
Monograph:
Ziauddin, S.M. and Zalzala, A.M.S. (1994) Distributed Real-Time Adaptive Control of Mechanical Arms with Practical Implementation. Research Report. ACSE Research Report 494 . Department of Automatic Control and Systems Engineering

---

DISTRIBUTED REAL-TIME ADAPTIVE CONTROL OF MECHANICAL ARMS WITH PRACTICAL IMPLEMENTATION

S.M. Ziauddin and A.M.S. Zalzala

Robotics Research Group,
Department of Automatic Control and Systems Engineering,
University of Sheffield,
P.O.Box 600, Mappin Street, Sheffield S1 4DU, United Kingdom
Email:rrg@sheffield.ac.uk

Research Report #494
January 1994

Abstract

Adaptive controllers employing the highly coupled and non-linear dynamics of robot manipulators are computationally complex, which present a major obstacle in their real time implementation for industrial applications. This paper describes a solution to this problem by employing a parallel processing approach. The parallelism inherent in the adaptive controllers is exploited to obtain an efficient implementation and reduce the overall computation time to within the limit acceptable for real time control. The distributed globally stable adaptive controller is implemented on a network of T-800 transputers for the six joint PUMA 560 arm.

Keywords: Adaptive Control, Automation, Parallel processing, Robots.

1. Introduction

Present day commercial robot manipulators are equipped with simple single joint PID controllers based on the assumption that the highly coupled and non-linear dynamics of the manipulators can be approximated by a linear model. These simple control schemes degrade the performance of the manipulators especially when large variations in payloads or tasks are encountered. For this reason as well as the fact that model parameters of existing manipulators are not known exactly, more efficient control schemes, such as adaptive control, are required. Advanced adaptive controllers compute the motor commands via the full non-linear model of the manipulator with the significant advantage of providing for global stability. However, the practical implementation of such control schemes in real time has always been difficult due to their inherent computational complexity. In order to implement these controllers for a full six joint arm and have it in operation on-line it is essential to employ parallel and distributed processing techniques to facilitate the computations for real-time applications.

This paper presents a distributed globally stable adaptive controller for the six joint PUMA 560. Scheduling of tasks onto processors has been done using DF/IHS (depth first implicit heuristic search) algorithm[4, 6]. In addition the formulation is mapped on a network of transputers thus providing a practical multi-processor adaptive controller. Computation time of 9.3 m.sec. has been achieved for the six link arm which is well within the maximum limit of 16 m.sec. imposed by the mechanical resonance frequency of manipulators.

The paper starts with an overview of adaptive controllers for robots. The next section discusses application of parallel processing in robot control. Next, implementation issues, decomposition of the adaptive controller into sub-parts and its scheduling is discussed, hence describing the complete distributed system. Finally, results of practical implementation on a network of transputers are presented, and conclusions are drawn.

2. Adaptive robot control

Several approaches to adaptive controller design for manipulators exist in literature. The approach of initial efforts has been to model the manipulator as a linear system and to apply existing adaptive control methods [1]. The major assumption here is that good results can be achieved if the model parameters being identified are not changing rapidly. Stability has always been an issue in this approach and no proof has ever been given. Another approach to adaptive control of a manipulator has been to base the control on the full non-linear model of the manipulator [10, 13]. The significant advantage here is the ability of proving global stability. A famous example is the adaptive controller by Slotine and Li [10] in which [illegible]

Lyapunov function is used to prove stability by showing that the output errors converge to a sliding surface, which in turn implies that the tracking errors converge to zero. These Lagrangian based controllers can be broken down into sub-tasks rather easily and appear to be good candidates for parallel processing. However, the amount of computation required for a six joint arm is too much for economical implementations. Another class is that of recursive adaptive controllers [12, 17]. They have the same convergence properties as their Lagrangian counterparts. Their structure is similar to the recursive structure of the Newton Euler equations for the inverse dynamics of manipulators. This makes it difficult to map them onto a network of processors but because of their lesser computational complexity they are a better choice for parallel processing and have been used in our work. The recursive version of the adaptive controller by Slotine and Li is given below:

Initialise: $w(0) = -g$

Upward:

$$v(k) = v(k-1) + d(k)q^{\cdot}(k)$$
$$w(k) = w(k-1) + d(k)q_r^{\cdot}(k)$$
$$w^{\cdot}(k) = w^{\cdot}(k-1) + d(k)q_r^{\cdot\cdot}(k) + v(k-1) \times d(k)q_r^{\cdot}(k)$$
$$e(k) = v(k) - w(k)$$

Downward:

$$f_k^i = .5v(k) \times R_i w(k) + .5w(k) \times R_i v(k) + .5R_i w(k) \times v(k) + R_i w^{\cdot}(k)$$
$$F(k) = F(k+1) + \sum_{i=1}^{10} f_k^i a_k^i$$
$$\tau(k) = d'(k)F(k) - K_d s_k$$
$$a_k^{i\cdot} = -P_k^i e_k^{\cdot} f_k^i$$

The different terms of the controller are defined below. All the components are with respect to base link co-ordinates.

$v(k)$	Spatial velocity of link k.
$w(k)$	Spatial reference velocity of link k.
$w^{\cdot}(k)$	Spatial reference acceleration of link k.
f_k^i	Local force component of link k calculated from R_i the ith placement matrix.
$F(k)$	Summed force of kth link and above.
$\tau(k)$	Required torque at kth joint.
K_d, P	Strictly positive definite gain matrices.
a_k^i, R_i	They are defined as $I_k = \sum_{i=1}^{10} R_i a_k^i$ where I_k = the 6×6 inertia matrix of link k. R_i is the 6×6 sparse placement matrices with ones at the places of inertia matrix elements and zeros elsewhere.
$a_k^i \in \{J_{xx}, J_{yy}, J_{zz}, J_{xy}, J_{xz}, J_{yz}, p_x, p_y, p_z, m\}$	i. e. the set of inertia elements of kth link and $p_i = mr_i$ where m is the mass of link k and r_i vector from the link co-ordinates to the centre of mass of the link.
$d(k)$	Spatial vector representing the joint axis.
$q, q^{\cdot}, q^{\cdot\cdot}$	Joint angles and their derivatives.
$q_d, q_d^{\cdot}, q_d^{\cdot\cdot}$	Desired joint angles and their derivatives.

$\dot{q}_r$ $= \dot{q}_d - \Lambda(q - q_d)$ where Λ is a positive definite gain matrix.

s $= \dot{q} - \dot{q}_r$

g Spatial gravity vector

3. Parallel processing in robot control

Most of the research efforts of implementing robot control on parallel processors have been limited to inverse model calculation using Newton Euler (NE) equations. There have been two approaches to this. The former is the hardware approach which develops a dedicated hardware architecture as fits a computational flow of the NE equations. The latter is the software approach that develops scheduling algorithms to assign NE equations onto parallel processors in an arbitrary architecture. As an example of the former approach Nigam and Lee [5] proposed a pipelined structure which is not effective for a robot having six or less DOF due to the initial pipe delay. Lathrop [3] proposed two schemes for NE and Lagrange Euler (LE) equations both of which require special purpose VLSI chips. Lee, Mudge and Turney [5] proposed a special purpose processor which functions as an attached processor of a general purpose computer for computing joint torques. As example of the latter approach Kasahara [6,15], achieved a time of 8.4 m.sec. for NE equations on three 8086/8087 processor pairs. Hashimoto and Ohashi[14,16] implemented their resolved NE equations on four transputers to achieve a time of .66 msec. for a three DOF arm. In adaptive control application Sinha and Ho [18] achieved a time of 6.4 msec. for a single joint adaptive control algorithm which assumes a linear robot model. Chung and Daniel [21] achieved update rates of 780 Hz, 390 Hz and 195 Hz for servo, inertia and link parameters respectively for a three DOF arm. No application for any globally stable adaptive controller for a full six DOF arm has been reported. The need for an efficient and economical parallel processing scheme for adaptive control for an industrial manipulator has always been there.

4. The distributed system

4.1 Implementation Issues

A key simplifying aspect to the derivation of the controller is the use of the spatial vector notation of Featherstone [11] which allows translational and rotational quantities to be combined in a single six dimensional vector. This notation considerably reduces the complexity and number of equations of the resulting controller. Several practical considerations can greatly influence the actual efficiency of the algorithm. Using the Denavit-Hartenberg conventions the following choice of reference frames minimises the number of frame transportations: Velocities, accelerations, inertias and local force components should be expressed in the link's own frame of reference. Joint axes and summed forces should be expressed with respect to the frame beneath them. These transformations have been carried out and the adaptive controller in its final shape is given below.

Initialise: $w(0) = -g$

Upwards:

$$v_k(k) = X_k^{k-1}(v_{k-1}(k-1) + d_0\dot{q}(k))$$
$$w_k(k) = X_k^{k-1}(w_{k-1}(k-1) + d_0\dot{q}_r(k))$$
$$\dot{w}_k(k) = X_k^{k-1}(\dot{w}_{k-1}(k-1) + d_0\dot{q}_r(k) + v_{k-1}(k-1) \times d_0\dot{q}_r(k))$$
$$e_k(k) = v_k(k) - w_k(k)$$

Downwards: $f_k^i = .5v_k(k) \times R_i w_k(k) + .5w_k(k) \times R_i v_k(k) + .5R_i w_k(k) \times v_k(k) + R_i \dot{w}_k(k)$

$$F_k(k) = X_k^{K+1} F_{k+1}(k+1) + \sum_{i=1}^{10} f_k^i a_k^i$$

$$\tau(k) = d_0^{'} X_{k-1}^k F_k(k) - K_d(k) s(k)$$

$$a_k^{i\cdot} = -P_k^i e_k^{'}(k) f_k^i$$

In these equations the subscript k is to show that the spatial vector representing velocity, or acceleration of any link k or the force on any link k is referred to its own link co-ordinates. X_j^i is the 6×6 spatial transformation matrix which transforms the spatial vectors from link i co-ordinates to link j co-ordinates and $d_0 = [001000]'$, the superscript dash indicates spatial transpose operation and the operator $\times$ represents spatial cross product. Note that the joint k axis $d(k)$ is a constant vector d_0 in k-1 co-ordinates. Having made these transformations the next step was to customise the algorithm for PUMA 560 by putting the values of the appropriate kinamatic parameters and the inertial parameters [7, 8]. Also as the 6×6 matrix multiplications are not at all an efficient way of computing the equations, particularly when there are quite a number of sparse matrices involved, the whole algorithm was converted into symbolic form at equation level. For this purpose the symbolic mathematics package MAPLE was used. In addition to simplification through the use of MAPLE an important advantage gained was that dividing the algorithm into sub tasks became relatively easy as the grain size could now be easily altered. Two different grain sizes have been used in the scheduling as can be seen in the following section.

4.2 Adaptive controller decomposition

The recursive adaptive controller under consideration consists of a set of recursive equations employing spatial algebra [11] in which the linear and angular components of motion and force are combined into one six dimensional vector. The controller computes the motor torques for each joint and also updates the inertia parameters of each link. These two processes may be carried out in parallel.

Parallelism may be achieved if we divide the whole process into sub tasks that represent one of the following :

- Such basic operations as addition, subtraction, multiplication and division.
- Calculations associated with independent equations.
- An intermediate arrangement of one and two.

In the task generation process it appears that maximum parallelism is achieved by forming sub-tasks which represent such small fundamental operations as additions and multiplications and mapping these onto a set of processors. However, this would produce tasks that number in thousands. This is undesirable from the point of view of scheduling complexity and communication overhead. The other alternative is to form equation level tasks. The main bottleneck here is the computation of the local force components f_k^i. They represent a set of ten equations for each joint involving spatial operations. The above facts illustrate that the task sizes in the first two cases are not suitable for efficient parallel processing and an intermediate course of action has to be taken. In other words there has to be a trade off between parallelism and communication overhead. This was made possible by the conversion of the algorithm to symbolic form.

Considering the amount of communications involved two different grain sizes are being used in the scheduling process, one having an average task size of 438 μ sec. and the other having an average task size of 256 μ sec. i.e., roughly half of the previous one. The number of sub-

tasks for the two cases being 38 and 65 respectively. Task times have been obtained by running each task on the target transputer and are thus very reliable. The schedules obtained using these times are therefore expected to be very close to the real implementation except for the fact that communications among processors would not be taken into account. The effect of communications will be evident in the final section when the practical implementation results are given.

4.3 Scheduling

The problem of scheduling a controller algorithm onto a set of processors is deterministic in the sense that all information governing the scheduling decisions is known in advance. A task graph representing task dependencies can therefore be formed. The availability of the precedence constraints among the different sub tasks makes this problem suitable for static scheduling. There are few known polynomial time scheduling algorithms even when severe restrictions are placed on the task graph, the sizes of the sub-tasks and the number of parallel processors. The task graph of the problem at hand is very general which does not permit the use of polynomial time scheduling algorithms [19] therefore an optimisation approximation algorithm known as DF/IHS (depth first implicit heuristic search) [4] has been used. This scheme is a heuristically guided depth search for finding optimal or close to optimal schedules. The main feature of DF/IHS lies in the fact that it does not require the computation of heuristic function for all active nodes with the largest depth in order to find the next branching node. This is made possible by a pre-processing stage which also helps to generate a very accurate initial solution for the depth search thereby considerably reducing the time of the search. Computer memory requirement for the algorithm is also quite low and is of the order of $m \times n$ where m and n are the number of processors and the number of sub tasks respectively.

Using dynamic programming technique the critical path lengths for the coarse and finer grain task graphs were found to be 6.27 m. sec. and 3.84 m. sec. respectively. These times represent the minimum possible time for any schedule. DF/IHS was able to reach these times for both grain sizes. The results of schedules are shown in figures 1 and 2. It is evident that better schedules having increased processor utilisation are obtained using finer grain tasks. However the amount of communication involved rises sharply.

5. Practical implementation on transputers

The resulting parallel system is communication intensive. Thus the advantage gained by using smaller average sub-task size is lost when the increase in communication overhead is considered. For practical purposes the coarse grain schedule provides better results. The coarse grain schedule was implemented on networks of T-800 transputers having three and four transputers respectively. The transputer networks are shown in figure 3 and the times achieved are shown are shown in figure 4. DF/IHS schedule results are also given for the sake of comparison. Effects of communication are very obvious from the results of figure 4. In figure 3 (a) and (b) the transputer connected to the host (Sun Spark station)performed the duty of distributing data to the other processors and collecting the results from them. The actual algorithm was run on the remaining transputers. The program was developed using the ANSI C toolset provided by Inmos. Computation times achieved for two and three transputers are 10.7 and 9.3 m.sec respectively. These times are well within the maximum limit imposed for real time control which indicate the usefulness of employing effective parallel processing techniques to the adaptive robot control problem.

6. Conclusions

This paper presented a distributed implementation of a globally stable adaptive controller for a six link industrial robot. All the previous results found in literature are either for a single joint adaptive control algorithm or for an arm having three or less links. Choice of correct controller has been found to be the most important factor in achieving good results through parallel processing. Since the Lagrangian-based controllers are not the best choice for practical implementations because of their computational complexity, controllers based on Newton-Euler equations or having a similar recursive structure are better candidates for parallel processing. The communications-intensive nature of the problem imposes a lower limit on the average size of the sub-tasks. Keeping the above facts in view, the reported distributed adaptive controller for PUMA 560 running on three T-800 transputers achieved times well within the real-time limit.

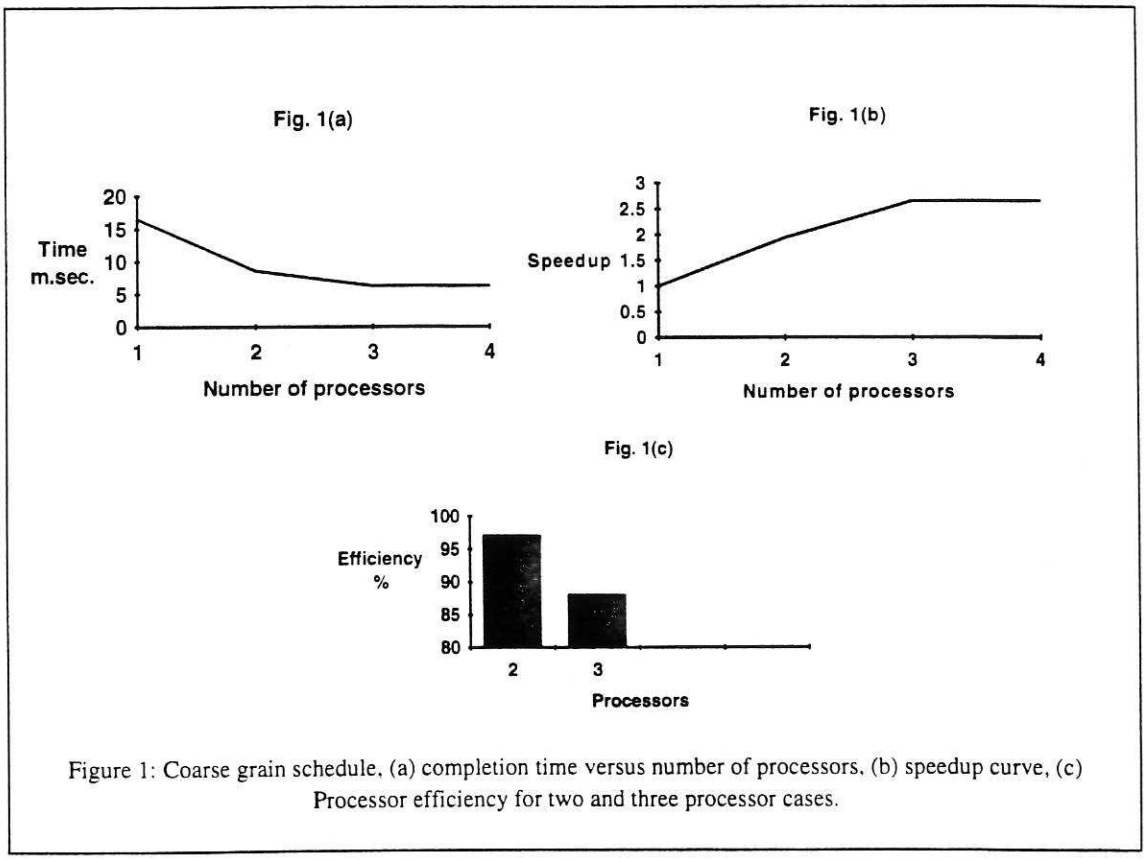

Figure 1: Coarse grain schedule, (a) completion time versus number of processors, (b) speedup curve, (c) Processor efficiency for two and three processor cases.

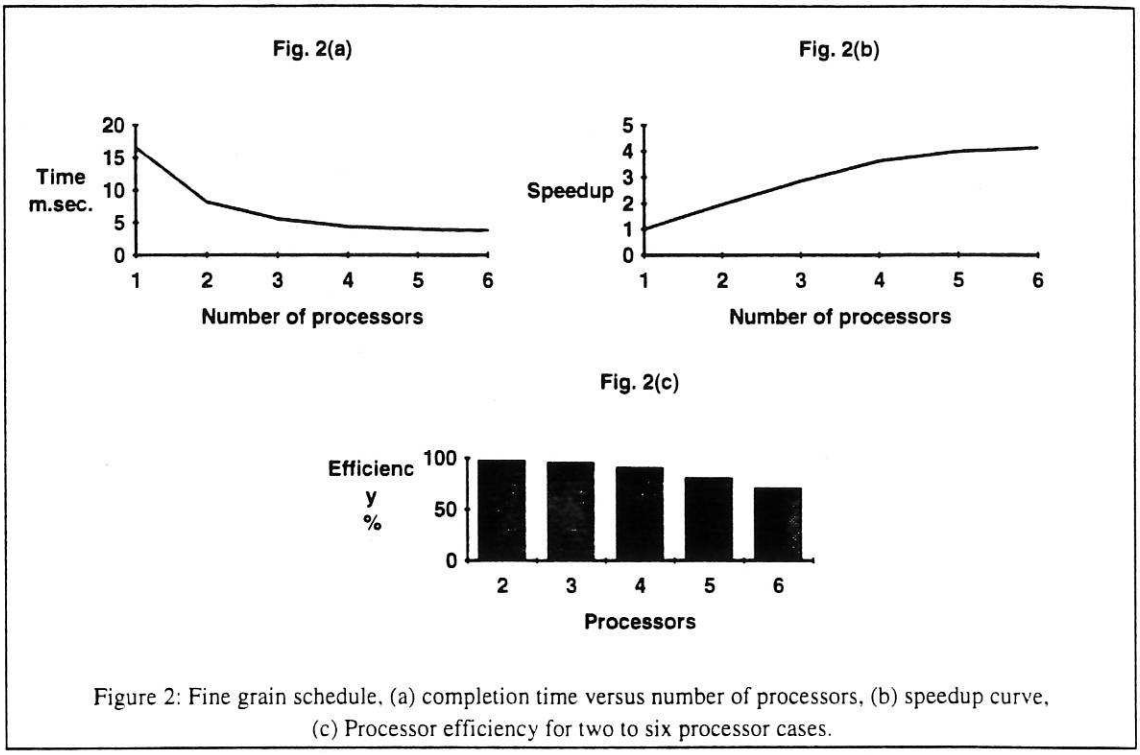

Figure 2: Fine grain schedule, (a) completion time versus number of processors, (b) speedup curve, (c) Processor efficiency for two to six processor cases.

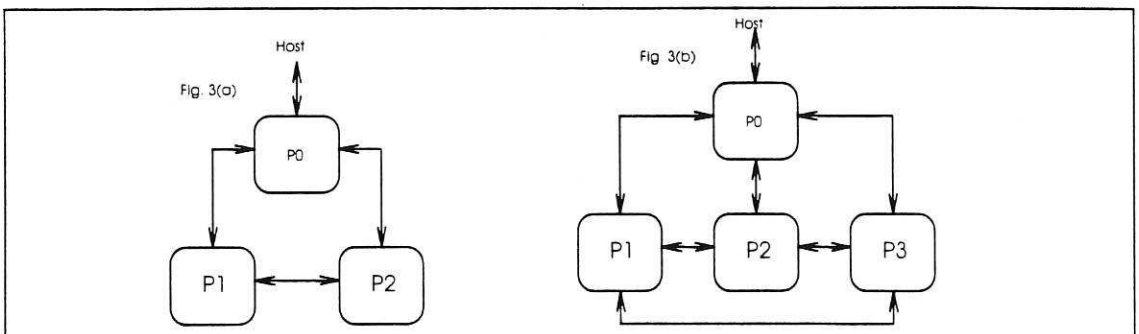

Figure 3: (a) three transputer (T-800) network,
(b) four transputer (T-800) network (P0 is for communication with host.)

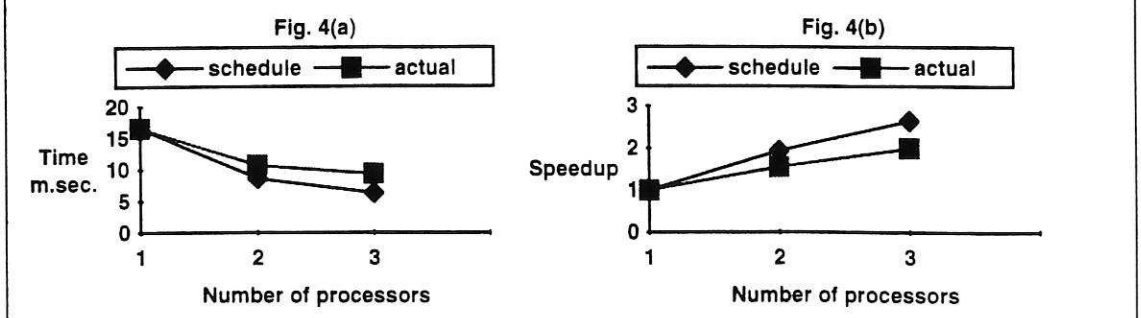

Figure 4: Actual implementation results (a) completion time versus number of processors, (b) speedup curve, (c). (schedule results are also given for comparison)

References

1. S. Dubowsky and D. T. Des Forges, "The application of model reference adaptive control to robotic manipulators, "ASME Jl. Dynam. Syst. Meas. Control, vol. 101, 1979.
2. J. Y. S. Luh and C. S. Lin, "Scheduling of parallel computation for a computer controlled mechanical manipulator," IEEE Trans. System Man and Cyber. vol SMC-12(2), 1982.
3. R. H. Lathrop, "Parallelism in arms and legs," MIT, M. Sc. thesis, 1982.
4. H. Kasahara and S. Narita, "Practical multiprocessor scheduling algorithm for efficient parallel processing," IEEE Trans. Computers vol. C-33(11), 1984.
5. R. Nigam and C. S. G. Lee, "A multiprocessor based controller for the control of mechanical manipulators," IEEE Jl. Robotics Automation vol. RA1(4), 1985.
6. H. Kasahara and S. Narita, "Parallel processing of robot arm control computation on a multiprocessor system," IEEE Jl. Robotics Automation vol. RA-1(2), 1985.
7. T. J. Tarn, A. K. Bejczy, S. Hans and X. Yun, "Dynamic equations for Puma 560 robot arm, "Department of Systems Science and Mathematics, Washington University St. Louis, Missouri 63130, 1985.
8. T. J. Tarn, A. K. Bejczy, S. Hans and X. Yun, "Inertia parameters of Puma 560 robot arm, "Department of Systems Science and Mathematics, Washington University St. Louis, Missouri 63130, 1985.
9. K. S. Fu, R. C. Gonzalez and C. S. G. Lee, "Robotics: Control Sensing Vision Intelligence," McGraw Hill Book Company, 1987.
10. J. J. E. Slotine and W. Li, "On the adaptive control of robot manipulators," Int. Jl. Robotics Research, vol. 6(3), 1987.
11. R. Featherstone, "Robot Dynamics Algorithms," Kluwer Academic Publishers, 1987.
12. M. W. Walker, "An efficient algorithm for the adaptive control of a manipulator," IEEE Int. Conf. Robotics Automation, vol. 2(682), 1988.
13. J. J. Craig, "Adaptive Control of Mechanical Manipulators," Addison-Wesley, 1988.
14. K. Hashimoto and H. Kimura, "A new parallel algorithm for inverse dynamics," The Int. Jl. Robotics Research, vol. 8(1), 1989.
15. H. Kasahara, "Parallel processing of robot arm dynamic control computation on multiprocessors," Microprocessors and Microsystems, vol. 14(1), 1990.
16. K. Hashimoto, K. Ohashi and H. Kimura, "An implementation of a parallel algorithm for real time model based control on a network of microprocessors," The Int. Jl. Robotics Research, vol. 9(6), 1990.
17. J. J. E. Slotine and N. Gunter, "Performance in adaptive manipulator control," The Int. Jl. Robotics Research, vol. 10(2), 1991.
18. P. K. Sinha and P. L. Ho, Transputer based real time parallel adaptive control of a robot manipulator," in Parallel and Distributed computing in Engineering Systems, Elsevier Science Publishers BV (North Holland), 1992.
19. T. G. Lewis and H. El Rewini, "Introduction to Parallel Computing," Prentice Hall, 1992.
20. I. J. Cornish and A. M. S. Zalzala, "Transputer benchmarks and performance comparison with other computing machines," University of Sheffield, Research Report #467, 1993.
21. Y. Z. Chung and R. W. Daniel, "Parallel processing networks for adaptive controllers, "IEE Proceedings-D, vol 140(1), 1993.