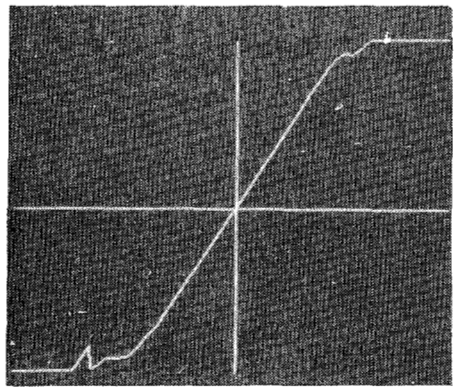

Fig. 8. Input–output characteristic.

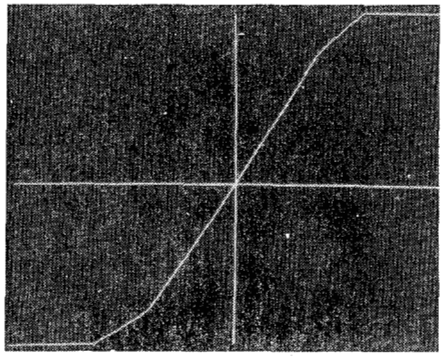

Fig. 9. Input–output characteristic.

Rotation

Due to the rotation property of the two-dimensional Fourier transform rotation of $f(x, y)$ through an angle ϕ_0 also rotates the spectrum by the same amount. It is also well known that the Fourier spectrum (modulus) is invariant to translation.

Now, suppose that we scan the Fourier spectrum $|F_s|^2$ along a circle centered at the origin. If circular sampling of $|F_s|^2$ is used, then by taking the Fourier transform of these samples, extracting the Fourier phase, and subtracting this phase from the reference phase, the rotation angle ϕ_0 can be obtained. In general, few circles may be required to obtain unique and accurate value for the rotation angle. The number of circles can be minimized if the radius and the corresponding samples are properly selected. The method can be summarized as follows.

Step 1: For each specific choice of radius ρ_i, extract K proper samples over the circle centered at the origin with radius ρ_i, where

$$\rho_i = (u^2 + v^2)^{1/2}.$$

Each sample is identified by the radius and its angle ϕ_j, where

$$\phi_j = \arctan \frac{v}{u}.$$

This gives

$$F_s^i(j) = |F_s(\rho_i, \phi_j)|^2, \qquad j = 1, 2, \cdots, k.$$

Step 2: The phase method can be used to determine the rotation angle ϕ_0. By taking the Fourier transform of the sequence

$$F_s^i(j), \qquad j = 1, 2, \cdots, k$$

for each circle. The corresponding phase θ_{ϕ_i} can be extracted.

Step 3: By comparing (subtracting) the incoming and the reference phases, the rotation angle ϕ_0 can be determined.

This approach has a potential of being useful in several important ways: 1) only a few circles may be required to obtain an accurate value for the rotation angle; 2) the same concept can be extended to take into account the dilation and identification problem; and 3) it is clear that the method is computationally simple and inexpensive.

IV. Conclusion

An attempt has been made to treat an image processor, which is used to locate an object, as a transducer. This includes a situation where the object is in motion or stationary object with moving camera.

A Fourier-phase method has been described. This method continuously and automatically produces signals related to the control, which are a function of the incoming image position with respect to the reference image position. The idea of the input–output characteristic is applicable to two-dimensional translation, rotation, and dilation.

Using projection yields one-dimensional characteristics with similar forms (almost) to accomodate motions in two-dimensional scenes.

References

[1] R. J. Grommes and C. J. Yi, "Analysis and simulation of a video tracking system," in *Proc. 1974 IEEE Syst., Man, Cybern. Conf.*, 1974, pp. 93–98.

[2] A. L. Gilbert, L. Alton, M. K. Giles, G. M. Flachs, R. B. Rogers, and U. Y. H. "A real-time video tracking system," *IEEE Trans. Patt. Anal. Mach. Intell.*, vol. PAMI-2, pp. 47–56, Jan. 1980.

[3] T. Uno, M. Ejinu, and T. Tokunaga, "A method of real-time recognition of moving objects and its application," *Patt. Recognition*, vol. 8, pp. 201–208, 1976.

[4] C. A. Winsor and F. J. Thomas, "TVAC—A television area correlation tracking system," in *25th Annual Southwestern Conf. Exhibition Record*, 1973, pp. 501–504.

[5] R. J. Schalkoff and E. S. McVey, "A model and tracking algorithm for a class of video targets," *IEEE Trans. Patt. Anal. Mach. Intell.*, vol. PAMI-4, pp. 2–10, Jan. 1982.

[6] R. M. Inigo and E. S. McVey, "CCD implementation of a three-dimensional video tracking algorithm," *IEEE Trans. Patt. Anal. Mach. Intell.*, vol. PAMI-3, pp. 230–240, Mar. 1981.

[7] E. S. McVey and W. B. Wooland, "A perturbation method for obtaining control signals in an image tracking system," in *Proc. 1979 Joint Automatic Control Conf.*

[8] M. R. Kabuka and E. S. McVey, "A position sensing method using images", in *Proc. IEEE Southeastern Sympo. Syst. Theory*, 1982.

[9] J. J. Pearson, D. C. Hines, Jr., S. Golosman, and C. D. Kuglin, "Video-rate image correlation processor," *SPIE*, vol. 119, pp. 197–205, 1977.

[10] W. K. Pratt, *Digital Image Processing*. New York: John Wiley, 1978.

Multi-Microprocessor-Based Cartesian-Space Control Techniques for a Mechanical Manipulator

CHANG-HUAN LIU, MEMBER, IEEE, AND YEN-MING CHEN

***Abstract*—A multi-microprocessor-based structure for controlling the manipulator motion in Cartesian space is presented. The objective is to investigate the feasibility of implementing hand-based control laws using**

Manuscript revised February 3, 1986.

C.-H. Liu is with the Department of Electrical Engineering and Technology, National Taiwan Institute of Technology, Taipei, Taiwan.

Y.-M. Chen is with the Department of Control Engineering, National Chiao-Tung University, Hsin-Tsu, Taiwan.

IEEE Log Number 8608268.

present day microprocessors. Computational complexity analysis indicates that the control schemes involving full inverse-kinematics and inverse-dynamics computations can be realized using the proposed control structure.

I. Introduction

The motion control of mechanical manipulators in Cartesian space (or task-oriented space) has been a subject of intensive research in recent years. The needs for controlling manipulator motion in Cartesian coordinates can be found in various manufacturing tasks such as conveyor-belt tracking, arc welding, spray painting, assembly, and many others. These tasks are carried out by precisely controlling the linear and angular position, velocity, and/or acceleration of the hand of a manipulator along a preplanned path in Cartesian space. Many schemes have been proposed for controlling the manipulator motion in Cartesian space [1]–[5]. To achieve the control of a desired path along which the hand of a manipulator travels, feedback control loops at either the joint or hand level are required. In either case the real-time transformation of a Cartesian trajectory into the corresponding joint trajectories, the so called inverse-kinematics problem [6] is needed. Once the inverse-kinematics computation is completed, the next issue is the on-line computation of control algorithms at the joint level. One of the more sophisticated approaches to joint-coordinate control is the "inverse problem" or "computed torque" technique [5], [7]. The method takes into account the nonlinear interaction effects among the joints. Assuming that there are no modeling errors, the resultant control action will lead to asymptotic zero tracking errors for all joints. Although the efficient Newton–Euler algorithm [8] for computing the inverse dynamics has been proposed for sometime, a great amount of on-line computation is still needed during the motion. The combined inverse-kinematics and inverse-dynamics computational costs are so prohibitive that neither the computed-torque method nor its variants [3]–[5] have been successfully implemented on general-purpose six-degree-of-freedom manipulators using present-day microprocessors.

Recently, several efficient algorithms [9]–[11] concerning the computation of either the inverse-kinematics or inverse-dynamics of the manipulator have been proposed. The algorithms utilize the fact that most industrial manipulators have relatively simple geometric structures and have only a few practical configurations. For general-purpose manipulators with six degrees-of-freedom, the first-three-link models are usually the Puma or the Standford manipulator type; the last-three-link models (i.e., the wrists) are usually designed to be spherical—i.e., each wrist with three intersecting axes of rotation. Featherstone [9] first took the advantage of the spherical-wrist configuration and proposed an efficient method for computing the inverse kinematic positions and velocities of the manipulator. Hollerbach and Sahar [10] extended Featherstone's results to include the computation of the inverse kinematic accelerations of the manipulator. The algorithms allow one to compute the complete inverse kinematic positions, velocities, and accelerations of the manipulator. The efficiency for computing the inverse dynamics of the manipulator has recently been improved by a technique proposed by Horak [11]. His scheme simplifies the inverse-dynamics computation by symbolic manipulation. When implemented on a single microprocessor, Horak's scheme is about five times faster than the Newton–Euler algorithms.

The efficient algorithms proposed by Featherstone, Hollerbach-Sahar, and Horak spur our interest in investigating the feasibility of designing hand-based control laws that involve full inverse-kinematics and inverse-dynamics computations. Furthermore, the control laws are aimed at implementing on present day microprocessors. Specifically, the six-degree-of-freedom Stanford manipulator [6], [7] is chosen for the study. Though the major results presented in this paper will be limited to the Stanford manipulator, the approach is general enough to be applicable to other general-purpose six-axis industrial manipulators with spherical wrists.

II. Inverse Kinematics

In applying Featherstone and Hollerbach-Sahar's methods, it is necessary to solve the inverse kinematic positions, velocities, and accelerations of the Stanford manipulator in a consecutive manner. For each of the inverse kinematic computation, four steps are executed that systematically compute the joint positions, velocities, or accelerations. Since the detailed procedures for formulating the inverse kinematic equations of six-axis manipulators with spherical wrists have been well documented in [9], [10], the equations for the Stanford manipulator will not be repeated here. Instead, the computational complexity of the derived scheme is evaluated and is tabulated in Table I. This is obtained by counting the number of required mathematical multiplication and addition operations. Table I clearly indicates the efficiency of the scheme as compared with the conventional inverse-kinematics computations that involve computing the Jacobian matrix for the manipulator.

III. Inverse Dynamics

The Lagrange–Euler equations of motion for the Stanford manipulator, excluding the actuator dynamics, gear friction and backlash, and external loading effect, can be expressed as [7]

$$D(q)\ddot{q}+C(q,\ \dot{q})+G(q)=\tau \tag{1}$$

where $D(q)$ is the 6×6 symmetric nonsingular moment of inertia matrix; $C(q,\ \dot{q})$ is the 6×1 vector specifying centrifugal and Coriolis effects; $G(q)$ is the 6×1 vector specifying the gravity effects; τ is the 6×1 vector of input generalized forces; and q represents 6×1 joint position vector. The inverse-dynamics problem is to compute the input generalized forces required for producing the given joint positions, velocities, and accelerations.

Hollerbach [12] showed that the most efficient inverse dynamic computational scheme was the recursive Newton–Euler algorithm. The Newton–Euler algorithm is a general formulation and can handle manipulators with any number of links and any configuration. However, by taking into account the practical configurations of present day industrial manipulators, especially those with spherical wrists, Horak [11] proposed a scheme that partitions the inverse-dynamics computation into two parts: the first part computes the inverse dynamics of the first three links using the Lagrange–Euler equations expressed in symbolic form and the second part computes the inverse dynamics of the wrist (i.e., the last three links) using the Newton–Euler algorithm, also in symbolic form. The resultant generalized forces are the combination of the two. Application of Horak's scheme to computing inverse dynamics of the Stanford manipulator is summarized as follows.

Step 1: Compute the (partial) input generalized forces of the first three links, which are denoted by τ_1', τ_2', and τ_3', respectively. By setting q_i, $\dot{q}_i$, and $\ddot{q}_i$, for $i = 4, 5, 6$, equal to zero and expanding (1), τ_1', τ_2', and τ_3' can be computed.

Step 2: Find the linear and angular velocities of the third link in its own coordinates.

Step 3: Compute the input generalized forces, (τ_4, τ_5, τ_6) for links 4, 5, and 6. The computation utilizes the results of Step 2. The force and the torque that the second part of the manipulator applies on link 3 can also be computed.

Step 4: Add the contributions due to the force and the torque that the second part of the manipulator applies on link 3 to τ_i' for $i = 1, 2,$

TABLE I
COMPUTATIONAL COMPLEXITY OF THE INVERSE KINEMATICS

Inverse-Kinematics Computations	Multiplications	Additions
Inverse-Kinematic Positions	58	36
Inverse-Kinematic Velocities	37	28
Inverse-Kinematic Accelerations	56	47
Total	151	111

TABLE II
COMPUTATIONAL COMPLEXITY OF THE INVERSE DYNAMICS

Step	Multiplications	Additions
1	39	26
2	14	11
3	121	103
4	14	19
Total	188	159

3. The results are the input generalized forces τ_1, τ_2, and τ_3 of the first three links.

The computational complexity of this scheme, when implemented on the Stanford manipulator, is summarized in Table II. The major factor that contributes to the overall computational efficiency of Horak's scheme is the use of closed form solutions in symbolic form that reduces the number of required arithmetic operations. As the symbolic manipulation languages such as MACSYMA are becoming available, the work of expressing part of the Lagrange–Euler or Newton–Euler equations in symbolic form will be simple. The total number of mathematical operations for the Stanford manipulator listed in Table II is less than the one given in [11], which requires 361 multiplications and 256 additions. This further reduction results from the fact that the link inertias for the Stanford manipulator contain the axial moments of inertia only, while the off-diagonal cross products of inertia are all set equal to zero [6], [7].

As seen from the description of Horak's scheme, Step 1 and Steps 2 and 3 are independent of each other and can be executed in parallel using two microprocessors. The computation of Step 4 is executed by either microprocessor. The advantage of this approach is that no special scheduling algorithms are required. The computational efficiency of Horak's scheme for the Stanford manipulator is in part due to the prismatic structure of the third link. When implemented on a rotary manipulatory, such as the Puma, the scheme becomes less efficient and requires between 500 to 650 arithmetic operations [11].

IV. CONTROL LAW FORMULATIONS

The problem of controlling manipulator motion in Cartesian space can be viewed as a tracking problem, where the control law is designed to drive the hand of the manipulator to track a desired Cartesian trajectory. The trajectory specifies the hand positions (and orientations), velocities, and accelerations. Since the actual motion control of the manipulator is done at the joint level, feedback control loops at either the joint or hand level are required. In either case, a hand-based control law must execute fairly complicated inverse kinematic and dynamic algorithms in real time. In fact most of the hand-based control laws proposed up to now must rely on large minicomputers to meet the performance requirement of the manipulator. The key objective in this paper is to demonstrate that, by using the multi-microprocessor approach, some of the existing hand-based control laws can be implemented using present-day microprocessors.

Specifically, three existing control methods that employ full inverse kinematic and dynamic computations are developed. The methods include the computed torque technique [5], the resolved-acceleration control [3], and an adaptive control strategy [13]. Among the three proposed, the first and the third control laws have feedback loops at the joint level, the second closes the feedback loop around the hand. The efficient algorithms discussed in Sections II and III are applied to implementing these control laws. A summary of the proposed control methods is given as follows.

A. Computed Torque Technique

In the computed torque technique, the required input generalized force vector τ is computed from

$$\tau = D_c(q)\ddot{q}_d^* + C_c(q, \dot{q}) + G_c(q) \tag{2}$$

where

$$\ddot{q}_d^* = \ddot{q}_d + K_v(\dot{q}_d - \dot{q}) + K_p(q_d - q); \tag{3}$$

q, $\dot{q}$ measured joint position and velocity vectors;
q_d, $\dot{q}_d$, $\ddot{q}_d$ desired joint position, velocity, and acceleration vectors;
K_v, K_p gain matrices;

and $D_c(q)$, $C_c(q, \dot{q})$, and $G_c(q)$ are the computed counterparts of $D(q)$, $C(q, \dot{q})$, and $G(q)$. If the computed elements are exactly equal to their actual counterparts (i.e., exact manipulator model), then substituting (2)–(3) into (1) yields

$$D(q)\{\ddot{q}_d - \ddot{q} + K_v[\dot{q}_d - \dot{q}] + K_p[q_d - q]\} = 0. \tag{4}$$

Let $e_q = q_d - q$ be the joint position error vector. Since the inertia matrix $D(q)$ is nonsingular, (4) becomes

$$\ddot{e}_q + K_v\dot{e}_q + K_p e_q = 0. \tag{5}$$

By properly selecting K_v and K_p, e_q approaches zero asymptotically.

A hand-based control law based on the computed torque technique can be realized for the Stanford manipulator using the inverse-kinematics and inverse-dynamics computational schemes discussed previously. The realization is formulated by a multi-microprocessor structure in which the first microprocessor computes the inverse-kinematics scheme and the second and third microprocessors compute the inverse-dynamics scheme. Computational complexity of the proposed approach is summarized in Table III.

B. Resolved-Acceleration Control

The resolved-acceleration control closes the feedback loop around the hand. Let $p_d(t)$, $\dot{p}_d(t)$, and $\ddot{p}_d(t)$ be the 3×1 desired position, velocity, and acceleration vectors of the hand in base coordinates, respectively. Using the measured joint positions, the actual rotation matrix A_h and position vector p_h of the hand can be computed. Define further $A_h \triangleq [n_h, s_h, a_h]$, where n_h, s_h and a_h are 3×1 column vectors of A_h. Assume that the desired rotation matrix of the hand is denoted by $A_d \triangleq [n_d, s_d, a_d]$, where n_d, s_d, and a_d are all 3×1 column vectors of A_d. Let $\omega_d(t)$ and $\dot{\omega}_d(t)$ be the desired 3×1 angular velocity and angular acceleration vectors of the hand in base coordinates. Then the linear feedback control is given by

$$\ddot{p}_d^*(t) = \ddot{p}_d(t) + k_1[\dot{p}_d(t) - \dot{p}_h(t)] + k_2[p_d(t) - p_h(t)] \tag{6}$$

and the angular feedback control is given by

$$\dot{\omega}_d^*(t) = \dot{\omega}_d(t) + k_1[\omega_d(t) - \omega_6(t)] + k_2 e_0(t) \tag{7}$$

TABLE III
COMPUTATIONAL COMPLEXITY OF THE COMPUTED TORQUE TECHNIQUE

Microprocessor	Functions	Multiplications	Additions
1	Inverse-kinematics computations: Table I	151	111
2	Inverse-dynamics computations: Table II, Steps 1 and 4, and (3) (Joints: 1, 2, 3)	59	57
3	Inverse-dynamic computations: Table III, Steps 2 and 3, and (3) (Joints: 4, 5, 6)	141	126

TABLE IV
COMPUTATIONAL COMPLEXITY OF THE RESOLVED-ACCELERATION CONTROL

Microprocessor	Functions	Multiplications	Additions
1	Resolved-acceleration: joint-to-hand transformation, (6)–(8), Table I, and inverse kinematic acceleration for computing (9)	174	154
2	Inverse-dynamics computations: Table II and Steps 1 and 4, (Joints: 1, 2, 3)	53	45
3	Inverse-dynamics computations: Table II and Steps 2 and 3 (Joints: 4, 5, 6)	135	114

where k_1 and k_2 are scalar gain constants, ω_6 is the actual angular velocity of the hand in base coordinates, and the orientation error e_0 is determined as

$$e_0(t)=\frac{1}{2}\left[n_h(t)\times n_d(t)+s_h(t)\times s_d(t)+a_h(t)\times a_d(t)\right]. \quad (8)$$

Using (6)–(8), the joint acceleration vector can be resolved according to

$$\ddot{q}_d^*(t)=J(q)^{-1}\left\{\begin{bmatrix}\ddot{p}_d^*(t)\\ \dot{\omega}_d^*(t)\end{bmatrix}-\dot{J}(q)\dot{q}(t)\right\} \quad (9)$$

where $J(q)$ is the Jacobian matrix. The input generalized forces for the Stanford manipulator are computed using (2).

Applying the multi-microprocessor-based approach, the computational complexity of the resolved-acceleration control is tabulated in Table IV. The most time-saving part in the computation is the application of Hollerbach and Sahar's inverse kinematic acceleration scheme for computing (9).

C. Adaptive Control

Recently various adaptive control techniques [13]–[16] have been proposed. For the purpose of comparison, an adaptive control strategy presented in [13] is summarized as follows. The control algorithm consists of two parts: a feedforward nominal control component and a variational control component computed via self-tuning adaptive joint controllers. The control structure is similar to the one proposed in [16]. Let τ_n be the 6×1 vector with entries τ_{ni} for $i = 1, \cdots, 6$ representing the nominal control component. It is computed by feeding the desired joint trajectories $(q_d, \dot{q}_d, \ddot{q}_d)$ into the inverse-dynamics computational routines. The variational control component is denoted by a 6×1 vector v with entries v_i, for $i = 1, \cdots, 6$. It is assumed that, for each joint, the nominal control τ_{ni} can compensate partially the nonlinear coupling effects acting on the joint. The compensated joint dynamics is regarded as linear in the vicinity of the desired trajectories and, in discrete-time, is represented by a second-order autoregressive moving average (ARMA) model:

$$\dot{q}_i(k)=a_1\dot{q}_i(k-1)+a_2\dot{q}_i(k-2)+b_1v_i(k-1)+b_2v_i(k-2) \quad (10)$$

for $k \geq 2$, where the subscript i denotes the ith joint; $\dot{q}_i(k)$ is the ith joint velocity at step k; $v_i(k)$ is the variational control for the ith joint at step k; and (a_1, a_2, b_1, b_2) are unknown parameters to be identified using input–output data. The unknown joint parameters are determined by using the standard recursive least-squares identification routine. $v_i(k)$ is determined from an one-step optimal performance criterion and is given by [13]:

$$v_i(k)=-\frac{\rho_i\hat{b}_1(k)}{\rho_i\hat{b}_1^2(k)+\epsilon_i}\left[\hat{a}_1(k)\hat{\dot{q}}_i(k)+\hat{a}_2(k)\dot{q}_i(k-1)+\hat{b}_2(k)v_i(k-1)-\dot{q}_{di}^*(k+1)\right] \quad (11)$$

where ρ_i and ϵ_i are the scalar constants; $\hat{a}_1(k)$, $\hat{a}_2(k)$, $\hat{b}_1(k)$ and $\hat{b}_2(k)$ are the estimated parameters;

$$\hat{\dot{q}}_i(k)=\hat{a}_1(k)\dot{q}_i(k-1)+\hat{a}_2(k)\dot{q}_i(k-2)+\hat{b}_1(k)v_i(k-1)+\hat{b}_2(k)v_i(k-2)$$

and

$$\dot{q}_{di}^*(k+1)=\dot{q}_{di}(k+1)+\alpha_i[\dot{q}_{di}(k-1)-\dot{q}_i(k-1)]+\beta_i[q_{di}(k-1)-q_i(k-1)]$$

with α_i and β_i representing the velocity and position regulation constants, respectively.

The computational complexity of the proposed adaptive control strategy, if realized by multi-microprocessors, is tabulated in Table V.

V. MICROPROCESSOR IMPLEMENTATION

In order to evaluate the feasibility of implementing the three hand-based control laws using present-day microprocessors, a computing time comparison of the control laws based on three microprocessors is conducted. The specifications of the microprocessors are summarized in Table VI. It is assumed that all the computations are carried out using floating-point arithmetic. The Intel 80287 serves as a floating-point coprocessor to the 16-bit 80286. The MC 68881 also functions as a coprocessor and is designed specifically to operate with the 32-bit MC68020. It can also be used as a memory-map peripheral

TABLE V
COMPUTATIONAL COMPLEXITY OF THE ADAPTIVE CONTROL STRATEGY

Microprocessor	Functions	Multiplications	Additions
1	Inverse-kinematics computations: Table I	151	111
2	Inverse-dynamics computations: Table II and Steps 1 and 4, (Joints: 1, 2, 3)	53	45
3	Inverse-dynamics computations: Table II and Steps 2 and 3 (Joints: 4, 5, 6)	135	114
Joint Microprocessor Controller (adaptive control only)	Parameter identification (RLS routine) and control law computation (11)	81	58

TABLE VI
SPECIFICATIONS OF THE MICROPROCESSORS

Microprocessor	Multiplication (Single (32-bit) Precision)	Addition (Single (32-bit) Precision)
Intel 80286/80287 (8-MHz Clock)	11.8 μs	8.75 μs
Motorola MC68000/MC68881 (12.5-MHz Clock)	7.9 μs[1]	7.6 μs[1]
Motorola MC68020/MC68881 (12.5-MHz Clock)	3.1 μs	2.8 μs

[1] Estimated value.

TABLE VII
MULTI-MICROPROCESSOR-BASED EXECUTION TIME COMPARISON

	Control Law								
	Computed Torque			Resolved-Acceleration			Adaptive Control		
Microprocessor	Intel 80286/ 80287 (8 MHz)	Motorola MC68000/ MC68881 (12.5 MHz)	Motorola MC68020/ MC68881 (12.5 MHz)	Intel 80286/ 80287 (8 MHz)	Motorola MC68000/ MC68881 (12.5 MHz)	Motorola MC68020/ MC68881 (12.5 MHz)	Intel 80286/ 80287 (8 MHz)	Motorola MC68000/ MC68881 (12.5 MHz)	Motorola MC68020/ MC68881 (12.5 MHz)
1	2.8 ms	2.04 ms	0.8 ms	3.4 ms	2.55 ms	0.97 ms	2.8 ms	2.01 ms	0.8 ms
2	1.2 ms	0.9 ms	0.35 ms	1.02 ms	0.76 ms	0.30 ms	1.02 ms	0.76 ms	0.3 ms
3	2.8 ms	2.07 ms	0.8 ms	2.6 ms	1.93 ms	0.74 ms	2.6 ms	1.93 ms	0.74 ms

processor to any M68000 family processors such as a 16-bit MC68000, but with some performance degradation. In Table VI, the execution times of the MC68000/MC68881 are estimated values; it is assumed that, for each multiplication and addition operation, two memory fetches are required. The microprocessor execution times for the proposed three control methods are tabulated in Table VII.

An evaluation of the execution-time performance must be based on the selection of sampling rate for the Stanford manipulator. Since the servo system for the manipulator is configured to be a sampled-data type, the servo rate must be at least 15 times the link structural frequency. In the case of the Stanford manipulator, the sampling rate would be 300 Hz, or a sampling period of 3.3 ms [6, pp. 211–212]. Based on the 3.3-ms sampling-period upperbound, it appears that all three control methods can be implemented using the microprocessors listed in Table VI, except the number one microprocessor for the resolved-acceleration control that uses the Intel 80286/80287. However, execution times given in Table VII do not include other floating-point operations such as sine, cosine, arctangent, division, and square root. This additional execution time for the number one microprocessor is about 0.1 ms using MC68020/MC68881 and 0.63 ms using MC68000/MC68881, respectively. The Intel 80286/80287 is expected to execute longer. The additional execution times for the number two and the number three microprocessors are negligible as compared with the values given in Table VII. Thus in order to successfully implement the three proposed control methods based on the 3.3-ms criterion, the MC68020/MC68881 and MC68000/MC68881 microprocessors can be selected.

In the above comparison, we have adopted the multiple microprocessor-based structure for implementing the control algorithms. The number one microprocessor executes the complete inverse-kinematics algorithm by receiving Cartesian trajectory commands from a host system. The number two and number three microprocessors execute the complete inverse-dynamics algorithm by using the desired and the feedback joint information. During on-line operation,

microprocessor 1 and microprocessors 2 and 3 can be executed in parallel, since Cartesian trajectory segments may be computed one step ahead of the current desired joint set positions, velocities, and accelerations. The multiprocessor system is operated under uniform sampling rate. The estimated execution times of the microprocessors given in Table VII do not include overhead time such as memory management for loading and storing data and system management for bus arbitration. However, the proposed inverse kinematic and dynamic algorithms have already been simplified symbolically. They are straightforward flow algorithms that contain no subroutines or loops. Thus Table VII gives reasonable estimated execution times.

There are other multiprocessor systems that have been proposed for computing the inverse dynamics of manipulators; they usually require special scheduling routines to achieve parallel computation [17]–[19]. Microprocessors 2 and 3 in our proposed multiple microprocessor-based structure can be reconfigured to adapt to these parallel processing algorithms, though further computational complexity analysis is needed in order to meet the sampling period requirement. Recently, several authors have also proposed parallel or pipeline algorithms for computing the inverse dynamics and the Jacobian matrix, which are implementable using very large-scale integration (VLSI) devices [20], [21]. Their approach is quite general and is applicable to various types of manipulators. Still, it would take sometime for VLSI robotic control chips to become commercially available. The approach presented in this paper can be readily implemented for a class of industrial manipulators with spherical wrists using currently available microprocessors.

One may note that the total execution times for the three control methods using only one MC68020/MC68881 are 1.95 ms, 2.01 ms, and 1.84 ms, respectively. This indicates that one MC68020/MC68881 is fast enough to execute the Cartesian-based control algorithms and satisfies the 3.3-ms sampling period requirement. But this conclusion is only valid for the Stanford manipulator considered in this paper, since the algorithms are symbolically simplified by taking the advantages of the spherical wrist and diagonal link inertias of the manipulator. The complexity for computing the inverse dynamics of general-purpose six-axis manipulators is quite high [22], thus necessitating the use of multiprocessing architecture so as to satisfy the sampling period requirement. With the rapid advancement of VLSI technology, it can be expected that the 32-bit microprocessors are better suited for robotic control applications when speed and performance are considered. Though only one 32-bit microprocessor is used for performance evaluation, other 32-bit microprocessors are also available for applications and their performance has constantly being improved [23]. For example, the recently announced 16.67-MHz MC68020/MC68881 or the Intel 80386/80387 should ensure the implementation of hand-based control laws with complete inverse-kinematic and inverse-dynamic computations.

VI. Conclusion

In this paper the design of Cartesian space control techniques for a mechanical manipulator, the Stanford manipulator, has been presented. The key objective is to implement hand-based control laws using present day microprocessors. By applying the efficient inverse-kinematic and inverse-dynamic computational algorithms proposed recently, three existing control methods are shown to be implementable using currently available 16-bit and 32-bit microprocessors. Computational complexity analysis of the control laws indicates that the Cartesian-based schemes can be executed within the required sampling period. The multi-microprocessor-based approach presented in this paper can be applied to the Cartesian motion control of a class of six-axis well-structured (i.e., with spherical wrists) industrial manipulators.

References

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

[2] ——, "The mathematics of coordinated control of prosthetic arms and manipulators," *J. Dynamic Syst. Measurement, Contr., Trans. ASME*, vol. 94, pp. 303–309, Dec. 1972.

[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. 468–474, June 1980.

[4] C. S. G. Lee and B. H. Lee, "Resolved motion adaptive control for mechanical manipulators," *J. Dynamic Syst. Measurement, Contr., Trans. ASME*, vol. 106, pp. 134–141, June 1984.

[5] E. G. Gilbert and I. J. Ha, "An approach to nonlinear feedback control with applications to robotics," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-14, pp. 879–884, Nov./Dec. 1984.

[6] R. P. Paul, *Robot Manipulators: Mathematics, Programming, and Control*. Cambridge, MA: M.I.T., 1981.

[7] A. K. Bejczy, "Robot arm dynamics and control," Jet Propulsion Lab., tech. memo. 33-669, Feb. 1974.

[8] J. Y. S. Luh, M. W. Walker, and R. P. C. Paul, "On-line computational scheme for mechanical manipulators," *J. Dynamic Syst. Measurement, Contr., Trans. ASME*, vol. 102, pp. 69–76, June 1980.

[9] R. Featherstone, "Position and velocity transformations between robot end-effector coordinates and joint angles," *Int. J. Robotics Res.*, vol. 2, pp. 35–45, Summer 1983.

[10] J. M. Hollerbach and G. Sahar, "Wrist-partitioned, inverse kinematic accelerations and manipulator dynamics," *Int. J. Robotics Res.*, vol. 2, pp. 67–76, Winter 1983.

[11] D. T. Horak, "A simplified modeling and computational scheme for manipulator dynamics," *J. Dynamic Syst. Measurement, Contr., Trans. ASME*, vol. 106, pp. 350–353, Dec. 1984.

[12] 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.

[13] C.-H. Liu, "A comparison of controller design and simulation for an industrial manipulator," *IEEE Trans. Ind. Electron.*, vol. IE-33, pp. 59–65, Feb. 1986.

[14] S. Dubowsky and D. T. Des Forges, "The application of model referenced adaptive control to robotic manipulators," *J. Dynamic Syst. Measurement, Contr., Trans. ASME*, vol. 101, pp. 193–200, Sept. 1979.

[15] A. J. Koivo and T.-H. Guo, "Adaptive linear controller for robotic manipulators," *IEEE Trans. Automat. Contr.*, vol. AC-28, pp. 162–171, Feb. 1983.

[16] C. S. G. Lee and M. J. Chung, "An adaptive control strategy for mechanical manipulators," *IEEE Trans. Automat. Contr.*, vol. AC-29, pp. 837–840, Sept. 1984.

[17] J. Y. S. Luh and C. S. Lin, "Scheduling of parallel computation for a computer-controlled mechanical manipulator," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-12, pp. 214–234, Mar. 1982.

[18] H. Kasahara and S. Narita, "Parallel processing of robot-arm control computation on a multimicroprocessor system," *IEEE J. Robotics Automat.*, vol. RA-1, pp. 104–113, June 1985.

[19] R. Nigam and C. S. G. Lee, "A multiprocessor-based controller for the control of mechanical manipulators," in *Proc. 1985 IEEE Int. Conf. Robotics Automat.*, 1985, pp. 815–821.

[20] R. H. Lathrop, "Parallelism in manipulator dynamics," *Int. J. Robotics Res.*, vol. 4, pp. 80–102, Summer 1985.

[21] D. E. Orin, H. H. Chao, K. W. Olson, and W. W. Schrader, "Pipeline/parallel algorithms for the Jacobian and inverse dynamics computations," in *Proc. 1985 IEEE Int. Conf. Robotics Automat.*, 1985, pp. 785–789.

[22] P. K. Khosla and C. P. Neuman, "Computational requirements of customized Newton-Euler algorithms," *J. Robotic Syst.*, vol. 2, pp. 309–327, Fall 1985.

[23] G. Zorpette, "The beauty of 32 bits," *IEEE Spectrum*, vol. 22, pp. 65–71, Sept. 1985.