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 system 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. However, a direct Newton-Euler (N-E) formulation coupled with an appropriate special purpose processor has been suggested as a solution.

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 alphabetics while matrices are in boldface upper case alphabetics.

Current Control Methods

As noted above, given the equations of motion of a manipulator, the control problem is to find appropriate torques/forces to serve 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, (ii) The Cerebellar Model Articulation Controller, (iii) The near-minimum-time control, (iv) The suboptimal control, and (v) The model reference adaptive control. 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...
VAL\(^1\) commands, and (ii) subtask coordination with the six 6503 microprocessors to carry out the command. The online 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 \(\theta_1, \theta_2, \cdots, \theta_6\) 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) acknowledgment 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 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 proscribed. 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 servicing. 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' AP1208, a high performance numerically oriented attached processor, and the Intel 8087, 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 AP1208

---

\(^1\) VAL is a registered trademark 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
closer word size (32 bits versus 38 bits). It differs from
the 8086 by having its own chip program memory,
input/output buffers to facilitate real-time applications,
and two independent function units. However, the 8086 has a
more flexible number format, and can deal with several vari-
ants of the IEE 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 ISL (Isoplanar Integrated
Injection Logic) technology developed by Fairchild Corpora-
tion. 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 265x32 register file (RF).
4. A 32x32 bit input buffer (IB).
5. A 32x32 bit output buffer (OB).
6. A 1kx60 bit program memory (PM).
7. A 4x10 bit program counter stack (PCS).
8. A 1x50 bit program memory data register (PMDR).
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 Sodano and Conway, indicates
that 60% 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, lof the number of active devices
required by the chip indicates that 90% will be con-
tained 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 5K 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 were
anticipated.

The floating point number format used in the design
study is the 32 bit proposed IEEE standard described in
Coenen. Both the AU and the MU were designed to handle
this format. However, the rounding modes, rounding preci-
sion 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 perform-
ing 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 fol-
lowed by a 6 (log 2 24) 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 per-
formed 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 (pro-
gram logic arrays) 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 frac-
tion, and the final stage performing normalization, truncation
to 24 bits, and exponent addition. The design of the multi-
plier 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 5 (log 2 24)
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 gen-
erate signals which feed another level of lookahead units.
This process is continued in the standard fashion to produce
a lookahead tree of height 3 (log 2 48). The total time
to add is thus (3x2)x60x150 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 one position right shift if numbers are
represented in the format above. The final stage in stage
carry-exponent addition—is performed using a simple 8 bit
ripple carry adder. The delay for 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 MP is to be realized as a 1kx60 bit dynamic memory.
The design is based on the standard single
transistor dynamic memory cell. The memory is organized as
60 '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 addresses are kept in a 5 bit counter that is incre-
mented 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 for-
mat is shown in Fig. 3. There are two basic types of
instructions distinguished by the least significant two bits.

Type 1 control the AU and MU indicating which regis-
ters 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 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.\textsuperscript{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 instruc-
tions also handle branching. A 10 bit next address field (NA
in the format of Fig. 3) is stacked on the RCS if the con-
dition indicated by the CC Field is met. Conditions include: IB 
full; OB full; result of add positive; result of add negative; 
result of add; 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 MI 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 intro-
duction 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 micro-
code development than standard program development.

The RF is a 256x32 bit static memory. The design is 
based on the standard 64 transistor static memory cell.\textsuperscript{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 MI 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 MI 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 MI, 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 MI output in two steps, 
one step per output. The complete transfer takes an M-
cycle; each step takes 250 nS. Data for a 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 syn-
chronously 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.\textsuperscript{16} 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 pro-
cessor.

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 Danavit-Hartenberg matrix notation form. 
The equations of motion derived from the Lagrangian and 
Newtonian approach are briefly presented below.

Lagrange-Euler Formulation\textsuperscript{16} 

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

$$\tau_i = \frac{d}{dt} \left( \frac{\partial L}{\partial \dot{q}_i} \right) - \sum_{j=1}^{n} \sum_{m=1}^{m} Tr \left\{ \frac{\partial T_j^m}{\partial \theta_j} \int \left( \frac{\partial T_j^m}{\partial \theta_j} \right) \right\} \dot{q}_j$$

$$+ \sum_{m=1}^{m} \sum_{j=1}^{n} Tr \left( \frac{\partial T_j^m}{\partial \theta_j} \int \frac{\partial T_j^m}{\partial \theta_j} \right) \dot{q}_j$$

$$- \sum_{j=1}^{n} \sum_{m=1}^{m} T_j^m \dot{q}_j ; \text{ for } i=1,2,...,n$$

where $Tr$ indicates the Trace operator, $\theta_i$ is the inertial 
tensor expressed in the $i^{th}$ coordinate frame, $m_i$ is the mass 
of the $i^{th}$ link and $f_j$ is the position of the center of mass 
of link $j$.

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

$$D(q)\ddot{q} + H(q,\dot{q}) + G(q) = \tau$$

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

Newton-Euler Formulation\textsuperscript{17} 

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

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

$$\omega_l = R_l^{-1}(\omega_{l-1} + \dot{\alpha}_{l-1})$$

$$a_l = R_l^{-1}(a_{l-1} + c_{l-1} \times \ddot{\alpha}_{l-1} + \dddot{\alpha}_{l-1})$$

$$m_l = c_l \times (\omega_l \times f_l) + a_l \times f_l + a_l$$

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:

$$f_i = m_i \dot{r}_i + R_i^T \dot{f}_{i+1} = F_i + R_i^T f_{i+1}$$
\[ n_i = l_i \alpha_i + \omega_i \times (l_i \omega_i) + m_i (\dot{r}_i + \tau_i) \times \bar{r}_i + R_i^{11} \dot{r}_{i+1} + R_i^{11} \dot{q}_{i+1} \]  
\[ \tau_i = (R_i^{-1} \dot{r}_i + \dot{\theta}_i) n_i \]  

where the \( \dot{\theta}_i \)'s, \( \dot{\omega}_i \)'s, and \( \ddot{\omega}_i \)'s are the relative angles, velocities, and accelerations between link \( i-1 \) and link \( i \) for \( i = 1, \ldots, 6 \); and  
\( \omega_i \) = the angular velocity of link \( i \) with respect to the \( i \)-th coordinate system;  
\( \alpha_i \) = the angular acceleration of link \( i \) with respect to the \( i \)-th coordinate system;  
\( \bar{r}_i \) = the center of the \( i \)-th mass relative to the \( i \)-th coordinate system;  
\( a_i \) = the linear acceleration of link \( i \) in the \( i \)-th coordinate system;  
\( \ddot{a}_i \) = the linear acceleration of the center of mass of link \( i \) with respect to the \( i \)-th coordinate system;  
\( l_i \) = 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;  
\( M_i \) = the total moment exerted on link \( i \) with respect to the \( i \)-th coordinate system;  
\( \tau_i \) = 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 serve 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 formulation or the N-E equations of motion. Paul 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 second for a Stanford arm. Lee 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. Thecontrol 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 as all the complex trigonometric functions are implemented as table look-ups. 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 \( D(q), H(q, \dot{q}), \) and \( G(q) \) in Eq. 2 to minimize their nonlinear effects, and use a position plus derivative control to serve the joints. Thus, the structure of the control law has the form:

\[ \tau = D(q) \ddot{q} + K_p (\dot{q} - \dot{q}^d) + K_v (q - q^d) \]  
\[ + H(q, \dot{q}) \ddot{q} + G(q) \]  

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

Substituting the \( \tau \) from Eq. 10 into Eq. 2, we have:

\[ D(q) \ddot{q} + H(q, \dot{q}) \ddot{q} + G(q) = \]

\[ D(q) \ddot{q} + K_p (\dot{q} - \dot{q}^d) + K_v (q - q^d) \]

If \( D(q), H(q, \dot{q}), \) and \( G(q) \) are equal to \( D(q), H(q, \dot{q}), \) and \( G(q) \) respectively, then Eq. 11 reduces to:

\[ D(q) \ddot{q} + K_p \dot{q} + K_v q = 0 \]  

Since \( D(q) \) 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 \( [K_v, K_p K_v, \ldots, K_p^{n-1} K_v] = n \), then \( \lim_{t \to \infty} \dot{q}(t) = 0 \).

The analogous control law derived from the computed torque technique based on Eqs. 3-9 can be obtained by substituting \( \dot{q} \) in these equations with:

\[ \ddot{q} = \sum_{s=1}^{n} K_p (\ddot{q}_s - \ddot{q}_s) \]

or

\[ \ddot{q} = \sum_{s=1}^{n} K_p \ddot{q}_s + \sum_{s=1}^{n} K_v \dot{q}_s \]

where \( K_p \) and \( K_v \) 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 \( K_v \) and the velocity feedback gain matrix \( K_p \) can be chosen as in Paul.

**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


<table>
<thead>
<tr>
<th>Controller based on N-E Equations of Motion</th>
<th>Multiplications</th>
<th>Additions</th>
</tr>
</thead>
<tbody>
<tr>
<td>\omega_i</td>
<td>9n</td>
<td>7n</td>
</tr>
<tr>
<td>\omega_i</td>
<td>9n</td>
<td>9n</td>
</tr>
<tr>
<td>\omega_i</td>
<td>27n</td>
<td>22n</td>
</tr>
<tr>
<td>\omega_i</td>
<td>15n</td>
<td>14n</td>
</tr>
<tr>
<td>F_i</td>
<td>3n</td>
<td>0</td>
</tr>
<tr>
<td>f_i</td>
<td>9(n−1)</td>
<td>9n−6</td>
</tr>
<tr>
<td>N_i</td>
<td>24n</td>
<td>18n</td>
</tr>
<tr>
<td>\sum (n_e f_i + K_e \omega_i + K_p \omega_i)</td>
<td>2n</td>
<td>4n</td>
</tr>
<tr>
<td>Total Math Operations</td>
<td>119n−24</td>
<td>107n−21</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>DA</th>
<th>SA1</th>
<th>SA2</th>
<th>DM</th>
<th>SM1</th>
<th>SM2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>Normal Type 1</td>
<td>0 1</td>
<td>Type 1, but ignore AU fields</td>
<td>1 0</td>
<td>Type 1, but ignore MU fields</td>
</tr>
<tr>
<td>1 1</td>
<td>Type 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 3. Instruction Formats.

<table>
<thead>
<tr>
<th>IB</th>
<th>OB</th>
<th>SA</th>
<th>CC</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>Normal Type 2</td>
<td>0 1</td>
<td>Type 2, but ignore IB field</td>
</tr>
<tr>
<td>1 0</td>
<td>Type 2, but ignore OB field</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 1</td>
<td>No operation, ignore all other fields</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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