Proceedings of 23rd Conference
on Decision and Control
Las Vegas, NV, December 1984

Real-Time Control of CMU Direct-Drive Arm II Using Customized Inverse Dynamics[1]

Takeo Kanade, Pradeep K. Khosla and Nobuhiko Tanaka

The Robotics Institute

Carnegie-Mellon University

Pittsburgh, PA 15213

Abstract

Real-time control of fast manipulators requires efficient control algorithms to achieve high sampling rates. Practical implementation of the *inverse dynamics* to achieve high sampling rates demands an efficient algorithm which utilizes the capabilities of modern digital hardware. To reduce the computational requirements of the Newton-Euler (N-E) algorithm for *real-time* applications, we propose the concept of *customizing* the algorithm for specific manipulator designs. We analyze the computational requirements of the algorithm for designs incorporating kinematic and dynamic parameter simplifications. We illustrate our approach by customizing the N-E algorithm for the CMU DD Arm II (the second version of the CMU direct drive arm). The customized algorithm reduces the computational requirements of the general-purpose algorithm by 56 percent. We also describe the hardware system for real-time control of the CMU DD Arm II and the implementation of our customized algorithm on a Marinco processor and highlight its impact on Manipulator control.

1. Introduction

The recursive Newton-Euler (N-E) formulation of robot dynamics has become a standard algorithm for *real-time* simulation and control because of its comparative computational advantages. With this formulation, the solution of the *inverse dynamics problem* (i.e., the problem of computing the joint torques required to achieve the desired positions, velocities and accelerations) has become feasible for *real-time* control. Using the Newton-Euler algorithm, control sampling periods of 16-20 *ms* have been achieved and are considered adequate for industrial applications[1, 2]. While sampling periods of 16-20 *ms* may be adequate for industrial manipulators, they are inadequate for high-speed manipulation using direct drive arm technology[3].

Practical implementation of the *inverse dynamics* algorithm to achieve higher sampling rates demands an efficient algorithm which utilizes the capabilities of modern digital hardware. To reduce the computational load of the N-E algorithm, the authors[4, 5] have proposed to *customize* the algorithm for specific manipulators. To customize the N-E formulation we take advantage of the sparsity of *all* matrices and vectors in the formulation[6].

The objective of this paper is to highlight the customized design and prototype implementation of our Newton-Euler algorithm on a Marinco[7] processor for the real-time control of the anthropomorphic direct drive arm (DD Arm II) being developed at Carnegie-Mellon University. We have implemented our algorithm and measured the execution time. Our experimental data indicate that we can implement our *inverse dynamics* algorithm with a control sampling period of 1.2 *ms* (and thereby realize a sampling frequency of approximately 830 Hz).

This paper is organized as follows: In Section 2, we outline the Newton-Euler algorithm. The computational requirements of the N-E algorithm under a variety of physical simplifications are enumerated in Section 3. The design implications of these simplifications are then noted. We describe, in Section 4, the implementation of our customized N-E algorithm for the CMU DD Arm II. In Section 5, we highlight the hardware system of our experimental prototype for the *real-time* control of the CMU DD Arm II and illustrate the effect of fast sampling on manipulator control. Conclusions are then advanced in Section 6.

2. The Newton-Euler Formulation

In this section, we summarize the Newton-Euler formulation[8]. For our application, we outline the algorithm for a *rotary* manipulator.

To formulate the dynamic equations-of-motion for an N degree-of-freedom (DOF) manipulator, the Denavit-Hartenberg convention[9] defines the N+1 coordinate frames (the i-th coordinate frame being attached to the i-th link, beginning with link 0 - the base, and ending with link N - the end-effector). The four Denavit-Hartenberg parameters, which define homogeneous transformations between successive coordinate frames, are listed in Table 2-1. For a revolute joint *i*, the joint position variable is the angle θ_i; and the remaining three are constants.

The kinematic and dynamic parameters and variables, which appear in the Newton-Euler algorithm, are compiled in Table 2-2. Equations (1)-(7) outline the *forward* recursions (from the base to the end-effector). The initial conditions (for i = 0) assume that the manipulator is at rest in a gravitational field. The propagation of the linear velocities v_i in (3), which is included for completeness, is not evaluated in efficient implementations[8]. Equations (8)-(10) display the backward recursions of the forces and moments at the end effector and compute the *joint torques*.

$$\omega_{i+1} = A_{i+1}^T \left[\omega_i + z_0 \dot{\theta}_{i+1} \right] \tag{1}$$

$$\dot{\omega}_{i+1} = A_{i+1}^T \left[\dot{\omega}_i + z_0 \ddot{\theta}_{i+1} + \omega_i \times (z_0 \dot{\theta}_{i+1}) \right] \tag{2}$$

$$v_{i+1} = A_{i+1}^T \left[v_i \right] + \omega_{i+1} \times p_{i+1} \tag{3}$$

$$\dot{v}_{i+1} = A_{i+1}^T \left[\dot{v}_i \right] + \dot{\omega}_{i+1} \times p_{i+1} + \omega_{i+1} \times (\omega_{i+1} \times p_{i+1}) \tag{4}$$

$$\dot{v}_i^* = \dot{\omega}_i \times s_i + \omega_i \times (\omega_i \times s_i) + \dot{v}_i \tag{5}$$

$$F_i = m_i \dot{v}_i^* \tag{6}$$

$$N_i = J_i \dot{\omega}_i + \omega_i \times (J_i \omega_i) \tag{7}$$

$$f_i = A_{i+1} \left[f_{i+1} \right] + F_i \tag{8}$$

$$n_i = A_{i+1} \left[n_{i+1} \right] + p_i \times f_i + N_i + s_i \times F_i \tag{9}$$

[1]This research was supported, in part, by the ONR Contract No. N00014-81-0503.

$$\tau_i(t) = n_i^T (A_i^T z_0) \qquad (10)$$

Initial Conditions

$$\omega_0 = \dot{\omega}_0 = v_0 = 0$$

$\dot{v}_0 = [g_x \; g_y \; g_z]^T$ (gravitational acceleration of the manipulator base)

Terminal Conditions

f_{N+1} is a known externally applied force at the end-effector.

n_{N+1} is a known externally applied moment at the end-effector.

Table 2-1: Denavit-Hartenberg Link Parameters (Scalars)

Definitions of the Parameters
θ_i *is the angle of rotation about the* z_{i-1} *axis*
a_i *is the length of translation along the rotated* x_i *axis*
d_i *is the length of translation along the* z_{i-1} *axis*
α_i *is the angle of rotation about the* x_i *axis*

Table 2-2: Kinematic and Dynamic Parameters

m_i	Total mass of link i
$\tau_i(t)$	Joint torque at joint i
ω_i and $\dot{\omega}_i$	Angular velocity and acceleration of the ith coordinate frame
v_i and $\dot{v}_i$	Linear velocity and acceleration of the ith coordinate frame
v_i^* and $\dot{v}_i^*$	Linear velocity and acceleration of the center-of-mass of link i
F_i and N_i	Net force and moment exerted on link i
f_i and n_i	Force and moment exerted on link i by link (i – 1)
p_i	Position of the ith coordinate frame with respect to the (i – 1)st coordinate frame: $p_i = [\, a_i \;\; d_i \sin(\alpha_i) \;\; d_i \cos(\alpha_i) \,]^T$
s_i	Position of the center-of-mass of link i: $s_i = [\, s_{ix} \;\; s_{iy} \;\; s_{iz} \,]^T$
z_0	$= [0\,0\,1]^T$
J_i	Classical inertia tensor of link i about the center-of-mass of link i; with principal inertias J_{ixx}, J_{iyy} and J_{izz}; and cross-inertias J_{ixy}, J_{ixz} and J_{iyz}.
A_i	Orthogonal rotation matrix which transforms a vector in the (i-1)st coordinate frame to a coordinate frame which is parallel to the ith coordinate frame:

$$A_i = \begin{bmatrix} \cos(\theta_i) & -\cos(\alpha_i)\sin(\theta_i) & \sin(\alpha_i)\sin(\theta_i) \\ \sin(\theta_i) & \cos(\alpha_i)\cos(\theta_i) & -\sin(\alpha_i)\cos(\theta_i) \\ 0 & \sin(\alpha_i) & \cos(\alpha_i) \end{bmatrix}$$

for i = 1,2,...,N, where $A_{N+1} \equiv I$.

3. Computational Requirements

In this section, we quantify the computational requirements for the *inverse dynamics* algorithm implementations of Section 2. The computational requirements are analyzed, in increasing order of simplifications, for the following implementations:

1. Straightforward
2. General-Purpose
3. Parallel/Perpendicular Axes
4. Parallel/Perpendicular Axes and Spherical Wrist
5. Diagonal Inertia Tensors
6. Customized Implementation for DD Arm II

We have quantified the number of floating-point operations for the case of 6 DOF arms and these are enumerated in Table 3-1.

3.1. Straightforward Implementation

Our *straightforward* implementation of the N-E algorithm incorporates *only* the savings introduced by the sparse z_0 vector. This implementation is convenient from the programming point-of-view and requires 129N multiplications and 103N additions/subtractions. For a six degrees-of-freedom manipulator the computational requirements are 774 multiplications and 618 additions/subtractions.

3.2. General-Purpose Implementation

The *general-purpose* implementation incorporates the savings introduced by the zero (or the [3,1]) element of the homogeneous transformation A_i matrices, the sparse z_0 vector, the zero initial conditions, and the gravitational acceleration $\dot{v}_0 = [0\ 0\ \text{-}g]^T$ of the manipulator base. The stationary-base assumption creates identically zero elements in recursions (1), (2), (4) and (6)-(8) which can be eliminated from the algorithm[5]. An N degree-of-freedom manipulator requires 123N-60 multiplications and 96N-55 additions/subtractions. For a six degrees-of-freedom manipulator the computational requirements are 678 multiplications and 521 additions/subtractions. The general-purpose implementation, thus, reduces the computational requirement by 96 multiplications and 97 additions compared to the straightforward implementation.

3.3. Parallel/Perpendicular Axes

Most of the existing manipulators have adjacent axes which are either parallel or perpendicular. For this orientation of the axes, the computational load is 91N-46 multiplications and 74N-41 additions/subtractions. A 6 degree-of-freedom manipulator thus requires 500 multiplications and 403 additions/subtractions. This reduction in the floating-point operations are a direct consequence of the following observations:

1. Generation of the A_i matrices requires neither multiplication nor addition. This property of parallel/perpendicular axes leads to a reduction of 4N or 24 multiplications for a 6 DOF manipulator.

2. Multiplication of a vector by an A_i matrix requires 4 multiplications and 2 additions instead of 8 multiplications and 5 additions for the general-purpose implementation.

3. Calculation of the joint torques $\tau_i(t)$ in (10) does not require any computation. This fact accrues a saving of 2N multiplications and N additions/subtractions for an N DOF manipulator. For 6 degrees-of-freedom manipulator), 12 multiplications and 6 additions/subtractions are saved.

3.4. Spherical Wrist

Pieper[10] showed that a wrist with *three* intersecting axes-of-rotation is a kinematic configuration that always leads to a closed-form solution for the reverse kinematic problem. A spherical wrist is characterized by the zero terminal position vectors[11], $p_N = p_{N-1} = [\,0\ 0\ 0\,]^T$. If we assume that the manipulator has parallel/perpendicular axes *and* a spherical wrist:

1. We save 28 multiplications and 22 additions/subtractions in the computation of the linear accelerations of links N and N-1 in (4); and

2. We save 8 multiplications and 8 additions/subtractions in the calculation of the link moments n_N and n_{N-1} in (8).

A spherical wrist thus saves 36 multiplications and 30 additions/subtractions.

For a rotary manipulator with parallel/perpendicular axes and a spherical wrist, the computational requirements become 91N-82 multiplications and 74N-71 additions/subtractions. A 6 degree-of-freedom manipulator requires 464 multiplications and 373 additions/subtractions.

3.5. Inertial Parameters

If we assume parallel/perpendicular axes *and simplified inertial* parameters (i.e., diagonal J_i tensors), we only accrue additional computational savings in the calculation of the net link moments N_i in (7). The reduction is 12N-1 multiplications and 12N-4 additions/subtractions. The computational load becomes 79N-45 multiplications and 62N-37 additions/subtractions. A 6 DOF rotary manipulator thus requires 429 multiplications and 335 additions/subtractions.

If we combine the savings introduced by parallel/perpendicular axes, a spherical wrist and diagonal inertia tensors, we reduce the computational burden to 79N-81 multiplications and 62N-61 additions/subtractions. A 6 DOF rotary manipulator requires 393 multiplications and 305 additions/subtractions. The computational savings introduced by the aforementioned simplified kinematic and inertial parameters are independent because they affect different recursions in the N-E algorithm: simplified kinematic parameters affect (4) and (9); and simplified inertial parameters affect (7).

3.6. Computational Savings

We have enumerated, in Table 3-1, the reduction in computational requirements achieved by introducing practical simplifications. The following observations emerge directly from Table 3-1:

1. The fractional savings in multiplications parallel the fractional savings in additions/subtractions for each practical simplification.

2. The dominant computational savings arise from the practical assumption that adjacent axes are either parallel or perpendicular. This axis arrangement saves 25 percent of the computational load of the general-purpose implementation (in Table 3-1). This finding reinforces the design of manipulators with adjacent axes which are parallel or perpendicular.

3. The assumption of diagonal inertia tensors saves an additional 10-15 percent, and the symmetry of the J_i tensors saves 3N storage locations. From the practical point-of-view, we note that the designer has little control over the structure of the J_i tensor for a link, and hence this simplification may be difficult to realize.

4. A spherical wrist, combined with parallel/perpendicular axes, reduces the computational load by 30 percent, and these savings are significant.

5. The remaining simplifications arise from *not* implementing multiplication by zero or one. These simplifications *customize* the implementation for specific applications. For example, each zero position vector $p_i = [0\ 0\ 0]^T$ saves 18 multiplications and 15 additions/subtractions. A spherical wrist with $p_N = p_{N-1} = 0$ saves 5-10 percent. Additional computational savings accrue from the zero elements of the s_i vector from the link frame origin to the center-of-gravity of the link.

Table 3-1: Computational Requirements of the N-E Algorithm (6 DOF Rotary Manipulator)

Design	Implementation	Multiplications	Additions/Subtractions
1	Straightforward	774	618
2	General-Purpose	678	521
3	Parallel/Perpendicular Axes	500	403
4	Spherical Wrist and (3)	464	373
5	Diagonal Inertia Tensors and (3)	429	335
6	Spherical Wrist and (5)	393	305
7	DD Arm II (Customized)	303	226

In this section, we have analyzed the computational requirements for *inverse dynamics*, under a variety of physical simplifications. By judicious mechanical design, we can reduce these requirements and the increased sampling rate has profound implications for the *real-time* control of high-speed manipulators. In the next section, we design our customized N-E algorithm for the DD Arm II and reduce further the actual computational requirements for our application.

4. Customized N-E Algorithm for the DD Arm II

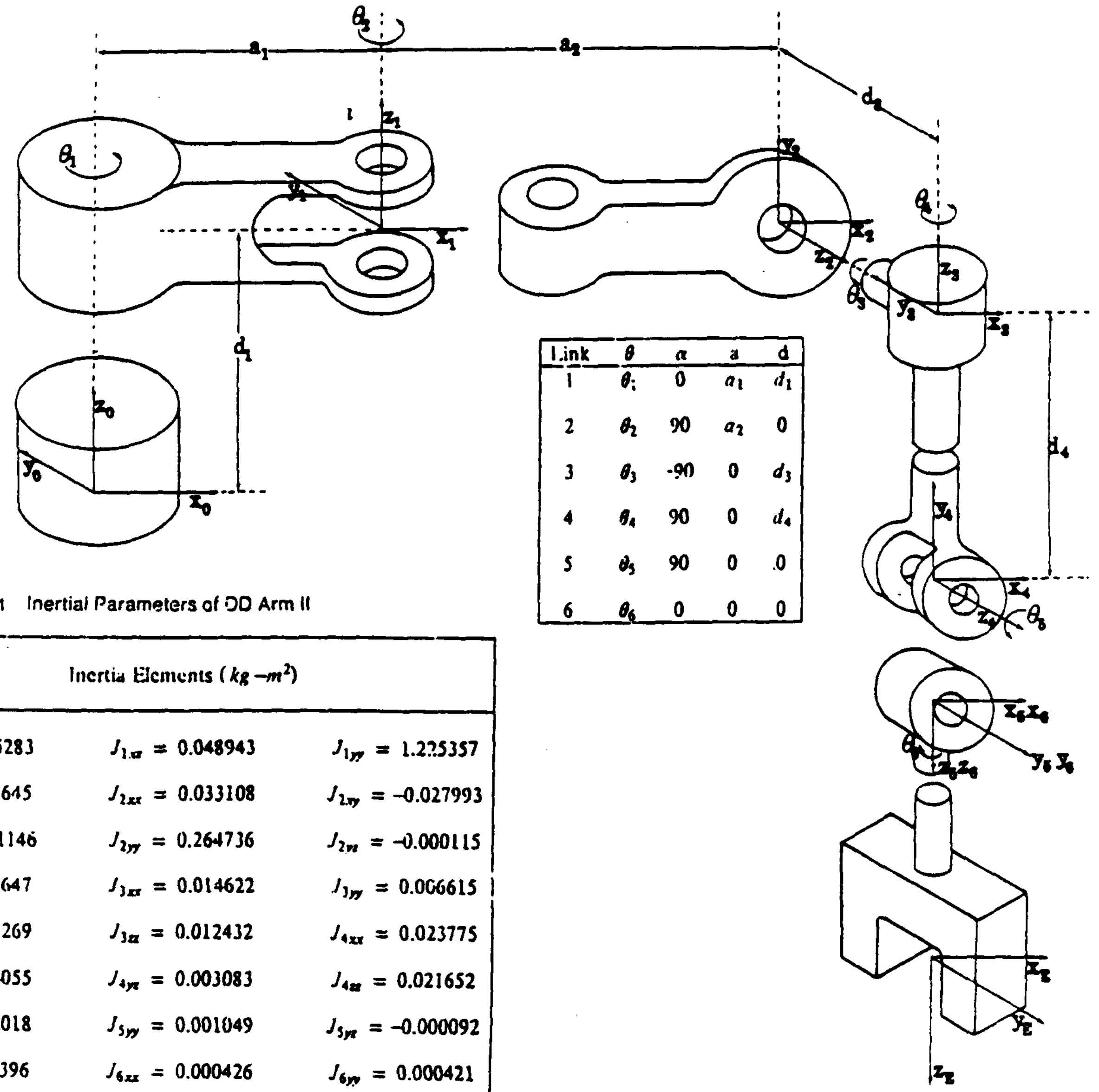
In this section, we customize the N-E algorithm for our six degree-of-freedom DD Arm II[12]. In Section 3, we observed that reduction in computational requirements can be achieved by not implementing multiplications by zero and unity elements. We propose a new convention which gets rid of redundant operations and establishes a framework for customizing the N-E algorithm. Using this convention it is possible to derive implementations which incorporate physical simplifications. Hence, in this section we go a step further and take advantage of the sparse link inertia tensor (J_i) and vectors to center-of gravity s_i and thereby *customize* the Newton-Euler algorithm for DD Arm II.

4.1. CMU DD Arm II

We have developed, at CMU, the concept of direct-drive robots in which the links are directly coupled to the motor shaft. This construction eliminates undesirable properties like friction and gear backlash. The CMU DD Arm II is the second version of the CMU direct-drive manipulator and is designed to be faster, lighter and more accurate than its predecessor DD Arm I[3]. We have used brushless *rare-earth magnet* DC torque motors to reduce the friction and the *ripple*. The SCARA-type configuration of the arm reduces the torque requirements of the first two joints and also simplifies the dynamic model of the arm. To achieve the desired accuracy, we use very high precision (16 bits/rotation) rotary absolute encoders. The arm weighs approximately 70 pounds and is designed to achieve maximum joint accelerations of 10 rad/sec^2.

The link coordinates of the CMU DD Arm II, depicted schematically in Figure 4-1, are assigned by the Denavit-Hartenberg convention[13]. The numerical values of the non-zero inertial parameters (i.e.,the masses, the center-of-gravity vectors and inertia tensors) of the links and their variable names as used by the customized *algorithm* are listed in Table 4-1. We model each link as a composite of hollow cylinders, solid cylinders, prisms and rectangular parallelopipeds. We then use our Arm Modeling Package[8] to compute the inertia tensor and the vector to the center-of-gravity of all of the links and generate a data-base for our DD Arm II.

Figure 4-1: Coordinate Assignments and Kinematic Link Parameters of the DD Arm II



Link	θ	α	a	d
1	θ_1	0	a_1	d_1
2	θ_2	90	a_2	0
3	θ_3	-90	0	d_3
4	θ_4	90	0	d_4
5	θ_5	90	0	0
6	θ_6	0	0	0

Table 4-1 Inertial Parameters of DD Arm II

Inertia Elements ($kg-m^2$)		
J_{1xx} = 0.135283	J_{1xz} = 0.048943	J_{1yy} = 1.225357
J_{1zz} = 1.193645	J_{2xx} = 0.033108	J_{2xy} = −0.027993
J_{2xz} = −0.001146	J_{2yy} = 0.264736	J_{2yz} = −0.000115
J_{2zz} = 0.286647	J_{3xx} = 0.014622	J_{3yy} = 0.006615
J_{3yz} = 0.001269	J_{3zz} = 0.012432	J_{4xx} = 0.023775
J_{4yy} = 0.004055	J_{4yz} = 0.003083	J_{4zz} = 0.021652
J_{5xx} = 0.002018	J_{5yy} = 0.001049	J_{5yz} = −0.000092
J_{5zz} = 0.001396	J_{6xx} = 0.000426	J_{6yy} = 0.000421
J_{6zz} = 0.000047		

Centers-of-Gravity (m)		
s_{1x} = −0.300000	s_{1z} = 0.100399	s_{2x} = −0.131742
s_{2y} = −0.013219	s_{2z} = 0.001105	s_{3y} = −0.039703
s_{3z} = −0.012487	s_{4y} = 0.071645	s_{4z} = 0.022866
s_{5y} = 0.005481	s_{5z} = −0.017932	s_{6z} = 0.008176

Link Masses (kg)		
m_1 = 19.753	m_2 = 7.894	m_3 = 2.801
m_4 = 1.881	m_5 = 0.936	m_6 = 0.269

4.2. Customization Procedure

In order to develop a systematic procedure for *customization* of the Newton-Euler algorithm, we propose the following convention:

The non-zero elements of a vector or a matrix are denoted by subscripted variables, and the zero and unity elements by a 0 and 1, respectively. We propagate the non-zero elements as variables and the zero elements as zeros[4, 5].

With this convention, we label the non-zero elements of the vector s_i to the center-of-gravity of link i by the variable s_{ij}, where the subscript i denotes the link number and the subscript j (for j = x,y or z) denotes the x, y or z components of the vector, respectively. The inertia tensors of the links are symbolized with the same convention as the vector to the center-of-gravity. The non-zero elements of the inertia tensor of link i are denoted by J_{ijk}, where j and k are the coordinate components (x, y and z) of the inertia tensor J_i.

We use the link parameters and the inertial parameters to generate our customized Newton-Euler algorithm for the DD Arm II. We illustrate our approach by developing the angular velocities in (1) and linear accelerations in (4). The initial conditions for the forward recursions are utilized and the gravitational acceleration of the manipulator base is:

$$v_0 = [0 \;\; 0 \;\; -g]^T .$$

By using (1) and the initial conditions, we formulate the angular velocities of the six links. The angular velocity ω_1 of link 1, expressed in terms of the angular velocity of link 0 is

$$\omega_1 = \begin{bmatrix} 0 \\ 0 \\ \dot{\theta}_1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \omega_{1z} \end{bmatrix}$$

Generation of ω_1 thus only requires the measurement of the joint angular velocity $\dot{\theta}_1$. Since the N-E algorithm is recursive and the angular velocity ω_2 is expressed in terms of the angular velocity ω_1, we pass the fact that two of the elements of ω_1 are zero (according to our convention) to the computation of ω_2. The angular velocity ω_2 is

$$\omega_2 = \begin{bmatrix} 0 \\ \omega_{1z} + \dot{\theta}_2 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ \omega_{2y} \\ 0 \end{bmatrix}$$

Computation of ω_2 thus requires only one addition. Upon continuing the recursion, we compute the angular velocity vector ω_3 according to:

$$\omega_3 = \begin{bmatrix} S_3\omega_{2y} \\ -\dot{\theta}_3 \\ C_3\omega_{2y} \end{bmatrix} = \begin{bmatrix} \omega_{3x} \\ \omega_{3y} \\ \omega_{3z} \end{bmatrix}$$

where $C_3 = \cos(\theta_3)$ and $S_3 = \sin(\theta_3)$.

Since ω_3 is a full vector, computation of ω_4, ω_5 and ω_6 is a generic operation and each requires 4 multiplications and 3 additions/subtractions. If we had not passed zero elements as zero, we would have required 24 multiplications and 18 additions/subtractions. Our approach thus reduces the computational requirements to 14 multiplications and 10 additions/subtractions for a saving of 10 multiplications and 8 additions/subtractions.

Efficient computation of the angular accelerations $\dot{\omega}_i$ in (2) parallels the foregoing generation of the angular velocities ω_i in and require 23 multiplications and 18 additions/subtractions.

We illustrate our approach further by detailing the computation of the linear accelerations of the links. From (4), the initial conditions and the link parameters, we obtain each of the constituent components of the linear acceleration of link 1 as follows:

$$A_1^T[\dot{v}_0] = \begin{bmatrix} 0 \\ 0 \\ -g \end{bmatrix}$$

$$\dot{\omega}_1 \times p_1 = \begin{bmatrix} 0 \\ -\dot{\omega}_{1z}a_1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ b_{12} \\ 0 \end{bmatrix}$$

$$\omega_1 \times p_1 = \begin{bmatrix} 0 \\ -\omega_{1z}a_1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ b_{22} \\ 0 \end{bmatrix}$$

and

$$\omega_1 \times (\omega_1 \times p_1) = \begin{bmatrix} -\omega_{1z}b_{22} \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} b_{31} \\ 0 \\ 0 \end{bmatrix}$$

The linear acceleration of link 1 is then:

$$\dot{v}_1 = \begin{bmatrix} b_{31} \\ b_{12} \\ -g \end{bmatrix} = \begin{bmatrix} \dot{v}_{1x} \\ \dot{v}_{1y} \\ \dot{v}_{1z} \end{bmatrix}$$

Computation of $\dot{v}_1$ requires only 3 multiplications. The savings arise from the fact that the simple structure of ω_1 and of $\dot{\omega}_i$ was passed to the next computation using our convention. The computation of the linear accelerations for all the six links require 42 multiplications and 24 additions/subtractions. The computational requirements for a six degree-of-freedom rotary manipulator with parallel/perpendicular axes and a spherical wrist are 70 multiplications and 47 additions/subtractions. We thus save 28 multiplications and 23 additions/subtractions by our approach.

We have followed the abovementioned approach to customize our Newton-Euler algorithm for the DD Arm II[4]. Our customized algorithm requires 303 multiplications and 226 additions/subtractions For our DD Arm II, we reduce the computational requirements of the general-purpose implementation by 56 percent. These dramatic reductions transcend the physical simplifications outlined in Table 3-1.

Since the DD Arm II is a manipulator with a spherical wrist, we can achieve further reduction in the computational requirement (for cartesian space control) by *customizing* Hollerbach and Sahar's[11] scheme of *Wrist Partitioned Kinematic Inverses*. In the next section, we outline the prototype implementation of our N-E algorithm.

5. Real-Time Implementation of the Customized Algorithm

We have implemented the customized N-E algorithm to obtain real-time performance for the control of DD Arm II. In the ensuing paragraphs, we describe the hardware of the arm control system and the performance evaluation of our customized algorithm.

5.1. System Description of the DD Arm II Control Hardware

The hardware of the DD Arm II control system, as shown in Figure 5-1, consists of three integral components: the Motorola M68000 microcomputer, the Marinco processor and the TMS-320 microprocessor-based individual joint controllers. The following paragraphs describe the function of each component of the control system.

The Motorola M68000 microcomputer

The Motorola M68000 microcomputer is the host system and controls the associated Marinco array processor and the TMS-320 based joint controllers. At each sampling instant the M68000 system reads the joint positions and velocities from the shared memory locations and transfers these values to the Marinco processor. The M68000 system and the TMS-320 share a common memory segment through which all the communications take place. The M68000 system sees the Marinco processor as its own memory and communicates by writing data in the Marinco memory. After depositing the joint data, the M68000 system starts the execution of the inverse dynamics program on the Marinco processor.

Individual Joint Controllers

Each individual joint controller consists of a TMS-320 microprocessor chip. At each sampling instant the TMS-320 system reads the digital value of the individual joint positions, and computes the corresponding velocities and accelerations. These values are then written into a shared memory location from where the M68000 can read them. When the joint controller is using the shared memory it is not possible for the M68000 system to access it and memory contention problems are avoided. When the data is ready to be read by the M68000 system, the joint controller gives control of the memory to the M68000 system.

The APB-3024 Marinco Processor

The Marinco board operates as an arithmetic processor for the Motorola M68000 microprocessor. The Marinco board appears as internal memory to its host processor and hence is a *memory mapped* device. This architecture speeds up the data transfer operations because the *handshaking* commands for external communications are avoided. The Marinco has *triport* (or sharable) memory which allows both the host and the array processor to access any portion of the array processors memory at all times and hence results in parallel I/O and computation. The starting and stopping of the Marinco board is controlled by the host Motorola M68000 microprocessor. When the *inverse dynamics* computation is completed, the Marinco board interrupts the M68000 microprocessor which transfers the computed torque values to the individual joint motor controllers.

5.2. Implementation of the Customized Algorithm on a Marinco Processor

The Marinco processor is an inexpensive array processor with an instruction cycle of 125 ns. The board has fixed-point multiplier and additional hardware to perform the floating-point operations. Each of our floating-point multiply and add routines execute in approximately 1 μs. Negation takes one instruction or 125 ns. Our floating-point subtract (a negation followed by a floating-point add) thus executes in 1 125 μs. In our implementation we obtain the rotational components $\cos(\theta_i)$ and $\sin(\theta_i)$ of the transformation A_i matrices through subroutine calls. Computation of the sine/cosine pair takes 15 μs. Since 6 quadrature pairs are required for the 6 DOF rotary DD Arm II, a total of 90 μs is required to complete these calculations. The execution times cited for the arithmetic floating-point operations do not include overheads for data preparation and data storage. The total execution time of the N-E algorithm, however includes all the overheads in the program. We estimate a 1.26 *ms* computational cycle of our *customized* N-E *algorithm (in Section 4) for the DD Arm II* in Table 5-1.

The measured execution time of our hardware implementation is 1.2 *ms* and we are able update the joint motor torques at a sampling frequency of 830 Hz. Our prototype implementation demonstrates an order-of-magnitude reduction in the sampling periods compared to the published figures[2] of 16-20 *ms*. The achievable sampling rate of 830 Hz opens horizons for the design and practical implementation of novel manipulator controllers.

5.3. Effect of Fast Sampling on Manipulator Control

We have developed the CMU DD Arm II for high-speed manipulation [12]. The arm is designed to achieve maximum joint velocities of 12 rad/sec and maximum joint accelerations of 10 rad/sec^2. In this section we analyze the impact of the achievable fast sampling rates on manipulator control schemes.

If the manipulator model is exact then the joint torques computed at each sampling instant are exact. However, modeling inaccuracies lead to a discrepancy between the model and the manipulator and result in tracking errors. Since the model of the DD Arm II is reasonably accurate, we can reduce significantly the errors between the desired and actual joint positions, velocities and accelerations by closing the servo-loop around each joint and implementing the computed torque[14, 15] control signal

$$\ddot{q}_i(k) = \ddot{q}_{id}(k) + K_v[\dot{q}_{id}(k) - \dot{q}_i(k)] + K_p[q_{id}(k) - q_i(k)]$$

where K_p and K_v are the position and velocity gains, respectively.

To test the effect of fast sampling on the control algorithm we conducted computer simulations using our Dynamics Simulation Package[6] on a Vax 11/750 computer. The task was to move joint 2 from its rest position of zero degrees to 90 degrees and back to zero degrees in a sinusoidal manner while commanding the other joints to maintain their zero positions. In our simulation we also included a model mismatch of 10% in the link inertias.

We conducted the simulation experiment for sampling periods of 1, 5 and 10 ms. In all the three cases, joint two was commanded to move with a maximum angular velocity of 8 rad/sec. We placed the poles of the error equation at -20 sec^{-1} to obtain the values of K_p and K_v as 400 and 40 respectively. Figures 5-2, 5-3 and 5-4 depict the position error curves for joints 1, 2 and 3 respectively. We have not included the error curves for joints 4, 5 and 6 for the lack of space. For a sampling period of 10 *ms* the maximum positioning errors along the trajectory for joints 1, 2 and 3 are 2.6, 5.3 and 7.9 degrees, respectively. Decreasing the sampling period to 1 *ms* reduces the positioning errors for joints 1, 2 and 3 to 0.25, 1.06 and 1.1 degrees, respectively. This reduction of the positioning errors is a consequence of the reduction in the control sampling period. Our computer simulations show that the positioning errors of the individual joints can be reduced, for a given set gains and mismatch in the model, by decreasing the control sampling period.

Table 5-1: Execution times of the Marinco Processor

Operation	Number	Execution Time(μs)	Total Execution Time (ms)
Multiplication	303	2.0	0.606
Addition	226	2.0	0.552
Negation	80	0.125	0.010
Sine-Cosine	6	15	0.09
Total			1.258

6. Conclusions

To reduce the computational requirements of the Newton-Euler algorithm, we have developed the concept of *customizing* the algorithm for specific manipulators. This concept can be applied to any manipulator, including manipulators with translational degrees-of-freedom. With this technique it is required to derive the Newton-Euler algorithm only once for every manipulator. Our *customized* N-E algorithm for the CMU DD Arm II reduces the computational requirements of the general-purpose N-E algorithm by 56 percent. The proposed *customized* algorithm has been implemented in hardware and we have achieved execution times of 1.2 *ms*. This demonstrates an order of magnitude increase in the sampling rates compared to previously reported figures of 16-20 *ms*. We have shown, through computer simulations, that increasing the sampling frequency leads to a reduction in the joint positioning errors along the specified trajectory.

The increased sampling rates have a tremendous impact on manipulator control. The sampling rate can be further increased by implementing the floating-point operations fully in hardware. Implementation of schemes which compute the joint torques in parallel will reduce the computation time phenomenally and lead to higher sampling frequencies. We are, at present, conducting experimental hardware evaluation of the manipulator control scheme.

Figure 5-1: System Configuration of the DD Arm II Control System

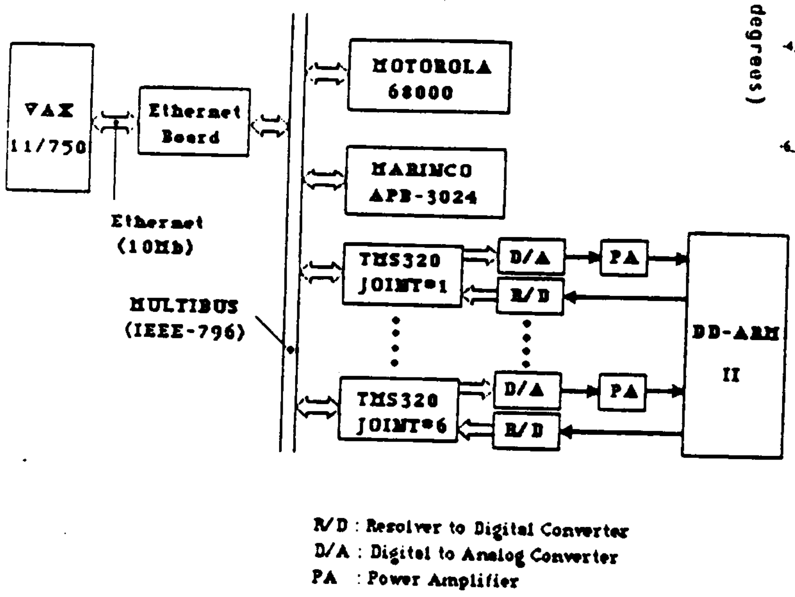
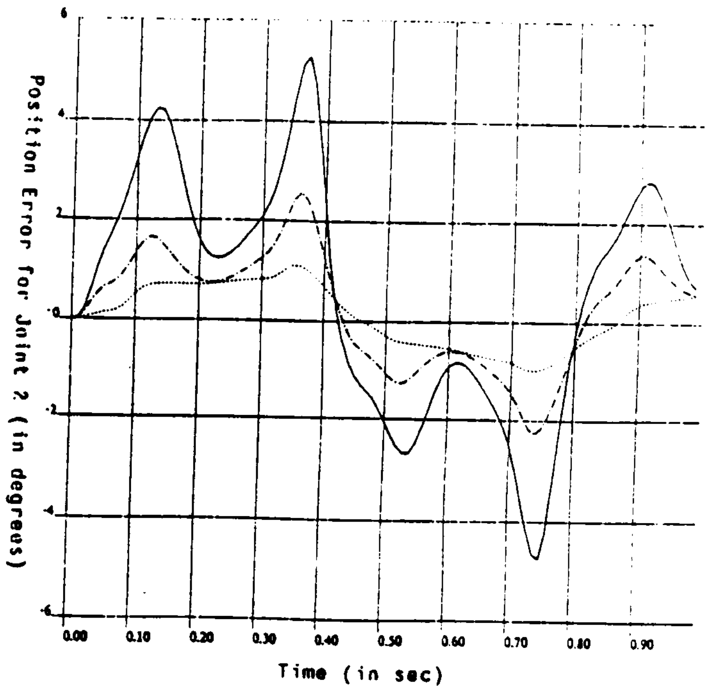


Figure 5-3: Position Tracking Error of Joint 2



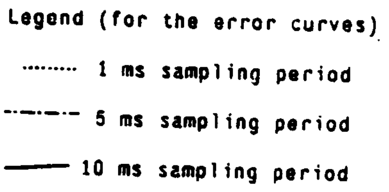


Figure 5-2: Position Tracking Error of Joint 1

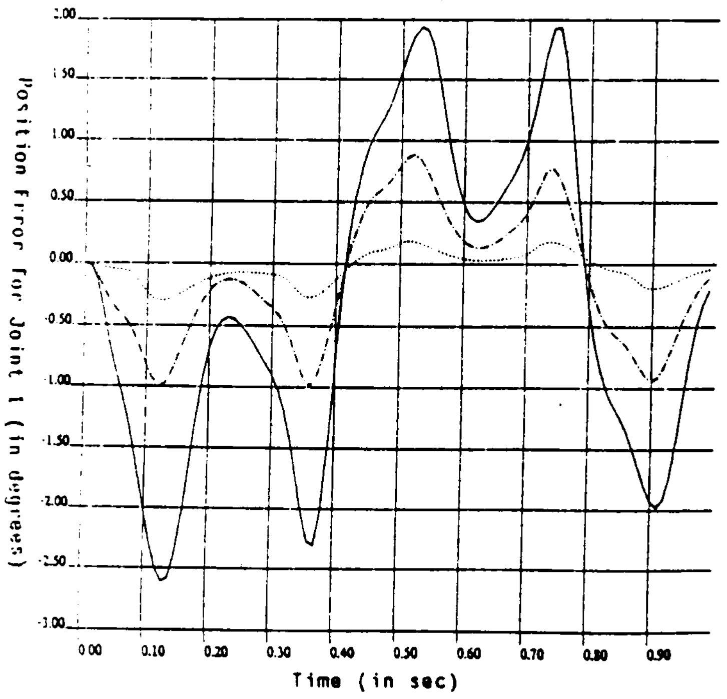
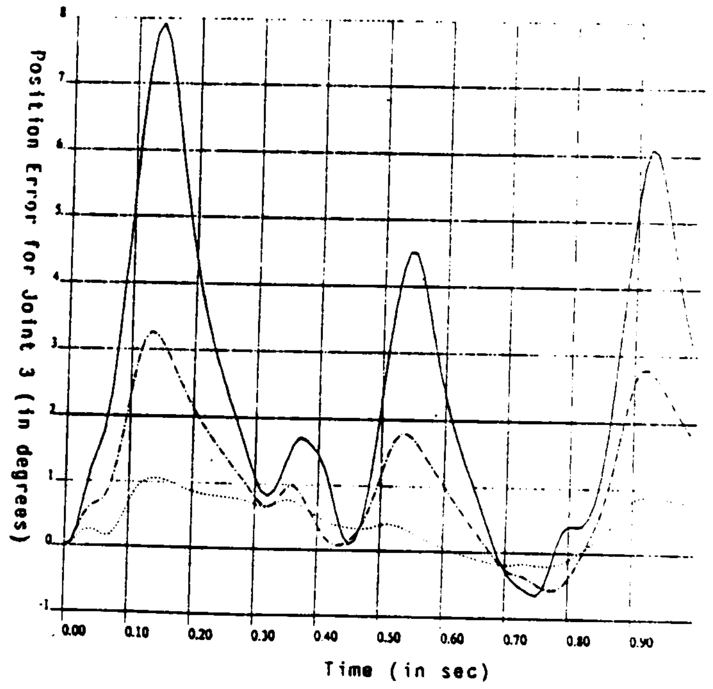


Figure 5-4: Position Tracking Error of Joint 3



References

1. Luh J. Y. S., Walker, M. W. and Paul R. P. C., "Resolved-Acceleration Control of Mechanical Manipulators," *IEEE Transactions on Automatic Control*, Vol. 25, No. 3, June 1980, pp. 468-474.

2. Luh J. Y. S., Walker, M. W. and Paul R. P. C., "On-Line Computational Scheme for Mechanical Manipulators," *Journal of Dynamic Systems, Measurement, and Control*, Vol. 102, No. 2, June 1980, pp. 69-76.

3. Asada, H. and Kanade, T., "Design of Direct Drive Mechanical Arms," *Journal of Vibration, Stress, and Reliability in Design*, Vol. 105, No. 1, July 1983, pp. 312-316.

4. Khosla, P. K. and Kanade, T., "Dynamic Equations of DDArm II," Robotics Institute Document, Vision Laboratory RVL-021, *Carnegie-Mellon University*, 1983.

5. Khosla, P. K. and Kanade, T., "Real-Time Implementation of Arm Dynamics," Robotics Institute Document, Vision Laboratory RVL-022, Carnegie-Mellon University, 1983.

6. Khosla, P. K. and Neuman, C. P., "Computational Requirements of Customized Newton-Euler Algorithms," *Technical Report, Department of Electrical Engineering*, Carnegie-Mellon University, Pittsburgh, PA 15213, 1984.

7. Marinco APB-3024M Array Processor Board, *Reference Manual*, Marinco, Inc., 3878-A Ruffin Road, San Diego, CA 92123, 1983.

8. Swartz, N. M., "Arm Dynamics Simulation," *Journal of Robotic Systems*, Vol. 1, No. 1, 1984, .

9. Denavit, J. and Hartenberg, R. S., "A Kinematic Notation for Lower Pair Mechanisms Based on Matrices," *Journal of Applied Mechanics*, Vol. 77, No. 2, June 1955, pp. 215-221.

10. Pieper, D. L., *The Kinematics of Manipulators under Computer Control*, PhD dissertation, Department of Computer Science, Stanford University, 1968.

11. Hollerbach, J. M. and Sahar, G., "Wrist Partitioned Inverse Kinematic Accelerations and Manipulator Dynamics," *Proceedings of the First International IEEE Conference on Robotics*, Paul, R. P., ed., Atlanta, GA, March 13-15, 1984, pp. 152-161.

12. Schmitz, D. and Ratner, D., "Design and Construction of the CMU Direct Drive Arm II," Robotics Institute Document, Vision Laboratory RVL-019, Carnegie-Mellon University, 1984.

13. Khosla, P. K. and Kanade, T., "Kinematics of DDArm II," Robotics Institute Document, Vision Laboratory RVL-017, Carnegie-Mellon University, 1983.

14. Bejczy A. K., "Robot Arm Dynamics and Control," Technical Memorandum 33-669, Jet Propulsion Laboratory, Pasadena, CA, February 1974.

15. Markiewicz, B. R., "Analysis of the Computer Torque Drive Method and Comparision with the Conventional Position Servo for a Computer-Controlled Manipulatror," Technical Memorandum 33-601, Jet Propulsion Laboratory, Pasadena, CA., March 1973.

16. Neuman, C. P. and Tourassis, V. D., "Discrete Dynamic Robot Models," Technical Report, Department of Electrical and Computer Engineering, Carnegie-Mellon University, 1984.