A New Distributed Real-Time Controller for Robotics Applications

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

Abstract

This paper describes a dual board real time distributed control module based on the INMOS T414/T800 Transputers. The CPU board provides fast external memory, support for the four 10 MHz serial Transputer links including two fiber optic links, and an I/O expansion connector. The board's backplane connector is pin compatible with the INMOS ITEM Development System. The plug-in I/O board provides a bidirectional latched 32 bit I/O bus with full handshaking support. Half of this board is allotted to a wire-wrap prototyping area allowing for easy customization to specific I/O needs.

An easily configurable network built from this low cost modular design should be able to tackle the most demanding real time control applications, with respect to computation as well as I/O requirements. We describe two particular applications presently underway in the Yale Robotics Laboratory. The first is a three node network that supports a simple juggling robot. The second is a twelve node network for control of a standard industrial robot manipulator.

1 Introduction

Advances in theoretical understanding of the dynamical properties of nonlinear mechanical systems [7, 13, 14, 15, 23], and some recent empirical experience with many-degree-of-freedom robot arms [2, 9] suggest that the time is approaching when their control will be as routine a matter as that of comparable linear time invariant systems. It begins to seem as though the most critical present obstacles to such a possibility lie in the practical realm of actuator and computational technology. This paper addresses an approach to the constraints of the latter domain.

We document the specifications and preliminary performance characteristics of a low-cost computer intended as the representative prototype of a single node in a highly interconnected parallel computing and sensing environment dedicated to the real-time control of a complex nonlinear system. While there is no single aspect of this system which cannot be equalled (or even bettered) by some other commercially existing (or soon to exist) device, we feel that the combination of flexibility in inter-processor connection schemes, simplicity of programming and convenience of development environment, and raw computational and I/O speed, along with its very low cost make it an ideal vehicle for exploring the practical computational issues in robot control alluded to above. We presume that the same considerations lend this computer a certain attraction within the industrial world as well.

We continue this introduction with a brief review of computational issues in robotics and a comparison of some alternative technologies for addressing those issues. The second section of the paper provides a short account of the advantages to this approach, a physical description of the prototype system, and a description of the improved fabricated dual board — the Yale XP/DCS Version 1.0. Finally some application examples are discussed in the last section.

1.1 Computational Needs in Robotics

A generally accepted model for robot arm dynamics takes the form

$$M(q)\ddot{q} + B(q,\dot{q})\dot{q} + k(q) = \tau \qquad (1)$$

where q is a vector of joint meaurements, τ is a vector of control torques or forces exerted on each joint by the actuators, M arises from the "inertial" properties of the links, B, from the coriolis and centripetal forces of motion, and k represents the gravitational forces. Even for kinematically simple robots, the entries of the matrices M, B, and the vector, k, are extremely complex high order polynomials in transcendental functions of the joint variables, q. An important early study by Bejczy [3] shows that these nonlinear terms are not simply analytical artifacts, but represent cross-coupling forces which may vary by as much as three orders of magnitude over the robot workspace.

As of this writing, the most widely discussed control algorithm to achieve "robot tracking" around some à priori specified reference trajectory, q_d, is the so-called "computed torque" or

[1]This work is supported in part by Inmos Corporation, GMF Robotics Corporation, PMI Corporation, 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.

"inverse dynamics" algorithm [6, 19, 24],

$$\tau \triangleq k(q) + B(q,\dot{q})\dot{q} + M(q)\left[\ddot{q}_d + K_2(\dot{q}_d - \dot{q}) + K_1(q_d - q)\right], \quad (2)$$

which corresponds to the general linear strategy of pole-placement via state feedback, preceded by a forward-loop compensator intended to invert dynamics of the feedback compensated plant. Implementation of this strategy evidently requires computation of the full robot dynamical model (1) — on the face of it, roughly 10^5 flops for a six degree of freedom arm — in real time. Work by Hollerbach [7] indicates that the full model may be computed at the cost of only 10^3 flops by taking advantage of the intrinsic structure of the dynamics, and a recent analysis by Lathrop [17] indicates a great deal of this computation may be implemented in parallel. It should be noted as well, that each sample requires two input measurements (a position and a velocity) and one output command (desired torque or force) per degree of freedom.

In fact, algorithm (2) represents only one of a variety of different approaches to the robot tracking problem, which itself, is only one of many control problems which arise within the context of robotics. Even within the tracking paradigm, if one does not assume à priori knowledge of the robot link and robot load dynamical parameters (i.e. mass, centroid, moment of inertia matrix) then an adaptive version of algorithm (2) [10, 22, 11] might be attempted, and the computational cost would greatly increase. On the other hand, one might depart entirely from this paradigm in favor of the "natural control" methodology [14, 12] wherein a dynamical "reference model" is substituted for the reference signal, q_d. In this paradigm, off-line planning is almost eliminated and the real-time controller is responsible for almost all computation. For example, in robot obstacle avoidance problems, the natural controller is required to compute the gradient vector of a complex "navigation function" [16]. Previous research demonstrates that this problem is PSpace hard [20], thus the computational complexity of any such algorithm may be expected to increase exponentially with the number of robot degrees of freedom. The prospect of computing feedback algorithms which grow exponentially with the control problem raises unprecedented demands for performance and expandability of the real-time computational engine.

To the complexity of robot control algorithms must be added the burden of intelligent sensing. In addition to position and velocity of the robot's joints, most interesting robotics applications will involve such data intensive sensory modalities as vision or distributed contact forces. These add a considerable computational and I/O load to the previously described performance specifications.

In summary, the computational complexity of algorithm (2) represents merely a lower bound on what we might expect out of our real-time robot controllers. Moreover, it seems safe to assert that no fixed computational capacity will suffice given the ineluctable desire for additional mechanical degrees of freedom and more sophisticated sensory capabilities. It seems useful, however, to keep (2) in mind when considering general problems of robot control.

1.2 Sampling Rates for Distributed Architectures

If we take (2) as the minimal exemplar of the real-time computational load, then there are some 10^3 flops for each sampling period required from a centralized processor. In order to complete the analysis of the computational power required to implement real-time robot control algorithms it is now necessary to specify a target sampling rate.

A large body of theory presently exists concerning discrete time control of a continuous linear time invariant systems. In particular, according to the Nyquist Sampling Theorem, we know that the sampling rate must be at least twice the highest frequency of the bandwidth over which the system is to be controlled. For practical considerations, one generally picks a factor of ten [5]. In contrast, there is no widely applicable understanding of how to control continuous nonlinear systems with discrete controller. The issue of sampling rate is complicated since the very notion of a "time constant" is not viable for systems whose coefficients change with time or state such as the one characterized by (1).

A reasonable rule of thumb is that most dc servo motors mechanically coupled to links of representative mass have individual time constants of between 20 and 100 msec. Thus, from the point of view of linear theory, it should suffice to sample each motor at least every 10 msec. Indeed, out of the great number of commercially available robots today, we are unaware of any which achieve a sampling rate much faster than 100 Hz. In fact. there is some justification (beyond commercial viability) for this circumstance, since (to the best of our knowledge) all existing commercial robots employ control schemes which ignore the rigid body dynamics (1), and assume that the robot consists of a set of dynamically decoupled linear servo motors. No such justification, however, may be given for control schemes like algorithm (2). While reasonable heuristic arguments may be given for computing certain components of such strategies at much higher rates than others [8], there might always remain the nagging possibility that failures or lackluster improvements in robot performance attending the implementation of sophisticated control schemes arise from limitations of sampling.

It is widely accepted in the field of robotics that parallel processing represents the only reasonable approach to the computational requirements as represented above. We will advance the argument in the sequel that from the point of view of cost, reliability, and expandability, distributed parallel architectures hold the most practical promise for robot controllers. However, this recourse further complicates the question of sampling rate: communications occur not merely across the robot input and output channels but "inside the computer" as well between individual distributed processing components. Thus, each motor experiences a *local* sampling rate dictated by the external I/O speeds of its individual supervisory node, while the overall coupled system of motors and nodes gives rise to a more complex set of *cross latencies* which characterize the "freshness" of the data from the i^{th} node as used to compute the new values at the j^{th} node.

Our point of view is that advances in technology should obviate the need for any consideration of the effects of discrete time

controllers insofar as local sampling rate is concerned. If sufficiently fast digital computer and sensory instruments are available for each local node, then we may implement what appears to the analogue controlled system as a continuous control signal. The effect of the cross latencies introduced by our distributed architecture upon the overall stability and performance of the coupled closed loop system is a more subtle phenomenon, which we have begun to study independently [25].

For the present purposes of specifying the computational capabilities of our distributed engine, we set the goal of supporting the minimal computational load and all the low level I/O with one node (although we may use more than one for such a task in practice). We set the (intuitive, rather than analytically justifiable) goal of sampling at two orders of magnitude above the Nyquist rate of the robot's motors. We presume a six degree of freedom robot with joint sensors which deliver position and torque information for each degree of freedom and a separate power amplifier for each which accepts desired torque commands from the controller. Thus, each node must be capable of delivering at least 10^3 flops, reading from and writing to 18 parallel I/O channels at a rate of 1 KHz.

1.3 Commercially Available Technology

The architecture reported here represents a compromise between the seemingly endless horizon of contemporary hardware power and the limited ability of a small research group with many other pressing priorities to gain the advantages thereof. This limitation translates into the decision to restrict attention to reasonably cheap, commercially available hardware components with facile and well documented development environments. Since our situation is by no means unusual within the engineering community, this account may be of interest on more than a purely technical level.

Before examining hardware options within that narrower context, it is worth reviewing the requirements arising from the the domain of application discussed in the previous section. In general, the overall system should consist of simple and inexpensive nodes. Inter-node communication ought to be fast, since, as has been noted in the previous section, real-time control applications are as much data driven as computationally bound. The communication scheme must be easily reconfigurable, and node expandible with minimal deterioration of bandwidth. Such flexibility with respect to size and interconnection pattern allows for a variety of applications, and admits comparison of competing algorithms or different computational distributions of the same algorithm — all critical aspects of the experimentation we envision. The performance of a single node should be close to or exceed that of a "standalone" commercial microprocessor in order to achieve the desired computation rate with the minimal number of nodes. Furthermore, since robot sensor and actuator technology is in a state of rapid flux, each node must be easily adaptable to a variety of analog and digital inputs and outputs in an expandable fashion. Finally, the parallel hardware has to be matched with a suitable language which supports parallelism as well as a development environment for creating, distributing and debugging code.

Truly *parallel systems* employing a distributed architecture amenable to multiple instruction flow are already commercially available (e.g., Floating Point Systems, Intel (hypercube), etc.). While these systems typically exhibit great processing power they often have limited internode communication bandwidth or inadequate I/O capability. Cost represents a major consideration in our project as well as in industrial robot control systems: investments in the range of up to several hundred thousands dollars disqualify commercially available distributed networks for our purposes. On the other hand, *Array processors*, offered by companies like Systolic Systems, Marinco Computer Procucts or Mercury Computer Systems, represent an alternative means of boosting computing power for conventional mainframes or even personal computers, to which they interface via memory map. They commonly feature rates between one and five million floating point operations per second, substantial user libraries and cost around $5000. Unfortunately, their I/O capability is typically limited. Moreover, array processors cannot afford the ease of expandability of truly distributed architectures, nor can they admit experimentation with interprocess communication and synchronization.

The innerent caveats of purchased systems (i.e. cost, manufacturer dependence, inflexibility, "non-transparency", hardware overhead) may be eliminated by recourse to customized design, and an increasing number of such "homemade" distributed, real time controllers, are indeed being built. Generally these systems evolve around off the shelf commercial microprocessors, e.g. one of TI's TMS320, Motorola's 68000 or National Semiconductor's 32000 family. Of course, a number of commercially developed single board computers based on these processors are becoming available as well.

Whichever of these processors (or of the myriad number of chips similar in architecture) 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). Moreover, due to the lack of a suitable language, software issues become more and more problematic as the hardware increasingly exploits parallelism.

In this paper, we present a solution to the problem of real-time distributed control based upon the INMOS Transputer Chips We describe the development of the XP/DCS, a real-time control node with the floating point computational power, internode communication bandwidth, I/O interface and development environment required for advanced distributed control applications. The CPU board provides fast external memory, support for the four 10 MHz serial Transputer links including two fiber optic links, and an I/O expansion connector. The board's backplane connector is pin compatible with the INMOS ITEM Development System. The plug-in I/O board provides a bidirectional latched 32 bit I/O bus with full handshaking support. Half of this board is allotted to a wire-wrap prototyping area allowing for easy customization to specific I/O needs. The cost of the board set at the time of writing is slightly over $2000.

2 Design: Philosophy and Prototypes

The choice of the INMOS product line represents a strategy which standardizes and places the burden of parallelism — interprocessor communications support, software, and development environment — around a commercial product, while customizing the computational "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 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 System Service connections). 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.

The Transputer Development System (TDS), consisting of an evaluation board (B004) and supporting software and documentation, satisfies the need for a coherent prototyping environment. Using an IBM AT, or a more powerful engineering workstation (e.g. Sun, Apollo, Vax, etc.) as a host, the user can generate, debug, compile, and download code to a target node or group of nodes. The network configuration utility included in TDS minimizes the software changes attending the addition of a node: the new node's name (i.e. PROCESSOR 5 T8) along with processes targeted to run on it are adjoined to an existing network configuration file; assignment of download link and interprocessor data transfer links completes the specification. The OCCAM compiler contained within the TDS supports the creation of processes that use "channels" for communication. These soft channels are mapped into physical links when a program is configured. Given this capability, programs intended for a particular interconnection scheme using a specified number of nodes can be simulated on variant networks, or even a single transputer, if desired.

In summary, from our point of view, the Transputer's primary advantage over the other comparable CPU's mentioned in the previous section is its intrinsic inter-processor communications capability. This capability is notable both with respect to hardware performance as well as the relative sophistication of the commercially provided development environment for parallel and concurrent applications. We now describe the present state of evolution of a board design for a real-time motion controller based upon this technology. In Section 2.1 we describe some hardware studies designed to convince us of the feasibility of achieving "customized computational identity" at various nodes via co-processor technology. In Section 2.2 we describe the standardized PC board set with homogeneous computational capability (based solely upon the Inmos T800 CPU) and "customized I/O identity" which has become our general purpose laboratory work horse for real-time control applications.

2.1 The Prototype Version: Hardware Exploration

The addition of special purpose coprocessors to particular nodes represents an integral aspect of the hardware development path we have chosen to pursue. The advent of the application specific integrated circuit (ASIC), and associated design tools make the consideration of future customized coprocessors an attractive alternative versus adjusting algorithms to the limitations of current (or near term) hardware. For the XP/DCS Prototype we selected the well known WEITEK 1164,65 64/32 bit IEEE floating point Multiplier and ALU together with the T414 Transputer. This served the dual purpose of an exercise in transputer/coprocessor design as well as boosting the T414's floating point computational capabilities which is of central importance in robotics. At the time this work was begun, around summer of 1986, the T414's software floating point processing ability fell short of our computational needs. INMOS had announced the T800 [4], the next generation transputer with on-chip floating point support, but the date of availability was unclear.

A detailed report on this first generation design is available in a separate technical report [18]. For the purposes of this paper, we offer a summary feature list for the XP/DCS Prototype as follows: the T414 15 MHz 32 bit RISC microprocessor with four 10Mbps serial communication links; 30K bytes of external zero wait state static RAM; WEITEK 1164/1165 IEEE 64/32 bit floating point chip set; Five 32 bit I/O ports; Eight individually addressable flags.

2.2 The Second Generation: XP/DCS Version 1.0

The previously described prototype had been in daily operation for almost one year in a three node configuration controlling a juggling apparatus in the Yale Robotics Laboratory described in the next section. Over that time, we upgraded the nodes as experience dictated and as the availability of appropriate new technology allowed. It became clear that easy customization of a node's I/O capability was as important as tailoring its computational characteristics. We set about in our second generation design to achieve a general purpose I/O board which would standardize the interface between our computational network and the physical world. The essential features of this new board include an upgrade from T414 to T800 processor, the addition of fiber optic link interface capability, and modularization into a mother-daughter board set. The first feature provides reasonable computational power (1 Mflop) with no special purpose coprocessors. The second improvement turns out to be a vital step in so EMI hostile an environment as an electrical servo actuated robot. Fiber optic technology meant the long awaited end to interference problems and impedance matching "artistry" for the 7m communication lines which link the transputer network. The main structural innovation is the clear separation between computation and I/O on two separate boards. All I/O functions (and eventually, any coprocessors will) reside on a "daughter card"

which plugs into a "bus expansion connector" on the "mother card". The latter — the CPU board proper — can therefore be used alone as a floating point node or with an attached I/O card as a complete data acquisition and control node.

2.2.1 The XP/DCS CPU board

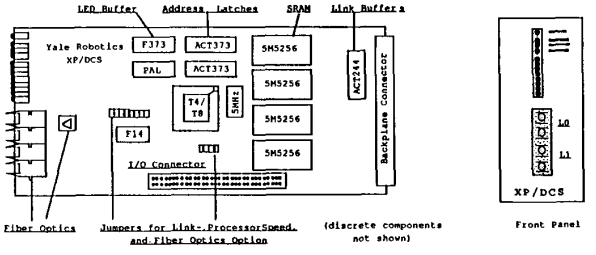

Figure 1: XP/DCS - CPU board

The high line density typical for 32 bit designs dictated the use of a *four layer Printed Circuit Board* with the attending benefits of reduced ground noise and enhanced signal integrity (the prototype was plagued by electrical and mechanical wire wrap failures). We 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.

The CPU board supports both the T800 as well as the T414 at jumper selectable clock speeds of 15-20Mhz. A power-up reset signal provided on the board ensures that the processor is reset properly at power-up. Without this provision, random external bus accesses may occur until a program is downloaded. This is especially bad for I/O devices which may thus be accessed arbitrarily and cause potential disaster.

This version is much simpler than the prototype. The advent of the T800 with its on chip floating point support has eliminated the need for the WEITEK chipset and its support circuitry. Second, we have moved all TTL logic into one PAL. Third, all I/O related functions have moved to the I/O board. This halves the chip count for the CPU board to 12 devices.

By exploiting the development of fast 32k x 8 SRAMS we have *increased the local memory* by a factor of four compared to the prototype to 128K bytes of which 124K bytes (for T8) or 126K bytes (for T4) are accessible. To satisfy the requirement of zero wait states, 70 ns chips from Mitsubishi are used. As before each XP/DCS node has access to the host's (larger) memory through a serial link connection. We also keep open the option to add additional local SRAM memory on the I/O card. Only further experience with real life control situations will determine if larger local memory is needed.

The Transputer's four high speed *serial links* are made available on the rear edge connector compatible to the INMOS ITEM boards. The link speeds are user selected. All inputs are Schottky diode clamped. For purposes of improving the impedance matching in critical cases the output lines' series resistances can be altered. 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.

To serve as *visual software status indicators* or aid in system debugging, eight LEDs are located at the front edge of the card. The *system service* lines to and from the Transputer pass through the rear edge connector and have pin-outs compatible with those in the INMOS ITEM unit. All three, Error, Analyse, and Reset are visually displayed by the LEDs.

The *I/O connector* (DIN type 'B') 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 kind of add-on board can to be attached that interfaces to the transputer memory interface. However it is required that all signals be buffered after the I/O connector.

The *performance* of this CPU board is determined by the Transputer employed and the number of wait states for external memory cycles. As we use zero wait states memory (for CPU speeds up to 20MHz) all the performance measures given by INMOS in [21] for the transputer apply for our CPU board, as well.

As mentioned above, the XP/DCS CPU board is *pin compatible* with all existing INMOS evaluation products using the B201-1 10 slot card cage. The I/O daughter board derives its power from the CPU board . Analog supply voltages are passed from the rear edge connector to the I/O connector on several pins. These rear edge pins are not connected in the standard INMOS ITEM chassis.

2.2.2 The XP/DCS I/O board

The first I/O innovation was, as mentioned above, to separate computational and I/O functionality. Second, the test period with the prototype has shown it had more digital I/O than we ever needed, and was too inflexible, providing a fixed number (5) of input and output ports, each 32 bits wide. Moreover, virtually every I/O device is now available with tri state output, eliminating the need for a separate I/O latch for each unit, and permiting operation in a parallel fashion via the same I/O latch.

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

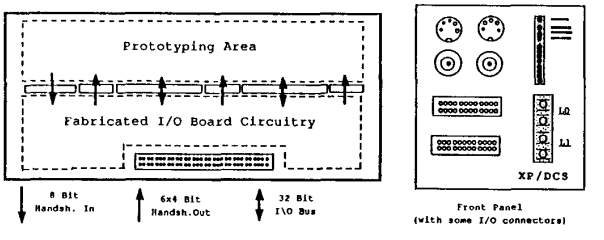

Figure 2: XP/DCS - I/O board

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. Recall that a typical robotic application, as outlined in the first section, requires 18 I/O operations and 1000 flops per sampling time. Thus we can easily see, that with these timing considerations the I/O time requirements are negligeible compared to the computational requirements. The typical three external memory cycles are: (1)address/enable device via handshaking out; (2) read from / write to device (if necessary, after delay); (3) disable device.

However, in general (for example for some ASICS, or if one needs the faster I/O cycle time — generally a gain of less than 1μsec), the board allows for direct bus interfacing to the transputer called *synchronous* I/O mode. This is possible by removing the bidirectional bus latches. However, no default latched mode is simultaneously possible and no further support is provided on board for this mode which is not considered the default use.

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 very briefly review our experiences with two particular pieces of robotic apparatus.

3.1 Control of a Juggling Apparatus

One line of research we pursue is directed towards understanding underlying principles in modeling, analyzing and controlling robots that repetitively catch or throw, hop, run — or juggle. For empirical validation of our theoretical models and as a source of insight, we have built a simple two dimensional juggling apparatus. The physical system 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 Figure 3. The figure depicts the simple three node XP/DCS newtork we employ in this application consisting of an intelligent sensing node to estimate puck positions and velocities from the digitizing table, a motor controller node, and a host/logging node.

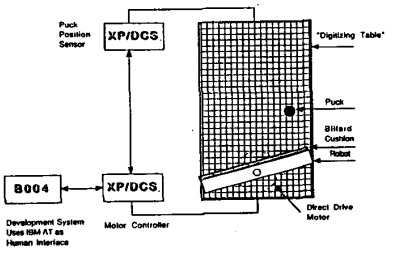

Figure 3: The Yale Juggler

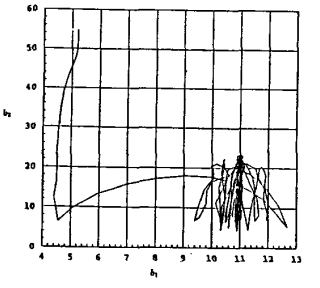

Figure 4: Convergence to a Vertical One-Juggle.

Figure 4, is a "recording" of a successful "vertical one-juggle" using an algorithm reported in [1]. The plot shows the actual horizontal (b_1-axis) and vertical (b_2-axis) trajectory of a planar free-falling puck subjected to a series of juggling impacts by the robot. The puck is controlled from the initial drop point (upper left) to a stable pure vertical periodic orbit.

3.2 An Advanced XP/DCS Based Robot Controller

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 shown in Figure 5, was chosen as the target mechanical unit.

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 INMOS Transputer 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 6

The servo transputers provide a clean interface to each actuator, thus freeing the designer of the control network from low level operational requirements of the particular motors used. Thus the

architecture of the control network will be dictated solely by the structure of the particular control algorithm chosen. We now briefly describe an advanced control algorithm and a distributed transputer network which realizes the algorithm.

Our first experiments will be directed toward computation of the computed torque strategy (2) discussed in the introduction. Our present implementation has each servo processor communicating directly to two processors: a dedicated computation processor and a dedicated communication server processor. The server processors communicate in a ring topology. Communicating directly with the i'th low level servo is the i'th computation node which computes the subexpression of equation 2 associated with the the i'th joint. The local computation consists of the entire feedback gain calculation, and the i'th rows of both the $M(q)$ and $B(q,\dot{q})$ matrices, followed by the appropriate multiplication and summation. The local computation receives the i'th state information directly from the i'th servo, and the $j, j \neq i$'th state information as well as the reference information from a dedicated server node. This local computation is currently executed on a single T-800, but one might imagine further granularization of this process. Each low level servo node also *reports* its state directly to its local server node which forwards this data to the remaining servers. An additional command server receives reference trajectory commands from the host and forwards this data to the servers. This topology is illustrated in Figure 7.

Since the servo control boards of Figure 6 are not yet complete at the time of writing, our epxerimentation to date has been limited to timing measurements of algorithm (2) implemented

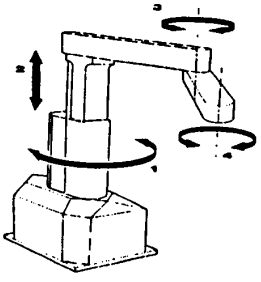

Figure 5: The GMF Model A-500

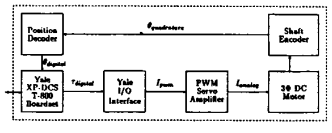

Figure 6: Servo Block Diagram.

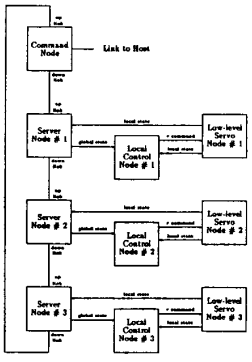

Figure 7: The Control Network Topology.

upon the network illustrated in Figure 7 with the low-level nodes generating simulated commutation commands and robot state information. These experiments are reported in some detail in [25], and it will suffice here merely to display a variety of "cross latency" matrices (as defined in the introduction of this paper) in Figures 8, 9, and 10, which obtain from a variety of distribution and communications schemes. All measurements are in microseconds.

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

Figure 8: Network Cross Latencies: Buffered.

node	1	2	3
1	817	706	922
2	982	832	930
3	860	731	829

Figure 9: Network Latency: Unbuffered.

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

Figure 10: Network Latency: Self Buffering.

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 one degree of freedom juggler in a two degree of freedom environment. In *Proc. IEEE Conference on Intelligent Systems and Robots*, page , Tokyo, Japan, Oct 1988.

[2] C. H. An, C. G. Atkeson, J. D. Griffiths, and J. M. Hollerbach. Experimental evaluation of feedforward and computed torque control. In *Sixth CISM-IFToMM Symposium on Theory and Practice of Robots and Manipulators*, Sep 1986.

[3] Antal K. Bejczy. *Robot Arm Dynamics and Control.* Technical Report 33-699, Jet Propulsion Laboratory, Pasadena, CA, 1974.

[4] INMOS Corporation. *IMS T800 Transputer.* Product Overview 72 TRN 117 01, INMOS Corporation, Nov 1986.

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

[6] E. Freund. Fast nonlinear control with arbitrary pole placement for industrial robots and manipulators. *The International Journal of Robotics Research*, 1(1):65–78, 1983.

[7] 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.

[8] Oussama Khatib. Real time obstacle avoidance for manipulators and mobile robots. *The International Journal of Robotics Research*, 5(1):90–99, Spring 1986.

[9] 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.

[10] Daniel E. Koditschek. Adaptive strategies for the control of natural motion. In *IEEE Proceedings 24th Conference on Decision and Control*, pages 1405–1409, Fort Lauderdale, Dec 1985.

[11] Daniel E. Koditschek. Adaptive techniques for mechanical systems. In *Fifth Yale Workshop on Applications of Adaptive Systems Theory*, pages 259–265, Center for Systems Science, Yale University, New Haven, CT, May 1987.

[12] Daniel E. Koditschek. Automatic planning and control of robot natural motion via feedback. In Kumpati S. Narendra, editor, *Adaptive and Learning Systems: Theory and Applications*, pages 389–402, Plenum, 1986.

[13] Daniel E. Koditschek. High gain feedback and telerobotic tracking. In *Workshop on Space Telerobotics*, pages 355–363, Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA, Jan 1987.

[14] Daniel E. Koditschek. Natural motion for robot arms. In *IEEE Proceedings 23rd Conference on Decision and Control*, pages 733–735, Las Vegas, Dec 1984.

[15] Daniel E. Koditschek. Robot kinematics and coordinate transformations. In *IEEE Proceedings 24th Conference on Decision and Control*, pages 1–4, Fort Lauderdale, Dec 1985.

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

[17] R. H. Lathrop. Parallelism in manipulator dynamics. *Int. J. Robotics Res.*, 4(2):80–102, Summer 1985.

[18] F. Levin, M. Bühler, and D. E. Koditschek. *A Prototype Processing Cell for Distributed Real Time Control.* Technical Report 8701, Center for Systems Science, Yale University, Mar 1987 .

[19] J. Y. S. Luh, M. W. Walker, and R. P. Paul. Resolved acceleration control of mechanical manipulators. *IEEE Transaction on Automatic Control*, AC-25:468–474, 1980.

[20] 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.

[21] Roger Shepherd and Peter Thompson. *Lies, Damned Lies, and Benchmarks.* Technical Note 27 72 TCH 027 00, INMOS Corporation, Jul 1987.

[22] Jean-Jacques E. Slotine and Weiping Li. On the adaptive control of robot manipulators. In *Proc. ASME Winter Annual Meeting*, page , Anaheim, CA., Dec 1986.

[23] Morikazu Takegaki and Suguru Arimoto. A new feedback method for dynamic control of manipulators. *ASME Journal of Dynamics Systems,Measurement, and Control*, 102:119–125, 1981.

[24] T. J. Tarn, A. K. Bejczy, A. Isidori, and Y. Chen. Nonlinear feedback in robot arm control. In *Proc. 23rd IEEE Conference on Decision and Control*, pages 736–751, Las Vegas, Nev., Dec 1984.

[25] 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.