6-23-1989

A Distributed Message Passing Computational and I/O Engine for Real-Time Motion Control

M. Buehler
Yale University

L. L. Whitcomb
Yale University

F. Levin
Yale University

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

Recommended Citation

M. Buehler, L. L. Whitcomb, F. Levin, and Daniel E. Koditschek, "A Distributed Message Passing Computational and I/O Engine for Real-Time Motion Control", *Proceedings of the American Control Conference* , 478-483. June 1989.

Abstract
This paper illustrates the use of the Yale XP/DCS - a dual board real-time distributed control module based upon the INMOS Transputer family of micoprocessors - for high performance real-time motion control applications. The XP/DCS complements the the Transputer's 1.5 Mflop computational rate and four independent on-chip 20 Mbps DMA communication engines, by providing a bidirectional latched 32 bit bus extension with full handshaking support for easy customization of the I/O capabilities of any node. After contrasting this design with commercially available alternatives we describe three particular applications presently underway in the Yale Robotics Laboratory. We conclude by reporting some initial experiments concerning the effect of code distribution and message passing protocols upon sampling rate.

1989 American Control Conference
Pittsburgh, PA

June 21-23 1989

A Distributed Message Passing Computational and I/O Engine for Real-Time Motion Control

M. Bühler, L. L. Whitcomb, F. Levin, D. E. Koditschek [1]
Center for Systems Science
Yale University, Department of Electrical Engineering

Abstract

This paper illustrates the use of the Yale XP/DCS — a dual-board real-time distributed control module based upon the INMOS Transputer family of microprocessors — for high performance real-time motion control applications. The XP/DCS complements the the Transputer's 1.5 Mflop computational rate and four independent on-chip 20 Mbps DMA communication engines, by providing a bidirectional latched 32 bit bus extension with full handshaking support for easy customization of the I/O capabilities of any node. After contrasting this design with commercially available alternatives we describe three particular applications presently underway in the Yale Robotics Laboratory. We conclude by reporting some initial experiments concerning the effect of code distribution and message passing protocols upon sampling rate.

1 Introduction

This paper describes a modular computational engine that has become the workhorse for almost all real-time motion control tasks that we encounter in the Yale Robotics Laboratory. Roughly speaking, these tasks amount to the marshalling of various data — from external sensors; from user specified commands; from motor joint variables — and rapid computation involving their appropriate combinations all in time to produce motor commands that will ensure high performance coordinated movement of many coupled mechanical degrees of freedom.

Our solution to the exigencies of high performance motion control rests largely upon the INMOS "Transputer" family of microprocessors. To the powerful built-in computation and communications capabilities of their hardware, INMOS has added a facile and integrated software/development environment extremely well suited to the generation of parallel and concurrent versions of numerically complex algorithms in a message passing environment. Our contribution to the existing technology lies in the design and implementation of a two board set of printed circuits reflecting the I/O and memory requirements of the present applications. Our purpose in writing this article is to alert control engineers to the intrinsic capabilities of Transputer networks in general and the Yale "XP/DCS" boardset in particular (Section 2); to provide illustrative examples of the use and performance of our boards in specific motion control applications (Section 3); and to suggest some of the novel theoretical issues that arise when real-time control is effected by a distributed network of message passing computers (Section 4).

2 The Yale XP/DCS: Design Requirements and Alternatives

Typical servo motors have time constants of between 10 and 100 msec., depending upon the load. While the theoretically prescribed *Nyquist sampling rate* is merely double the fastest frequency of the system to be controlled, practical wisdom in the control community has long dictated the choice of an order of magnitude higher [5]: we may thus take $1kHz$ as the desired sampling and control update frequency target. The coordination of multiply coupled degrees of freedom exacts a surprisingly high computational cost. For example, an "inverse dynamics" controller for a six degree of freedom industrial robot manipulator requires roughly 10^3 flops every sampling interval [6]. This algorithm, then, would require $1Mflop$ of raw computational capability for real-time implementation. Since the controller must also sample joint positions and velocities and deliver torque commands to all six degrees of freedom, the algorithm assumes that 18 parallel I/O operations are performed at the rate of $1kHz$ as well. In fact, the computational requirements of robotics applications are much more stringent if we are interested in autonomous robots and other machines which possess some greater independence from human operators. For example, robot navigation feedback laws for environments cluttered with obstacles [8] involve real-time computation which grows exponentially with the number of degrees of freedom [12]. Intuitively, it is clear that the closer we come to an anthropomorphic model of the machine, the more overwhelming the real-time computational load.

More relevant, perhaps, to the short term opportunities of the industrial world are recent developments in smart sensor and actuator technology. For example, a new class of "variable reluctance" motors promises relatively large torque/mass ratios at very low speeds suggesting great improvements in direct-drive positioning applications through the elimination of transmissions and linkages. However, satisfactory performance from these motors requires a great deal of attention to their commutation and current drive strategies [9], necessitating the dedication of a single processor to the production of torque itself in a single motor. Similar computational requirements in smart sensor applications will quickly convince the forward minded engineer that real-time motion control applications require not merely copious computational capacity but distributed processing and high capacity I/O at physically distinct locations.

[1]This work is supported in part by Inmos Corporation, GMF Robotics Corporation, PMI Motion Technologies, Weitek Corporation, and, in part, by matching funds from the National Science Foundation under the terms of a Presidential Young Investigator Award held by the last author.

Two years ago we initiated a search for a powerful, cost effective, and easily expandable solution to the problems of high performance real-time motion control in the Yale Robotics Laboratory. We sought a system consisting of relatively simple and affordable nodes, deployed in a reconfigurable network arrangement with high inter-node bandwidth. Since the theory regarding algorithm partitioning within a parallel processing environment is still in its infancy, and target control situations are legendary in their variability, we saw system reconfiguration flexibility as the most important goal. Furthermore, since robot sensor and actuator technology is in a state of rapid flux, each node would need to be easily adaptable to specific digital and analog interfacing problems. A most important consideration centered around the software and development environment. We wanted to experiment with motion control algorithms and emphatically did not want to write real-time operating system kernels or develop our own parallel/concurrent code distribution tools.The system chosen would need a suitable high level language that supports parallel processing, and has capabilities for writing, distributing and debugging code. Most critically, the hardware associated with the development and target system system needed to be affordable.

The options available in the marketplace were numerous when we began our search for solutions two years ago: they are almost bewildering now. Since we feel that our "XP/DCS" system remains attractive today with respect to the required capabilities (and particularly with respect to price/performance criteria) we will present a brief review of commercial alternatives below before describing the nature of our design.

2.1 Technology Review

Given clear upper limits on desired processor power and I/O capacity it would seem most desirable to use a single centralized CPU. Unfortunately, the open-ended nature of robotics research virtually guarantees that any fixed-capacity computational environment will be quickly exhausted by the ineluctable desire for more degrees of freedom and more complicated algorithms. Thus, a "granular" incrementally costly control architecture seems unavoidable.

2.1.1 Shared Bus Based Commercial Products

Commercial motion controllers based upon the current generation of 32 bit microprocessors designed as functional elements on one of the many shared bus arrangements are currently a very popular means to this desirable "finer grain" more flexible and exapandable architecture. As an example consider the offerings from Ironics Inc.,(Ithaca, NY) a VME board manufacturer specializing in multiprocesing applications. Their latest high-performance board, the IV-9001, is designed around AMD's RISC 29000 microprocessor. With a throughput of 17MIPS at 25MHz and 6MFLOPS with the 29027 coprocessor, the board has enough processing power to handle the range of tasks in question. The system memory held on a daughter board can have 2,8 or 16 Mbytes of static-column DRAM. For I/O applications they plan a similar daughter board with providing 100 Mbytes/sec throughput. Custom I/O cards are also possible. The IV-9001 base board costs $7995. To that must be added memory at $1995 for the 2MB board. The price for the I/O daughter board has yet to be announced. The development system consists of a 68020 based Unix workstation coupled to a VME chassis via serial link. Code is witten, compiled and then downloaded to the target boards in the development chassis. Monitors, and debuggers exist to aid in the process. The development system pricing with software starts at $16,000.

Other VME boards using the Motorola 68020 and 68030 CISC parts are somewhat less expensive but offer less performance. Heurikon Corp. (Madison, WI) will provide a 20MHz 68030 VME cpu card with 4Mb DRAM and a 68882 coprocessor for about $5600. With the Motorola parts, the floating point performance becomes a question for our application. At about .3 MFLOPS the 68030/68882 won't provide the computational power we desire. Although the VME bus has a bandwidth specification of 40 Mbytes/sec, a 680x0 based multiprocessor system would probably average no greater than about 3 - 5 Mbytes/sec of available inter-processor communications bandwidth. Difficulties arise concerning bus latency and the 'partitioning of code in such a system.

Similar considerations apply to the Multibus. As part of the IEEE 1296 specification for the Multibus II, the concept of message passing seeks to accomodate multiple microprocessor systems. A single chip termed the Message Passing Coprocessor was designed by Intel to enable any microprocessor to interface easily to the IEEE 1296 bus. Expected bus bandwidth is improved over that of the VME bus to a respectable 13 MBytes/sec in actual applications. Costs for MultibusII boards are still fairly high however. Heurikon's HK68/M220 board uses the 32 bit 68020 and has 4MB DRAM, 256k EPROM and a 68881 coprocessor and lists for $6295.00

2.1.2 Limitations of Shared Bus Architectures

The high costs per node for these commercial VME and Multibus processor boards represent only one aspect of their deficiency regarding the intended application. For those problems requiring the power of a multiprocessor solution, the major limitations of these systems stem, ironically, from their fundamental strength, the shared bus architecture, or, as it has sometimes been described, the "von Neumann bottleneck." As more processors are added to a system, the bus' ability to service them diminishes. The point at which the number of processors becomes problematic varies with the application but problems with bus loading and latency are inevitable [11]. Faced with these intrinsic limitations, designers have increasingly sought to minimize bus traffic as much as possible. Their solutions take the form of more autonomous "nodes" i.e. processors having more local resources e.g. dedicated memory and I/O ports and systems with more efficient bus handling e.g. the MPC chip on the Multibus II. The enhanced bus bandwidth (100MB/s) expected with the next generation VME bus, variously called Futurebus or Ruggedbus, will provide designers some relief but won't alter the more fundamental constraint limiting performance for multiprocessor systems.

2.1.3 Summary of Commercial Alternatives

In summary, whichever of the conventional microprocessors is chosen as the basis of such a customized distributed architecture, all share two big disadvantages. First, they are not designed to be interconnected for parallel processing and thus, by themselves, do not afford inter-processor communication. Thus one generally is forced to resort to a bus based approach — the basic structure for almost all parallel real-time control systems built in the past. Unfortunately, the bus communication bandwidth decreases at least linearly with the number of nodes and

I/O units which are attached in a parallel fashion. While it may be feasible to build such parallel systems with only few nodes, expandability quickly becomes limited (depending on the communication requirements of the specific application). Second, due to the lack of a suitable language, software issues become more and more problematic as the hardware increasingly exploits parallelism.

2.2 The Yale XP/DCS Controller Node

In this section we present a description of a system we feel represents a "best fit" of technology to task: a reconfigurable distributed network of I/O customizable boards costing less than $2000 each for which a complete off-the-shelf commercial development environment may be purchased for roughly $5000. Moreover, since network links are standardized, our nodes may be trivially interconnected to any of the hundreds of additional existing third party vended boards based upon this technology. We have described the design and performance characteristics of our boards elsewhere in detail [4, 2], and will be content with a brief sketch here.

2.2.1 The INMOS Transputer

The choice of the INMOS product line represents a strategy which standardizes and places the burden of parallelism — inter-processor communications support, software, and development environment — around a commercial product, while customizing the computational or I/O "identity" of particular nodes by recourse to special purpose hardware.

The Transputer is a 32-bit RISC microprocessor with fast on-chip RAM, interrupt and DMA support, an internal architecture supporting multi-processing, and four high speed DMA serial interprocessor communication links. The latter capability represents the most important feature of this chip relative to its competitors. The four links circumvent the constraints of bus based interprocessor communication schemes both with regard to reconfigurability as well as bandwidth. The result is a topology to which nodes are added or deleted simply by physically connecting a four wire serial cable (and a three wire system service connection). Through the parallel processing constructs of the associated programming language, OCCAM, one can equally simply address the software requirements of process concurrency. Whether multitasking on one transputer, or engaged in parallel implementation on a network of transputers, the desired relationships between software processes and hardware processors may be specified with ease and flexibility.

2.2.2 The XP/DCS Motherboard

Our motherboard consists of a Transputer chip, 128 Kbytes bytes of zero wait state SRAM, address decoding hardware, and a bus extension connector. We have standardized to the increasingly popular Eurocard form factor, using a board size of 100mm x 220mm, the so-called Single Extended Eurocard. The rear edge connector is pin compatible with INMOS' evaluation cards for the ITEM system. In particular, the four high speed *serial links* are made available on this rear edge connector. To ensure signal integrity in harsh EMI, two of the links may be routed to *fiber optic ports* located on the board's front edge. Current hardware choices limit the fiber bandwidth to 5Mbps, the link speed which will be supported by all transputer family products.

The *I/O connector* located on the lower edge of the board, passes the 32 bit Data/Address bus with all interrupt-, DMA-, and bus-control lines. Thus any add-on board can to be attached that "fits" the transputer memory interface. However it is required that all signals be buffered after the I/O connector.

2.2.3 The XP/DCS I/O board

Virtually every I/O device is now available with tri state interface, eliminating the need for a separate latch for each unit, and permitting operation in parallel on a common I/O bus. Thus, it became soon clear that in order to simultaneously maximize flexibility as well as ease of use, we needed a latched 32 bit bidirectional I/O bus which provides for a virtually unlimited number of I/O devices with minimal chip count. In order to further minimize the custom I/O effort when accommodating a specific I/O device, we provide six individually addressable sets of four latched handshaking output lines as well as a total of eight handshaking input lines. In order to prevent arbitrary latched outputs of the handshaking output lines at power up to cause disaster (for example, enabling a robot joint while the torque command is not under control), these outputs wake up in a high impedance state and can be jumpered to either polarity on the fabricated section of the I/O board. For programming and debugging convenience, all handshake output lines can be read back.

This implementation provides the user with a maximum of support for custom I/O needs: Most tristate I/O devices should be able to interface to this bus without any additional support chips at all. If desired, several devices can be accessed simultaneously: for example, two 16 bit or three 10 bit D-to-A or A-to-D converters could be accessed by attaching the different chips to different portions of the I/O bus.

We call the mode previously described the *asynchronous* I/O mode. Devices can be accessed independent of their speed. A complete I/O cycle would take in this default mode would take about 1 μs, or roughly the time for one flop. Examples of this mode of operation are to be found in Sections 3.1 and 3.3 where quadrature output from shaft encoders is required for motor position control. However, in general the board allows for direct bus interfacing to the transputer in a manner that we call *synchronous* I/O mode. This is possible by removing the bidirectional bus latches. An example of this mode of operation is to be found in Section 3.2 where the I/O board carries fast video memory.

2.2.4 Estimated Cost

It is perhaps unfair to compare the cost of commercially marketed electronics with "homemade" equipment. Yet we are now using a printed circuit version of the XP/DCS motherboard that has been produced in (small) volume by a local design company for general use in the Yale Robotics Laboratory at a cost of less than $2000 for each stuffed and tested board. Surely, given the economies of scale involved in higher volume production, this represents the upper limit of cost that would be incurred by a mass market. A printed circuit version of the XP/DCS I/O board discussed above is presently in preparation so we have as yet no good estimate of its volume cost (present applications use wire-wrap prototypes). In any event, the daughterboard concept invites customization of I/O capability — a very different board has been prototyped for the transmission of video data in the application describe in Section 3.2 — and per board cost will vary greatly with the particular design.

3 Applications

The XP/DCS system was designed to be the general workhorse for real-time motion control experiments within the Yale Robotics Laboratory. In this section we will briefly review our experiences with three particular robotic devices. References to more elaborate discussions of each apparatus will be given in the text below.

3.1 A Simple Juggling Robot

A program of research into dexterous manipulation in intermittent dynamical environments [1, 7] leads to experimentation with the juggling robot depicted in Figure 1.

The physical apparatus consists of a puck, which slides on an inclined plane and is batted successively by a simple "robot": a bar with billiard cushion rotating in the juggling plane as depicted in that figure. All intelligent sensor and controller functions are performed by a four node XP/DCS network.

In order to move the bar according to some puck dependent control algorithm, the puck's position and velocity in both directions on the plane must be measured. Presently, this is accomplished by placing an oscillator inside the puck and burying a grid in the juggling plane, thus imitating a big digitizing tablet. On the back of the plane, a simplified XP/DCS system, the Puck Sensor Node, is used as a smart sensor. It measures the voltages induced in the sensing grid by the puck. The puck position in the plane is computed from the zero and first order moments. This information is used to estimate the puck's state: we use a standard linear observer to reduce measurement noise in position and velocity data. Each puck state measurement is communicated asynchronously via fiber optics to the Computation Node. This sampling and communication process is performed at a rate of 1kHz (when tracking one puck). In the near future we intend to introduce an XP/DCS based real time stereo vision system described in the next section in order to move off the plane into three space. We are fairly confident that the attendant decrease in sampling rate to 60Hz will not affect the experimental results significantly.

The Computation Node receives puck state information from the sensor node, reports logging data to the logging node, implements the control algorithm and issues the resulting desired robot states to the Motor Control Node. Various additional tasks like detecting the puck motion status (up, top, down, impact), predicting puck states (both used for extracting logging data) as well as extensive error checking and housekeeping tasks have to be performed on this node as well. The sampling time can vary between 500 and 1000 μs. The Motor Control Node is dedicated to commanding a high torque dc servo actuator (built by PMI Motion Technologies) at a rate of 2kHz.

The experiences with the XP/DCS, the transputer and the development environment derived from this application are very encouraging. No single number can capture the ease of use and the little time spent with system overhead. Given the T800's intrinsic floating point capability, and the mathematical function library, formulas were programmed (in OCCAM, the native compiler) almost directly from the blackboard with no attempt at code optimization. In spite of substantial calculations, and a great deal of data logging and error handling overhead, very high sampling rates were achieved. The system operates capably in a high EMI environment in consequence of the low cost 5Mbps fiber optic units from Hewlett Packard built into the Yale XP/DCS boards.

3.2 A Field Rate Camera System

The need for real time stereo vision arises from our plans to move to a three degree of freedom robot which juggles in three space. The motivation to develop our own vision system as opposed to buying some commercial unit is an outgrowth of that underlying the original XP/DCS design. Most of the computational hardware burden is shouldered by the existing XP/DCS node: we need merely provide an effective interface from the video signal to a motherboard through the bus extension connector described in Section 2.2. In fact, the prototype working version of the Cyclops system described here was designed and tested by two masters students working in the Yale robotics laboratory on a *semester project* for a computer architecture course offerd in the fall of 1988 [13]. Moreover, as is always true of "homebuilt" boards, we end up with a much more powerful and flexible system than could be purchased with our modest laboratory equipment grants.

We have successfully tested the system detailed below in a tracking task requiring the position and velocities of two "point masses" moving in three dimensions in structured lighting at field rate (60 Hz). To the best of our knowledge the only commercially available products which achieve this performance [3, 10] cost considerably more than the system reported here. [1] We are confident that far more complicated objects may be tracked at near field rates as well, but have not yet developed the necessary software.

Cyclops consists of five main elements as depicted in Figure 2: a camera, a digitizer board, a filter board and one or more video memory boards, each of which connects to a XP/DCS CPUboard. The functions are straightforward — the camera provides interlaced fields of video data to the digitizer, which converts the analog signal into an eight bit pixel stream, accompanied with some synchronization signals. The video bus containing the pixel stream is brought to the filter board. Here any two dimensional filter or convolution with a 6x14 pixel size window can be applied to the data in real time. Finally the video data is written into one or several video memory "daughter boards" where it can be accessed by an XP/DCS motherboard over the bus connector. All memory boards attach to the video bus in parallel: the maximum possible number is restricted only by drive and noise limitations.

Even though Cyclops satisfies a specific need, it is sufficiently flexible in its operation as to allow its application to most vision tasks. In order to illustrate its generality, we will describe three different distribution schemes for the video fields and the implications for update rate and latency. By update rate we mean the rate at which results are computed by any of the processing boards on the video bus whereas latency refers to the total processing time from raw video data to end result.

Group Mode: the video fields are loaded into all memory boards simultaneously. Depending on the application, each processor then picks a dynamic or static partition of the full field for processing. A new field is loaded after all processors

[1] Since the video memory boards, unlike the XP/DCS motherboards have not yet been mass produced, it is difficult to give a good cost estimate. A complete system incurs roughly $5000. in electronics parts — a commercially viable product might be sold for double that amount.

have finished processing — thus latency is equal to update rate.

Scan Mode: for many applications, the video data is not easily separable into partitions without extensive inter-processor communication and decreased performance. In this case one can achieve an increased update rate while keeping the same latency as with only one memory board by sequentially scanning successive fields into different memory boards. Maximally eight memory boards can receive different fields.

Mixed Mode: this mode combines scan and group mode. Successive fields are scanned into groups of boards as opposed to into single board as in scan mode. This reduces latency when compared to the scan mode.

3.3 An Industrial Robot Manipulator

An advanced robot controller based on the XP/DCS is under construction for both testing new robot control algorithms and investigating issues of distributed real time control. For this task the GMF Robotics Model A-500, a four degree of freedom SCARA type arm, was chosen as the target mechanical unit. A more detailed account of this work may be found in [14].

Like virtually all currently 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. For our experiments it is necessary to be able to directly and independently specify the torque being delivered by each joint of the robot. Since the original control system does not allow this type of interface at any level, it was necessary replace the manufacturer's control system with our own system. For each of the robot's joints, the new interface consists of a dedicated XP/DCS node which directly commutates (in software) the currents in the DC brushless motors at the robot joints. The system block diagram for a single joint is shown in Figure 3.

A primitive protocol has been defined for the interface between the servo transputer and the control network, thus abstracting them as "perfect actuators" which report their state in floating point units of radians, and receive torque commands in floating point units of Newton-meters. The simple servo I/O structure gives the designer a clean interface to experiment with high level control algorithm design and implement arbitrary network topologies using standard INMOS compatible nodes. We have found this approach to offer a powerful and flexible environment superior to bus-based multiprocessing environments. The speed and simplicity of processor interconnection has proven indispensible in the ease of rapid prototyping and testing of network concepts. The relation of the control network to the low level servo nodes is illustrated in Figure 4 which depicts a topology we have found particularly well suited to implementations of the "computed torque" algorithm.

4 Conclusion

The XP/DCS has considerably advanced our real-time control capabilities within the Yale Robotics Laboratory both with respect to computational power and simplicity of deployment. We have sketched in Section 2.2 a hardware design based upon the INMOS Transputer yielding a large number of 1.5 Mflop engines, each possessed of zero wait-state direct memory mapped I/O capability whose cost (at time of writing) is less than $2000 each. Since typical robotic actuators and intelligent sensors cost at least twice this amount our strategy of assigning (at least) an individual XP/DCS node to each such device can be easily defended against criticisms of profligacy in today's research environment. Given the historically accelerated decreases in electronics cost relative to mechanical hardware we may confidently predict this strategy's commercial cost effectiveness as well within a few years time. We have provided an incomplete but illustrative review of commercial alternatives to the XP/DCS strategy in Section 2.1 suggesting that our design is less than half the present cost of anything comparable to be found on the market. More critically, as we point out in that section, given the intended level of granularity in deployment, our Transputer based design seems to offer the only alternative to the contradiction between more processors and higher communications traffic intrinsic to shared bus parallelism.

A uniform, granular, computationally powerful system of individual nodes with completely reconfigurable interconnection network focusses research attention upon innovation in algorithm design and physical experimentation rather than code optimization, operating system hacks, and interface hardware development. The diverse applications presented in Section 3 — a standard industrial manipulator; a frame-rate stereo vision system; a novel juggling robot — suggest just how versatile this notion of a "laboratory workhorse" can be. Since each physical device with its dedicated XP/DCS node has the computational identity of yet another Transputer in the network, the standard INMOS concurrent development environment and network debugging tools may be applied directly. Once a device has been "wedded" to its computational mate, the integration or re-assignment to another assembly or experimental apparatus is no harder than disconnecting and reconnecting the twisted pair or fiber-optic cables that form the network links.

There do, however, seem to be a number of new theoretical issues concerning the efficacy of real-time controllers whose only mode of communication consists of messages passed around a distributed network (as opposed, for example, to the possibility of "instantaneous" broadcasts or shared memory schemes made possible by the single bus architectures we rejected in Section 2.1). In a recent paper [14] we have added to the familiar and contrasting notions of *update rate* versus *latency* the slightly more refined distinction between *self latency* and *cross latency* in a distributed real-time message passing controller. Self latency is, roughly speaking, the average time it takes a given node (or set of nodes) in the control network to adjust the output commands it is producing to the nature of new sensory data received locally. Cross latency, by contrast, is the average time it takes a given node (or set of nodes) to adjust the output commands it is producing in response to sensory data received at a distant node (or set of nodes) and transmitted via a message across the network. We have found experimentally that slight variations in code distribution and message passing protocols can have dramatically different effect upon the resulting *network latency matrix* . It is to be expected that the the network latency matrix — a much more complicated variation on the familiar theme of controller time constants — will, in turn, have a noticeable impact upon the resulting closed loop behavior of the controlled system. We suspect that the effective design of real-time controllers built from distributed message passing nodes, attractive though such hardware may be, will require a careful analysis of which network latency properties are desirable and how they may be guaranteed by appropriate principles of synthesis.

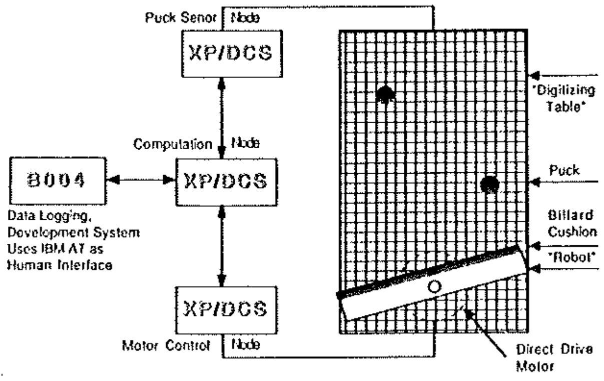

Figure 1: The Yale Juggler

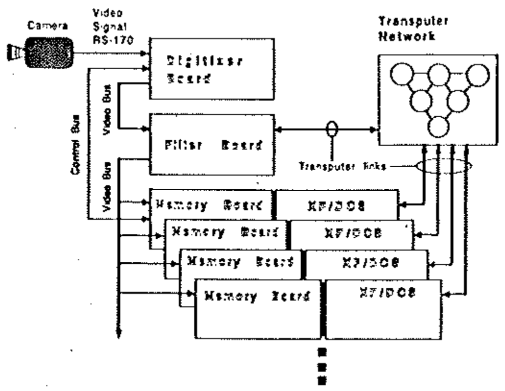

Figure 2: The Cyclops Vision System

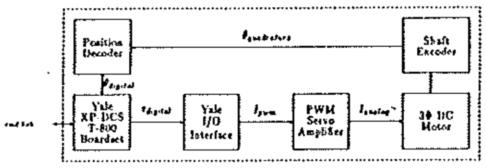

Figure 3: Servo Block Diagram.

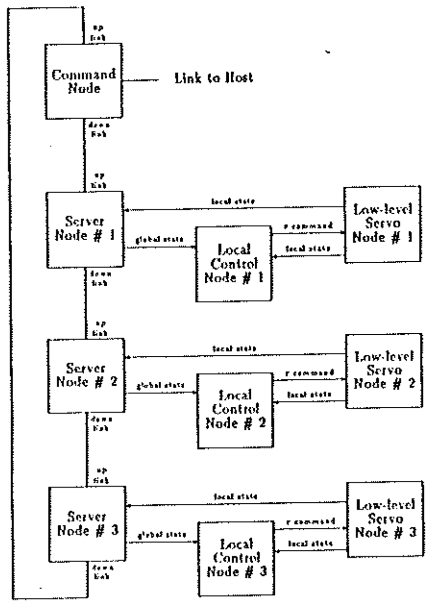

Figure 4: A Sample Control Network Topology.

Acknowledgements

We would like to thank Professors Peter Kindlmann and Alfred Ganz for numerous technical and philosophical contributions to the design reported here, as well as their continuing generous pedagogical efforts on our behalf.

References

[1] M. Bühler, D. E. Koditschek, and P.J. Kindlmann. A family of robot control strategies for intermittent dynamical environments. In *Proc. IEEE International Conference on Robotics and Automation*, (to appear), Arizona, May 1989.

[2] M. Bühler, L. Whitcomb, F. Levin, and D. E. Koditschek. A new distributed real-time controller for robotics applications. In *Proc. 34th IEEE Computer Society International Conference — COMPCON*, pages 63–68, IEEE Computer Society Press, San Francisco, CA, Feb 1989.

[3] Datacube. *Max Video User's Manual*. Technical Report, Datacube, Inc., 4 Dearborn Rd., Peabody, MA 01960, 1986.

[4] F. Levin and M. Bühler and L. Whitcomb and D. E. Koditschek. Transputer computer juggles real-time robotics. *Electronic Systems Design*, 19(2):77–82, Feb 1989.

[5] Gene F. Franklin and J. David Powell. *Digital Control of Dynamic Systems*. Addison-Wesley, Reading, MA, 1980.

[6] J. M. Hollerbach. A recursive formulation of manipulator dynamics and a comparative study of dynamics formulation and complexity. In Brady et al., editor, *Robot Motion*, pages 73–87, MIT Press, 1982.

[7] D. E. Koditschek and M. Bühler. Analysis of a simplified hopping robot. *Int. J. Rob. Research*, (to appear), 1989 .

[8] Daniel E. Koditschek and Elon Rimon. Robot navigation functions on manifolds with boundary. *Advances in Applied Mathematics*, (to appear).

[9] J. H. Lang, G. C. Verghese, and M. Ilic-Spong. Opportunities in estimation and control of electrical motors. In *Proc. 25th IEEE Conference on Decision and Control*, Athens, Greece, Dec 1986.

[10] Motion Analysis. *RTEV Real-Time Camera Manual*. Technical Report, Motion Analysis, Inc., 3650 N. Laughlin Rd., Santa Rosa, CA 95403, 1986.

[11] Per Stenström. Reducing contention in shared-memory multiprocessors. *Computer*, 21(11):26–37, Nov 1988.

[12] Jacob T. Schwartz and Micha Sharir. *On the "Piano Movers" Problem I. The Case of a Two-Dimensional Rigid Polygonal Body Moving Amidst Polygonal Barriers*. Technical Report 39, N.Y.U. Courant Institute Department of Computer Science, New York, 1981.

[13] C. J. Taylor, N. Vlamis, M. Bühler, and A. Ganz. The cyclops vision system. In *Proc. North American Transputer Users Group Meeting*, Salt Lake City, UT, Apr 1989.

[14] Louis L. Whitcomb, M. Bühler, and D. E. Koditschek. Preliminary experiments real-time distributed motion control. In *Proc. North American Transputer Users Group*, NY, Oct 1988.