HIERARCHICAL CONTROL STRUCTURE USING SPECIAL PURPOSE PROCESSORS FOR THE CONTROL OF ROBOT ARMS*

C. S. G. Lee, T. N. Mudge, J. L. Turney

Robot Systems Division
Center for Robotics and Integrated Manufacturing
The University of Michigan
Ann Arbor, Michigan 48109

Abstract

This paper presents a proposal for a hierarchical control structure that uses special purpose processors for the control of industrial robots. The systems consist of a general purpose computer as a supervisory machine with attached special purpose processors that perform the bulk of the numerically intensive computations in the real-time control algorithms. The host machine performs trajectory planning, coordinate systems transformations, and coordination among various robots. The special purpose processor is proposed as a single chip processor which performs real-time interpolation between set points from the host machine, computes the correction torque for each joint of the robot arm. Comparisons between the proposed control structure and current industrial control technique are discussed. The architecture of the processor is outlined. The gross motion control strategy is briefly discussed and the computational complexity of the control law is tabulated. Finally, the results of a functional simulation of the processor executing part of a control program are noted.

Introduction

Present computer technology provides cost effective solutions to many problems which were not too long ago considered infeasible. With the increasing availability of inexpensive memory and the burgeoning of VLSI technology, it is now cost-effective to design special purpose attached processors that are tailored to specific but complex computational problems. Normally, these problems could only be solved by expensive mainframe machines. One such problem is the real-time control of a robot arm.

The purpose of robot arm control is to maintain a prescribed motion for the arm along a desired arm trajectory by applying corrective compensation torques to the actuators to adjust for any deviations of the arm from that desired trajectory. Several modes of manipulator control have evolved during the last three decades. The computer-controlled mode is the center of current development trends. This technique promises to extend the use of robot arms far beyond the domain of repetitive tasks.

Conventional servomechanism techniques are being used in present day computer-controlled manipulators. However, the motion dynamics of an "n" degree-of-freedom manipulator is inherently nonlinear and can only be described by a set of "n" highly coupled nonlinear second order ordinary differential equations. The nonlinearities arise from inertial loading, coupling between neighboring joints, and gravitational loading of the links. Furthermore, the dynamic parameters of a manipulator vary with the position of the joint variables which are themselves related by complex trigonometric transformations. The servomechanism approach models the varying dynamics of a manipulator inadequately and neglects the coupling effects of the joints. As a result, manipulators controlled this way move at slow speeds with unnecessary vibrations. This reduces their application to tasks which can tolerate limited precision.

A priori information needed for control is a set of equations of motion describing the dynamic behavior of the manipulator. The dynamic equations of motion formulated by the Lagrange-Euler (L-E) method have been shown to be computationally inefficient[1,2]. However, a direct Newton-Euler (N-E) formulation coupled with an appropriate special purpose processor has been suggested as a solution[3].

This paper presents a proposal for a hierarchical control structure that uses special purpose attached processors. This proposal, it is argued, would allow the real-time control of industrial robots, in particular the PUMA robots. The main advantages of this approach are: (i) The computation of the joint torques is based on the dynamic model of a robot arm allowing a faster more responsive control system to be constructed. (ii) The dynamic model facilitates variable feedback gains to accommodate varying payloads.

In the following sections vectors are represented in boldface lower case alphabets while matrices are in boldface upper case alphabets.

Current Control Methods

As noted above, given the equations of motion of a manipulator, the control problem is to find appropriate torques/forces to servo all the joints of the manipulator in real-time to track a desired trajectory as closely as possible. Several control methods have been developed to accomplish this task. Notable among these are: (i) The resolved motion rate control[4], (ii) The Cerebellar Model Articulation Controller[5], (iii) The near-minimum-time control[6], (iv) The suboptimal control[7], and (v) The model reference adaptive control[8]. Of particular relevance to this discussion is the current PUMA robot arm control scheme, which we will briefly describe.

The controller consists of an LSI-11/02, and six 6503 microprocessors each with a joint encoder, a digital-to-analog converter (DAC), and a current amplifier. The control structure is hierarchically arranged. At the top of the system hierarchy is the LSI-11/02 microcomputer which serves as a supervisory computer. At the lower level are the six 6503 microprocessors--one for each degree of freedom (see Fig. 1--all figures are at the end of the paper). The LSI-11/02 computer performs two major functions: (i) online user interaction and subtask scheduling from the user's

* This work was supported in part by the National Science Foundation Grant ECS-8106954 and the Robot Systems Division of the Center for Robotics and Integrated Manufacturing (CRIM) at The University of Michigan, Ann Arbor, MI. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the funding agencies.

VAL[1] commands, and (ii) subtask coordination with the six 6503 microprocessors to carry out the command. The on-line interaction with the user includes parsing, interpreting, and decoding the VAL commands, in addition to reporting appropriate error messages to the user. Once a VAL command has been decoded, various internal routines are called to perform scheduling and coordination functions. These functions which reside in the EPROM of the LSI-11/02 computer include: (i) coordinate systems transformations (e.g. from the world coordinates XYZOAT to the joint coordinates $\vartheta_1, \vartheta_2, \cdots, \vartheta_6$ or vice versa) (ii) joint-interpolated trajectory planning; this involves sending incremental location updates corresponding to each set point to each joint every 28 ms. (iii) acknowledging from the 6503 microprocessors that each axis of motion has completed its required incremental motion. (iv) two instruction lookahead to perform the continuous path interpolation if the robot is in continuous path mode.

At the lower level in the system hierarchy is the joint controller which consists of a digital servo board, an analog servo board, and power amplifiers. The 6503 microprocessor is an integral part of the joint controller which directly controls each axis of motion. Each microprocessor resides on a digital servo board with its EPROM and DAC. It communicates with the LSI-11/02 computer through a Unimation-designed interface board which functions as a demultiplexer that routes set points information to each joint controller. The interface board is in turn connected to a 16 bit DEC parallel interface board (DRV-11) which transmits the data to and from the Q-Bus of the LSI-11/02 (see Fig. 1). The microprocessor computes the error signal and sends it to the analog servo board which has a lead-lag compensator designed for each joint motor. The feedback gain of the compensator is tuned to run at a "VAL speed" of 100. There are two servo loops for each joint control (see Fig. 1). The outer loop provides position error information and is updated by the 6503 microprocessor about every millisecond. The inner loop consists of analog devices and a compensator with derivative feedback to put damping on the velocity variable. Both servo loop gains are constant and tuned to perform as a "critically-damped joint system" at a normal speed of 100 (VAL speed). The main functions of the microprocessor include:

(1) Every 28 ms, receive and acknowledge set points from the LSI-11/02 computer and perform interpolation between the current joint value and the desired joint value.

(2) Every millisecond (approximately), read the register value which stores the incremental values from the encoder mounted at each axis of rotation.

(3) Update the error actuating signals derived from the joint-interpolated set points and the values from the axis encoders.

(4) Convert the error actuating signal to voltage using the DAC's, and send the voltage to the analog servo board which moves the joint.

It can be seen that the PUMA robot control scheme is basically a position plus derivative control method. One of the main disadvantages of this control scheme is that the feedback gains are constants and prespecified. It does not have the capability of updating the feedback gains under varying payloads. Since an industrial robot is a highly nonlinear system, the inertial loading, the coupling between joints and the gravity effects are all position dependent terms. Furthermore, at high speeds the inertial loading term can change drastically. Thus, the above control scheme using constant feedback gains to control a nonlinear system does not perform well under varying speeds and payloads. In fact, the PUMA arm moves with noticeable vibrations at reduced speeds.

One solution to this problem is the use of digital control with feedforward components computed by a special purpose processor. The proposed control structure would provide an improvement over the existing PUMA robot arm control technique.

Proposed Digital Control Scheme

As noted above, the PUMA robot arm control scheme suffers from the fact that the feedback gains are constant and a simple servomechanism is used to servo a nonlinear system. We argue that a better solution is the use of a special purpose processor (APAC--Attached Processor for Arm Control) to compute all the joint torques plus the correction torques based on a complete dynamic model of the robot arm. This has the advantage of being able to change the feedback gains in the digital servo loop if the load is changing within a task cycle.

Overall System Configuration

The proposed overall system control structure is also hierarchically arranged. The LSI-11/02 computer still serves as a supervisory computer while the APAC performs the dedicated functions of servo control. The LSI-11/02 performs the same functions as before, while the APAC replaces all the functions of the 6503 microprocessors performing dedicated control according to the dynamic model of the robot arm. The APAC controls the robot arm as a whole system whereas the 6503 microprocessors perform individual joint servoing. The system is similar to that depicted in Fig. 1. At one end the APAC communicates with the LSI-11/02 computer through a DEC parallel interface board. At the other end it communicates with the analog devices such as power amplifiers through a multiplexer. All the joint position information from the encoders are fed back to the APAC also through a multiplexer. Based on the dynamic model formulated by the N-E method and the feedback information from all the joints, the APAC computes all the joint torques and the correction torques and feeds the required signals to the power amplifiers within one millisecond. The dynamic model and the control equation used by the APAC are discussed in detail in the next two sections.

Special Purpose Processor Architecture

This section presents the preliminary specification for a very large scale integrated (VLSI) circuit implementation of our proposed APAC, a single chip processor for dedicated numerically intensive control applications. Circuit densities commensurate with levels of integration projected for the mid-1980s are assumed. The proposed APAC is suitable for real-time control where sophisticated control strategies require very large numbers of high precision arithmetical operations to be performed for every input/output transaction. In particular, the APAC is intended for the real-time control of a robot arm. The APAC functions as an attached processor of a general purpose minicomputer. It operates on 32 bit floating point data. Conceptually, it lies between Floating Point Systems' AP120B, a high performance numerically oriented attached processor, and the Intel 8087[9], a single chip numerically oriented attached processor in the Intel 8086 family of components. All three work with floating-point numbers. The APAC differs from the AP120B

[1] VAL is a registered trade-mark of Unimation. It is the command language for the PUMA series of robot arms.

by being much simpler, less flexible, slower, and by having a smaller word size (32 bits versus 38 bits). It differs from the 8086 by having its own on chip program memory, input/output buffers to facilitate real-time applications, and two independent function units. However, the 8087 has a more flexible number format, and can deal with several variants of the IEEE floating point standard up to and including the 80 bit format. This preliminary study assumes the APAC will be implemented in nMOS. However, our eventual aim is to investigate the design of the APAC in a faster technology that still has the density of integration associated with nMOS. A prime candidate is the I3L (Isoplanar Integrated Injection Logic) technology developed by Fairchild Corporation. The major components are as follows:

1. A 32 bit floating point adder unit (AU).
2. A 32 bit floating point multiplier unit (MU).
3. A 256x32 register file (RF).
4. A 32x32 bit input buffer (IB).
5. A 32x32 bit output buffer (OB).
6. A 1Kx50 bit program memory (PM).
7. A 4x10 bit program counter stack (PCS).
8. A 1x50 bit program memory data register (PMDR).
9. A 16 bit loop counter (LC).
10. Condition code logic (CC).

The data path is shown in Fig. 2. A preliminary gate level logic design and layout of an nMOS realization of the chip, using the design rules given in Mead and Conway[10], indicates that 50% of the area will be occupied by the AU, MU, RF and PM. The other components occupy less than 10% of the area, and the buses, control signal lines, and bonding pads occupy the remaining 40% of the chip. An estimate, based on a logic gate count, of the number of active devices required by the chip indicates that 90% will be contained in just four of the components--the AU, MU, RF, and PM. The estimate shows 16K devices are required for the AU, 32K for the MU, 16K for the RF, and 56K for the PM. The estimate for the total device count works out to be 150K. This is well within projections for single chip systems in the mid 1980's. At that time 1 million devices/chip are anticipated[11].

The floating point number format used in the design study is the 32 bit proposed IEEE standard described in Coonen[12]. Both the AU and the MU were designed to handle this format. However, the rounding modes, rounding precision control, infinity arithmetic, denormalized arithmetic, most of the floating point exceptions, and the various extended formats called for by the proposed standard were not considered in the design of either the AU or the MU. Naturally, inclusion of any of these features would increase the complexity of both the AU and MU, and estimates of the device count would have to be adjusted accordingly.

The AU is a three stage pipeline with the first stage performing fraction alignment, the second stage performing fraction addition, and the final stage performing normalization. Alignment is performed using an 8 bit subtractor with full lookahead to determine the number of shifts needed followed by a 5 (= $\lceil \log\ 24 \rceil$) level 24 bit barrel shifter to execute the shifts. Fraction addition is performed using a standard 24 bit binary adder structure with partial carry lookahead across groupings of 4 bits. Normalization is performed using another 24 bit barrel shifter. The basic machine cycle (M-cycle) is targeted at 500 nS. Each stage of the pipeline completes its task within an M-cycle, thus when the AU is in streaming mode--operands are being fed to it as fast as possible--it produces a result every 500 nS. The AU is constructed from standard NOR/NOR PLAs (program logic arrays[10]) having an estimated delay of 50 nS. This is fast enough to be used as a building block in the construction of an alignment stage--potentially the most time consuming of the AU's three stages--that can operate within 500 nS. It is also fast enough to construct the binary adder for the fraction addition, as well as the barrel shifter for the normalization stage.

The MU is also a three stage pipeline with the first stage performing partial product generation and carry-save addition, the second stage performing carry propagation addition to produce the unnormalized 48 bit product fraction, and the final stage performing normalization, truncation to 24 bits, and exponent addition. The design of the multiplier is quite standard. Stage one uses a tree of 3-input to 2-output carry-save adders, implemented with PLAs as the basic building block. With a tree height of 8 (= $\lceil \log\ 24 \rceil$) and 50 nS delay per PLA the 500 nS time limit for a pipeline stage is easily met (generating the partial products requires only an array of AND's and adds only 15 nS to the delay time). Stage two uses a 48 bit adder with full lookahead. The lookahead is across groups of 4 bits, and is performed by lookahead units that are realized as PLAs. The lookahead units themselves produce group propagate and group generate signals which feed another level of lookahead units. This process is continued in the standard fashion to produce a lookahead tree of height 3 (= $\lceil \log\ 48 \rceil$). The total time to add is thus (3x2)x50=150 nS plus the delay through a full adder (50 nS). The third stage performs normalization using a shift register--normalization after multiply never requires more than a one position right shift if numbers are represented in the format above. The final step in stage three--exponent addition--is performed using a simple 8 bit ripple carry adder. The effect of normalization on the exponent is also accounted for by reusing this adder.

Notice that in both the design of the AU and the MU very conservative timing estimates were used. The only critical parts are stage one of the AU (alignment) and stage one of the MU (the carry-save adder tree).

The PM is to be realized as a 1Kx50 bit dynamic memory. The design is based on the standard single transistor dynamic memory cell. The memory is organized as 50 "planes" of 32x32 cells. The PM is addressed using a 4x10 bit program counter stack which allows convenient subroutine linkage between subroutines nested up to three deep. Refresh for the memory is achieved by cycle stealing every 16th instruction fetch (this has not been taken into account in the performance figure of the next section). The refresh address is kept in a 5 bit counter that is incremented every 16th M-cycle. The PM can be regarded as being a writable control store, i.e., programming the APAC is essentially done at the microcode level. The instruction format is shown in Fig. 3. There are two basic types of instructions distinguished by the leftmost two bits.

Type 1 control the AU and MU indicating which registers in the RF are the sources for their operands and which registers are the destinations for their results. Fields SA1 and SA2 indicate sources for the AU, and field DA indicates a destination for the AU's result. Similarly for the MU--see Fig. 3. Provision is made to specify a no-operation for the AU and/or the MU. Since both the AU and the MU are three stage pipelines and since it takes one M-cycle to move data to these units (see later), the destination field information is not needed by the control logic until four M-cycles after the source field information. To account for this both the

destination fields of the PMDR are piped through their own four stage delay lines before being decoded by the control logic. The leftmost two bits must also be piped through a four stage delay to allow the control logic to determine whether or not to ignore the output of the destination field delay lines. This technique for controlling pipelines is explained in more detail in Kogge[13].

Type 2 instructions control data transfers from the head of the IB FIFO to registers in the RF, as well as from the registers in the RF to the tail of the OB FIFO (specified by fields IB and OB in the format of Fig. 3). Type 2 instructions also handle branching. A 10 bit next address field (NA in the format of Fig. 3) is stacked on the PCS if the condition indicated by the CC field is met. Conditions include: IB full; OB full; result of add positive; result of add negative; result of add zero; always true, i.e. an unconditional branch. Detection of the conditions is performed by the condition code logic--CC in Fig. 3. To use the APAC efficiently type 2 instructions should be kept to a minimum.

Notice that the instruction format is very "horizontal" allowing concurrent operation of the AU and the MU to be specified. The job of taking advantage of this potential for concurrency is left entirely up to the user. This means that program preparation is quite complex if maximum use is to be made of the APAC. However, as stated in the introduction the APAC is intended for dedicated environments where it is likely to execute only a very small set of programs. The development of these programs should be considered as part of the overall system design. As noted earlier, this approach to program development is more in line with microcode development than standard program development.

The RF is a 256x32 bit static memory. The design is based on the standard six transistor static memory cell[14]. It is organized as 256 32 bit registers. The registers share a single 32 bit wide output bus and a single 32 wide input bus (see Fig. 2). The output bus connects the registers to the two AU inputs, to the two MU inputs, and to the tail of the output buffer, OB. During type 1 instructions the use of the output bus is multiplexed. Data is moved from the RF registers to the two AU inputs and the two MU inputs in four steps--one step per input. The complete transfer takes an M-cycle; each step takes 125 nS. The input bus connects the output of the AU, the output of the MU, and the head of the IB to the RF registers. As with the output bus, during type 1 instructions, the input bus is multiplexed. Data is moved from the AU output and the MU output in two steps, one step per output. The complete transfer takes an M-cycle; each step takes 250 nS. For data to make a round trip from a register through a function unit and back to a register takes five M-cycles.

The IB and the OB are 32 word FIFO buffers for input and output respectively. In the case of the IB, data can be added to the tail and removed from the head asynchronously, unless the buffer is full. Adding to the tail is under the control of an external clock which need not run synchronously with the chip timing. This requires a synchronizer circuit. Designing correctly operating synchronizers can be very involved; however, it need not be since the problem has been thoroughly studied in Stucki[15]. The operation of the OB is also asynchronous in a similar fashion.

Finally, the PM and the PCS can be loaded through the input port to allow the chip to function as an attached processor.

The next subsection presents the control equations and the dynamic model of a PUMA robot arm suitable for implementation in the APAC.

Dynamic Model of a Robot Arm

The dynamic equations of motion for a robot arm can be obtained from known physical laws (Newtonian mechanics) and physical measurements (link inertias and parameters). The actual derivation can be based on either a Lagrangian or Newtonian approach applied to open articulated chains represented in Denavit-Hartenberg matrix notation form. The equations of motion derived from the Lagrangian and Newtonian approach are briefly presented below.

Lagrange-Euler Formulation[16]

Applying the L-E equations of motion to the Lagrangian function of the robot arm yields the necessary generalized torque τ_i for joint i to drive the i^{th} link of the arm:

$$\tau_i = \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\vartheta}_i}\right) - \frac{\partial L}{\partial \vartheta_i} = \sum_{k=i}^{6}\sum_{j=1}^{k} Tr\left\{\frac{\partial T_0^k}{\partial \vartheta_j} J_k \left(\frac{\partial T_0^k}{\partial \vartheta_i}\right)^t\right\}\ddot{\vartheta}_j + \sum_{m=i}^{6}\sum_{j=1}^{m}\sum_{k=1}^{m} Tr\left\{\frac{\partial^2 T_0^m}{\partial \vartheta_j \partial \vartheta_k} J_m \left(\frac{\partial T_0^m}{\partial \vartheta_i}\right)^t\right\}\dot{\vartheta}_j\dot{\vartheta}_k - \sum_{j=i}^{6} m_j g \frac{\partial T_0^j}{\partial \vartheta_i}\bar{r}_j \quad ; \text{for } i=1,2,\ldots,6 \tag{1}$$

where Tr indicates the Trace operator, J_k is the inertial tensor expressed in the k^{th} coordinate frame, m_i is the mass of the i^{th} link and $\bar{r}_i$ is the position of the center of mass of link i.

Because of its matrix structure, this formulation is appealing from a control viewpoint in that it gives a set of closed form differential equations:

$$D(\vartheta)\ddot{\vartheta} + H(\vartheta,\dot{\vartheta}) + G(\vartheta) = \tau \tag{2}$$

This form allows one to design a control law that compensates for all the nonlinear effects easily. Computationally, however, the L-E formulation is extremely inefficient compared to the following formulation.

Newton-Euler Formulation[17]

The N-E equations of motion of a manipulator consist of a set of compact forward and backward recursive equations. They have significantly less operations than the L-E formulation. The formulation, based on a modification of Luh's approach[2,17], is presented below.

The forward recursive equations propagate linear velocity, linear acceleration, angular velocity, angular acceleration, total link forces and moments from the base to the end-effector of the manipulator. For manipulators having all the rotary joints, these equations are:

$$\omega_i = R_i^{i-1}(\omega_{i-1} + \dot{\vartheta}_i z_{i-1}) \tag{3}$$

$$\alpha_i = R_i^{i-1}(\alpha_{i-1} + \omega_{i-1} \times \dot{\vartheta}_i z_{i-1} + \ddot{\vartheta}_i z_{i-1}) \tag{4}$$

$$a_i = \omega_i \times (\omega_i \times r_i) + \alpha_i \times r_i + R_i^{i-1} a_{i-1} \tag{5}$$

$$\bar{a}_i = \omega_i \times (\omega_i \times \bar{r}_i) + \alpha_i \times \bar{r}_i + a_i \tag{6}$$

The backward recursive equations of motion propagate, from the end-effector to the base of the manipulator, the forces and moments exerted on link i by link i-1, as follows:

$$f_i = m_i\bar{a}_i + R_i^{i+1} f_{i+1} = F_i + R_i^{i+1} f_{i+1} \tag{7}$$

$$n_i = I_i\alpha_i + \omega_i \times (I_i\omega_i) + m_i(\bar{r}_i + r_i) \times \bar{a}_i + r_i \times R_i^{i+1} f_{i+1} + R_i^{i+1} n_{i+1} \quad (8)$$

$$\tau_i = (R_i^{i-1} z_{i-1})^t n_i \quad (9)$$

where the ϑ_i's, $\dot{\vartheta}_i$'s, and $\ddot{\vartheta}_i$'s are the relative angles, velocities, and accelerations between link i-1 and link i for i = 1,...,6; and

- ω_i = the angular velocity of link i with respect to the i^{th} coordinate system;
- α_i = the angular acceleration of link i with respect to the i^{th} coordinate system;
- r_i = the origin of the i^{th} frame with respect to the $i-1^{th}$ frame;
- $\bar{r}_i$ = the center of mass of link i with respect to the i^{th} frame;
- a_i = the linear acceleration of link i with respect to the i^{th} coordinate system;
- $\bar{a}_i$ = the linear acceleration of the center of mass of link i with respect to the i^{th} coordinate system;
- I_i = the inertia about center of mass of link i with respect to the i^{th} coordinate system;
- F_i = the total external force exerted on link i with respect to the i^{th} coordinate system;
- N_i = the total moment exerted on link i with respect to the i^{th} coordinate system;
- f_i = the force exerted on link i by link i-1 with respect to the i^{th} coordinate system;
- n_i = the moment exerted on link i by link i-1 with respect to the i^{th} coordinate system;
- τ_i = the torque exerted on link i.

Given the equations of motion of a manipulator as in Eq. 2 (L-E formulation) or Eqs. 3-9 (N-E formulation), the control problem is to find appropriate torques/forces to servo all the joints of the manipulator in real-time to track a desired position trajectory as closely as possible. One of the basic control schemes is the computed torque technique based on the L-E[1] or the N-E equations of motion[18]. Paul[1] concluded that closed loop digital control is impossible if the complete L-E equations of motion are used. It requires 2,000 floating point multiplications and 1,500 floating point additions to compute all the joint torques per set point for a Stanford arm. Lee[18] applied the computed torque technique to the N-E equations of motion and derived an efficient control law in the joint space to servo a PUMA robot arm. The control law is computed iteratively using the N-E equations of motion. Using a conventional uniprocessor computer such as a PDP-11/45, the feedback control equation can be computed within 3 ms if all the complex trigonometric functions are implemented as table look-up. If the APAC is used in place of a PDP-11/45, not only can the computation be speeded up (see next section), but even more complex models can be considered in which friction and backlash can be accounted for.

The computed torque technique assumes that one can accurately compute the counterparts of $D(\vartheta)$, $H(\vartheta, \dot{\vartheta})$, and $G(\vartheta)$ in Eq. 2 to minimize their nonlinear effects, and use a position plus derivative control to servo the joints[1]. Thus, the structure of the control law has the form:

$$\tau = D_a(\vartheta)\left[\ddot{\vartheta}^d + K_v(\dot{\vartheta}^d - \dot{\vartheta}) + K_p(\vartheta^d - \vartheta)\right] + H_a(\vartheta, \dot{\vartheta}) + G_a(\vartheta) \quad (10)$$

where K_v is a 6x6 velocity feedback gain matrix, K_p is a 6x6 position feedback gain matrix, $D_a(\vartheta)$, $H_a(\vartheta, \dot{\vartheta})$ and $G_a(\vartheta)$ are the counterparts of $D(\vartheta)$, $H(\vartheta, \dot{\vartheta})$ and $G(\vartheta)$ respectively in Eq. 2.

Substituting the τ from Eq. 10 into Eq. 2, we have:

$$D(\vartheta)\ddot{\vartheta} + H(\vartheta, \dot{\vartheta}) + G(\vartheta) = D_a(\vartheta)\left[\ddot{\vartheta}^d + K_v(\dot{\vartheta}^d - \dot{\vartheta}) + K_p(\vartheta^d - \vartheta)\right] + H_a(\vartheta, \dot{\vartheta}) + G_a(\vartheta) \quad (11)$$

If $D_a(\vartheta)$, $H_a(\vartheta, \dot{\vartheta})$, $G_a(\vartheta)$ are equal to $D(\vartheta)$, $H(\vartheta, \dot{\vartheta})$, $G(\vartheta)$ respectively, then Eq. 11 reduces to:

$$D(\vartheta)\left[\ddot{e} + K_v\dot{e} + K_p e\right] = 0 \quad (12)$$

Since $D(\vartheta)$ is always non-singular, and if (i) K_v is a symmetric non-negative definite matrix, (ii) K_p is a symmetric positive definite matrix, and (iii) the rank of

$$\left[K_v \mid K_pK_v \mid \cdots \mid K_p^{n-1}K_v\right] = n, \text{ then } \lim_{t\to\infty} e(t) \to 0$$

The analogous control law derived from the computed torque technique based on Eqs. 3-9 can be obtained by substituting $\ddot{\vartheta}_i$ in these equations with:

$$\ddot{\vartheta}_i^d + \sum_{s=1}^{n} K_v^{is}(\dot{\vartheta}_s^d - \dot{\vartheta}_s) + \sum_{s=1}^{n} K_p^{is}(\vartheta_s^d - \vartheta_s)$$

or

$$\ddot{\vartheta}_i^d + \sum_{s=1}^{n} K_v^{is}\dot{e}_s + \sum_{s=1}^{n} K_p^{is}e_s \quad (13)$$

where K_v^{is} and K_p^{is} are the derivative and position feedback gains for joint i respectively. The physical interpretation of putting Eq. 13 into the N-E recursive equations can be viewed as follows:

(1) The first term will generate the desired torque for each joint if there is no modeling error and the system parameters are known. However, there are errors due to backlash, gear friction, uncertainty about the inertia parameters, and time delay in the servo loop so that deviation from the desired joint trajectory will be inevitable.

(2) The remaining terms, in the N-E equations of motion, will generate the correction torque to compensate for small deviations from the desired joint trajectory.

(3) The control law is a position plus derivative control and has the effect of compensating the inertial loading, coupling effects, and the gravity loading of the links.

Computational complexity of the control equations as in Eq. 13 is tabulated in Table 1. Using the APAC, Eq. 13 and Eqs. 3-9 can be computed in 1 ms. Since an industrial robot is a highly nonlinear system, care must be exercised in choosing the feedback gains in the control equations. In order to achieve a "critically damped" system for each joint subsystem (which in turn loosely implies that the whole system behaves as a "critically-damped" system), the position feedback gain matrix K_p and the velocity feedback gain matrix K_v can be chosen as in Paul[19].

Simulation of Gross Motion for the PUMA Arm

To illustrate the effectiveness of the APAC a functional simulation was performed using APL. The details are discussed in [3]. The forward and backward recursive equations for computing the actuator torques were used as a benchmark, since they are the major computational task in the proposed arm control strategy. These were programmed for the APAC. A listing of the program showing how the function units can be efficiently scheduled can be found in [3]. The APAC is operating at its maximum rate when both function units are in streaming mode. In this mode it is producing the results of two floating-point operations every M-cycle, i.e., it is operating at a rate of 4 MFLOPS. Our simulation showed that about 73% of the time the function units produced

results, i.e. the APAC was operating at an average of 2.93 MFLOPS for this benchmark. This corresponds to a torque computation (i.e., actuator signals for all six joint motors) in about 250 μs. To achieve this considerable time was spent hand optimizing the program. Scheduling two pipelined function units is time consuming. Support software to help with this aspect of program preparation would be a necessity in a production environment.

Conclusion

This paper has presented a proposal for a hierarchical control structure that uses a special purpose attached processors--the proposed APAC. We have argued that this approach will allow the real-time control of an industrial robot, in particular a PUMA robot. The encouraging performance figures cited above support this argument.

References

1. Paul, R. "Modeling, Trajectory Calculation, and Servoing of a Computer Controlled Arm," Stanford Artificial Intelligence Laboratory Memo AM-177, November 1972.
2. Turney, J., Mudge, T. N., Lee, C. S. G., "Equivalence of Two Formulations for Robot Arm Dynamics," SEL Report 142, ECE Department, University of Michigan, December 1980.
3. Turney, J., Mudge, T. N., "VLSI Implementation of a Numerical Processor for Robotics," *Proceedings of the 27-th International Instrumentation Symposium*, Indianapolis, Indiana, April 1981, pp. 169-175.
4. Whitney, D. E., "Resolved Motion Rate Control of Manipulators and Human Prostheses," *IEEE Transaction on Man-Machine Systems*, Vol. MMS-10, no. 2, June 1969, pp 47-53.
5. Albus, J. S., "A New Approach to Manipulator Control: The Cerebellar Model Articulation Controller (CMAC)," Journal of Dynamic Systems, Measurement and Control, Transaction ASME, Series G, Vol. 97, No. 3, Sept. 1975.
6. Kahn, M. E., Roth, B., "The Near-Minimum-Time Control of Open-Loop Articulated Kinematic Chains," *Transaction of the ASME, Journal of Dynamic Systems, Measurement, and Control*, September 1971, pp 164-172.
7. Saridis, G. N., Lee, C. S. G., "An Approximation Theory of Optimal Control for Trainable Manipulators," *IEEE Transaction on Systems, Man and Cybernetics*, Vol. SMC-9, no. 3, March 1979, pp 152-159.
8. Dubowsky, S., DesForges, D.T., "The Application of Model Referenced Adaptive Control to Robotic Manipulators," *Transaction of the ASME, Journal of Dynamic Systems, Measurement and Control*, Vol. 101, Sep., 1979, pp 193-199
9. Palmer, J., "The Intel 8087 Numeric Data Processor," *Proc. 7th Annual Symposium on Computer Architecture*, La Baule, France, May 1980, pp. 174-181.
10. Mead, C. A., and L. A. Conway, *Introduction to VLSI Systems*, Addison-Wesley, 1980.
11. Patterson, D. A., and C. H. Sequin, "Design Considerations for Single-Chip Computers of the Future," *IEEE Trans. Computers*, Vol. C-29, No. 2, Feb. 1980, pp. 108-116.
12. Coonen, J. T., "An Implementation Guide to a Proposed Standard for Floating-Point Arithmetic," *Computer Magazine*, Jan. 1980, pp. 68-79.
13. Kogge, P. M., "The Microprogramming of Pipelined Processor," *Proc. of the 4th Annual Symposium on Computer Architecture*, March 1977, pp. 63-69.
14. Ohzone, T., J. Yasui, T. Ishihara, and S. Horiuchi, "An 8Kx8 Bit Static MOS RAM Fabricated by n-MOS/n-Well CMOS Technology," *IEEE J. Solid-State Circuits*, Vol. SC-15, No. 5, Oct. 1980, pp. 854-861.
15. Stucki, M. J., and J. R. Cox, "Synchronization Strategies," *Proc. Caltech Conf. on VLSI*, Jan. 1979.
16. Bejczy, A. K., "Robot Arm Dynamics and Control," Technical Memo 33-669. Jet Propulsion Laboratory, February 1974.
17. Luh, J. Y. S., M. W. Walker, and R. P. C. Paul, "On-Line Computational Scheme for Mechanical Manipulators," *Trans. ASME: Journal of Dynamic Systems, Measurement, and Control*, Vol. 120, June 1980 pp. 69-76.
18. Lee,C. S. G., Chung, M. J., Mudge, T. N., and Turney, J. L., "On the Control of Mechanical Manipulators," *6 th IFAC Symposium on Identification and System Parameter Estimation*, June 7-11, 1982, Washington, D.C.
19. Paul, R., Robot Manipulators, MIT Press, 1981.

Controller based on N-E Equations of Motion	Multiplications	Additions
ω_i	$9n$	$7n$
α_i	$9n$	$9n$
$\mathbf{a}_i$	$27n$	$22n$
$\bar{\mathbf{a}}_i$	$15n$	$14n$
$\mathbf{F}_i$	$3n$	0
$\mathbf{f}_i$	$9(n-1)$	$9n-6$
$\mathbf{N}_i$	$24n$	$18n$
$\mathbf{n}_i$	$21n-15$	$24n-15$
$\ddot{\vartheta}_i^d + K_v\dot{e}_i + K_p e_i$	$2n$	$4n$
Total Math. Operations	$119n-24$	$107n-21$

Table 1

Breakdown of Mathematical Operations of the Controller Based on the Newton-Euler Formulation

where n = number of degree-of-freedom of the robot arms

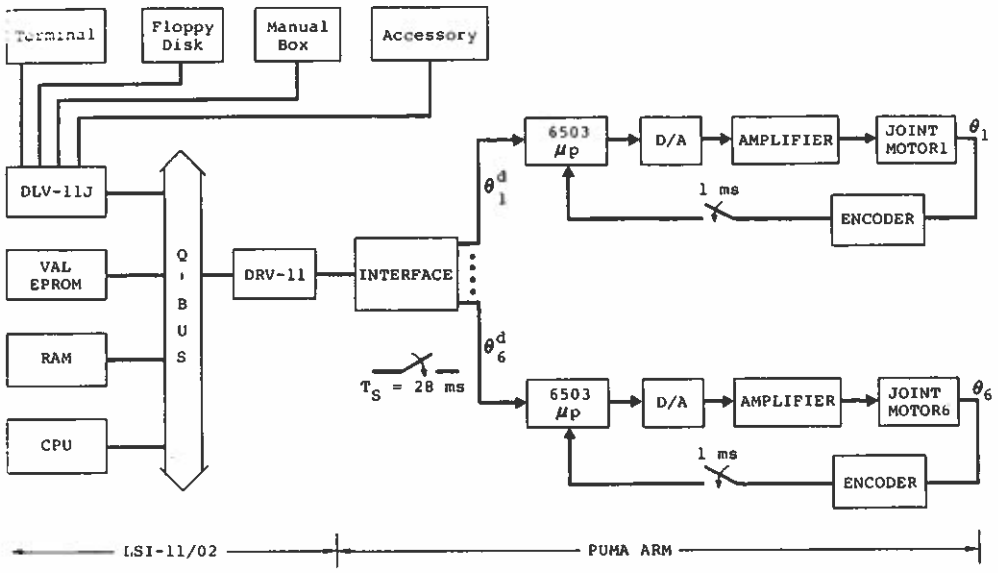

Figure 1. PUMA Robot Arm Control Structure Diagram.

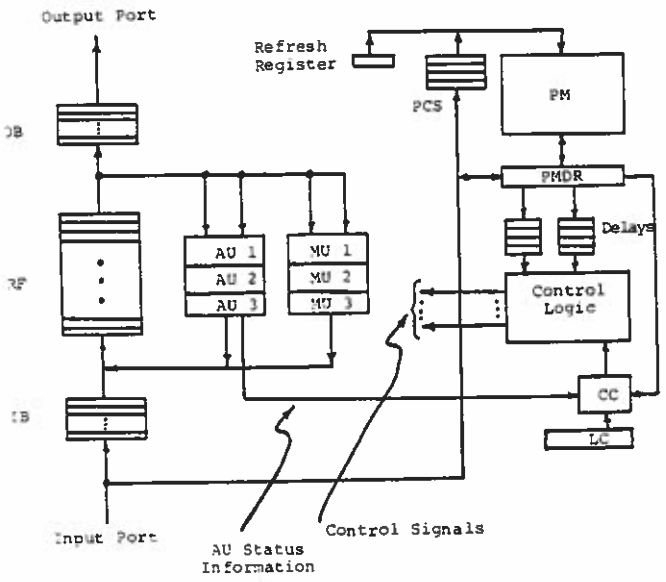

Figure 2. Data Path Block Diagram of the Proposed APAC.

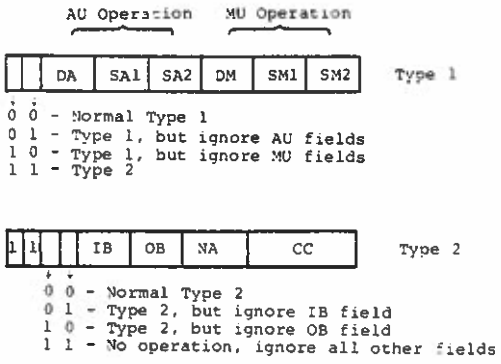

Figure 3. Instruction Formats.