Preliminary Experiments in Real Time Distributed Robot Control. [1]

Preprint of a Paper to Appear in the
Proceedings of the 1988 North American Transputer Users Group Meeting
New York, New York
October, 1988

Center for Systems Science
Yale University, Department of Electrical Engineering

Louis L. Whitcomb, Martin Bühler, Daniel E. Koditschek

October 2, 1988

[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 last author.

Abstract

We investigate the computational needs of advanced real-time robot control. First, sampling rate issues in the control of nonlinear systems are discussed. Second, a representative nonlinear robot control algorithm using an explicit robot dynamical model is derived. Some typical terms of the exact equations are given for two industrial robot arms. Third, we define some performance criteria of interest in real-time control. Finally, we compare a variety of implementations of the above control algorithm on a network of INMOS Transputers.

1 Introduction: Real Time Computational Needs in Robotics

Advances in theoretical understanding of the dynamical properties of nonlinear mechanical systems [Hol82, Kod87, Kod84, Kod85, TA81], and some recent empirical experience with many-degree-of-freedom robot arms [AAGH86, KK86] 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.

A large body of theory now exists concerning discrete time control of a continuous linear time invariant dynamical system. 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 [FP80]. In contrast, there is no widely applicable understanding of how to control continuous nonlinear systems with a 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.

A generally accepted model for robot arm dynamics takes the form [1]

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

where q is a vector of joint measurements, 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, C, 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, C, and the vector, k, are extremely complex high order polynomials in transcendental functions of the joint variables, q. An important early study by Bejczy [Bej74] 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 generally discussed control algorithm to achieve "robot tracking" around some à priori specified reference trajectory, q_d, is the so-called "computed torque" or "inverse dynamics" algorithm [Fre83, LWP80, TBIC84],

$$\tau \stackrel{\triangle}{=} k(q) + C(q,\dot{q})\dot{q} + M(q)\left[\ddot{q}_d + K_2(\dot{q} - \dot{q}_d) + K_1(q - q_d)\right], \tag{2}$$

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 [Hol82] 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 [Lat85] 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.

[1] This equation is derived in appendix B.

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. 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) [Kod85, SL86]

might be attempted, and the computational cost would greatly increase. On the other hand, one might use a much simpler robust high-gain feedback scheme [Kod87] whose computational cost is correspondingly less. Analogously, most interesting robotics applications will involve additional sensory modalities such as vision or force, which will drive up the required I/O capability as well. It seems useful, however, to keep algorithm (2) in mind when considering general problems of robot control, and assume that each sampling interval will incur a cost of at least 10^3 floating point operations.

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 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 mS. Thus, from the point of view of linear theory, it should suffice to sample at least every 10 mS. 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. 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 [Kha86], 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.

Our point of view is that advances in technology will ultimately obviate the need for any consideration of the effects of discrete time controllers. [2] It would be possible to implement what appears to the controlled system as a continuous control law, if a sufficiently fast digital computer and sensory instrumentation were available. For the purposes of our research, we have set the (intuitive, rather than analytically justifiable) goal of sampling at two orders of magnitude above the Nyquist rate. In the case of a six degree of freedom arm this translates into instrumentation capable of delivering 10^3 flops and at least 18 I/O operations in less than 1 mS. Of course, this is merely a lower bound on desired overall performance. Apart from the demands of more computationally intensive control algorithms, and more sensory data collection, a longer term goal of our research is the integration of as much "intelligence" as possible at the level of control in order to relieve the higher level from inappropriate time consuming computation [Kod86]. Thus, a suitable real-time control architecture must admit the possibility of easy upward expansion both with regard to greater computation and I/O rates as well as "higher level" functioning.

It is becoming accepted in the field of robotics that parallel processing represents the only reasonable approach to expanding computational requirements as represented here. There arise immediately the inevitable questions concerning processor hardware, interconnection scheme, communications protocol, development environment, etc. In previous papers [LBK7 , LBK88] we have described the design of a new distributed real time control engine, the XP-DCS motherboard/daughterboard set, based upon the INMOS T-800. In this paper we report upon some preliminary experiments with a network of such nodes, targeting certain performance measures in the context of a particular robot control algorithm.

[2] Of course, the same cannot be said regarding the problem of discrete numbers. The effects of quantization errors may be expected to play a significant role in the behavior of controllers.

2 A Well Known Control Algorithm for Robotic Tracking

2.1 General Structure

The general structure of the closed loop system is, in picture form, the following:

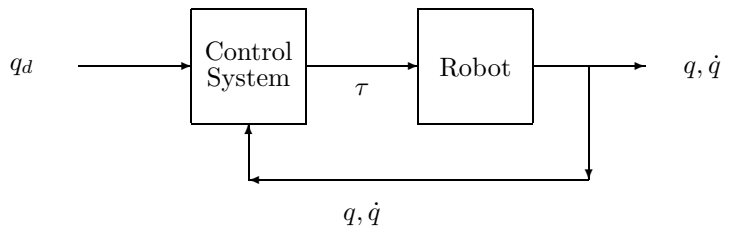

Figure 1: Closed Loop System.

The robot is described by the second order nonlinear differential equation given in 1, and is completely derived in appendix B. The controller, described by the relation given in 2, and is derived in appendix C. This is the so-called "computed torque control algorithm," which is well known in the robotics literature [Fre83, LWP80, TBIC84]. The relative merits of this procedure are a matter of real concern in implementing actual systems. For example, the algorithm assumes à priori knowledge of the complete inertial distribution of the robot, including payload, which is often difficult to obtain in practice. The purpose of this paper, however, is to investigate *implementation issues of real time control algorithms on distributed architectures.* As such we present the computed torque procedure not as being "superior" or "optimal", but simply as *representative* of a class of current state-of-the-art control algorithms.

2.2 Computer Derivation of Explicit Equations

Thus far we have written the control equations using the abstract notation of vector calculus. This allows for a simple and precise derivation of the equations of motion in very general fashion which reveals the general structure of the expression independent of a robot's number of joints or particular geometry. The previous equations hold for a large class of robots, including most of the currently available industrial units such as the Unimation PUMA 560, the GMF A-510, the Adept-1, and others. It is our belief that the *structure* of this class of expressions may be exploited for evaluation in real time on an appropriately designed parallel architecture. We hope to obviate the need for hand-optimization of code by constructing a control system which will offer predictable and satisfactory performance over the entire class of expressions. Thus we seek to design a control architecture which is independent of the specific details of a robot's kinematic type, and extensible to higher degrees of freedom in a uniform fashion.

Our intent is to *implement* real controllers. Given a general expression as developed in the previous section and a specific target robot, we must be able to derive the explicit equations which apply to that machine. An n jointed robot, in this model, is completely specified by a set of $14n$ parameters and n variables. First, each joint is classified as being either revolute or prismatic. Second, the frame transformation between successive joints is then uniquely specified by three kinematic parameters and one joint position variable. This serves to uniquely specify the geometric configuration of the robot. Describing the mass distribution of each link requires an additional ten parameters, and serves to uniquely specify the dynamical character of the robot. In practice, the $14n$ parameters are usually measured

experimentally, although in principal they could be calculated directly from a CAD data base. For computational simplicity, these parameters are often represented by an embedding in a set of n 4x4 kinematic matrices and n 4x4 dynamical matrices.

Given a completely specified robot, in the form of a set of kinematic and inertial parameters, the equations of motion are obtained by performing elementary algebraic and differential calculus operations. For simple robots of, say, two or three degrees of freedom this is a straightforward exercise in freshman calculus which involves a great deal of trigonometric simplification to reduce the intermediate and final equations to a relatively simple form. For degrees of freedom higher than about three or four the intermediate expressions in the derivation may easily involve thousands of terms which are polynomial in transcendentals and parameters. For higher degree of freedom robots it becomes impractical to derive the explicit equations by hand.

We have automated the explicit derivation procedure using the computer symbolic math package SMP [3] symbolic manipulation environment, though many of the available symbolic packages such as MACSYMA should be equally well suited.

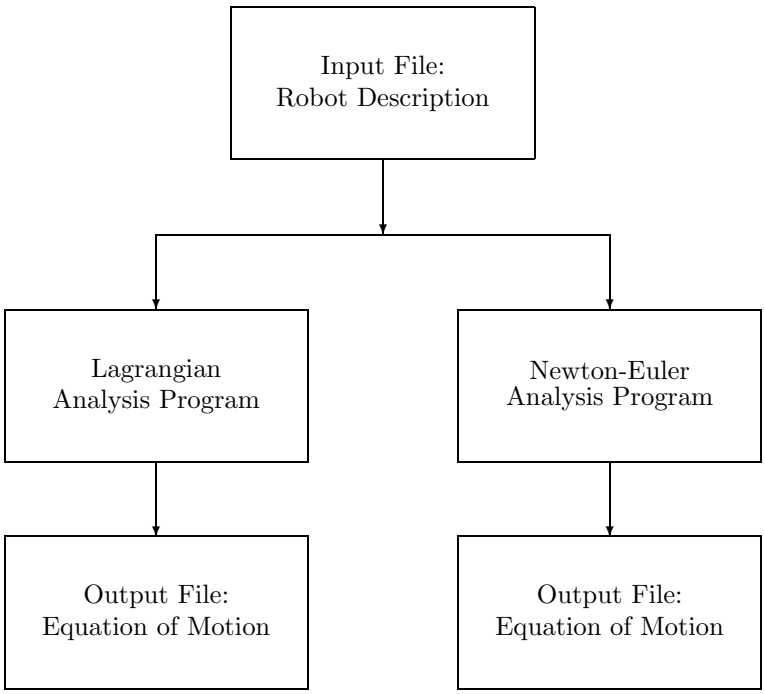

Figure 2: SMP Equation of Motion Derivation.

Verification of correctness of the resulting expressions is a problem. For all but simple robots it is virtually impossible to attempt manual verification. Two independent programs sharing no common code were written, each of which accept as input a file containing the kinematic and inertial matrices which completely characterize a robot and outputs a complete, simplified equation of motion for the robot. One program uses the exact lagrangian derivation which is presented in this paper. The other program uses a completely different approach, a Newton-Euler free body analysis [LWP80] method which is well known in the literature. The two programs have proven to be functionally equivalent in evaluating the equations of motion for all of the many robots tested, and the results for simple cases agree with our manual derivations. Appendix D contains a sample robot description file which completely characterizes

[3]Developed at California Institute of Technology by S. Wolfram, and commercially available from Inference Corporation.

a simple three degree of freedom robot. Finally, appendix F gives the actual (1,2) elements of the coriolis matrix generated by the SMP derivation program for both the simple robot of appendix D and an actual industrial robot.

3 Implementation on the Yale XP-DCS System

Our objective has been to implement the computed torque algorithm for a real robot. The GMF Robotics Model A-500, a four degree of freedom SCARA type arm shown in figure 3, was chosen as the target mechanical unit.

Figure 3: The GMF Model A-500

3.1 Hardware and Low Level Servos

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. It was therefore necessary replace the manufacturer's control system with our own low level interface. 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 4

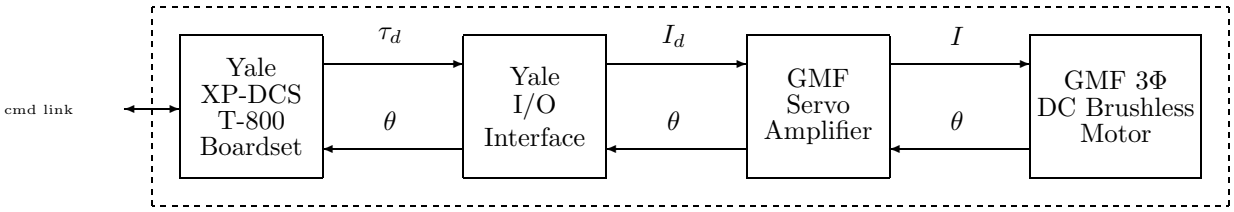

Figure 4: Servo Block Diagram.

3.2 Input/Output and Controller Structure

The servo transputer has five primary tasks:

- Receive torque commands from a higher level controller via standard INMOS links.
- Report current state information $(\theta_i, \dot{\theta}_i, \sin\theta_i, \cos\theta_i)$ to the higher level controller.
- Perform low-level commutation of currents in the three phases of the dc servo's windings.
- System monitoring and software safety interlock (a hardware safety interlock system is, of course, also implemented.)
- Servo initialization on power up.

The computer used in this servo is an XP-DCS, described previously, and control software is implemented in the OCCAM language under the TDS development environment. A prototype servo system realizing figure 4 has been constructed and tested, and successfully commutates the dc brushless motor at a frequency of 10Khz. Implementation for the remaining joints is currently underway.

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.

With the low level servo transputers providing a clean interface to the actuators, we now describe a transputer network which realizes the computed torque control algorithm described earlier in the paper. This control algorithm requires a *reference trajectory* which is presumably generated by a higher level host, see figure 1, and I/O to each servo, thus we arrive at the gross conceptual structure shown in figure 5.

3.3 Performance Criteria

Our performance objectives are to evaluate the control expression of equation 2 at sampling intervals on the order on 1 mS. The 1 mS target was chosen because it is about an order of magnitude faster than the frequency response of the servos themselves.

In the familiar world of single-cpu digital controllers we have a precise notion of "sampling interval" as a measureable quantity which is in good correspondence with our intuitive understanding. The "sampling interval" of a controller is generally agreed to be the time it takes for the controller (1) to read its input port,(2) evaluate a sequential control expression, and (3) assert the results on its output port. In the case of a controller based upon a distributed architecture, however, the notion of "sampling interval" is problematic because the network may have many I/O ports as well as computational elements which may operate asynchronously. A distributed controller might be conducting I/O operations at some nodes while evaluating control expressions at other nodes in an asynchronous and overlapping fashion - thus it is unclear how one might measure the classical "sampling interval" of a distributed controller.

We therefore define the concepts of "latency" and "update period" as measureable quantities associated with a variable in a node's memory. These concepts are well known in the literature as natural extensions of the "sampling period" concept to distributed systems networks. We view them as particularly important in the context of real time distributed controllers, as they offer perhaps the most fundamental benchmark for measuring a system's performance.

Update Period: Define *update period* of a variable in a node's memory space to be the average interval between successive assignments to that variable. This is includes both assignments by the node's cpu as well as by DMA devices such as links.

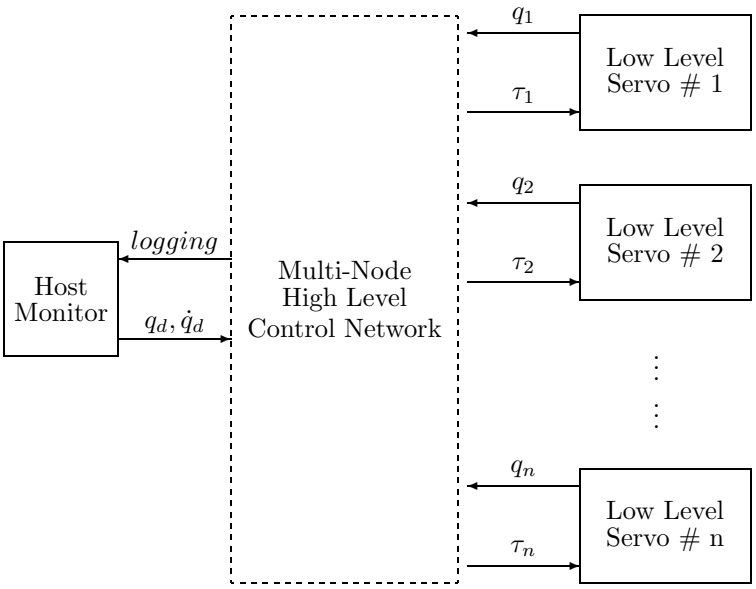

Figure 5: Implementation Gross Block Diagram.

Latency: Assume that, in the normal course of execution of a particular network, the value of variable y at a node is functionally dependent on the the value of a variable x which may reside somewhere else in the network, i.e. $y = f(x, \cdots)$. Define the *latency* between the two variables to be the average time taken for a value of x to manifest itself as a corresponding valid value assignment to y.

The definition of "update period" corresponds to our usual notion of system throughput. The definition of "latency" is an attempt to make more precise the notion of data latency through a network. It could be argued that the classical concept of "sampling period" is a special case of these definitions when applied to a single processor. The definitions are somewhat weak in their use of "average", by which we simply mean that these quantities are to be experimentally measured in the usual fashion with repeated trials.

With a control network of the general type pictured in figure 5 we now name some special instances of the above definitions which are of particular interest from a control viewpoint:

Self Latency: The latency between the local *state* variables of a low-level servo, and its associated *torque* command variable.

Cross Latency: The latency between the local *state* variable of one low-level servo and the associated *torque* command variable in a *another* servo.

Self latency is, roughly speaking, the average time it takes the control network to perform the following: (1) input a servo's state, (2) use the state value to compute a valid torque command, and (3) write the resulting torque command back to the servo. Cross latency is likewise the average time it takes the control network to input a servo i's state, use the state value to compute a valid torque command for joint j, and write the resulting torque command to servo j.

3.4 Partitioning the Control Equation

The explicit analytical control expression we have derived is a *function* whose *domain* is $5n$ dimensional space consisting of a $2n$ dimensional space of robot states plus a $3n$ dimensional space of reference trajectories, and whose *codomain* is the n dimensional space of joint torques. Thus we may accurately conceptualize our control problem as the repeated evaluation of a vector valued function

$$\tau = f(q, \dot{q}, q_d, \dot{q}_d, \ddot{q}_d) \tag{3}$$

where $\tau, q, \dot{q}, q_d, \dot{q}_d$, and $\ddot{q}_d$ are n dimensional vectors and f is the computed torque equation. One can imagine that any single-cpu controller's performance in this control task will degrade as the dimensionality of the robot increases. Alternatively, we can imagine an extensible system using many cpu's so that the system's computational power could be chosen to be commensurate with the computational burden. Such a system might, at the expense of the attendant hardware, offer satisfactory performance independent of system's dimensionality.

Our point of view is that for a large class of control problems, we *need not* attempt to solve the general computer science problem of distributed systems. Rather, we propose to identify a *class* of control algorithms which is sufficiently rich so as to admit desired performance of the resulting closed loop system, yet has an intrinsic structure which simplifies the task of partitioning the control burden across a distributed architecture.

Let us examine several approaches to implementing real time networks for evaluating functions of the general form equation 3 with the specific structure given in equation 2.

A popular approach to distributed computing of some computations is the so-called "processor farm" approach in which a given task is divided into discrete subtasks represented as data packets which are queued and routed through a network to the next available processor. For some computations, such as

ray tracing, this approach provides for nearly 100% processor utilization and the system performance increases linearly with the number of processors. This approach tends to provide extremely high data update rates ("throughput") at the expense of rather lengthy and unpredictable latencies. The latencies are a result of the time a subtask packet may spend being routed through the network or in queue. Real-time control systems, where we are specifically interested in minimizing system latency, do not appear to lend themselves to this type of implementation.

Alternatively, one might imagine partitioning the computation and assigning specific subtasks to particular processors. Promising applications of this approach include finite element modeling and numerical solutions to partial differential equations on appropriately dimensioned networks. In the robotics context, vector valued functions of the form in equation 3 might easily be mapped onto n processors, each of which evaluates the i'th component of the vector valued function. For general f, however, this will not necessarily result in reduced latency unless *each* of the independent f_i requires less computation then the entire vector. For example if the f_i were recursive functions then a distributed implementation might offer no advantage over a conventional implementation.

Happily, such is not the case for computed torque robot control, and an inspection of appendix C will reveal that there are obvious way of partitioning the computed torque control expression into smaller independent subtasks. Several approaches for mapping the computed torque algorithm onto an architecture are immediately obvious: One might allocate processor to evaluate each of the subexpressions $k(q), C(q,\dot{q})\dot{q}, M(q)$, and $[\ddot{q}_d + K_2(\dot{q} - \dot{q}_d) + K_1(q - q_d)]$ along with a few extra processors to assemble the results. Extending this concept further, one might imagine assigning to each processor an *element* of the matrix valued functions $C(q,\dot{q})\dot{q}$ and $M(q)$, and the vector valued functions $k(q)$ and $[\ddot{q}_d + K_2(\dot{q} - \dot{q}_d) + K_1(q - q_d)]$.

A completely different parallel approach is proposed in a paper by Lathrop [Lat85]. It begins with an alternative Newton-Euler derivation of the robot equations of motion, and proposes two parallel architectures whose topology is dictated explicitly by the structure of the derivation. The systems proposed, while ignoring communication costs, are respectively linear and $\log_2$ in the degrees of freedom of the manipulator.

Regardless of the particular network structure proposed one must ensure that, as the computational burden per node decreases, the I/O requirements remain manageable. Adding extra processors to a network, while certainly increasing raw computational power, is of no benefit to real time control should the increased I/O requirements result in a net increase in system latency. We view this tradeoff between "computation costs" and "communication cost" as one of the fundamental issues in the design and implementation of real-time distributed systems. Surprisingly, to the best of our knowledge, the voluminous body of robotics literature on distributed control strategies uniformly omits latencies introduced by inter-processor communication.

We have implemented several of the above mentioned partitioning strategies in the Yale Robotics Laboratory using INMOS Transputers. After numerous trials, experiments, and abandoned attempts, we have arrived at a partitioning which appears to perform well in terms of distribution of computation while simultaneously achieving efficient interprocessor communication.

4 Experimental Results

Our objective is to develop a *constructive theory* of implementation for real time robot control networks, and the experimental component of this quest has been the implementation of *actual* control distributions. To this end we have found the Transputer to be a superb platform for real time controller development and implementation. We have benefited immeasurably by the ability to quickly construct, evaluate, and modify an arbitrary topology of extremely powerful processors. The exceptional scalar floating point power of the T-800, its simple interconnect, and the availability of the simple TDS development environment were absolutely essential in permitting us to quickly construct distributed control networks that would have taken years to build in our previous bus-based environment. We have con-

structed dozens of real time distributed control topologies and primitive experiments in an attempt to understand the issues involved. We shall attempt to relate the essential elements through brief descriptions of two different distribution approaches.

4.0.1 An Early T4 Implementation.

An early implementation of the computed torque algorithm for a three degree of freedom robot used T4s. Allocating one processor to each of the subexpressions $C(q,\dot{q})\dot{q}, M(q)$, and $[\ddot{q}_d + K_2(\dot{q} - \dot{q}_d) + K_1(q - q_d)]$ we obtained the single-input single-output control network pictured in figure 6.

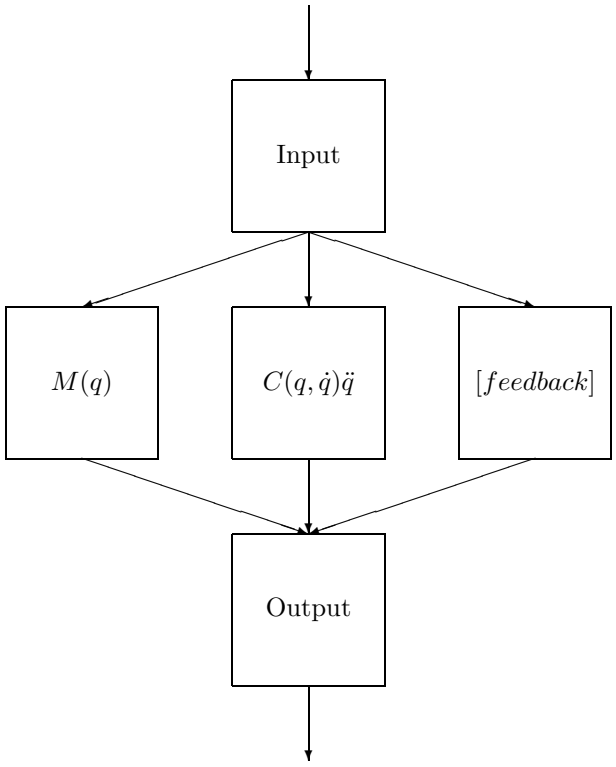

Figure 6: A 5 Node network.

The input processor collects state information from each servo node (not shown.) Each computation processor receives a complete copy of the global state and reference data from the input node, calculates its subexpression, and forwards the value of its subexpression. The subexpressions are assembled in the output processor which then delivers torque commands to the motors.

Experimentally measuring the update and latency periods for this network, we found that the distributed implementation indeed had an *update* period of about five times that of a uniprocessor implementation. This indicates that the distributed implementation had about five times greater throughput than a single processor implementation. Surprisingly, though, the *latency* through the network was approximately equal to that of the uniprocessor implementation! The results, in units of uS are tabulated in figure 7. (Note that the network is single-input and single-output, thus the self latency and cross latency are exactly equivalent.) The substantial latency of the distributed controller is the result of the I/O overhead associated with communicating a substantial amount of data and subexpression terms between the processors. The results were essentially similar for like implementations using an

intermediate number of processors.

nodes	1	5
Update Period	4291	859
Latency	5095	5212

Figure 7: Early T4 Networks.

This early implementation also suffers from two additional fatal flaws: First, it has a single-input single-output I/O structure which does not lend itself well to interfacing with more than a few servo nodes. Second,since adding additional nodes makes no improvement on system latency, the system does not scale well to higher degrees of freedom and additional processors.

4.0.2 A Recent T8 Implementation

A more recent implementation of the exact same control expression using a different partition of the equation and a different resulting network topology offered substantially better results.

This 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 $C(q, \dot{q})$ 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 8 and *actual measured latencies* are tabulated as a matrix in figure 9. The row index of the matrix indicates the "source" servo and the column index indicates the "target" servo. For example the (1,1) element of the matrix represents the self latency of the network for servo 1, while the (1,2) element represents the cross latency from servo 1 through the network to servo 2. Units are microseconds.

Note the slight diagonal dominance of the matrix; the additional latency for communication across the network of servers is manifested in the off-diagonal terms. This distribution offers a natural I/O structure for interfacing to servos, and it is extensible in a regular fashion to higher degrees of freedom by simply adding more sites on the server ring. It appears to represent a satisfactory compromise between computation granularization and communication overhead, while offering regular extensibility.

4.0.3 Communication Protocol Effects

Within the task partitioning and network architecture of each of the implementation we have observed substantially different performance characteristics for different types of device drivers. The most notable effect on real-time latency results from the use of buffering on a processor's input channels. First let us define two basic types of input device drivers which a computation process might use as a front end for incoming data.

Buffered: The device driver stores the *most recent* copy of incoming data in a buffer, and asynchronously services requests from the computation process for this data. This type of device driver might be used to interface two inherently asynchronous processes.

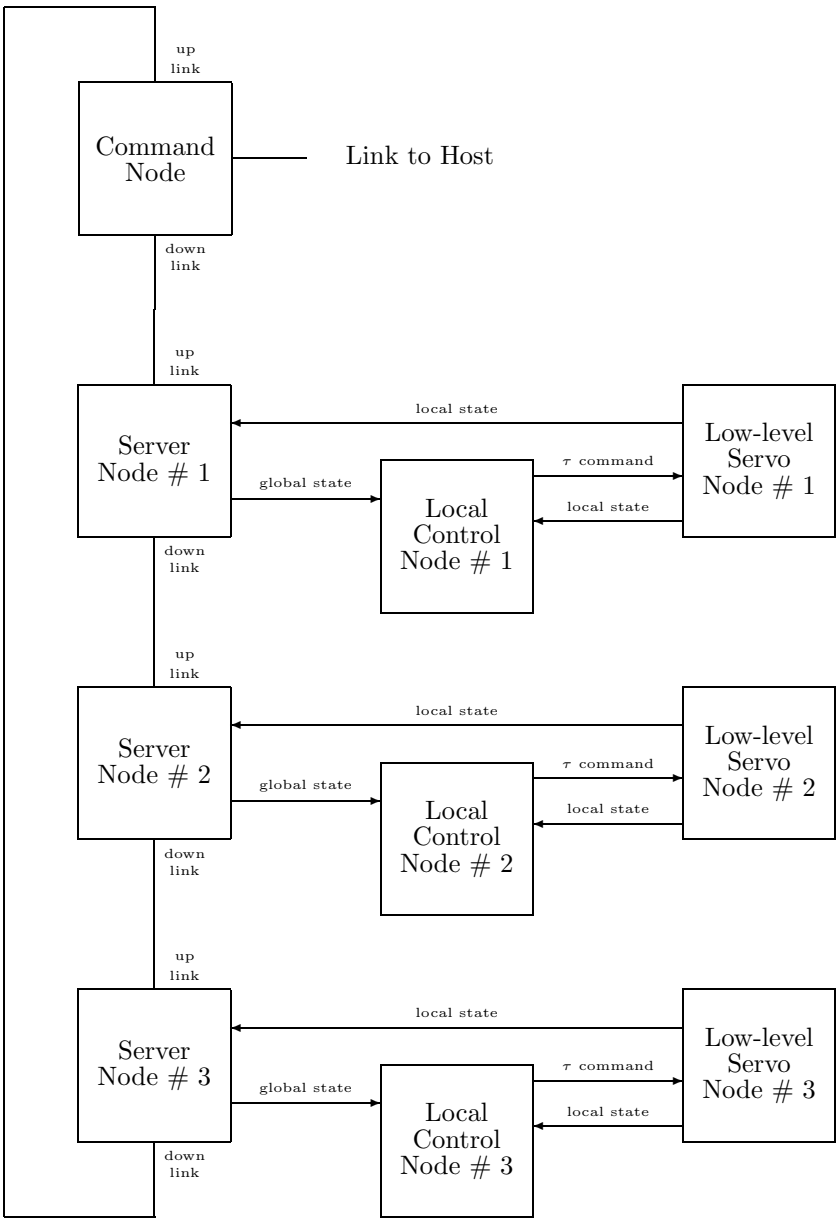

Figure 8: The Control Network Topology.

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

Figure 9: Network Latency: Buffered.

Unbuffered: The computation process, when needing new data, simply waits until receipt of the next incoming data packet before continuing with the computation. This operation corresponds to an OCCAM channel input, and requires that the two processes synchronize.

The latency figures listed in this section are the result of regular computation processes, pictured in figure 8, with two inputs and one output. The first input is from the local servo, the second input is from the local server, and the output is to the local servo. In figure 9 both computation inputs were buffered.

Conducting some primitive experiments confirmed that input buffering on computation processes has a beneficial effect on update rate, but has an *adverse* effect on latency. An input buffer and associated computation process which receives asynchronous inputs with average update rate x will, in general, have a latency $\frac{1}{2}x$ greater than would the simple unbuffered computation alone. This is the effect of the fact that the input buffer, which is updated every x time units has an average latency of $\frac{1}{2}x$.

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

Figure 10: Network Latency: Unbuffered.

Figure 10 tabulates latencies for an otherwise identical network implemented with no buffering whatsoever. The implementation achieves the original performance objectives with the *total latency* under 1mS in all cases. It can be seen that the latencies are roughly uniform for both cross and self latency, because the unbuffered network tends to be self synchronizing.

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

Figure 11: Network Latency: Self Buffering.

Finally, using an otherwise identical network, buffering was implemented *only* on the computation nodes' server inputs, while the inputs from the local servo remain unbuffered. We would expect to see improved self latencies in this case at the expense of degraded cross latencies. The actual measured latencies for this case, tabulated in figure 11, substantiate this conjecture.

5 Conclusion

We have attempted to demonstrate the that there is an intrinsic correspondence between the form of a control expression and its effective implementation on a distributed control system. Indeed it appears that the inherent structure revealed by different derivation methods for an expression may give rise to substantially different task partitioning and network topologies (architectures) in implementation, with correspondingly different limitations in ultimate system performance.

It has been demonstrated that, within a given task partitioning and network topology, different interprocess communication protocol conventions give rise to substantially different latency characteristics.

We are currently developing a primitive technique for predicting the internode latencies resulting from a given computation, task partition, topology, and communication protocol. We suggest that interprocess communication protocol, in addition to producing in substantially different performance measures, is fundamental in determining the "robustness" of a control network in actual implementation, and that this too may be predicted.

Clearly the most compelling question is the following: Given a control expression and a performance criterion, constructively generate a class of distributed controllers which satisfy the specifications. It is our view is that by sufficiently restricting the class of control expressions and primitive network building blocks one might be able to move towards a constructive theory of real-time control implementation.

A Notation

Frame Transformation: A 4x4 matrix denoted ${}^{i}_{j}T$ which represents the position of frame j frame with respect to frame I . Frame composition operates in the obvious fashion:

$$ {}^{a}_{b}T \triangleq \left\{ \begin{array}{ll} (a < b) & \prod_{b-1}^{i=a} {}^{i}_{i+1}T \\ (a = b) & I \\ (a > b) & \prod_{a-1}^{i=b} {}^{i}_{i+1}T \end{array} \right\} $$

and by definition

$$ {}^{a}_{b}T = {}^{b}_{a}T^{-1} $$

This convention was originally developed in [DH55] and more recently presented in the robotics context in [Pau81, Cra86], to which the reader is referred to for detailed explanation.

Point: The symbol ${}^{i}r$ denotes a point in affine 3-space with respect to frame i, and is represented by a 4x1 vector whose first three elements are the points usual coordinates and whose fourth entry is unity.

Stack: If M is an n by m matrix of numbers, then M^S is the $n \times m$ vector formed by stacking the successive columns of M.

Kronecker Product: Denoted by $\otimes$ and operating in the usual fashion.

B Derivation of The Equations of Motion

Let us now derive explicitly the equations of motion for an arbitrary robot which we thus far have written in the general form (1) .

B.1 Kinetic Energy

The first step is to develop an expression for the total kinetic energy of the robot in terms of the state variables

$$q \stackrel{\triangle}{=} \begin{vmatrix} \theta 1 \\ \vdots \\ \theta n \end{vmatrix} \qquad\qquad \dot{q} \stackrel{\triangle}{=} \begin{vmatrix} \dot{\theta}_1 \\ \vdots \\ \dot{\theta}_n \end{vmatrix}$$

where q is the vector of joint positions, and $\dot{q}$ is the vector of joint velocities. We assume that the frame zero, the "base frame", is an inertial frame of reference, and derive from the definitions an expression for the velocity of an arbitrary point in an arbitrary link with respect to the base frame.

$$\begin{aligned} {}^a_b T &= {}^b_a T^{-1} \\ {}^0 r &= {}^0_n T \ {}^n r \\ {}^0 \dot{r} &= {}^0_n \dot{T} \ {}^n r \\ &= \sum_{i=1}^{n} {}^0_{(i-1)} T \ {}^{(i-1)}_i \dot{T} \ {}^i_n T \ {}^n r \end{aligned}$$

We can write down the expression for the kinetic energy for the i'th link of the robot

$$\begin{aligned} d\mathcal{K} &= \frac{1}{2} \ {}^0 \dot{r}^2 dm \\ &= \frac{1}{2} \ {}^0 \dot{r}^T \ {}^0 \dot{r} dm \\ d\mathcal{K}_i &= \frac{1}{2} \ {}^n r^T \ {}^0_n \dot{T}^T \ {}^0_n \dot{T} \ {}^n r dm \\ \mathcal{K}_i &= \frac{1}{2} \int_{vol} {}^0 \dot{r}^T \ {}^0 \dot{r} dm_i \\ \mathcal{K} &= \sum_{i=1}^{n} (\frac{1}{2} \int_{vol} {}^0 \dot{r}^T \ {}^0 \dot{r} dm_i) \\ &= \frac{1}{2} \sum_{i=1}^{n} \int_{vol} trace[\dot{r} \dot{r}^T dm_i] \\ &= \frac{1}{2} \sum_{i=1}^{n} \int_{vol} trace[\ {}^0_i \dot{T} \ {}^i r (\ {}^0_i \dot{T} \ {}^i r)^T dm_i] \\ &= \frac{1}{2} \sum_{i=1}^{n} \int_{vol} trace[\ {}^0_i \dot{T} \ {}^i r \ {}^i r^T \ {}^0_i \dot{T}^T dm_i] \\ &= \frac{1}{2} \sum_{i=1}^{n} trace[\ {}^0_i \dot{T} \int_{vol} (\ {}^i r \ {}^i r^T dm_i) \ {}^0_i \dot{T}^T] \\ \mathcal{K}_i &= \frac{1}{2} \sum_{i=1}^{n} trace[\ {}^0_i \dot{T} J_i \ {}^0_i \dot{T}^T] \end{aligned}$$

where $J_i \stackrel{\triangle}{=} \int_{vol} {}^i r \, {}^i r^T dm_i$ is the expression for the i'th link's inertial tensor with respect to the link's coordinate system. Continuing, we have

$$\begin{aligned}
\mathcal{K} &= \frac{1}{2}\sum_{i=1}^{n} trace[\, {}^0_i\dot{T} J_i \, {}^0_i\dot{T}^T] \\
&= \frac{1}{2}\sum_{i=1}^{n} [(\, {}^0_i\dot{T} J_i)^S]^T \, {}^0_i\dot{T}^S \\
&= \frac{1}{2}\sum_{i=1}^{n} [(J_i \otimes I)^T \; {}^0_i\dot{T}^S] \; {}^0_i\dot{T}^S \\
&= \frac{1}{2}\sum_{i=1}^{n} [\, {}^0_i\dot{T}^S]^T (J_i \otimes I) \; {}^0_i\dot{T}^S \\
&= \frac{1}{2}\sum_{i=1}^{n} [(D_q \; {}^0_i T^S)\dot{q}]^T (J_i \otimes I)(D_q \; {}^0_i T^S)\dot{q} \\
&= \frac{1}{2}\sum_{i=1}^{n} \dot{q}^T [(D_q \; {}^0_i T^S)]^T (J_i \otimes I)(D_q \; {}^0_i T^S)\dot{q} \\
&= \frac{1}{2}\dot{q}^T \{\sum_{i=1}^{n} [(D_q \; {}^0_i T^S)]^T (J_i \otimes I)(D_q \; {}^0_i T^S)\}\dot{q} \\
\mathcal{K} &= \frac{1}{2}\dot{q}^T M(q)\dot{q}
\end{aligned}$$

is the expression for the robot's total kinetic energy where the matrix $M(q)$ matrix formed by the summation of outer products

$$M(q) \stackrel{\triangle}{=} \sum_{i=1}^{n} [(D_q \; {}^0_i T^S)]^T (J_i \otimes I)(D_q \; {}^0_i T^S)$$

B.2 Lagrangian Derivation

Writing the lagrangian [Arn78] where $\mathcal{K}$ represents the total kinetic energy and $\mathcal{P}$ represents the potential energy

$$\mathcal{L} = \mathcal{K} - \mathcal{P}$$

we will set the potential term (due to gravitation) to zero for simplicity of exposition

$$\mathcal{L} = \frac{1}{2}\dot{q}^T M(q)\dot{q}$$

and use lagrange's formula

$$\tau^T = \frac{d}{dt}(D_{\dot{q}}\mathcal{L}) - D_q\mathcal{L}$$

thus

$$\begin{aligned}
D_{\dot{q}}\mathcal{L} &= \dot{q}^T M(q) \\
\frac{d}{dt}(D_{\dot{q}}\mathcal{L}) &= \ddot{q}^T M(q) + \dot{q}^T \dot{M}(q)
\end{aligned}$$

and since $(ABC)^S \equiv (C^T \otimes A)B^S$

$$\begin{aligned}
\dot{q}^T\dot{M}(q) &= (\dot{M}(q)\dot{q})^T \\
\dot{q}^T\dot{M}(q) &= ((\dot{q}^T \otimes I)\dot{M}(q)^S)^T \\
\dot{q}^T\dot{M}(q) &= ((\dot{q}^T \otimes I)D_q[M(q)^S]\dot{q})^T
\end{aligned}$$

so

$$\frac{d}{dt}(D_{\dot{q}}\mathcal{L}) = \ddot{q}^T M(q) + ((\dot{q}^T \otimes I)D_q[M(q)^S]\dot{q})^T$$

and for the other term:

$$\begin{aligned}
D_q\mathcal{L} &= \frac{1}{2}\dot{q}^T D_q[M(q)]\dot{q} \\
&= \frac{1}{2}\dot{q}^T(\dot{q}^T \otimes I)D_q[M(q)^S] \\
&= \dot{q}^T\{\frac{1}{2}(\dot{q}^T \otimes I)D_q[M(q)^S]\}
\end{aligned}$$

and putting it all together yields:

$$\begin{aligned}
\tau^T &= \frac{d}{dt}(D_{\dot{q}}\mathcal{L}) - D_q\mathcal{L} \\
&= \ddot{q}^T M(q) + \dot{q}^T\{(\dot{q}^T \otimes I)D_q[M(q)^S]\}^T - \dot{q}^T\{\frac{1}{2}(\dot{q}^T \otimes I)D_q[M(q)^S]\} \\
&= \ddot{q}^T M(q) + \dot{q}^T\{(\dot{q}^T \otimes I)D_q[M(q)^S]\}^T - \dot{q}^T\frac{1}{2}\{(\dot{q}^T \otimes I)D_q[M(q)^S]\} \\
&= \ddot{q}^T M(q) + \dot{q}^T[\{(\dot{q}^T \otimes I)D_q[M(q)^S]\} - \frac{1}{2}\{(\dot{q}^T \otimes I)D_q[M(q)^S]\}^T]^T \\
\tau^T &= \ddot{q}^T M(q) + \dot{q}^T C(q,\dot{q})^T \\
\tau &= M(q)\ddot{q} + [\{(\dot{q}^T \otimes I)D_q[M(q)^S]\} - \frac{1}{2}\{(\dot{q}^T \otimes I)D_q[M(q)^S]\}^T]\dot{q}
\end{aligned}$$

Thus the equations of motion are

$$\tau = M(q)\ddot{q} + C(q,\dot{q})\dot{q}$$

where:

$$M(q) = \sum_{i=1}^{n}[(D_q \ {}^0_i\dot{T}^S)]^T(I_i \otimes I)(D_q \ {}^0_i\dot{T}^S)$$

$$C(q,\dot{q}) = \{(\dot{q}^T \otimes I)D_q[M(q)^S]\} - \frac{1}{2}\{(\dot{q}^T \otimes I)D_q[M(q)^S]\}^T$$

C Derivation of the Computed Torque Algorithm

The plant is described by the second order nonlinear differential equation

$$\tau \stackrel{\triangle}{=} k(q) + M(q)\ddot{q} + C(q,\dot{q})\dot{q}$$

and the controller is described by the relation

$$\tau \stackrel{\triangle}{=} k(q) + C(q,\dot{q})\dot{q} + M(q)[\ddot{q}_d + K_2(\dot{q} - \dot{q}_d) + K_1(q - q_d)]$$

as previously given in (1) and (2) . Equating the two equations yields

$$M(q)\ddot{q} + C(q,\dot{q})\dot{q} = C(q,\dot{q})\dot{q} + M(q)[\ddot{q}_d + K_2(\dot{q} - \dot{q}_d) + K_1(q - q_d)]$$

$$0 = M(q)[-(\ddot{q} - \ddot{q}_d) + K_2(\dot{q} - \dot{q}_d) + K_1(q - q_d)]$$

which, defining $e \stackrel{\triangle}{=} q - q_d$, we may rewrite as the following:

$$0 = M(q)[-\ddot{e} + K_2\dot{e} + K_1 e]$$

The matrix $M(q)$ is nonsingular, allowing us to multiply on the left by $M(q)^{-1}$ thus

$$0 = [-\ddot{e} + K_2\dot{e} + K_1 e]$$

$$\ddot{e} = K_2\dot{e} + K_1 e$$

which we may rewrite as the first order system where

$$\begin{aligned} x_1 &\stackrel{\triangle}{=} e \\ \dot{x}_1 &\stackrel{\triangle}{=} x_2 \\ \dot{x}_2 &\stackrel{\triangle}{=} K_1 x_1 + K_2 x_2 \end{aligned}$$

yielding

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & I \\ K_1 & K_2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

which is clearly linear in e, the system tracking error. It can be shown that a sufficient condition for the asymptotic stability of this first order linear time invariant is for the "gain" matrices K_1 and K_2 be negative definite.

D Sample Robot Description File

SMP robot description input file for RRR Planar arm.

```
 /* KINEMATICS: Link Transformations:  */
T[1]   :  { { Cos[q[1]], - Sin[q[1]],  0,    0 }, \
     { Sin[q[1]],   Cos[q[1]],  0,    0 }, \
     {         0,           0,  1,    0 }, \
     {         0,           0,  0,    1 }  }

T[2]   :  { { Cos[q[2]], - Sin[q[2]],  0, l[1] }, \
     { Sin[q[2]],   Cos[q[2]],  0,    0 }, \
     {         0,           0,  1,    0 }, \
     {         0,           0,  0,    1 }  }

T[3]   :  { { Cos[q[3]], - Sin[q[3]],  0, l[2] }, \
     { Sin[q[3]],   Cos[q[3]],  0,    0 }, \
     {         0,           0,  1,    0 }, \
     {         0,           0,  0,    1 }  }

T[4]   :  { {         1,           0,  0, l[3] }, \
     {         0,           1,  0,    0 }, \
     {         0,           0,  1,    0 }, \
     {         0,           0,  0,    1 }  }

 /* DYNAMICS: Link Inertia Tensors:   */
M[1]   :  { { l[1]^2 m[1],  0,  0,   l[1] m[1] }, \
     {           0,  0,  0,           0 }, \
     {           0,  0,  0,           0 }, \
     {   l[1] m[1],  0,  0,        m[1] }  }

M[2]   :  { { l[2]^2 m[2],  0,  0,   l[2] m[2] }, \
     {           0,  0,  0,           0 }, \
     {           0,  0,  0,           0 }, \
     { l[2]   m[2],  0,  0,        m[2] }  }

M[3]   :  { { l[3]^2 m[3],  0,  0,   l[3] m[3] }, \
     {           0,  0,  0,           0 }, \
     {           0,  0,  0,           0 }, \
     { l[3]   m[3],  0,  0,        m[3] }  }
```

E Lagrangian Analysis Source Code

```
/*=============================================================================*/
/*                                                                             */
/*                      Lagrangian Robot Dynamics.                             */
/*                                                                             */
/*=============================================================================*/
/*=============================================================================*/
/*  Complete arm inertia tensor:  MM(q) = MM(theta ,theta ,theta )             */
/*                                          1      2      3                   */
/*=============================================================================*/
mbig[ $transform, $q, $IM, $zerolink, $lastlink ] ::                    \
 ( mbig[ $a, $b, $c, $d, $e] :                                          \
   Trigimpl[Trigexpl[                                                   \
   Sum[ mlittle[ $transform, $q, $IM, $zerolink, $lastlink, %link ],    \
       {%link, $zerolink + 1, $lastlink }]                              \
    ]] )

/*=============================================================================*/
/*  The Lagrangian.    L = K - P = Kinetic - Potential                      */
/*=============================================================================*/
lagrangian[ $transform, $q, $qdot, $IM, $zerolink, $lastlink ] ::       \
  ( lagrangian[ $a, $b, $c, $d, $e, $f] : (1/2) Trans[$qdot].           \
    mbig[ $transform, $q, $IM, $zerolink, $lastlink ].$qdot )

/*=============================================================================*/
/*  Robot Equations of Motion.                                             */
/*=============================================================================*/
mbar[ $transform, $q, $qdot, $IM, $zerolink, $lastlink ] ::             \
( mbar[ $a, $b, $c, $d, $zerolink, $lastlink] :                         \
  Kron[Trans[$qdot],Ident[ Len[q]] ].                       \
  Trigexpl[ DM[mbig[ $transform, $q, $IM, $zerolink, $lastlink],$q] ]   \
     )
coriolis[ $transform, $q, $qdot, $IM, $zerolink, $lastlink ] ::             \
( coriolis[ $a, $b, $c, $d, $e, $f] :                                       \
  mbar[ $transform, $q, $qdot, $IM, $zerolink, $lastlink ] -          \
  (1/2)  Trans[ mbar[ $transform, $q, $qdot, $IM, $zerolink, $lastlink ]]   \
     )

/*=============================================================================*/
/*  Now the equation of motion.                                            */
/*=============================================================================*/
torque[ $transform, $q, $qdot, $qdotdot, $IM, $zerolink, $lastlink ] ::     \
( torque[ $a, $b, $c, $d, $e, $f, $g] :                                     \
  mbig[ $transform, $q,  $IM, $zerolink, $lastlink ].$qdotdot +     \
  coriolis[ $transform, $q, $qdot, $IM, $zerolink, $lastlink ].$qdot )
```

F Explicit equations of Motion

The following is the symbolic derivation of the [1,2] element of the coriolis matrix for a simple three degree of freedom planar arm:

$c_{1,2} = -2l_1l_2m_2\dot{\theta}_1 \sin\theta_2 - l_1l_2m_2\dot{\theta}_2 \sin\theta_2 - 2l_1l_2m_3\dot{\theta}_1 \sin\theta_2 - l_1l_2m_3\dot{\theta}_2 \sin\theta_2 - 2l_1l_3m_3\dot{\theta}_1 \sin(\theta_2+\theta_3) - l_1l_3m_3\dot{\theta}_2 \sin(\theta_2+\theta_3) - l_1l_3m_3\dot{\theta}_3 \sin(\theta_2+\theta_3)$

The following is the exact expression for the [1,2] term of the coriolis matrix for the Unimation PUMA 560, a six degree of freedom industrial robot arm:

$$\dot{\theta}_1(-1.03937\cos\theta_3\sin^2\theta_2 + 0.0215632\cos\theta_3\sin 2\theta_2 + 0.0209956\cos 2\theta_3\sin^2\theta_2 + 0.31519\cos 2\theta_3\sin 2\theta_2 + 1.03937\cos^2\theta_2\cos\theta_3 - 0.0209956\cos^2\theta_2\cos 2\theta_3 + 0.0215632\cos^2\theta_2\sin\theta_3 - 0.0215632\sin^2\theta_2\sin\theta_3 - 1.03937\sin 2\theta_2\sin\theta_3 - 0.00248717\cos\theta_3\cos\theta_5\sin^2\theta_2 - 0.63038\cos\theta_3\sin^2\theta_2\sin\theta_3 + 0.0419912\cos\theta_3\sin 2\theta_2\sin\theta_3 - 2.30400*^-5\cos 2\theta_3\cos 2\theta_4\sin 2\theta_2 + 1.16928*^-4\cos 2\theta_3\cos\theta_5\sin^2\theta_2 + 0.00249466\cos 2\theta_3\cos\theta_5\sin 2\theta_2 + 6.91200*^-5\cos 2\theta_3\cos 2\theta_5\sin 2\theta_2 - 0.00248717\cos\theta_5\sin 2\theta_2\sin\theta_3 + 0.00248717\cos^2\theta_2\cos\theta_3\cos\theta_5 + 0.63038\cos^2\theta_2\cos\theta_3\sin\theta_3 - 1.16928*^-4\cos^2\theta_2\cos 2\theta_3\cos\theta_5 - 0.00248717\cos\theta_3\cos\theta_4\sin 2\theta_2\sin\theta_5 + 4.60800*^-5\cos\theta_3\cos 2\theta_4\sin^2\theta_2\sin\theta_3 - 0.00498931\cos\theta_3\cos\theta_5\sin^2\theta_2\sin\theta_3 + 2.33856*^-4\cos\theta_3\cos\theta_5\sin 2\theta_2\sin\theta_3 - 1.38240*^-4\cos\theta_3\cos 2\theta_5\sin^2\theta_2\sin\theta_3 - 0.00249466\cos 2\theta_3\cos\theta_4\sin^2\theta_2\sin\theta_5 + 1.16928*^-4\cos 2\theta_3\cos\theta_4\sin 2\theta_2\sin\theta_5 + 2.30400*^-5\cos 2\theta_3\cos 2\theta_4\cos 2\theta_5\sin 2\theta_2 + 0.00248717\cos\theta_4\sin^2\theta_2\sin\theta_3\sin\theta_5 - 4.60800*^-5\cos^2\theta_2\cos\theta_3\cos 2\theta_4\sin\theta_3 + 0.00498931\cos^2\theta_2\cos\theta_3\cos\theta_5\sin\theta_3 + 1.38240*^-4\cos^2\theta_2\cos\theta_3\cos 2\theta_5\sin\theta_3 + 0.00249466\cos^2\theta_2\cos 2\theta_3\cos\theta_4\sin\theta_5 - 0.00248717\cos^2\theta_2\cos\theta_4\sin\theta_3\sin\theta_5 - 2.33856*^-4\cos\theta_3\cos\theta_4\sin^2\theta_2\sin\theta_3\sin\theta_5 - 0.00498931\cos\theta_3\cos\theta_4\sin 2\theta_2\sin\theta_3\sin\theta_5 - 4.60800*^-5\cos\theta_3\cos 2\theta_4\cos 2\theta_5\sin^2\theta_2\sin\theta_3 - 1.84320*^-4\cos 2\theta_3\cos\theta_4\cos\theta_5\sin^2\theta_2\sin\theta_5 + 2.33856*^-4\cos^2\theta_2\cos\theta_3\cos\theta_4\sin\theta_3\sin\theta_5 + 4.60800*^-5\cos^2\theta_2\cos\theta_3\cos 2\theta_4\cos 2\theta_5\sin\theta_3 + 1.84320*^-4\cos^2\theta_2\cos 2\theta_3\cos\theta_4\cos\theta_5\sin\theta_5 - 3.68640*^-4\cos\theta_3\cos\theta_4\cos\theta_5\sin 2\theta_2\sin\theta_3 \text{ si} 1.03904\sin 2\theta_2) + \dot{\theta}_2(0.41984\cos\theta_2 - 0.00374785\cos\theta_2\cos\theta_3 + 0.19026\cos\theta_2\sin\theta_3 + 0.19026\cos\theta_3\sin\theta_2 + 0.00374785\sin\theta_2\sin\theta_3 + 4.32288*^-4\cos\theta_2\cos\theta_5\sin\theta_3 + 0.00124358\cos\theta_2\sin\theta_4\sin\theta_5 + 4.32288*^-4\cos\theta_3\cos\theta_5\sin\theta_2 + 4.60800*^-5\cos\theta_2\cos\theta_3\cos\theta_4\sin\theta_4 + 4.32288*^-4\cos\theta_2\cos\theta_3\cos\theta_4\sin\theta_5 - 5.84640*^-5\cos\theta_2\cos\theta_3\sin\theta_4\sin\theta_5 + 0.00124733\cos\theta_2\sin\theta_3\sin\theta_4\sin\theta_5 + 0.00124733\cos\theta_3\sin\theta_2\sin\theta_4\sin\theta_5 - 4.60800*^-5\cos\theta_4\sin\theta_2\sin\theta_3\sin\theta_4 - 4.32288*^-4\cos\theta_4\sin\theta_2\sin\theta_3\sin\theta_5 + 5.84640*^-5\sin\theta_2\sin\theta_3\sin\theta_4\sin\theta_5 - 4.60800*^-5\cos\theta_2\cos\theta_3\cos\theta_4\cos 2\theta_5\sin\theta_4 + 9.21600*^-5\cos\theta_2\cos\theta_5\sin\theta_3\sin\theta_4\sin\theta_5 + 9.21600*^-5\cos\theta_3\cos\theta_5\sin\theta_2\sin\theta_4\sin\theta_5 + 4.60800*^-5\cos\theta_4\cos 2\theta_5\sin\theta_2\sin\theta_3\sin\theta_4 + 0.313635\sin\theta_2) + \dot{\theta}_3(-0.00374785\cos\theta_2\cos\theta_3 + 0.19026\cos\theta_2\sin\theta_3 + 0.19026\cos\theta_3\sin\theta_2 + 0.00374785\sin\theta_2\sin\theta_3 + 4.32288*^-4\cos\theta_2\cos\theta_5\sin\theta_3 + 4.32288*^-4\cos\theta_3\cos\theta_5\sin\theta_2 + 4.60800*^-5\cos\theta_2\cos\theta_3\cos\theta_4\sin\theta_4 + 4.32288*^-4\cos\theta_2\cos\theta_3\cos\theta_4\sin\theta_5 - 5.84640*^-5\cos\theta_2\cos\theta_3\sin\theta_4\sin\theta_5 + 0.00124733\cos\theta_2\sin\theta_3\sin\theta_4\sin\theta_5 + 0.00124733\cos\theta_3\sin\theta_2\sin\theta_4\sin\theta_5 - 4.60800*^-5\cos\theta_4\sin\theta_2\sin\theta_3\sin\theta_4 - 4.32288*^-4\cos\theta_4\sin\theta_2\sin\theta_3\sin\theta_5 + 5.84640*^-5\sin\theta_2\sin\theta_3\sin\theta_4\sin\theta_5 - 4.60800*^-5\cos\theta_2\cos\theta_3\cos\theta_4\cos 2\theta_5\sin\theta_4 + 9.21600*^-5\cos\theta_2\cos\theta_5\sin\theta_3\sin\theta_4\sin\theta_5 + 9.21600*^-5\cos\theta_3\cos\theta_5\sin\theta_2\sin\theta_4\sin\theta_5 + 4.60800*^-5\cos\theta_4\cos 2\theta_5\sin\theta_2\sin\theta_3\sin\theta_4) + \dot{\theta}_4(-4.60800*^-5\cos\theta_2\sin\theta_3 - 4.60800*^-5\cos\theta_3\sin\theta_2 + 4.60800*^-5\cos\theta_2\cos 2\theta_5\sin\theta_3 + 4.60800*^-5\cos\theta_3\cos 2\theta_5\sin\theta_2 - 0.00124358\cos\theta_4\sin\theta_2\sin\theta_5 + 0.00124733\cos\theta_2\cos\theta_3\cos\theta_4\sin\theta_5 + 5.84640*^-5\cos\theta_2\cos\theta_4\sin\theta_3\sin\theta_5 - 4.32288*^-4\cos\theta_2\sin\theta_3\sin\theta_4\sin\theta_5 + 5.84640*^-5\cos\theta_3\cos\theta_4\sin\theta_2\sin\theta_5 - 4.32288*^-4\cos\theta_3\sin\theta_2\sin\theta_4\sin\theta_5 - 0.00124733\cos\theta_4\sin\theta_2\sin\theta_3\sin\theta_5 + 9.21600*^-5\cos\theta_2\cos\theta_3\cos\theta_4\cos\theta_5\sin\theta_5 - 9.21600*^-5\cos\theta_4\cos\theta_5\sin\theta_2\sin\theta_3\sin\theta_5) + \dot{\theta}_5(9.21600*^-5\cos\theta_2\cos\theta_3\sin\theta_4 + 4.32288*^-4\cos\theta_2\cos\theta_3\sin\theta_5 - 0.00124358\cos\theta_5\sin\theta_2\sin\theta_4 - 9.21600*^-5\sin\theta_2\sin\theta_3\sin\theta_4 - 4.32288*^-4\sin\theta_2\sin\theta_3\sin\theta_5 + 0.00124733\cos\theta_2\cos\theta_3\cos\theta_5\sin\theta_4 + 4.32288*^-4\cos\theta_2\cos\theta_4\cos\theta_5\sin\theta_3 + 5.84640*^-5\cos\theta_2\cos\theta_5\sin\theta_3\sin\theta_4 + 4.32288*^-4\cos\theta_3\cos\theta_4\cos\theta_5\sin\theta_2 + 5.84640*^-5\cos\theta_3\cos\theta_5\sin\theta_2\sin\theta_4 - 0.00124733\cos\theta_5\sin\theta_2\sin\theta_3\sin\theta_4)$$

References

[AAGH86] 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.

[Arn78] V. I. Arnold. *Mathematical Methods of Classical Mechanics*. Springer-Verlag, N.Y., 1978.

[Bej74] Antal K. Bejczy. *Robot Arm Dynamics and Control*. Technical Report 33-699, Jet Propulsion Laboratory, Pasadena, CA, 1974.

[Cra86] John J. Craig. *Introduction to Robotics Mechanics and Control*. Addison Wesley, Reading, MA, 1986.

[DH55] J. Denavit and R. S. Hartenberg. A kinematic notation for lower pair mechanisms based upon matrices. *Journal of Applied Mechanics*, 215–221, Jun 1955.

[FP80] Gene F. Franklin and J. David Powell. *Digital Control of Dynamic Systems*. Addison-Wesley, Reading, MA, 1980.

[Fre83] 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.

[Hol82] 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.

[Kha86] Oussama Khatib. Real time obstacle avoidance for manipulators and mobile robots. *The International Journal of Robotics Research*, 5(1):90–99, Spring 1986.

[KK86] 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.

[Kod84] Daniel E. Koditschek. Natural motion for robot arms. In *IEEE Proceedings 23rd Conference on Decision and Control*, pages 733–735, Las Vegas, Dec 1984.

[Kod85] Daniel E. Koditschek. Robot kinematics and coordinate transformations. In *IEEE Proceedings 24th Conference on Decision and Control*, pages 1–4, Fort Lauderdale, Dec 1985.

[Kod86] 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.

[Kod87] Daniel E. Koditschek. High gain feedback and telerobotic tracking. In *Workshop on Space Telerobotics*, page , Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA, Jan 1987.

[Lat85] R. H. Lathrop. Parallelism in manipulator dynamics. *Int. J. Robotics Res.*, 4(2):80–102, Summer 1985.

[LBK7] 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 .

[LBK88] F. Levin, M. Bühler, and D. E. Koditschek. The Yale Real-Time Distributed Control Node. In *Oregon State University Conference on Parallel Processing*, page , Oregon State University, Portland, Ore., Apr 1988.

[LWP80] 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.

[Pau81] Richard P. Paul. *Robot Manipulators: Mathematics Programming and Control*. MIT Press, Cambdrige, MA, 1981.

[SL86] 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.

[TA81] 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.

[TBIC84] 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.