 multiplications and 19 additions to determine the $\boldsymbol{\alpha}_i{}^{\text{lin}}$ and $\boldsymbol{a}_i{}^{\text{lin}}$ at each joint. In strobing the ith joint the calculations proceed from the ith joint to the hand. This process of strobing results in $(24n(n+1))/2$ multiplications and $(19n(n+1))/2$ additions. See Table I for a breakdown. As noted (30) can be solved for $\ddot{\boldsymbol{\vartheta}}(t)$ using Gaussian elimination. Gaussian elimination takes $(n^3/3) + (n^2/2) - (5n/6)$ multiplications and $(n^3/3) - (n/3)$ additions for the elimination and $(n^2/2) + (n/2)$ multiplications and $(n^2/2)$ — (n/2) additions during the back solving. The D matrix becomes singular when the arm is asked to execute impossible motion, such as overreaching. D also becomes singular when two joints are aligned in such a way that either joint could make the desired contribution to the hand's $\boldsymbol{ac}_n$ six vector. In the case of the first type of singularity, the arm trajectory could be constrained to its legal bounds in the trajectory planning phase. In the case of the second type of singularity, a policy might be established whereby the joint closer to the base is allowed to perform the needed motion, while the joint closer to the hand is maintained at its previous value of velocity and acceleration during the backsolving process. This prevents any breakdown in the solution even though a singularity may exist in D. A complexity analysis for the resolved motion is given in Table I.

Applying the results of Table I to the PUMA of Fig. 1 yields a computation count for resolved motion of 982 multiplications and 790 additions ($n = 6$). Closer inspection reveals that many of these computations are unnecessary because of the simple form of the arm vectors $\boldsymbol{r}_i$. The net result is that the computation count can be reduced to about a third of that predicted by the complexity analysis, i.e., about 330 multiplications and 270 additions. If this computation is mapped onto the single chip processor discussed in [10], [15], the computation time per set point works out to about 205 μs.

Gross Motion

In the process of determining the RJA accelerations $\ddot{\boldsymbol{\vartheta}}$, the kinematics of the N–E equations are essentially determined. The $\boldsymbol{\omega}_i$, *the* $\boldsymbol{\alpha}_i{}^{\text{nonlin}}$, and the $\boldsymbol{a}_i{}^{\text{nonlin}}$ are calculated, and the $\boldsymbol{\alpha}_i$ and $\boldsymbol{a}_i$ can be determined with relatively few steps. The $\boldsymbol{\alpha}_i{}^{\text{lin}}$ and $\boldsymbol{a}_i{}^{\text{lin}}$ can be determined from partial terms used in finding the D matrix. To be precise, it requires $3n(n+1)$ multiplications and $3n^2 + 9n$ additions to compute the $\boldsymbol{\alpha}_i$, A_i, and $\boldsymbol{a}_i$.

The next step is to determine the dynamics of the arm using (10)–(15). The torques τ_i obtained from (13) are the torques needed to move the arm along the desired trajectory. The computational complexity of this calculation is given in Table II.

Friction can be modeled in the standard way as Coulombic static friction and Coulombic dynamic friction where the frictional moment is proportional to the force normal to the axis of the actuator, i.e., $\tau_i{}^{\text{coul}} = \mu_{\text{static}} f_i{}^{\text{normal}}$, and when

TABLE I
COMPLEXITY ANALYSIS FOR RESOLVED MOTION

Terms	multiplications	additions
$\mathbf{ac}_n$ converted to hand frame	18n	12n
$\boldsymbol{\omega}_i$	9n	7n
$\boldsymbol{\alpha}_i^{\text{nonlin}}$	11n	8n
A_j^{nonlin}	6n	9n
$A_i\,\mathbf{r}_i + R_i^{i-1}\,\mathbf{a}_{i-1}$	18n	15n
subtotal from $\mathbf{ac}_n^{\text{nonlin}}$	44n	39n
$\boldsymbol{\alpha}_i^{\text{lin}}$	$\frac{9n(n+1)}{2}$	$\frac{7n(n+1)}{2}$
$\boldsymbol{\alpha}_i^{\text{lin}} \times \mathbf{r}_j$	$\frac{6n(n+1)}{2}$	$\frac{3n(n+1)}{2}$
$\boldsymbol{\alpha}_i^{\text{lin}} \times \mathbf{r}_i + R_i^{i-1}\,\mathbf{a}_{i-1}$	$\frac{9n(n+1)}{2}$	$\frac{9n(n+1)}{2}$
subtotal for D	$\frac{24n(n+1)}{2}$	$\frac{19n(n+1)}{2}$
Forward solving	$\frac{n^3}{3}+\frac{n^2}{2}-\frac{5n}{6}$	$\frac{n^3}{3}-\frac{n}{3}$
Backsolving	$\frac{n^2}{2}+\frac{n}{2}$	$\frac{n^2}{2}-\frac{n}{2}$
Total	$\frac{n^3}{3}+13n^2+\frac{221}{3}n$	$\frac{n^3}{3}+10n^2+\frac{179}{3}n$

TABLE II
COMPLEXITY ANALYSIS FOR GROSS MOTION

N–E terms	multiplications	additions
$\ddot{\boldsymbol{\vartheta}}^{\text{adj}}$	2n	4n
$\boldsymbol{\omega}_i, \boldsymbol{\alpha}_i, A_i, \mathbf{a}_i$	3n(n+1)	$3n^2+9n$
$m_i\bar{\mathbf{a}}_i$	12n	9n
$\mathbf{f}_i$	9(n−1)	9(n−1)
$\mathrm{Tr}\,\{A_j K_j (Q^\gamma)^t\}$	6n	3n
$m(\mathbf{r}_i+\bar{\mathbf{r}}_i)\times\bar{\mathbf{a}}_i$	6n	3n
$\mathbf{r}_i \times R_i^{i+1}\mathbf{f}_{i+1}$	15n	9n
$R_{i+1}^i \mathbf{n}_{i+1}$	9n	6n
add n parts	0	15n
friction	2n	2n
Total	$3n^2+64n-9$	$3n^2+69n-9$

motion starts $\tau_i^{\text{coul}} = \mu_{\text{dynam}} f_i^{\text{normal}}$. In addition, viscous friction can be included where the force is proportional to the velocity, $\tau_i^{\text{visc}} = \gamma\dot{\vartheta}_i$. In all cases, the N–E formulation provides the force f_i and the moment n_i applied to the actuator i, along with the angular velocity $\dot{\vartheta}_i$ and the angular acceleration $\ddot{\vartheta}_i$. Thus models depending on these variables for Coulombic and viscous friction (such as the ones given earlier) can easily be employed. Even if friction is highly nonlinear and does not fit any model, lookup tables containing friction forces for various forces, moments and accelerations can be used. The tables may be experimentally determined prior to arm assembly.

In the gross motion stage the N–E equations must incorporate a control loop to compensate for the shortcomings of the model (see earlier discussion). A possible control strategy is explained later. It is adapted from the one proposed by Lee *et al.* [19].

The Lagrangian equations [14], a computationally inefficient alternative to the N–E equations, are useful in that they reveal a relation between the actuator torques $\boldsymbol{\tau}$ (represented here as an n-vector) and the RJA trajectories $\boldsymbol{\vartheta}$, velocities $\dot{\boldsymbol{\vartheta}}$, and accelerations $\ddot{\boldsymbol{\vartheta}}$:

$$\boldsymbol{\tau} = M(\boldsymbol{\vartheta})\ddot{\boldsymbol{\vartheta}} + C(\dot{\boldsymbol{\vartheta}}, \boldsymbol{\vartheta}) + G(\boldsymbol{\vartheta}). \tag{31}$$

These terms are implicitly calculated in the N–E algorithm. For discussion purposes, assume that C, the nonlinear centrifugal and Coriolis terms, and G, the nonlinear gravitational terms, can be calculated close to their true values. Then (31) can be linearized by subtracting C and G from $\boldsymbol{\tau}$ to obtain $\boldsymbol{\tau}^{\text{lin}}$.

$$\boldsymbol{\tau}_a^{\text{lin}} = M(\boldsymbol{\vartheta}_a)\ddot{\boldsymbol{\vartheta}}_a. \tag{32}$$

The subscript a refers to the actual or sensed values of the RJA trajectories, velocities, and accelerations. In the case of torque, it refers to the net value. The subscript d will refer to the desired or calculated values.

If when one inserts the resolved RJA trajectories values into the N–E algorithm, one adjusts $\ddot{\boldsymbol{\vartheta}}_d$ by adding in sensor measured terms

$$\ddot{\boldsymbol{\vartheta}}_d^{\text{adj}} = \ddot{\boldsymbol{\vartheta}}_d + k_v(\dot{\boldsymbol{\vartheta}}_d - \dot{\boldsymbol{\vartheta}}_a) + k_p(\boldsymbol{\vartheta}_d - \boldsymbol{\vartheta}_a) = \ddot{\boldsymbol{\vartheta}}_d + k_v\dot{\boldsymbol{\vartheta}}_e + k_p\boldsymbol{\vartheta}_e, \tag{33}$$

feedback is introduced. The e subscript represents error, indicating in this case, that RJA velocity and positional error are fed back.

With the adjusted $\ddot{\boldsymbol{\vartheta}}$, the Newton–Euler algorithm gives a relation:

$$\boldsymbol{\tau}_d^{\text{lin}} = M(\boldsymbol{\vartheta}_a)(\ddot{\boldsymbol{\vartheta}}_d + k_v\dot{\boldsymbol{\vartheta}}_e + k_p\boldsymbol{\vartheta}_e). \tag{34}$$

The value of the torque at the actuators $\boldsymbol{\tau}_a$ is an approximation to the desired, or calculated, torque. Subtracting (32) from (34), one obtains

$$\boldsymbol{\tau}_d^{\text{lin}} - \boldsymbol{\tau}_a^{\text{lin}} = M(\boldsymbol{\vartheta}_a)(\ddot{\boldsymbol{\vartheta}}_e + k_v\dot{\boldsymbol{\vartheta}}_e + k_p\boldsymbol{\vartheta}_e) \tag{35}$$

a system of damped harmonic equations with zero steady-state error. The coefficient k_p must be chosen large enough to achieve the "stiffness" necessary for precise motion but not too large to promote instability. The coefficient k_v must be chosen to produce a damped response. Wu and Paul [18] have shown that k_v should not be taken as constant but rather k_v is inertial load dependent and hence is $\boldsymbol{\vartheta}$ dependent. Rather than

calculate k_v, tables could be provided to look up k_v values. The tables would not be excessively large since the load varies little over wide angles.

The complexity analysis for the gross motion stage is given in Table II. Applying the results of Table II to the PUMA of Fig. 1 yields a computation count for the gross motion stage 483 multiplications and 513 additions ($n = 6$). Again, closer inspection reveals that many of these computations are unnecessary because of the simple form of the arm vectors r_i. The net result is that the computation count can be reduced to about a third of that predicted by the complexity analysis, i.e., about 170 multiplications and 180 additions. If this computation is mapped onto the single chip processor mentioned earlier, the computation time per set point works out to about 120 μs. Combining this with the set point time for the resolved motion stage gives a computation time of about 325 μs. In other words, resolving the motion between consecutive set points and calculating an adjusted torque to achieve that motion requires about 1/3 ms. This compares favorably to present controllers [10].

Fine Motion

During the terminal phase of arm motion when RJA velocities and accelerations are small, Coriolis and centrifugal terms need not be calculated. Inertial linear and angular accelerations no longer need be calculated. In addition, gravity terms may be updated less often since angles are not varying rapidly. Assume the fractional time for gravitational updates is ζ. (Table III displays the contribution of these terms.) To achieve the desired motion of the hand the resolved motion part of the gross motion stage can still be used. However, it should be applied to the hand's linear and angular velocity rather than the acceleration. This is what was originally proposed by Whitney [1].

To achieve desired output moments or forces by the hand for use in insertion and tool manipulation, pseudo-forces and -moments can be introduced into the equations. The computer can generate force and moment inputs to the hand which do not acutally exist. The arm responds by compensating for these artificially introduced forces and moments. For example, suppose it were desirable to scribe a straight line on a sheet of metal with a 1 N force. The trajectory can be planned so as to move the hand with constant velocity holding the scribe perpendicular to the metal, and the arm can be tricked into believing a -1 N force is pushing against it by setting $f_n = f_{\text{hand}} = -1$ N. The arm reacts by pushing with an extra 1 N force in an effort to maintain its trajectory against the pseudo-force (see Fig. 5). The advantage of the pseudo-force technique is that it requires no extra computation and "falls out" of the N–E model in a natural way.

In the following the pseudo-forces and -moments concept will be shown to be equivalent to an alternative fine motion strategy, the static forces and moments developed in Paul [7]. Assume the arm is static and that gravity has already been compensated for. Equations (11) and (12) can be written

$$f_i = R_i^{i+1} f_{i+1} \tag{36}$$

$$n_i = r_i \times R_i^{i+1} f_{i+1} + R_i^{i+1} n_{i+1}. \tag{37}$$

TABLE III
COMPLEXITY ANALYSIS FOR COMPLETE FINE MOTION

Terms	multiplications	additions
ω_i	9n	7n
α_i^{grav}	$9n\zeta$	$6n\zeta$
A_i	6n	$3n+6n\zeta$
D matrix	$\frac{24n(n+1)}{2}$	$\frac{19n(n+1)}{2}$
Forward solving	$\frac{n^3}{3}+\frac{n^2}{2}-\frac{5n}{6}$	$\frac{n^3}{3}-\frac{n}{3}$
Backsolving	$\frac{n^2}{2}+\frac{n}{2}$	$\frac{n^2}{2}-\frac{n}{2}$
a_i^{grav}	$\frac{3n(n+1)}{2}\zeta$	$\frac{n(n+1)}{2}\zeta$
$m\bar{a}_i^{grav}$	$12n\zeta$	$9n\zeta$
f_i	9(n−1)	9(n−1)
Tr $\{A_i K_i (Q^T)^t\}$	6n	3n
$m(r_i+\bar{r}_i) \times \bar{a}_i^{grav}$	$6n\zeta$	$3n\zeta$
n	15n	24n
friction	2n	2n
Total	$\frac{n^3}{3}+13n^2+\frac{352n}{6}-9$ $+\zeta(\frac{3}{2}n^2+\frac{57}{2}n)$	$\frac{n^3}{3}+10n^2+\frac{340}{6}n-9$ $+\zeta(\frac{n^2}{2}+\frac{53}{2}n)$

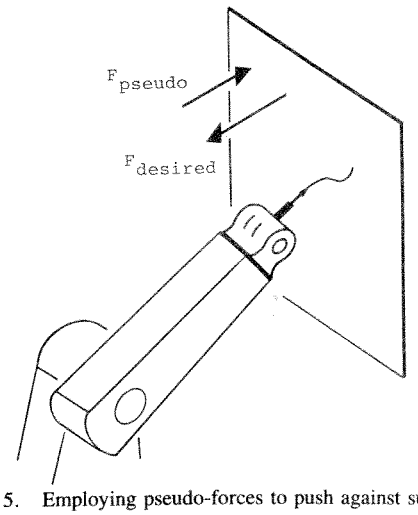


Fig. 5. Employing pseudo-forces to push against surface.

Through recursion, pseudo-forces and -moments input at the hand can be passed down to the ith coordinate frame (since the pseudo-forces and -moments are introduced in the hand frame, $f_n = f_{\text{pseudo}}$ and $n_n = n_{\text{pseudo}}$):

$$f_i = R_i^n f_{\text{pseudo}} \tag{38}$$

$$\boldsymbol{n}_i = \boldsymbol{r}_i \times R_i{}^n \boldsymbol{f}_{\text{pseudo}} + R_i{}^{i+1}\boldsymbol{r}_{i-1} \times R_i{}^n \boldsymbol{f}_{\text{pseudo}} + \cdots R_i{}^n \boldsymbol{r}_n \times R_i{}^n \boldsymbol{f}_{\text{pseudo}}. \tag{39}$$

In terms of a matrix equation, we can express the contribution of the force $\boldsymbol{f}_{\text{pseudo}}$ or moment $\boldsymbol{n}_{\text{pseudo}}$, experienced at the hand, to the force $\boldsymbol{f}_i$ or moment $\boldsymbol{n}_i$ as

$$\begin{bmatrix} \boldsymbol{f}_i \\ \boldsymbol{n}_i \end{bmatrix} = \begin{bmatrix} R_i{}^n \boldsymbol{x}_n & R_i{}^n \boldsymbol{y}_n & R_i{}^n \boldsymbol{z}_n & \boldsymbol{p}_{i-1}{}^n \times R_i{}^n \boldsymbol{x}_n & \boldsymbol{p}_{i-1}{}^n \times R_i{}^n \boldsymbol{y}_n & \boldsymbol{p}_{i-1}{}^n \times R_i{}^n \boldsymbol{z}_n \\ 0 & 0 & 0 & R_i{}^n \boldsymbol{x}_n & R_i{}^n \boldsymbol{y}_n & R_i{}^n \boldsymbol{z}_n \end{bmatrix} \begin{bmatrix} \boldsymbol{f}_{\text{pseudo}} \\ \boldsymbol{n}_{\text{pseudo}} \end{bmatrix} \tag{41}$$

where $\boldsymbol{p}_{i-1}{}^n$ is given by

$$\boldsymbol{p}_{i-1}{}^n = \boldsymbol{r}_i + R_i{}^{i+1}\boldsymbol{r}_{i+1} + \cdots R_i{}^n \boldsymbol{r}_n. \tag{42}$$

See Fig. 3. Note that representation is in the ith frame, not in the base frame. This is identical to the results obtained by Paul [7].

The fine motion portion of the algorithm along with its complexity analysis is detailed in Table III. Included in the analysis is on the fly resolved motion. As before, the total computation count is for one set point. For a PUMA arm the number of multiplications is $415 + 225\zeta$, and the number of additions is $763 + 117\zeta$. Again, simplifications allow us to reduce these numbers to about a third, i.e., $140 + 75\zeta$ multiplications and $260 + 39\zeta$ additions. On the processor mentioned earlier, this requires about $(140 + 40\zeta)$ μs per set point. Recall that ζ is typically a small fraction.

CONCLUSION

A unified approach to three stages of robot arm control has been presented. The unification is based on the N–E equations of motion. Apart from the conceptual advantages of this unification, there are computational savings over treating stages 2, 3, and 4 as separate computations. In particular, fewer computations are required for arms with no more than about a dozen joints (a number unlikely to be exceeded for most arms, at least in the near future). Coupled with recent advances in VLSIC technology, this now makes it possible to perform stages 2, 3, and 4 on the fly. In other words, current real-time constraints (updating a set point each millisecond) can be met even if the trajectory changes each time the arm moves. Of course, computations for the trajectory planning stage must still be done off-line.

In the future, extending the accuracy of the arm model appears promising. Improving the model improves the overall performance of the arm. The underlying philosophy we are promoting is to employ smart sensors and computers to improve the performance of the arm rather than relying on highly rigid arms with precision joints driven by large powerful motors. Trends in current technology suggest that this philosophy will result in cheaper higher performance robots, simply because of the dramatic cost reductions being achieved in the VLSIC technology central to both the production of computer chips and solid-state sensors. No such comparable improvement in technologies associated with other aspects of robot arm fabrication is occurring.

Finally, a considerable amount of research is to be done on trajectory planning algorithms. Once again, the dramatic cost reductions being achieved in VLSIC technology (in particular memory fabrication) suggests that table-lookup techniques may play a role in meeting real-time constraints.

REFERENCES

[1] D. E. Whitney, "Resolved motion rate control of manipulators and human prostheses," *IEEE Trans. Man-Mach. Syst.*, vol. MMS-10, pp. 47–53, June 1969.

[2] M. W. Walker and J. Y. S. Luh, "Manipulator control," School of Elec. Eng., Purdue Univ., West Lafayette, IN, Tech. Rep. TR-EE78-12, Mar. 1978.

[3] 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, pp. 468–473, June 1980.

[4] ——, "On-line computational scheme for mechanical manipulators," *Trans. ASME: J. Dynamic Syst., Meas., Contr.*, vol. 120, pp. 69–76, June 1980.

[5] J. L. Turney, T. N. Mudge, and C. S. G. Lee, "Equivalence of two formulations for robot arm dynamics," Syst. Eng. Lab., Univ. of Michigan, Ann Arbor, SEL Rep. 142, Dec. 1980; reissued as Center for research in Integrated Manufacturing Rep. RSD-TR-3-82.

[6] ——, "Connection between robot arm dynamic formulation with applications to simulation and control," Univ. of Michigan, Ann Arbor, Center for Research in Integrated Manufacturing Rep. RSD-TR-4-82, Nov. 1981.

[7] R. Paul, *Robot Manipulators.* Cambridge, MA: MIT Press, 1981.

[8] ——, "Manupulator Cartesian path control," *IEEE Trans. Syst. Man, Cybern.*, vol. SMC-9, pp. 702–711, Nov. 1979.

[9] T. Lozano-Perez and M. Wesley, "An algorithm for planning collision-free paths among polyhedral obstacles," *Comm. ACM,* vol. 22, pp. 560–570, Oct. 1979.

[10] C. S. Lee, T. N. Mudge, and J. L. Turney, "Hierarchical control structure using special purpose processors for the control of robot arms," in *Proc. IEEE Computer Society Conf. Pattern Recognition and Image Processing,* pp. 634–640, June 1982.

[11] J. Nevins, D. E. Whitney, *et. al.*, "Exploratory research in industrial modular assembly," C. S. Draper Lab., Cambridge, MA, NSF Project Rep. 1–4, Jan. 1974–Aug. 1976.

[12] S. H. Drake, "Using compliance in lieu of sensory feedback for automatic assembly," C. S. Draper Lab., Cambridge, MA, Rep. T-657, Sept. 1977.

[13] S. Simunovic, "Force information in assembly processes," in *Proc. 5th Int. Symp. Industrial Robots,* 1975, pp. 415–431.

[14] R. A. Lewis, "Autonomous manipulation on a robot: Summary of manipulation software functions," Jet Propulsion Lab., Pasadena, CA, Tech. Memo 33-679, Mar. 1974.

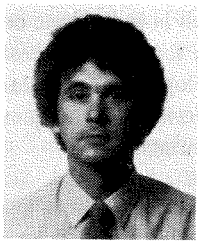
[15] J. L. Turney and T. N. Mudge, "VLSI implementation of a numerical processor for robotics," in *Proc. 27th Int. Instrumentation Symp.,* 1981, pp. 169–175; also presented at the Instrument Society of America Anaheim Conf., Oct. 1981.

[16] J. M. Hollerbach, "A recursive Lagrangian formulation of manipulator dynamics and a comparative study of dynamics formulation complexity," *IEEE Trans. Syst. Man, Cybern.*, vol. SMC-10, pp. 730–736, Nov. 1980.

[17] R. Horowitz and M. Tomizuka, "An adaptive control scheme for mechanical manipulators—compensation of nonlinearity and decoupling control," in *Proc. Dynamic Systems and Control Division ASME Winter Annu. Meeting,* Nov. 1980.

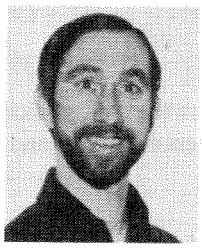
[18] C. H. Wu and R. P. Paul, "Manipulator compliance based on joint torque control," in *Proc. 19th IEEE Conf. on Decision and Control,* Dec. 1980, pp. 88–94.

[19] C. S. G. Lee, T. N. Mudge, M. J. Chung, and J. L. Turney, "On the control of mechanical manipulators," in *Proc. 6-th Int. Federation of Automatic Control Symp. on Identification and System Parameter Estimation,* June 1982.



Trevor N. Mudge (S'74–M'77–SM'84) received the B.Sc. degree in cybernetics from the University of Reading, England, in 1969, and the M.S. and Ph.D. degrees in computer science from the University of Illinois, Urbana, in 1973 and 1977, respectively.

He has been with the Department of Electrical and Computer Engineering at the University of Michigan since 1977 and currently holds the rank of Associate Professor. His research interests include computer architecture, operating systems, VLSI circuit design, computer vision, and robotics.



Jerry L. Turney received the B.S. degree in physics from Michigan State University, East Lansing, in 1972, the M.A. degree in physics from University of California, Berkeley, CA, in 1976, the M.S.E. degree in 1982 from University of Michigan, Ann Arbor, and is presently working toward the Ph.D. degree at the University of Michigan.

Currently he is employed as a Senior Research Engineer at the University of Michigan Robot Research Laboratory. His research interests include computer architecture, robotics and vision.