University of Pennsylvania
ScholarlyCommons

Departmental Papers (ESE) | Department of Electrical & Systems Engineering

May 1990

Robot control in a message passing environment: theoretical questions and preliminary experiments

Louis L. Whitcomb
Yale University

Daniel E. Koditschek
University of Pennsylvania, kod@seas.upenn.edu

Follow this and additional works at: http://repository.upenn.edu/ese_papers

Recommended Citation

Louis L. Whitcomb and Daniel E. Koditschek, "Robot control in a message passing environment: theoretical questions and preliminary experiments", . May 1990.

Copyright 1990 IEEE. Reprinted from *Proceedings of the IEEE Conference on Robotics and Automation,* , Volume 2, 1990, pages 1198-1123.

This material is posted here with permission of the IEEE. Such permission of the IEEE does not in any way imply IEEE endorsement of any of the University of Pennsylvania's products or services. Internal or personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution must be obtained from the IEEE by writing to pubs-permissions@ieee.org. By choosing to view this document, you agree to all provisions of the copyright laws protecting it.

NOTE: At the time of publication, author Daniel Koditschek was affiliated with Yale University. Currently, he is a faculty member in the Department of Electrical and Systems Engineering at the University of Pennsylvania.

Robot control in a message passing environment: theoretical questions and preliminary experiments

Abstract
The performance of real-time distributed control systems is shown to depend critically on both communication and computation costs. A taxonomy for distributed system performance measurement is introduced. A roughly accurate method of performance prediction for simple systems is presented. Experimental results demonstrate the effects of communication protocols on real-world system performance.

Comments
Copyright 1990 IEEE. Reprinted from *Proceedings of the IEEE Conference on Robotics and Automation, ,* Volume 2, 1990, pages 1198-1123.

This material is posted here with permission of the IEEE. Such permission of the IEEE does not in any way imply IEEE endorsement of any of the University of Pennsylvania's products or services. Internal or personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution must be obtained from the IEEE by writing to pubs-permissions@ieee.org. By choosing to view this document, you agree to all provisions of the copyright laws protecting it.

NOTE: At the time of publication, author Daniel Koditschek was affiliated with Yale University. Currently, he is a faculty member in the Department of Electrical and Systems Engineering at the University of Pennsylvania.

This conference paper is available at ScholarlyCommons: http://repository.upenn.edu/ese_papers/413

Robot Control in a Message Passing Environment: Theoretical Questions and Preliminary Experiments

Louis L. Whitcomb and Daniel E. Koditschek [1]

Center for Systems Science
Yale University, Department of Electrical Engineering

1 Abstract

The performance of real-time distributed control systems is shown to depend critically on *both* communication and computation costs. A taxonomy for distributed system performance measurement is introduced. A roughly accurate method of performance prediction for simple systems is presented. Experimental results demonstrate the effects of communication protocols on real-world system performance.

2 Introduction

This paper concerns the design and implementation of a new distributed controller intended to standardize all real-time computation within the Yale Robotics Lab and to support the interconnection of any combination of devices from our growing zoo of robotic sensors, actuators, and kinematic chains. We had set out with the intention of building "merely" an easily reconfigurable and incrementally cheap family of computational engines in order to get on with the "real work" of robotics. We quickly discovered that the very ubiquity of design that so facilitated interconnection led to a potentially bewildering variation in network behavior depending upon apparently innocent changes in topology and communication protocols. Thus, the paper constitutes, as well, a prelude to research in the theory of distributed message passing controllers.

We have distributed control computation over networks whose nodes are based upon one of the most powerful commercially available microprocessors — the Inmos T800 — and whose capacity can be easily expanded (in consequence of the T800's novel design) to meet almost any demand. However, mere capacity, in itself, is useless barring effective deployment. We provide preliminary experiments demonstrating our success in tapping this processing power to achieve digital sampling rates and floating point accuracy in the "benchmark" computed torque algorithm that compare favorably with any results previously reported in the literature. At the same time, these experiments point out the subtleties involved in deploying a controller over a network of processors: very different results obtain from apparently slight modifications in deployment strategies; the very notion of sampling rate itself must be revised within more general issues of "timeliness". Although these subtleties yield to intuition in the simple ten-node networks described here, we are interested in real-time computation of algorithms whose complexity must grow exponentially with the degrees of freedom [16]. Thus, we conclude that a much more systematic attack upon distributed real-time control design will be needed.

This paper is organized as follows. In the next section we offer a sketch of the large body of literature addressing such issues as correctness, latency, and models of concurrency that arise in this work. Section 3 then focuses upon certain aspects of timeliness in distributed controller performance that seem to have been relatively ignored in the literature to date. In particular, we introduce the notion of a *cross latency matrix* , and provide some experimental evidence of the subtlety in its causal mechanisms. Section 4 concerns the network designs we have chosen for the computed torque control of a torque actuated industrial manipulator and provides experimental data concerning its performance. A conclusion follows.

3 Physical and Computational Models

A burgeoning body of theory and working designs [13], attests to the growing belief that distributed processing represents the only viable solution to the expanding computational requirements of robotic systems. A meaningful discussion of distributed systems requires a taxonomy and model. Two widely accepted models of concurrent computation are the shared memory (SM) model and the communicating sequential process (CSP) model. The models provide fundamentally different mechanisms for interprocess communication [14, 7].

A simple shared memory system might have a common bus of some fixed bandwidth populated by many processors and a common memory store. This provides for efficient message *broadcasting*, wherein one process communicates an identical message to all other processors. However the communication bandwidth remains constant as the number of processors varies, often necessitating exotic bus design while restricting the system to relatively few (tens or so) useful processors [17].

A simple CSP system might have processors connected by individual point-to-point communication channels of fixed bandwidth. Actual systems systems often offer a fixed number of point-to-point communication connections for each processor, thus total system communication bandwidth scales linearly with the number of processors. At the expense of SM-style broadcasting, the CSP systems scalable communication bandwidth appears to offer the possibility of very large scale distributed systems.

[1]This work was supported in part by INMOS Corporation, GMF Robotics Corporation, Weitek Corporation, and the National Science Foundation under a Presidential Young Investigator Award held by the second author.

CH2876-1/90/0000/1198$01.00 © 1990 IEEE

3.1 CSP Architecture for Robot Control

We have used control systems based on both SM and CSP architectures, and found CSP based architectures to offer the following advantages for robot control applications:

Extendable Interprocessor Communication Bandwidth: The fixed bandwidth limitations of shared bus SM architectures restrict them either to applications requiring little interprocessor communication or to applications requiring relatively few processors. The CSP based architectures, whose communication bandwidth can increase with the number of processors, admits the possibility of practical large scale distributed controllers.

No Broadcasting: The control structures we are developing generally do not require that global information be broadcast to every processor.

Device I/O - Transmission and Isolation: The critical bus bandwidth requirements of SM architecture demand a fast parallel bus. The CSP architecture, with a multiplicity of communication channels, places less stringent bandwidth demands on individual channels. Relatively narrow (often serial) communication hardware channels suffice, vastly simplifying the hardware design issues of long distance transmission and electrical isolation.

The INMOS Transputer is a commercially available microprocessor designed for embedded control. The processor provides DMA support for interprocessor communication, hardware primitives for intra-processor communication between multitasked processes, sub microsecond context switching, and a hardware scheduler which obviates the necessity of a real-time operating system or kernel. The user has exceptional freedom and ease in mapping an abstract topology of processes and soft channels onto an actual hardware architecture of processors and hardware channels.

3.2 Distributed Computer Controllers for Continuous Dynamical Plants

The gap between continuous plant and digital discrete controller models is familiar to every practicing engineer. The proper modeling of system performance and machine arithmetic is an active concern of both the control [3, 15] and computer science [11, 2] communities. Investigators have also begun to directly address the performance of discrete time nonlinear controllers in continuous time closed loop systems [8] where, roughly speaking, we desire a "Nyquist sampling theorem" for nonlinear systems.

Concurrent control system implementations, which are used to accelerate computation, offer additional obstacles to accurate analysis and synthesis tools. The very notion of sampling period is compromised by a concurrent implementation. A conventional discrete time controller has its outputs assume constant values for discrete sample periods. A concurrent controller, however, may have no well defined sample period because the inputs and outputs may be serviced asynchronously and concurrently.

4 Timeliness: A Critical Aspect of Distributed Controller Performance

There is no reason to hope for quick theoretical solutions to the myriad design issues that arise from juxtaposition of continuous dynamical plants with digital discrete distributed controllers. Indeed, as we have tried to suggest above, not even the proper analytical framework for posing rigorous hybrid design problems is yet available. At least, however, most of the component issues have already attracted a considerable following, a growing literature, and we can expect a great deal of progress over the next decade.

Notably absent from many treatments of distributed real-time control is the explicit inclusion of *interprocessor communication costs* concomitant with computation costs of a distributed controller. Yet, in the course of building and running the robot controller described in the next section, we have observed significant variations in performance whose cause can be attributed to this phenomenon. The "timeliness" of computations throughout a network depends upon *both* communication *and* computation costs. In turn, the communication costs critically depend on both network topology and the interprocessor buffering paradigms employed.

4.1 Measures of "Timeliness"

A simple *sequential* model for a terminating computation has it read some operands from an input, evaluate an expression, and write the results to an output. We shall call the interval of time between reading the operand and writing the result (including the finite period consumed by i/o operations themselves) the **latency** of the computation. If the computation is performed at regular cycles, we shall term the interval of time between successive results the **update period** of the system. In the terminology of classical control theory, "sample period" corresponds to the above definition of update period.

A simple *concurrent* model of computation has the system reading from n inputs, computing an expression, and writing to m outputs all concurrently. The "Sample period" is ill defined in this case. However, a well defined interval of time elapses between data being read at input i and a valid result based upon this data appearing at output j. We shall term this interval the $i-j^{th}$ **cross latency** of the computation. There will, in general, be $n \cdot m$ cross latencies for such a system. If the computation is performed repetitively, we shall term the interval of time between successive results at the j'th output to be the j^{th} **update period** of the computation.

Note that, unlike the sequential model presented above where update period is at least as great as latency, the concurrent model has no such restriction because several computations may be simultaneously executing. We feel these quantities capture important qualities of concurrent computation systems essential to control applications, and in the sequel we discuss how internal system structure determines the values of these measures, as well as how they might relate to dynamical models of robot performance. We conclude this section with a discussion of how to measure **update** and **latency** in actual distributed systems.

4.1.1 Update and Latency Measurement

To measure the update rate of a process one can simply filter the output of the simple process through a dedicated update measurement process. As indicated in the picture of Figure 1 the update measurement process simply forwards the output of the simple process, and reports update measurements to the host over a special channel.

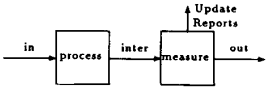

Figure 1: Update Measurement

Latency of one or more processes is measured by tagging incoming data with a timestamp. The timestamp is passed along with the results of each computation, and the process which reads the final output can read the timestamp to determine how long the data was in transit. This technique is pictured in Figure 2.

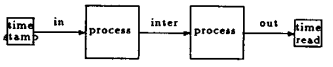

Figure 2: Latency Measurement

Note that measurement of latency is problematic if the directed graph corresponding to data flow contains any directed cycles. For in such a case, when the data "circulates", suitable criteria determining when the computation has finished are required. Many of our implementations have data flowing in undirected cycles, but we have never had the need for directed cycles.

Cross latency is measured in systems which have multiple data inputs and multiple data outputs. Latency between any individual data input and output is the actual cross latency for the input-output pair.

4.1.2 Global Clocks

Establishing a global time reference is a simple matter when the network is fully connected either via channels (CSP Model) or a common bus (SM Model). In practice we have designed asynchronous control networks. Controller operation does not require any global time reference — feedback from the physical plant itself provides the ultimate measure of "timeliness". Only when we wish additional diagnostic information from the system, such as cross latency or run-time execution tracing, does a global clock become necessary. A global clock of sufficient accuracy for most purposes may be established by providing stable local processor clocks which are synchronized at system startup.

4.2 Communication and Computation Performance Issues

A computer system often has predictable time costs associated with primitive computation operations such as arithmetic ($+, -, \times, \div$, etc...), elementary functions ($\sin(\cdot), \cos(\cdot)$, etc...), control flow, and the like. Distributed systems will also often have predictable time costs associated with primitive interprocessor communication operations. It is sometimes possible to estimate the execution time of simple deterministic programs by summing the time taken for each individual operation in the source code. This naive estimate of system performance, however, fails to account for significant system level effects which we shall discuss in this section. We will illustrate system level effects in the context of pipelined systems, because they are perhaps the simplest concurrent systems which clearly illustrate the phenomenon.

4.2.1 Buffering Effects: No Buffering

When several processes pass data over unbuffered links in a pipelined fashion the data update and latency depend in a curious fashion on the intervals at which data is made available to the first processor (p_{input}), the individual process latencies (p_i), the individual communication latencies (c_i), and the intervals at which the final results are read from the last process (p_{output}).

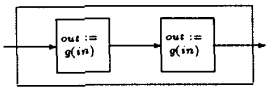

Figure 3: A Simple Pipelined Computation

Figure 3 shows a simple pipeline in which a process repeatedly performs a blocking read on its input link, evaluates a function, and performs a blocking write to its output link.

The average update rate of results from the pipeline is determined by the slowest process cycle in the pipe, $max(p_{input}, p_{output}, (p_i + c_i + c_{i+1})_{i=1,2})$. This is a consequence of the synchronization imposed by the unbuffered, blocking communication. At steady state the processes all eventually depend synchronously either directly or indirectly on the slowest process in the pipeline.

The average latency of the pipeline is more difficult to predict. The slowest process contributes the latency of its period. The "upstream" processes are directly or indirectly synchronized with the slowest process - waiting to write their results to their neighbor. Once a steady state is reached, each process upstream of the slowest contributes one multiple of the slowest's period to the overall latency. The downstream processes form their own pipeline which has its own slowest process, thus these processes must be recursively analyzed to determine their overall latency contribution to the pipeline. The latency analysis must be applied recursively to the downstream processes until the only remaining downstream item is the period with which results are read from the output of the pipeline, p_{output}.

4.2.2 Buffering Effects: Fifo Buffering

The case of communication via first-in first-out buffered links with blocking I/O is the same as unbuffered links when one represents buffered channels as an appropriately sized chain of computationless processes which simply input and output repeatedly.

4.2.3 Buffering Effects: Latest Buffering

In another kind of buffering, which we shall call **latest buffering**, two processes perform one way communication via a shared variable to which one process may perform atomic writes, and from which the other process may perform atomic reads. The reads and writes are discrete in time, and non blocking. The second process clearly may only read the "latest" value written

by the first process. This buffering technique can be useful for avoiding synchronization problems in concurrent systems, since the two processes are completely asynchronous.

The update period of such a process is simply that of the computation itself along with the internal and external communication times. Note that the update rate is independent of the data input rate. The latency period for data within a process is given by the sum of the update period, the time the data waited in the input buffer, and the time consumed by the actual input and output operations. Figure 4, which shows the experimentally measured latency of such a computational process as a function of the frequency at which the input buffer receives new data, shows that update rate is indeed independent of input rate and that average latency is equal to the update period plus half the input period.

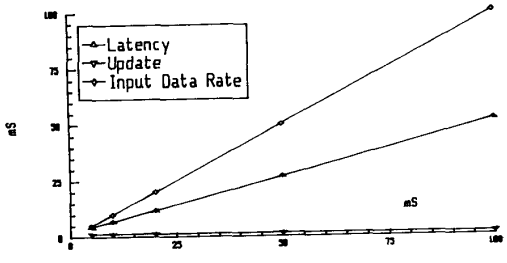

Figure 4: Update and Latency vs Data Input Rate.

5 A Distributed Robot Controller

We have implemented distributed real time control systems. This section offers a brief discussion of some design and implementation aspects of the actual control systems.

5.1 Device Independence: Control and Actuation

Where does the controller end and the plant begin? While the conceptual distinction between the two entities may be clear, the physical division is ambiguous. Is the division at the output shaft from the electric motor to the link? Is it at the command input to the motor power amplifiers? Or is it at the diodes of an optical shaft encoder?

We have found it convenient to place this conceptual division within the processor network itself, partitioning the network into an **actuator subsystem** considered to be part of the plant and a **control subsystem** which constitutes the controller. Each robot joint has a dedicated **actuator processor** which provides a device independent joint interface to the control subsystem. It handles the multitude of housekeeping task necessary to run an actual motor — possibly including commutation, position and velocity estimation, safety monitoring, and the like. The actuator process is a smart device driver which provides the control subsystem with a uniform interface to the many (often infuriatingly different) motor hardware interfaces of the robot.

Each actuator process has one channel to and one channel from the control subsystem. To the control subsystem it provides current state information — position and velocity of the joint — in floating point format in conventional units. From the control subsystem it receives torque (or force) commands, also in floating point format and conventional units. This modular design limits device dependent design to the actuator processors and frees the control subsystem to perform the purely "algorithmic" task of executing a control law.

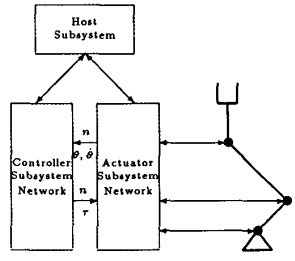

Figure 5: Control System Structure

We feel that the distinction between actuator and controller subsystems offers implementation advantages, by modularizing device dependence, as well as the conceptual clarity of isolating the more abstract algorithmic and computational control functions within the controller subsystem.

5.1.1 Computational Hardware

The computational hardware for these implementations was the Yale XP/DCS, a powerful real-time control node based on the INMOS Transputer floating point microprocessor. Students at the Yale robotics Lab currently use the XP/DCS *in actual working systems* which include: A robot juggler [4], a GMF A-500 SCARA arm [5] a field-rate real-time vision system, an advanced sensor-based obstacle avoidance system [6], a variable-reluctance motor commutation test bed, and a one degree of freedom pneumatic-muscle robot. Several new projects are underway.

5.1.2 The Plant

The GMF Robotics Model A-500, a four degree of freedom SCARA type arm shown in Figure 6, was chosen as the target mechanical unit. Each joint which has a motor capable of delivering torque and a position sensor. Like virtually all commercially available robot systems, the original A-500 system controller provides an integrated high level user interface which serves admirably in industrial applications, but precludes the low level servo intervention which is needed in the research laboratory. It was therefore necessary to replace the manufacturer's control system with our own low level interface. At present interfaces are fully operational for the two primary revolute axes.

5.2 Actual Controller Computational Performance

As a preliminary test for this CSP controller architecture we have implemented the well known "computed torque" nonlinear feedback control algorithm. While the performance of this algorithm has been extensively analyzed and simulated, its computational requirements have precluded its widespread use in actual systems. Several excellent experimental implementations have been reported in the literature. A purely feedforward

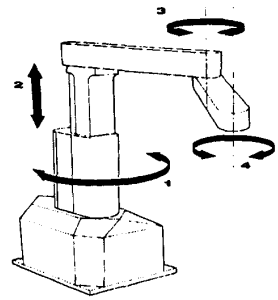

Figure 6: The GMF Model A-500

inverse dynamics implementation by Kanade at. al. [9] achieved a 1.2 mS update period for a six axis arm by exploiting structural features of the Newton Euler derivation for equations of motion. An implementation by Leahy et. al. [12] obtained a 14 mS sample period for a six axis arm. Khosla and Kanade [10] achieved a 2 mS sample period for a six axis arm using approximations of nonlinear terms. An et. al. [1] obtained a 7 mS sample period for a three axis arm in an exact implementation using scaled integer representation.

Rather than optimizing this algorithm to fit an available architecture, we constructed an architecture to suit the algorithm. We based our implementation on the exact Euler-Lagrange equations for the three dynamically coupled joints of a GMF A-500 SCARA arm, including all nonlinear terms of the derivation. Full 32 bit floating point representation was employed, and mathematical expressions were *not* optimized to omit special case operations on parameter values of 0.0, 1.0, and the like.

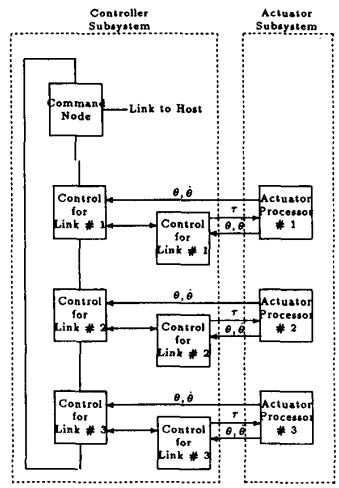

Figure 7: A Controller Hardware Topology.

The system was entirely programmed in the high level language OCCAM, and no assembly language code was used anywhere in the system. As pictured in the topology of Figure 7 the feedback expression for an individual joint, comprising the entire feedback gain calculation (including off-diagonal terms) and the appropriate rows of the generalized interia and coriolis matrices, were partitioned to processors directly connected to the corresponding joint actuator processor.

Figures 8, 9, and 10 tabulate actual measured cross latency matrices for this implementation. The control network has three inputs and three outputs corresponding to the respective robot actuator processors. The row index index of a matrix indicates the state input from each actuator and the column index indicates the command output to each actuator. For example the (1,1) element of the matrix represents the self latency of the network for actuator 1, while the (1,2) element represents the cross latency from actuator 1 through the network to actuator 2. Units are microseconds.

The network topology and computation distribution scheme was identical in each case, however different buffering strategies were employed for interprocess communication. Latencies uniformly under $1mS$, Figure 8 are observed for the unbuffered network. Introducing latest buffering, shown in Figure 9, is seen to significantly increase the off-diagonal terms — representing increased latency between concurrent partitions of the control algorithm. Figure 10 shows cross latencies obtained by employing unbuffered communication within the algorithm partitions and employing latest buffering between them. The strong diagonal dominance of the matrix shows "self latencies" uniformly under $600\mu S$ (for the computationally identical exact computed torque algorithm) at the expense of increased off-diagonal latencies.

These experimentally measured latencies reveal computational performance which compares favorably with other reported implementations, thus establishing the usefulness of CSP architectures for robot control. This data underscores the contribution of communication time to overall latency in concurrent controllers. The examples particularly demonstrate the significance of buffering strategy in determining the communication costs of a given network topology.

link	1	2	3
1	817	706	823
2	982	832	930
3	860	731	829

Figure 8: Actual Cross Latency: Unbuffered (μSec)

node	1	2	3
1	995	1320	1378
2	1166	1021	1270
3	1422	1296	967

Figure 9: Actual Cross Latency: Latest Buffering (μSec)

node	1	2	3
1	598	1456	1718
2	1486	572	1457
3	1707	1457	574

Figure 10: Actual Cross Latency: Combination of Un- and Latest-Buffering (μSec)

6 Conclusion

The substantial and ever increasing literature on distributed architectures within the robotics community suggests a consensus that concurrency offers the only practical solution to increasing computational needs.

We have argued that sample period, the classical measure of "timeliness" in discrete time controllers, is problematic for concurrent control systems, and have proposed the well defined quantities of update period (which reduces to sample period in the case of sequential implementations) and cross latency as generalized measures of controller computational performance.

We have constructed a working robot controller, using commercially available components and development tools, whose performance compares satisfactorily with both sequential and distributed applications in the literature. Moreover, our experiments demonstrate that update and latency in concurrent systems depend critically on *both computation and communication costs* even though the latter have been relatively ignored in the literature. We have shown that, in turn, communication costs depend critically on both the network topology and buffering strategy.

For simple (pipeline) topologies we have developed a roughly accurate tool for predicting the latency and update periods that result from the buffering paradigms of Section 4.2, as depicted in Figure 4. The surprising variation in cross latencies (Figures 8, 9, and 10) that obtains from schemes in the less trivial topologies (Figure 7) suggests the need for a richer set of theoretical techniques. For even though we do not know exactly what effect the various cross latencies will have on the performance of our physical closed loop systems (experiments of this kind are presently in progress) it seems virtually certain that these effects will become more dramatic in larger distributed controllers.

References

[1] Chae H. An, Christopher G. Atkeson, and John M. Hollerbach. *Model-Based Control of a Robot Manipulator.* MIT Press, Cambridge, MA, USA, 1988.

[2] Geoff Barrett. Verifying the transputer. In *NATUG1: Proceedings of the First Conference of The North American Transputer Useers Group*, pages 21–29, Salt Lake City, UT, USA, 1989.

[3] R. W. Brockett. On the computer control of movement. In *IEEE International Conference on Robotics and Automation*, pages 534–540, Philadelphia, PA, USA, 1988.

[4] M. Bühler , D. E. Koditschek, and P.J. Kindlmann. A Simple Juggling Robot: Theory and Experimentation. In V. Hayward and O. Khatib, editors, *International Symposium on Experimental Robotics*, page (to appear). Springer-Verlag, 1989.

[5] M. Bühler, L. Whitcomb, F. Levin, and D. E. Koditschek. A distributed message passing computational and i/o engine for real-time motion control. In *Proc. American Control Conference*, pages 478–488, Pittsburgh, PA, Jun 1989. American Control Society.

[6] Edward Cheung and Vladimir Lumelsky. Development of sensitive skin for a 3d robot arm operating in an uncertain environment. In *IEEE International Conference on Robotics and Automation*, pages 1056–1061, Scottsdale, AZ, USA, 1989.

[7] C. A. R. Hoare. *Communicating Sequential Processes.* Prentice-Hall, Englewood Cliffs, NJ, USA, 1985.

[8] Ping Hsu and Shankar Sastry. The effect of discretized feedback in a closed loop system. In *Proceedings of the 26th Conference on Decision and Control*, pages 1518–1523, Los Angeles, California, USA, 1987.

[9] Takeo Kanade, Pradeep K. Khosla, and Nobuhiko Tanaka. Real-time control of cmu direct-drive arm ii using customized inverse dynamics. In *IEEE Conference on Decision and Control*, pages 1345–1352, Las Vegas, Nevada, USA, 1984.

[10] Pradeep K. Khosla and Takeo Kanade. Real-time implementation and evaluation of model-based controls on cmu dd arm ii. In *Proceeding IEEE International Conference on Robotics and Automation*, pages 1546–1555, San Francisco, CA, Apr 1986.

[11] U. W. Kulisch and W. L. Miranker. The arithmatic of the digital computer: A new approach. *SIAM Review*, 28(1):1–40, March 1986.

[12] M. B. Leahy, Jr., K. P. Valavanis, and G. N. Saridis. The effects of dynamic models on robot control. In *IEEE International Conference on Robotics and Automation*, page 4954, San Francisco, CA, USA, 1986.

[13] C.S.G. Lee. IEEE Transactions on Robotics and Automation Special Issue on Robot Manipulators: Algorithms and Architectures., October 1989.

[14] A. Pnueli. Applications of temporal logic to the specification and verification of reactive systems: A survey of current trends. In J.W. Bakker, W.P. de Roever, and G. Rozenberg, editors, *Current Trends in Concurrency*, chapter 9, pages 510–584. Springer-Verlag, 1986.

[15] P. J. Ramadge and W. M. Wonham. Supervisory control of a class of discrete event processes. *SIAM J. Control and Optimization*, 25(1):206–230, Jan 1987.

[16] E. Rimon and D. E. Koditschek. Exact robot navigation in geometrically complicated but topologically simple spaces. In *Proc. IEEE International Conference on Robotics and Automation*, Cincinnati, OH, USA, May 1990.

[17] P. Woodbury, A. Wilson, B. Shein, B. Gertner, P. Y. Chen, J. Barttlet, and Z. Aral. Shared memory multiprocessors: The right approach to parallel processing. In *Proc. 34th IEEE Computer Society International Conference — COMPCON*, pages 72–80, San Francisco, CA, USA, February 1989. IEEE Computer Society Press.