A Design Methodology for Distributed Control Systems to Optimize Performance in the Presence of Time Delays*

J. K. Yook, D. M. Tilbury, N. R. Soparkar†
The University of Michigan
Mechanical Engineering and Applied Mechanics
†Electrical Engineering and Computer Science
Ann Arbor, MI 48109
{jyook, tilbury, soparkar}@umich.edu

Submitted for publication in the *International Journal of Control*
May 4, 2000

Abstract

When a control system is implemented in a distributed fashion, with multiple processors communicating over a network, both the communication delays associated with the network and the computation delays associated with the processing time can degrade the system's performance. In this case, the performance of the system may depend not only on the performance of the individual components but also on their interaction and cooperation. The approach taken in this paper assumes that the control has been designed without taking into account the network architecture. A theoretical framework is presented which allows the effect of time delays on the mechanical performance of the system to be precisely modeled, and these models are used to determine the optimal network architecture for the given control system. A design example of a two-axis contouring system is presented.

1 Introduction and Motivation

From monolithic, centralized control algorithms, there is an increasing trend towards decentralized, distributed real-time control implementations. This trend may be observed in application areas that range from robotic machining to autonomous vehicle control. The changes are dictated by pragmatic considerations that include inherent physical distribution of systems, control performance, and even economic factors. In distributed systems, modularity can be used to manage complexity. If the physical system to be controlled has natural points of division, a convenient way to modularize the control scheme would be to allocate one control module for each physical component of the system. However, this decoupled control strategy can only provide good overall performance if there is little physical coupling between the system components, or if the desired performance does not inherently depend on the coordination of the system components. In many cases, the mechanical system performance metrics (such as speed of response, trajectory following error, etc.) will be closely linked to the computing system performance metrics (processor speed, network bandwidth, laxity, quality of service, etc.).

*This research was supported in part by the NSF-ERC under grant EEC95-92125.

In general, the performance of the mechanical system depends not only on the performance of its individual components but also on their interaction and cooperation. In our research, we are working to explicitly characterize the performance of the control system, integrating the mechanical system performance metrics (such as speed of response, trajectory following error, etc.) with the computing system performance metrics (processor speed, laxity, communication delay, etc.), and thereafter, to employ quality of service (QoS) approaches from computer science to improve the overall performance. In this paper, we assume that traditional techniques are used to design a controller for the coupled mechanical system, ignoring any implementation effects. We then show how the best implementation for a given control system can be determined. Future work will address mechanisms for designing or re-designing a control system to take into account the limitations of the underlying computing infrastructure.

A distributed approach provides the following advantages over a centralized strategy:

- Physically inherent. Many target mechanical systems of interest are inherently distributed, and have autonomous control computing power.
- High performance. There can be far greater processing power than in centralized systems. Communication between the various parts of the system allows the overall system performance to be optimized.
- Flexibility. If the system is physically changed, with a good design, only the interconnections need to be re-specified and re-programmed. Rewiring is simplified by using networked communications.
- Fault-tolerant. If some parts of the system fail, others may still be able to work in a degraded mode (i.e., with reduced communication and processing ability).
- Commercially economical. Stringing a single network cable is often cheaper than wiring a point-to-point connection for each sensor and actuator. Also, components of a mechanical system provided by separate vendors may come with independently developed controllers which need to be integrated (e.g., by distributed coupling).

Although the term "distributed control" has been used to refer to a centralized control strategy with a distributed computing (multiprocessor) implementation (e.g., [6, 7, 24, 25, 26]), we will use the term "distributed control" to refer to control systems with physically distributed processing power and network communication. This type of control system architecture has also been referred to as a "networked control system" [33].

The outline of this paper is as follows. First, we briefly examine some related work on distributed control systems and the effect of time delays on system performance. In Section 3, we outline the framework we have developed for modeling the effects of time delay on control systems. We introduce our definition of a performance metric and define the performance degradation function associated with the time delay. Section 4 presents some preliminary analytic, simulation, and experimental results which validate the framework we have proposed. A design example of a two axis contouring system is presented in Section 5. Conclusions are given in Section 6.

2 Related Work

Traditionally, control algorithms have been designed without consideration of their implementation details. Digital control effects may be taken into account, but microprocessor clock speeds have increased so rapidly that in many cases, a continuous model is appropriate. However, when networks are used to carry feedback signals for a control system, the limited bandwidth induces unavoidable communication delays. As more

data is transmitted, the delay magnitude increases. In order to successfully analyze the behavior of a networked control system, the type and location of these delays must be characterized, and their effect on the performance of the control system understood. Although some parts of this problem have been studied previously, in this paper we study how the control architecture — the allocation of control tasks to processing nodes on a network — affects the mechanical performance.

2.1 Control network effects

When a network is used to transmit data, the delay between the initiation time of the message and the delivery time is influenced not only by the speed of the network (bit-rate) and message size but also by the amount of other traffic on the network and the communication protocol used. The communication protocol also affects the reliability and fault-tolerance of the networked system through its priority assignment and error-correction schemes. In [16], communication protocol requirements for real-time systems were discussed in detail. Attempts to improve existing communication protocols include a new token-passing mechanism for real-time access, fault detection, and recovery of transmission error [15] and a new LAN architecture accounting for the time constraints of the message environment [1]. To decrease the network induced delay, communication scheduling [14] and dynamic routing [12] have also been considered.

It is well-known that excessive delays in a feedback loop can destabilize a control system [8]. Thus, in a distributed control system, the effect of the network induced delays on the stability of the system should be analyzed. It has been shown in [3, 11] how network-induced delays can degrade the stability of a feedback control system where the components are interconnected via a common medium. Since stability analysis of the distributed control systems is strongly related to the communication protocol, preliminary stability analysis with a certain communication protocol was discussed in [14, 15]. In [33], analytic proofs of stability for networked control systems are given for common statically scheduled protocols as well as for a new protocol (try-once-discard); however, overly conservative results are obtained. Robust control [10, 23, 34, 22], state prediction control scheme [3], and recursive information flow system [22] have been proposed to compensate for the effect of time delays. However, these approaches do not allow a system with well-characterized and predictable time delays to be optimized. A study of the delay characteristics of popular control networks can be found in [18, 19]; depending on the network protocol and the amount of data to be sent, time delays can be constant, bounded, or random. Our work assumes that the network time delays can be characterized, and focuses on determining the *effects* of these types of delays.

2.2 Control architecture effects

The mechanical performance of a distributed control system depends on the control algorithms used and on the hardware implementation of those algorithms. The effects of synchronization [4, 27], sampling frequency [13], and delay [24, 35, 14] on the performance have been considered by other researchers. However, most of the existing work has focused on the single-input, single-output case, with one actuator, one sensor, and one processor. Even when multi-input, multi-output plants are considered, the control is typically centralized with the sensor and actuator data perhaps sent over a network, but there is no distributed coordination of physically separated mechanical systems. After control engineers develop block diagrams that describe the system behavior, the system integrator must answer fundamental implementation questions. For example, the granularity (number of nodes) of the system must be determined, along with the sampling rates, processing power of each node, and communication protocol, etc. These design issues will introduce different communication and computation delays in a system which will influence the overall performance of the system significantly. When choosing the architecture of a system, fewer nodes would decrease the

communication delay while increasing the computation task in each node. Due to this tradeoff between the communication and computation delay and the specification of hardware chosen, the system architecture will dictate the magnitude and the location of delays in the system. Not only the magnitude but also the location of the delay influences the system's performance significantly. In the absence of delay, the performance of any system architecture should be identical. However, since delay is unavoidable in a distributed system, the architecture which most efficiently arranges the delays should be chosen.

Existing studies on distributed control systems assume a predetermined distributed architecture; they rarely considered the impacts of different distributed control architectures on the performance of a distributed control system. That is, the physical layout of the distributed control systems is determined before any analysis attempts. The analysis of a distributed control systems has to overcome complexity of sampled-data system, and there is a lack of nonlinear systems control theory to analyze or synthesize such sampled control laws with respect to output performance indexes. For these reasons, no rigorous systematic approach to increase the performance of the system by proper implementation has been developed, although guidelines and simulation studies are presented in [2, 5, 21, 28, 30, 31, 32, 35]. Many of the studies which have been done on distributed control systems lead to straightforward conclusions such as to note that time delays should be minimized or that more axes can be handled if the axis control loops are closed at the axes rather than over the network. These preliminary investigations are useful as case studies to show the feasibility of distributed real-time control systems, but they are inadequate in terms of general principles that can be used to guide the design of distributed control systems. Therefore, a system integrator, in most cases, adopts a common-sense strategy — gathering closely related blocks onto a node, and minimizing any delay in the system — or heavily relies on simulation to make design decisions [24].

As we have indicated, the delays depend on the implementation architecture of the distributed system. In the best scenario, the control architecture design choices should be considered at the beginning of the control design process. In this paper, we propose an analytical approach to characterize the effects of the control architecture on the mechanical performance, given a predefined control algorithm. This will allow the system integrator to quickly evaluate many different scenarios and choose the best possible one. It will also enable future work whereby the control architecture will be designed together with the control algorithm.

3 Mathematical framework

In this section, we define the performance criteria for mechanical systems, and then the performance degradation function associated with different time delays. In Section 4, we present preliminary analytic, simulation, and experimental results which validate this framework.

3.1 Performance criteria

Within the context of this paper, the performance of a mechanical control system will be defined by how closely the system tracks a given reference trajectory. That is, given a desired reference trajectory $r(t)$ for the system, the performance is the difference between the actual system output $y(t)$ and the reference, $P = \|y - r\|$. Depending on the physical system and the application domain, one of many different norms may be used, including the maximum deviation from the trajectory, the average error along the trajectory,

or the endpoint error. The different norms give different performance criteria as defined below:

$$\begin{aligned}
\text{maximum error} \quad P_{max} &= \sup_{t\in[0,T]} \|y(t) - r(t)\| \\
\text{average error} \quad P_{av} &= \|\frac{1}{T}\int_0^T y(t) - r(t)dt\| \\
\text{average absolute error} \quad P_{abs} &= \frac{1}{T}\int_0^T \|y(t) - r(t)\|dt \\
\text{endpoint error} \quad P_{end} &= \|y(T) - r(T)\|
\end{aligned}$$

The reference trajectory may not necessarily have a pre-specified time scaling, but instead, be defined as a geometric path $r(\sigma)$ where σ takes values over the unit interval $[0, 1]$. In this case, any performance measure of the system must also consider the time-scaling of the trajectory. In general, a given trajectory can be followed with increasing accuracy only if the system moves more slowly. That is, if the time required to follow a trajectory is not *a priori* given, there is an inherent tradeoff between this total time and the error along the trajectory. In order to examine this trade-off, we will also consider a combined performance criteria given by:

$$P = p_0T + p_1P_{max} + p_2P_{iae} + p_3P_{end}$$

In this equation, the reference $r(t)$ for the P_i's should be interpreted as $r(\alpha(\sigma))$, where $\alpha : [0, 1] \to [0, T]$ is a smooth, monotonic function which describes the time scaling of the trajectory. The choice of the parameters $p_0, \ldots, p_3$ must be determined by the control engineer in the context of the particular application that is being considered.

3.2 Effect of time delays

As noted above, the focus of this work is on developing methodologies for implementing control laws using distributed computing systems, not on developing new control laws. Thus, the baseline performance will always be taken to be the expected value of the performance criteria with no time delay. This characterization allows us to isolate the effect of the time delay from the control design.

In a distributed control system, there are many different sources of delay depending on the architecture and the choice of network. To study the overall performance of the system, the effects of different time delays (which may be fixed or random and located in different places) must be characterized. In addition, the effects of multiple time delays on the overall performance must also be understood.

In this section, we define the performance degradation function as the difference between the performance of the system with and without time delay. We hypothesize that, for small time delays, the Taylor expansion of this function in terms of the time delays gives a valid approximation to the function. Using the Taylor expansion, the effects of the magnitude of an individual time delay can easily be seen, and the effects of many time delays may also be understood. Validation of this framework will be given in Section 4.

3.3 Performance degradation functions

Consider a control system with and without time delay. Let $r(t)$ be the reference, $y^*(t)$ be the output of the system without time delay, and $y(t)$ be the output of the system with the time delay. The nominal performance criteria is given by

$$P^* = \|y^* - r\|$$

We assume that the controller has been designed well, and that P^* is the "best" possible performance that we can achieve. With time delay, the performance criteria becomes

$$\begin{aligned} P &= \|y - r\| \\ &= \|y - y^* + y^* - r\| \\ &\leq \|y - y^*\| + \|y^* - r\| \\ &= \|\Phi\| + P^* \end{aligned}$$

where Φ represents the degradation in performance due to the time delay.

In general, there may be many time delays which contribute to the overall performance degradation function Φ. Consider a given distributed control architecture with n nodes (sensors, actuators, control modules) in the control system, and communication and computation delays $\tau_{i,j}$ as follows.

$$\begin{aligned} \tau_{i,i} &= \text{computation time at } i\text{th node} \\ \tau_{i,j} &= \text{communication delay from } i\text{th to } j\text{th node} \end{aligned}$$

The time delays need not be symmetric; $\tau_{i,j} \neq \tau_{j,i}$ is allowed. For example, if node i is a sensor, then i may need to send data to another node j over the network, giving some value for $\tau_{i,j}$, but if i does not receive any data, then $\tau_{j,i} = 0$ for all j. If there is some overhead associated with sending the sensor data, it can be included as $\tau_{i,i}$.

The performance degradation function Φ will depend on all of the time delays $\tau_{i,j}$. Its Taylor series expansion can be written as

$$\Phi(\tau_{1,1}, \tau_{1,2}, \ldots, \tau_{n,n}) = \Phi(0, 0, \ldots, 0) + \frac{\partial \Phi}{\partial \tau_{1,1}}\tau_{1,1} + \frac{\partial \Phi}{\partial \tau_{1,2}}\tau_{1,2} + \cdots + \frac{\partial \Phi}{\partial \tau_{n,n}}\tau_{n,n} + \text{higher order terms}$$

Since $\Phi(0, 0, \ldots, 0) = 0$ by definition, the first term on the right hand side is zero. Also, if the time delays are small, the higher order terms can be neglected. Thus, to a first-order approximation, the degradation in performance due to multiple time delays is equal to the sum of the performance degradation due to each time delay. In addition, the performance degradation due to a particular time delay is a linear function of the magnitude of the time delay.

We define the performance differential function, ϕ, to distinguish from the performance degradation function, Φ. The performance differential function $\phi_{i,j}$ is given by

$$\phi_{i,j} = \frac{\partial \Phi}{\partial \tau_{i,j}}$$

and represents the performance degradation due to a unit time delay between nodes i and j. We also use $\Phi_{i,j}$ to represent the performance degradation function for a single time delay between nodes i and j. Thus, the performance degradation function is given by the expression

$$\begin{aligned} \Phi &= \Phi_{1,1} + \Phi_{1,2} + \cdots + \Phi_{n,n} \\ &= \phi_{1,1}\tau_{1,1} + \phi_{1,2}\tau_{1,2} + \cdots + \phi_{n,n}\tau_{n,n} \\ &= \sum_{i}^{n}\sum_{j}^{n} \phi_{i,j}\tau_{i,j} \end{aligned} \tag{1}$$

The performance differential functions can be computed analytically for linear systems and numerically for nonlinear systems. For linear systems, we use the modified $\mathcal{Z}$-transform formulation for computation as described in Section 4.1. For nonlinear systems, simulation results must be used; see Section 4.2. The performance degradation function associated with a control architecture can be found by summing the products of the expected time delays with the performance differential functions as in equation (1), and this value can be used to compare with other control architectures.

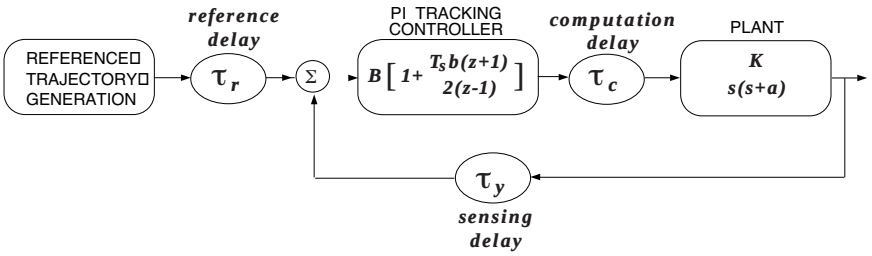


Figure 1: A control system with two control modules: a reference trajectory generation module which commands a ramp of unit slope, and a tracking control module which contains a simple PI controller. Other nodes in the system include the position sensor and the actuator. Possible delays are denoted by ovals; τ_c is the computational delay of the tracking control algorithm, τ_y is a sensing delay, and τ_r is due to communication delays in sending the reference command over a network.

4 Framework Validation

In the context of a very simple mechanical system, we have performed basic analysis, simulation studies and experiments to verify the mathematical framework presented in Section 3. First, we examined how the various time delays associated with distributed architectures impact the performance of the control algorithms. Next, the validation of equation (1) was done by both analysis and simulation. Finally, experimental validation of the above results was attempted.

4.1 Analytic results: Performance degradation functions

For the study, we considered a single degree-of-freedom system, consisting of a motor which moves a mass along an axis. Since this system was linear and low-order, the modified $\mathcal{Z}$-transform was used to compute the effects of delays. There is a reference trajectory generator which produces a ramp reference with unit slope; the plant is a second-order type I system with a PI servo control algorithm. Delays may occur in sending the reference and sensor data over a network, or in computing the control algorithm. Time delays in sending the actuator command over a network occur in the same place in the loop as the computational delay, and are not considered separately here. The block diagram of the system, with the potential time delays in the loop, is shown in Figure 1.

To compute the performance degradation functions for each of the three time delays, we first compute the output of the control system without any time delays using standard digital control techniques. Using the zero-order hold equivalent of the plant transfer function and the controller transfer function $C(z)$, we can compute the nominal output y^* as follows.

$$
\begin{aligned}
G(s) &= \frac{K}{s(s+a)} && \text{plant transfer function} \\
G(z) &= \frac{z-1}{z}\mathcal{Z}\left\{\frac{G(s)}{s}\right\} && \text{ZOH equivalent} \\
C(z) &= B\left[1+\frac{T_s b}{2}\frac{z+1}{z-1}\right] && \text{PI controller} \\
Y^*(z) &= \frac{C(z)G(z)}{1+C(z)G(z)}R(z) \\
y^*(k) &= \mathcal{Z}^{-1}\left\{Y^*(z)\right\}
\end{aligned}
$$

where $R(z)$ is the $\mathcal{Z}$-transform of the unit ramp reference and T_s is the sampling time.

Next, we compute the output of the system with time delays using the modified $\mathcal{Z}$-transform [9, 20]. Using this method, we can account for time delays which are not multiples of the sampling period. Unlike other methods for dealing with time delays (e.g., the Padé approximation), the modified $\mathcal{Z}$-transform does not rely on approximations; exact results are obtained.

For example, the zero-order hold equivalent discrete transfer function of the plant with computational time delay τ_c is computed using the modified $\mathcal{Z}$-transform as follows. Let the computational time delay $\tau_c = (\ell_c - m_c)T_s$, where ℓ_c is an integer and $0 < m_c < 1$ represents the fractional time delay. The main idea of the derivation is to account for the time delay in the continuous-time portion of the analysis, and then find the equivalent discrete-time representation.

$$\begin{aligned}
G_d(z) &= \frac{z-1}{z}\mathcal{Z}\left\{e^{-\tau_c s}\cdot\frac{G(s)}{s}\right\} && \text{ZOH equivalence}\\
&= \frac{z-1}{z}\mathcal{Z}\left\{e^{-(\ell_c-m_c)T_s s}\cdot\frac{K}{s^2(s+a)}\right\}\\
&= \frac{K}{a^2}\frac{z-1}{z^{\ell_c+1}}\mathcal{Z}\left\{a(t+m_cT_s)-1+e^{-a(t+m_cT_s)}\right\} && \text{time-domain solution}\\
&= \frac{K}{a^2}\frac{z-1}{z^{\ell_c+1}}\left((am_cT_s-1)\frac{z}{z-1}+\frac{aT_sz}{(z-1)^2}+\frac{e^{-am_cT_s}z}{z-e^{-aT_s}}\right)
\end{aligned}$$

The output of the system with a computational delay can be found using this delayed-equivalent plant as follows.

$$\begin{aligned}
Y_c(z) &= \frac{C(z)G_d(z)}{1+C(z)G_d(z)}R(z)\\
y_c(k) &= \mathcal{Z}^{-1}\{Y_c(z)\}
\end{aligned}$$

The performance degradation due to the computational time delay is then given by

$$\Phi_c = y_c - y^*$$

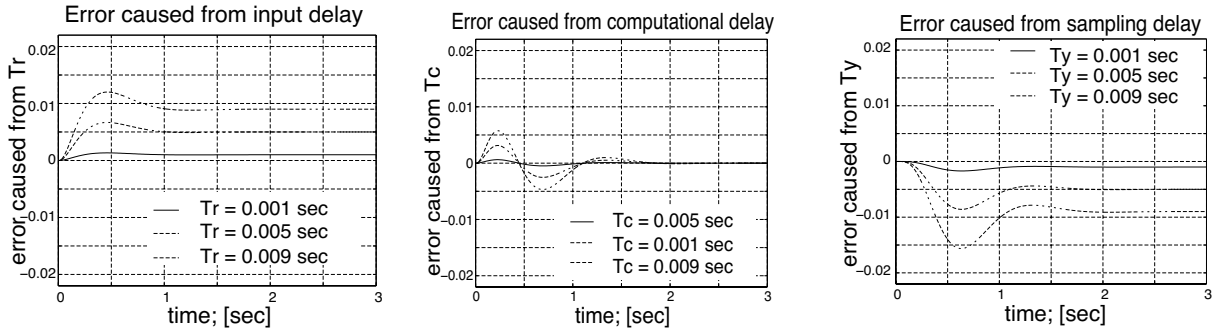
Similar analyses can be done for reference and sensing delays, and for combinations of delays. These analyses are summarized graphically in Figures 2 and 3.

Figure 2 shows the performance degradation functions as functions of time for the three individual time delays; three different magnitudes of time delay are shown for each delay location. As predicted by the Taylor expansion in Section 3.3, these functions Φ_r, Φ_c and Φ_y are linear functions of the magnitude of the time delay. Thus, our hypothesis that $\Phi_r = \phi_r\tau_r$, where ϕ_r is independent of the magnitude of the time delay, is validated for this example.

The performance degradation functions for combinations of different types of time delays in the system are shown in Figure 3. Comparing these results with those of Figure 2, it can be seen that the effects of time delays are additive, validating our hypothesis that the performance degradation function for the entire system can be computed by adding the individual performance degradation functions:

$$\Phi = \Phi_r + \Phi_c + \Phi_y = \phi_r\tau_r + \phi_c\tau_c + \phi_y\tau_y$$

One feature of this analysis is that it explains how the positive and negative steady-state errors associated with the reference and sensing delays cancel. This indicates that if one of these types of delays is unavoidably present in the system, the system designer may compensate for it, at least in the steady-state, by intentionally introducing a delay on the other. Such information will be invaluable in allocation of processing or communication resources in the underlying computing environment.

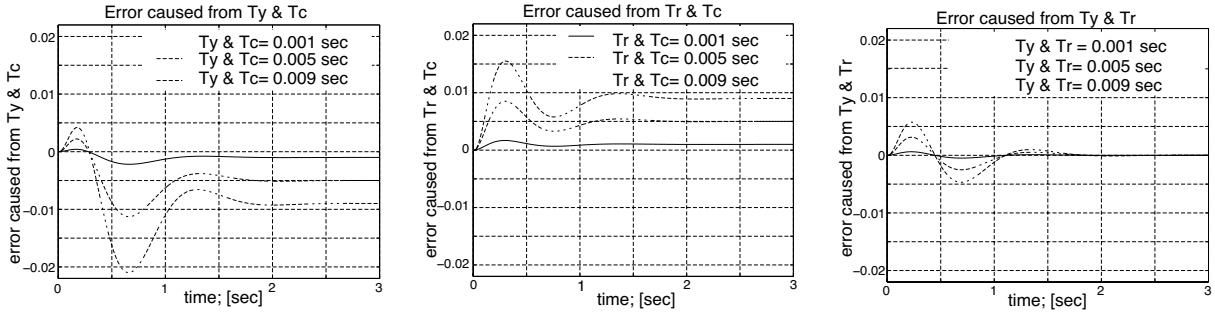


a. reference delay
$\Phi_r = y_r - y^* = \phi_r \tau_r$

b. computational delay
$\Phi_c = y_c - y^* = \phi_c \tau_c$

c. sensing delay
$\Phi_y = y_y - y^* = \phi_y \tau_y$

Figure 2: Performance degradation functions for different time delay locations and magnitudes. For example, (a) shows $\Phi_r = y_r - y^*$, where y_r is the output of the system with a reference time delay and y^* is the output of the system without any time delay. The function Φ_r is plotted for three different values of τ_r (0.001, 0.005, and 0.009 sec), showing that it is a linear function of τ_r. The sampling time T_s of both the servo controller and the interpolator is 0.01 sec.



a. Φ for computation delay τ_c and sensing delay τ_y
$\Phi = y - y^* = \phi_c \tau_c + \phi_y \tau_y$

b. Φ for computation delay τ_c and reference delay τ_r
$\Phi = y - y^* = \phi_c \tau_c + \phi_r \tau_r$

c. Φ for sensing delay τ_y and reference delay τ_r
$\Phi = y - y^* = \phi_y \tau_y + \phi_r \tau_r$

Figure 3: Performance degradation functions for different combinations of delays; $\Phi = y - y^*$. In each figure, three different magnitudes of delay are considered (0.001, 0.005, and 0.009 sec); the same magnitude is used for each type of delay. The sampling time of the servo controller in all cases is $T_s = 0.01$ sec.

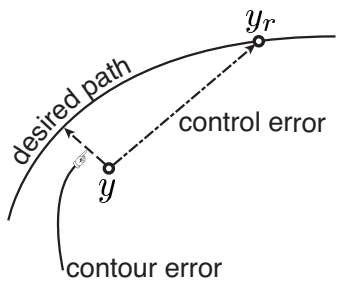


Figure 4: Comparing contour error versus control error. The contour error is a nonlinear function of the axis-level errors and the reference trajectory. Also, the estimator of the contour error in cross-coupled controller includes sinusoidal functions, making the system nonlinear. This nonlinearity makes the analytic method intractable.

4.2 Simulation results: Two-axis contouring system

Another example which we have considered in our preliminary work is a two-axis contouring system, namely an x-y positioning table. This system contains two copies of the previous single-axis system and a cross-coupled controller. Although both axes are linear with PI servo controllers, the cross-coupled controller has a non-linear term for estimation of contour error and the performance criteria; the *contour error* (see Figure 4) is a nonlinear function of the axis-level errors and the reference trajectory. This nonlinearity does not allow the analytic techniques of Section 4.1 to be directly applied; hence, we use simulations and experiments to validate the hypotheses. The simulation results are presented first; experimental results are discussed in Section 4.3.

For a two-axis contouring system, it is well-known that the addition of a cross-coupling controller can reduce the contour error in [17, 29]. The cross-coupling controller contains two parts: contour error approximator and contour error control. The estimation of the contour error from the reference input and the system output includes sinusoidal functions, making the system non-linear. The cross-coupled controller can be any conventional type of a controller such as PI or PID; in this paper we use a proportional controller.

The effect of control architectures and communication delays on the performance of the two-axis contouring system was evaluated through simulation studies and experiments on a small x-y table. Although the control algorithms used were very simple and would, in practice, almost always be implemented on a single processor, a distributed implementation was considered in order to study the communication requirements of the different types of controllers on a physical system model and the effects of delays on the control performance. We consider three different mappings of the system onto hardware:

1. One computer for all I/O (sensors and actuators) and control modules (Centralized architecture)
2. Two computers, one for each axis, connected by a communication network sending position data needed for the cross-coupling control (Decentralized architecture)
3. Two computers, one for each axis, no communication (Decoupled architecture)

These three architectures for the two-axis contouring system are depicted schematically in Figure 5. The third architecture does not use the cross-coupling control module.

Simulation results of the contour error for the three different architectures is shown in Figure 6a. As expected, the decoupled architecture, without any communication, performs much more poorly than the other two. Although in practice this will have some computational delays, in simulation these are not accounted

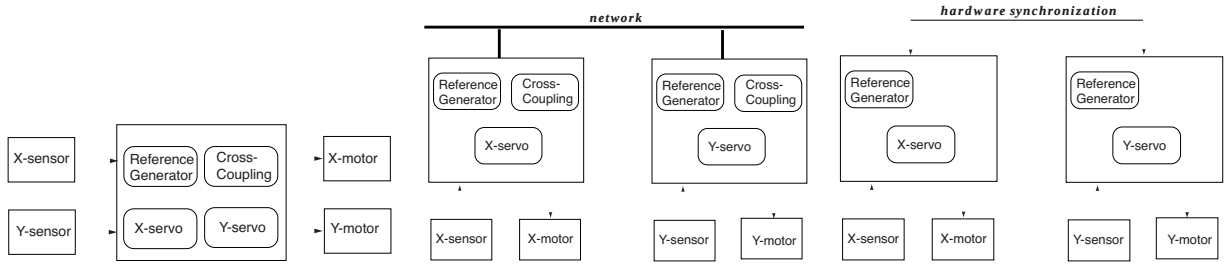
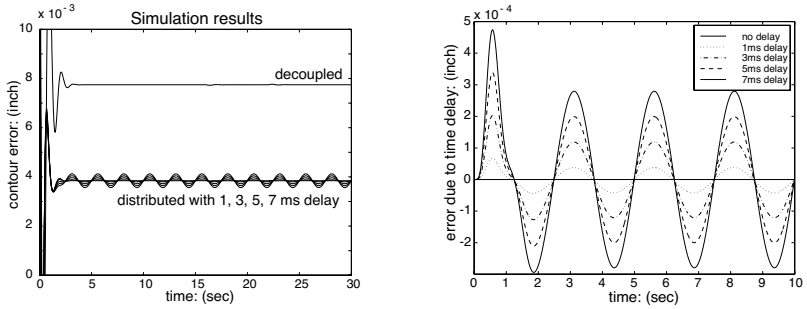


Figure 5: Three architectures which were considered for the two-axis contouring system. The centralized architecture is the traditional single-processor system with all I/O wired directly. The decentralized architecture uses two processors, and thus requires two copies of the reference trajectory generator and the cross-coupled controller. A network is used to send the axis position data between processors; this is needed for the cross-coupled control algorithm to work correctly. The decoupled architecture also uses two processors but there is no real-time communication; a hardware synchronization signal allows the two processors to start simultaneously.



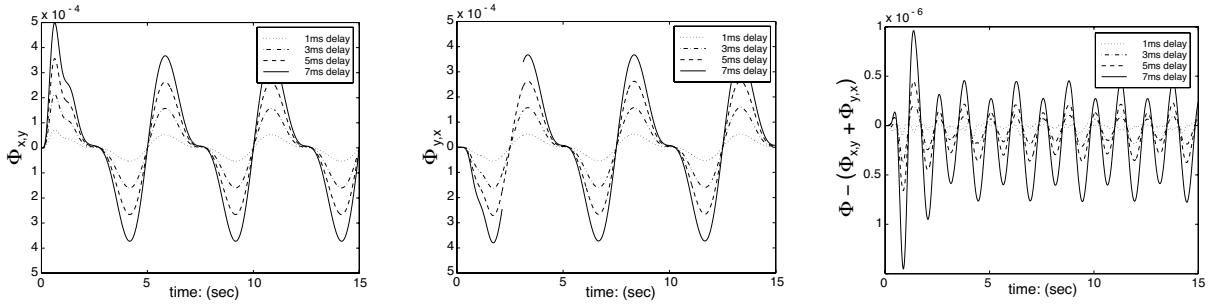
a. Contour errors b. Performance degradation functions

Figure 6: Contour error for the three different architectures shown in Figure 5. From the contour error, the performance degradation function for the decentralized case is computed.

for. Since we did not include the computation delay, the performance of the centralized architecture is identical to the nominal performance and is defined to be the "best." From these contour error results, we can find the performance degradation function Φ for the four different values of the delay τ which were considered: $\tau = 1, 3, 5, 7$ msec. These performance degradation functions, which are the difference between the contour error with time delay and without, are shown in Figure 6b.

The decentralized architecture contains two delays: $\tau_{x,y}$ and $\tau_{y,x}$, namely the x-position data to y axis system and vice versa. In Section 3.3, we claimed that the performance degradation function shown in Figure 6b is the sum of $\Phi_{x,y}$ and $\Phi_{y,x}$. Figure 7 shows that this hypothesis is quite valid although the 2-axis contouring system is non-linear. Note that the error $\Phi - (\Phi_{x,y} + \Phi_{y,x})$ shown in Figure 7c is approximately on the order of the delay squared.

In addition to looking at the contour error as a function of time, we can also tabulate the performance degradation function and consider different performance metrics. A tabulation of the contour errors for the two-axis system is given in Table 1. The performance functions used are the average, average absolute, rms, and maximum values of the contour error (e). The standard deviation of the contour error (σ) is also



a. delay in x data $\Phi_{x,y}$ b. delay in y data $\Phi_{y,x}$ c. error between $\Phi - (\Phi_{x,y} + \Phi_{y,x})$

Figure 7: The performance degradation function from summation individual performance degradation function is compared with simulated results. The error between simulation and computed from individual simulation is approximately order of delay square (high order term of Taylor series expansion).

computed. All simulations were run for 30 seconds, tracing three revolutions of the circular contour. The sample time was $T_s = 0.01$; thus $N = 3000$.

$$\begin{aligned} P_{av} &= \frac{1}{N}\sum_{k=1}^{N} e(k) \\ \sigma &= \sqrt{\frac{1}{N}\sum_{k=1}^{N}(e(k) - P_{av})^2} \\ P_{abs} &= \frac{1}{N}\sum_{k=1}^{N} |e(k)| \\ P_{rms} &= \sqrt{\frac{1}{N}\sum_{k=1}^{N} e(k)^2} \\ P_{max} &= \max_{1\le k\le N} \|y(k) - r(k)\| \end{aligned}$$

From the simulation results, the performance degradation function Φ was also tabulated. The "best" possible performance was taken to be the contour error with the centralized architecture, e^*. The norms of the different performance degradation functions, for different magnitudes of time delay, are reported in Table 2. The norms of these performance degradation functions are computed as follows:

$$\begin{aligned} \|\Phi_{av}\| &= |\frac{1}{N}\sum_{k=1}^{N} e(k) - e^*(k)| \\ \|\Phi_{abs}\| &= \frac{1}{N}\sum_{k=1}^{N} |e(k) - e^*(k)| \\ \|\Phi_{rms}\| &= \sqrt{\frac{1}{N}\sum_{k=1}^{N}(e(k) - e^*(k))^2} \\ \|\Phi_{max}\| &= \max_{1\le k\le N} \|e(k) - e^*(k)\| \end{aligned}$$

As predicted by the theory of Section 3.3, the magnitude of the performance degradation function is approx-

Table 1: Performance metrics for the two-axis contouring system, simulation results. Four different performance metrics are considered: the average contour error, the average absolute contour error, the rms contour error, and the maximum contour error. The standard deviation of the contour error is also shown. Since the error is always positive, the average and absolute errors are the same for this example. The simulations were run for 30 seconds with a sampling time of $T_s = 10$ms, thus, $N = 3001$. Units are milli-inches (0.001in).

	P_{av}	σ	P_{abs}	P_{rms}	P_{max}
Decoupled	7.7462	0.00390	7.7462	7.7462	7.7679
Centralized	3.8437	0.00190	3.8437	3.8437	3.8550
Decentralized, $\tau = 1ms$	3.8437	0.02814	3.8437	3.8438	3.8864
Decentralized, $\tau = 3ms$	3.8442	0.08456	3.8442	3.8451	3.9664
Decentralized, $\tau = 5ms$	3.8446	0.14102	3.8446	3.8472	4.0463
Decentralized, $\tau = 7ms$	3.8452	0.19748	3.8452	3.8503	4.1262

Table 2: Performance degradation functions for the two-axis contouring system, simulation results. For each case, the actual contour error e is compared to the contour error for the centralized simulation, denoted e^*. Units are milli-inches (0.001in).

	$\|\Phi_{av}\|$	$\|\Phi_{abs}\|$	$\|\Phi_{rms}\|$	$\|\Phi_{max}\|$
$\tau = 1ms$	0.00008	0.0254	0.0282	0.0400
$\tau = 3ms$	0.00051	0.0762	0.0847	0.1202
$\tau = 5ms$	0.00097	0.1270	0.1412	0.2002
$\tau = 7ms$	0.00155	0.1779	0.1976	0.2802

imately a linear multiple of the time delay, independently which norm is chosen. This linear relationship is further highlighted in Figure 11a.

4.3 Experimental results: Two-axis contouring system

A small two-axis table with DC motors was used for the experiment. A picture of the experimental set-up is shown in Figures 8 and 9. The transfer functions of the two axes were identified to be

$$\frac{X(s)}{V(s)} = \frac{23}{s(s+27)} \qquad \frac{Y(s)}{V(s)} = \frac{22.8}{s(s+27)}$$

Matlab/Simulink/Real-Time Workshop was used to implement the controllers on personal computers. Synchronization between the two computers was accomplished using a hardware switch which was flipped to start the experiments. Network communication was done using the bi-directional serial port, and the results were used to compare with the centralized architecture.

Several different experiments were run:

1. Centralized control, as shown in Figure 5a, with all computing and I/O on a single computer
2. Decentralized control, as shown in Figure 5b, with one computer for each axis, and communication over the serial port

Figure 8: The experimental set-up, with two computers and a two-axis table.

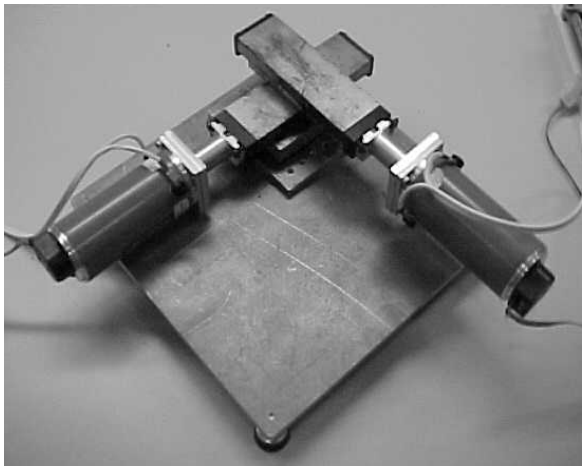
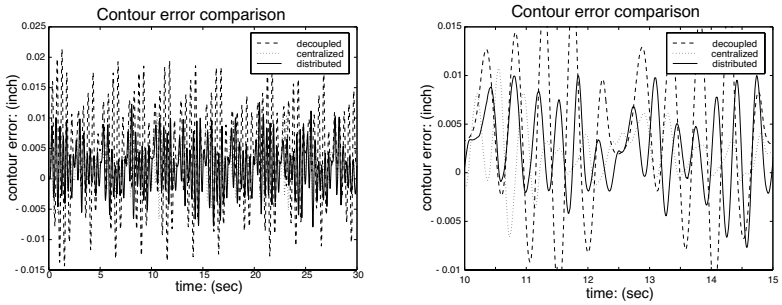


Figure 9: A close-up view of the two-axis table used for the distributed control experiment.



a. Contour errors b. Close-up of contour errors

Figure 10: Contour error for the three different architectures shown in Figure 5. A close-up is also given. Although the absolute values of the contour error do not match the simulation results, the trends that the centralized performs the best, decoupled the worst, hold true. The sample time is $T_s = 10ms$.

3. Decoupled control, as shown in Figure 5c, with one computer for each axis and no communication

The contour error for the three different architectures is shown in Figure 10. Although the absolute values of the contour error do not match the simulation results, the trends that the centralized performs the best, decoupled the worst, hold true. This result was expected since the computation delay is much smaller than the communication delay introduced by the serial connection. The experimental set-up contains friction and other nonlinearities which are not accounted for in the simulation.

In addition to looking at the contour error as a function of time, we can also tabulate the performance degradation function and consider different performance metrics and different magnitudes of time delay as we did for the simulation. In order to examine the effects of different magnitudes of delay, we needed to be able to specify the communication delay. For that reason, only one computer was used for the experiment with the time delay intentionally implemented in the control algorithm. A tabulation of the contour errors for the two-axis system is given in Table 3, and a tabulation of the performance degradation functions is given in Table 4. The performance metrics are plotted versus time delay in Figure 11b, showing a linear relationship for small time delays as predicted by the derivation in Section 3.3.

5 Design Example

In this section, we show how the performance degradation function framework can be used to evaluate and compare several different architectures for a distributed control system. A two-axis contouring system is considered again since it is relatively low-dimensional yet has tight coupling between the axes. It is also nonlinear, showing the broad applicability of the techniques.

5.1 Identification of basic modules

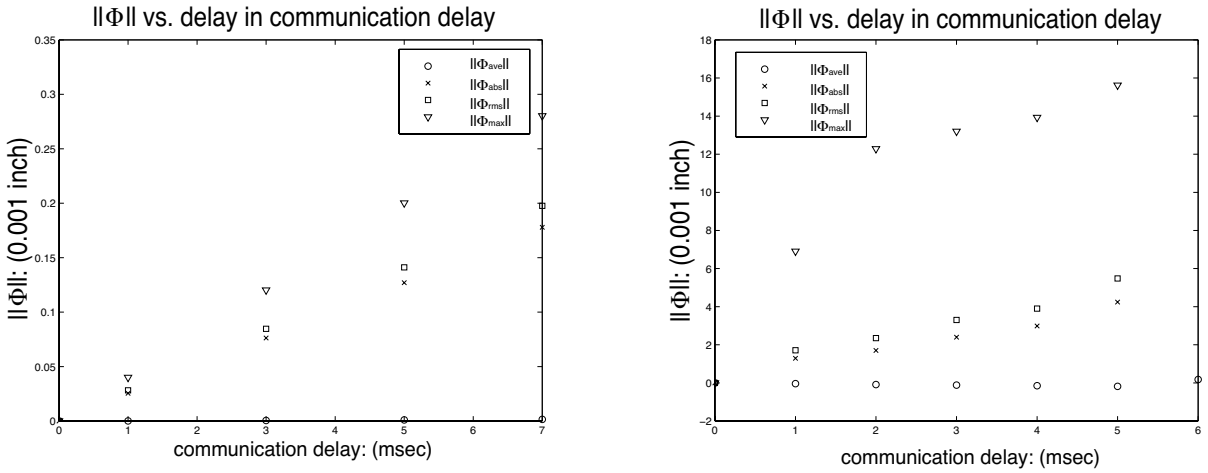
The first step in determining the architecture is to define the basic modules of the control system. A module can be a system component such as a sensor or actuator, an algorithm such as a servo controller, or a subsystem such as an axis. The inputs and outputs of each module must be defined, and their connections determined. The number of modules will determine the amount of communication that must be considered. By choosing the basic modules to be as small as possible in this step, the best performance can be guaranteed; however, smaller modules require more combinations to be examined.

Table 3: Performance metrics for the two-axis contouring system, experimental results. Four different performance metrics are considered: the average contour error, the average absolute contour error, the rms contour error, and the maximum contour error. The standard deviation of the contour error is also shown. Since the decentralized experiment only used one byte of data over the serial port, we also include "decentralized" experiments in which the time delay was intentionally included in the control program. We do not yet have a precise measurement of the actual delay time of the serial communication channel, although these results suggest it is on the order of 4ms. The experiments were run for 30 seconds with a sampling time of $T_s = 10$msec, thus, $N = 3001$. Units are milli-inches (0.001in).

	P_{av}	σ	P_{abs}	P_{rms}	P_{max}
Decoupled	3.983	7.349	7.015	8.358	21.302
Centralized	1.980	2.824	2.649	3.448	10.750
Decentralized, serial comm.	1.977	3.820	3.454	4.300	12.568
"Decentralized," $\tau = 1ms$	1.940	2.841	2.668	3.440	9.952
"Decentralized," $\tau = 2ms$	1.892	3.188	2.843	3.707	12.584
"Decentralized," $\tau = 3ms$	1.864	3.275	2.876	3.768	13.527
"Decentralized," $\tau = 4ms$	1.834	3.799	3.253	4.217	13.040
"Decentralized," $\tau = 5ms$	1.796	5.127	4.294	5.432	16.773
"Decentralized," $\tau = 6ms$	2.159	36.277	30.952	36.355	66.232

Table 4: Performance degradation functions for the two-axis contouring system, experimental results. For each case, the actual contour error e is compared to the contour error for the centralized experiment, denoted e^*. In some cases, the average contour error with the time delay is actually less than the average contour error for the centralized case. Since the contour error for the experiment is not as well-behaved as that for the simulation, these results are not as precise. They do, however, show similar trends, such that the performance degrades as the time delay increases. Units are milli-inches (0.001in).

	$\|\Phi_{av}\|$	$\|\Phi_{abs}\|$	$\|\Phi_{rms}\|$	$\|\Phi_{max}\|$
Decentralized, serial comm.	0.003	3.308	4.311	14.523
"Decentralized," $\tau = 1ms$	0.040	1.290	1.711	6.956
"Decentralized," $\tau = 2ms$	0.088	1.701	2.350	12.376
"Decentralized," $\tau = 3ms$	0.116	2.400	3.304	13.297
"Decentralized," $\tau = 4ms$	0.147	2.985	3.899	14.030
"Decentralized," $\tau = 5ms$	0.184	4.239	5.481	15.738
"Decentralized," $\tau = 6ms$	0.178	30.711	36.179	67.588



(a) Simulation results

(b) Experimental results

Figure 11: Performance degradation functions for different time delays and different performance metrics; simulation and experimental results. The circles represent Φ_{ave}, the crosses represent Φ_{abs}, the squares represent Φ_{rms}, and the triangles represent Φ_{max}. Because the contour error is a nonlinear function of the axis positions, simulation and experimental results must be used to compute the performance degradation function Φ. For small values of time delay, the performance degradation function is a linear function of the time delay, as predicted by the derivation in Section 3.3.

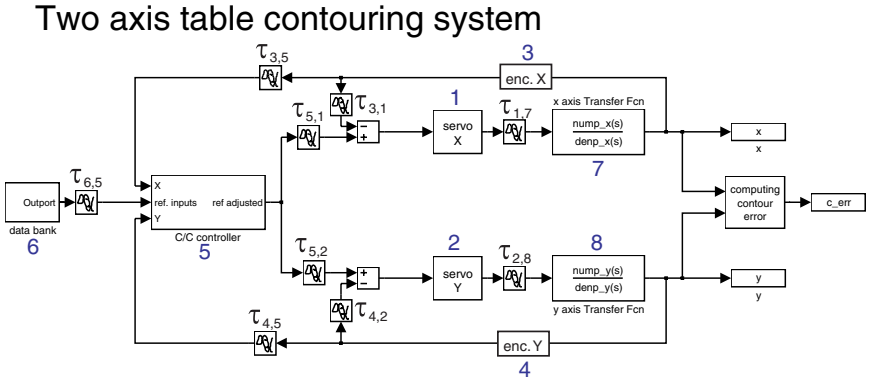


Figure 12: Two axis contouring system block diagram showing the eight basic modules and nine potential communication delays.

For the example system, we have chosen eight basic modules as shown in Figure 12:

- Two servo control (PI) modules, one each for the x and y axes, numbered 1 and 2
- Two position sensors (rotary encoders), one each for the x and y axes, numbered 3 and 4
- A cross-coupling control module to coordinate the two axis, numbered 5
- A reference trajectory generation module, numbered 6
- Two actuators (DC motors), one each for the x and y axes, numbered 7 and 8

Again, this set of the basic modules can be changed as desired. For example, the cross-coupling control module could be divided into two cross-coupling controllers for the x and y axes respectively, or the motor and the encoder for the x axis could become one basic module. Although the motor modules each include an amplifier, a gear, a leadscrew, and a tachometer, there is no need for network communication between them since these components are physically coupled. On the other hand, an encoder is considered as a basic module since it can easily be networked.

Once the basic modules have been identified, a block diagram of the system should be drawn showing the delay blocks at the input and output of each basic module. This identifies all possible delay locations if each basic module is connected directly to the network. The block diagram with all possible delays for the two-axis contouring system with the eight basic modules is shown in Figure 12. The basic modules are numbered from 1 to 8, and the time delay between modules i and j is denoted $\tau_{i,j}$. For this logical control architecture with eight basic modules, there are nine possible time delay locations.

5.2 Computation of performance differential functions, ϕ

After the basic modules and time delay locations have been identified, the performance differential functions associated with each time delay can be found either analytically or by simulation as discussed in Section 3. Computation of $\phi_{i,i}$ is not necessary since computation delay affects the system in the same way as communication delay. For example, the effect of a computation delay τ in the ith module is the same as the effect of a communication delay of the same magnitude τ between the ith module and all the modules it sends data to. Thus, $\phi_{i,i}$ can be computed using the following equation.

$$\phi_{i,i} = \sum_{j \neq i} \phi_{i,j} \tag{2}$$

Equivalently, the computation delay at node i could be added to all communication delays emanating from node i. We will use the former representation to maintain the distinction between communication and computation delays.

For a system with n basic modules, the maximum possible number of the performance differential functions that must be computed is n^2. In most practical cases, not every node will communicate with every other node, and the actual number of performance differential functions that must be computed will be significantly less. These performance differential functions can be put into a matrix $\underline{\phi} = [\phi_{i,j}]$. Each $\phi_{i,j}$ can be stored as a vector; the exact functional expression is not needed to compute the performance degradation function Φ. Note that some of the $\phi_{i,j}$ will be zero, indicating that there is no coupling between the corresponding modules. In addition, $\phi_{i,i}$ is zero if there is no computational task at the ith node. The performance differential function matrix for the example system is shown in Figure 13a. The size of the performance differential function matrix is 8×8 since the number of the basic modules for the example is 8. Note that

$$\underline{\phi} = \begin{bmatrix} \phi_{1,1} & 0 & 0 & 0 & 0 & 0 & \phi_{1,7} & 0 \\ 0 & \phi_{2,2} & 0 & 0 & 0 & 0 & 0 & \phi_{2,8} \\ \phi_{3,1} & 0 & \phi_{3,3} & 0 & \phi_{3,5} & 0 & 0 & 0 \\ 0 & \phi_{4,2} & 0 & \phi_{4,4} & \phi_{4,5} & 0 & 0 & 0 \\ \phi_{5,1} & \phi_{5,2} & 0 & 0 & \phi_{5,5} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \phi_{6,5} & \phi_{6,6} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \phi_{7,7} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \phi_{8,8} \end{bmatrix} \quad \underline{\tau} = \begin{bmatrix} \tau_{1,1} & 0 & 0 & 0 & 0 & 0 & \tau_{1,7} & 0 \\ 0 & \tau_{2,2} & 0 & 0 & 0 & 0 & 0 & \tau_{2,8} \\ \tau_{3,1} & 0 & \tau_{3,3} & 0 & \tau_{3,5} & 0 & 0 & 0 \\ 0 & \tau_{4,2} & 0 & \tau_{4,4} & \tau_{4,5} & 0 & 0 & 0 \\ \tau_{5,1} & \tau_{5,2} & 0 & 0 & \tau_{5,5} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \tau_{6,5} & \tau_{6,6} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \tau_{7,7} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \tau_{8,8} \end{bmatrix}$$

a. performance differential function matrix b. time delay matrix

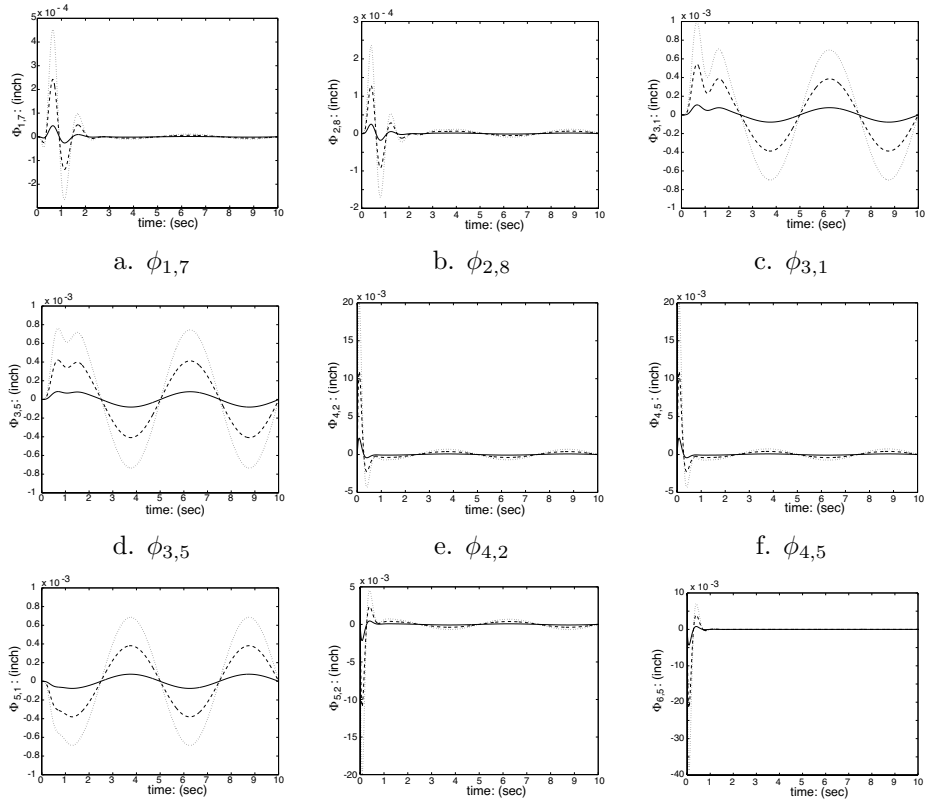
Figure 13: The performance differential functions $\phi_{i,j}$ and time delays $\tau_{i,j}$ of the system in Figure 12 are shown in matrix formation. Note that a number of elements of $\underline{\phi}$ and $\underline{\tau}$ are zero, indicating no communication between the respective modules. While $\underline{\phi}$ depends on the logical architecture (block diagram) of the system, it does not change with different implementation architectures. On the other hand, $\underline{\tau}$ depends only on the implementation architecture of the system (number of processing nodes, network protocol, operating system, etc.) The combination $\Phi = \sum_{i,j} \phi_{i,j}\tau_{i,j}$ determines the overall performance of the system.

a number of entries in the performance differential function matrix are zeros indicating that there is no communication between the respective modules.

Computation of the performance differential functions $\phi_{i,j}$ for the example system is done by simulation; the results are shown in Figure 14. To compute each $\phi_{i,j}$, the system is simulated with a delay $\tau_{i,j}$ and with no delay. The difference between the two simulation results divided by the magnitude of $\tau_{i,j}$ is $\phi_{i,j}$. When computing $\phi_{i,j}$, continuous-time models of the servo and cross-coupled controllers were used since the appropriate sampling period is not known at this stage. Although there are 17 non-zero entries for the performance differential function matrix in Figure 13a, only 9 performance differential functions are shown in Figure 14. This is because the diagonal terms $\phi_{i,i}$ can be computed using equation (2) as indicated above and detailed below.

$$\begin{aligned} \phi_{1,1} &= \phi_{1,7} \\ \phi_{2,2} &= \phi_{2,8} \\ \phi_{3,3} &= \phi_{3,5} + \phi_{3,1} \\ \phi_{4,4} &= \phi_{4,5} + \phi_{4,2} \\ \phi_{5,5} &= \phi_{5,2} + \phi_{5,1} \\ \phi_{6,6} &= \phi_{6,5} \\ \phi_{7,7} &= 0 \\ \phi_{8,8} &= 0 \end{aligned} \tag{3}$$

Associated with each nonzero performance differential function $\phi_{i,j}$, there will be a time delay $\tau_{i,j}$. The $\underline{\tau}$ matrix is shown in Figure 13b. The magnitude of $\tau_{i,j}$ will be determined by the architecture of the control system and network and choice of computing hardware and software. If two basic modules are placed together on a single node, the time delay between them will be zero.

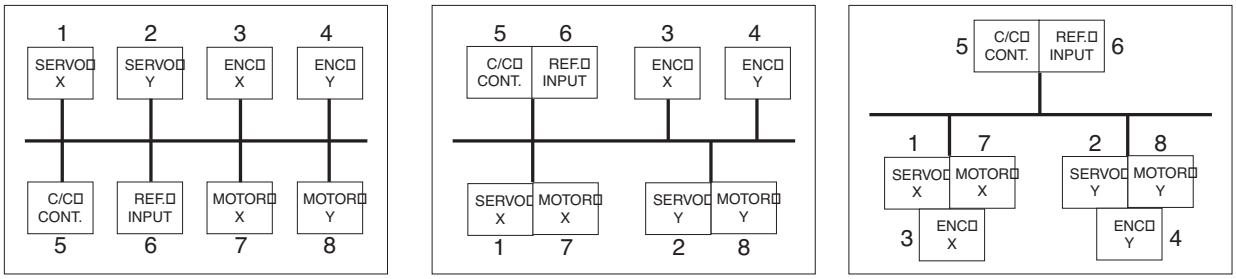


a. $\phi_{1,7}$ b. $\phi_{2,8}$ c. $\phi_{3,1}$

d. $\phi_{3,5}$ e. $\phi_{4,2}$ f. $\phi_{4,5}$

g. $\phi_{5,1}$ h. $\phi_{5,2}$ i. $\phi_{6,5}$

Figure 14: Individual performance degradation functions for different time delay locations and magnitude are shown. The function $\Phi_{i,j}$ is plotted for three different values of τ_r (0.001, 0.005, and 0.009 sec), showing that it is a linear function of $\tau_{i,j}$. Each performance differential function is obtained diving $\Phi_{i,j}$ by $\tau_{i,j}$



a. Completely distributed ($N = 8$) b. Lumped servo/motor ($N = 5$) c. Lumped axis control ($N = 3$)

d. Independent axes ($N = 2$) e. Centralized ($N = 1$)

Figure 15: Several possible network control architectures. The eight basic modules are assigned to between one and eight nodes. The number of nodes on the network, together with the communication protocol, will determine the communication delay; the number of basic modules per node, together with the operating system used, will determine the computation delay.

5.3 Determination of potential architectures

With n basic modules, there are $\sum_{i=1}^{n} \frac{n!}{i!(n-i)!}$ possible architectures. For each potential architecture, the corresponding time delay matrix $\underline{\tau}$ needs to be found. The performance degradation function for all cases can then be easily compared and the best architecture chosen. In this design example, we tested the five architectures shown in Figure 15. The two extreme cases — completely distributed and centralized — were chosen as candidates, along with three other natural groupings in which the servo controller is together with the motor. By grouping basic modules together on a single node, communication delays between the basic modules are eliminated. Many of the possible architectures would place modules together which don't require any communication between them; these cases would result in worse performance are not evaluated here.

5.4 Determination of $\underline{\tau}$ associated with each architecture

The delay matrix $\underline{\tau}$ contains the magnitudes of the communication and computation delays. These delays depend on a number of factors related to the system implementation. Communication delay mainly depends on the type of network, protocol used, data size, and number of nodes in the system. Computation delay depends on the processor power, operating system, and number of computing tasks assigned to each node. Finding accurate characterizations for computation and communication delay is another challenging problem and will not be addressed seriously in this paper (see [18, 19] for some work in this area). We made the following assumptions to get reasonable values for the communication and computation delay.

- each basic module has a task to be computed which takes $t_{tk} = 0.8$ msec
- the time required to switch between tasks is $t_{sw} = 0.3$ msec
- a motor module requires no computation time
- the computation delay is the time required for all other computation tasks on the node to be completed
- the message transmission time is $t_{tx} = 0.5$ msec, regardless of the size of the message
- the communication delay is the time required for all other nodes on the network to transmit their messages

Let N be the number of nodes on the network, and n_k be the number of basic modules assigned to node k. Using these assumptions, the communication delays are all equal:

$$\tau_{i,j} = t_{tx}(N-1) \tag{4}$$

For a basic module i on node k, the computation delay is

$$\tau_{i,i} = t_{tk}n_k + (n_k - 1)t_{sw} \tag{5}$$

Using these values, the time delay matrix $\underline{\tau}$ can be computed for the five different control architectures shown in Figure 15; the result is shown in Figure 16.

Even though a simplified model was used to estimate $\underline{\tau}$, well defined communication and computation delay characteristics, if available, can be employed to more accurately determine the time delay matrix $\underline{\tau}$. For example, if we are concerned with the releasing policy or priority assignment in the communication protocol, the time delays $\tau_{i,j}$ can take this into account. In this case, each different priority assignment or releasing policy for the same architecture would result in a different time delay matrix $\underline{\tau}$.

$$\underline{\tau}_a = \begin{bmatrix} 0.8 & 0 & 0 & 0 & 0 & 0 & 3.5 & 0 \\ 0 & 0.8 & 0 & 0 & 0 & 0 & 0 & 3.5 \\ 3.5 & 0 & 0.8 & 0 & 3.5 & 0 & 0 & 0 \\ 0 & 3.5 & 0 & 0.8 & 3.5 & 0 & 0 & 0 \\ 3.5 & 3.5 & 0 & 0 & 0.8 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3.5 & 0.8 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

a. Completely distributed ($N = 8$)

$$\underline{\tau}_b = \begin{bmatrix} 0.8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.8 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0.8 & 0 & 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0.8 & 2 & 0 & 0 & 0 \\ 2 & 2 & 0 & 0 & 1.9 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1.9 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

b. Lumped servo/motor ($N = 5$)

$$\underline{\tau}_c = \begin{bmatrix} 1.9 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1.9 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1.9 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1.9 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1.9 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1.9 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

c. Lumped axis control ($N = 3$)

$$\underline{\tau}_d = \begin{bmatrix} 4.1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 4.1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 4.1 & 0 & .5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 4.1 & .5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 4.1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 4.1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

d. Independent axes ($N = 2$)

$$\underline{\tau}_e = \begin{bmatrix} 7.4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 7.4 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 7.4 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 7.4 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 7.4 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 7.4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

e. Centralized ($N = 1$)

Figure 16: $\underline{\tau}$ is computed for distributed control architectures shown in Figure 15 using equations (4) and (5).

5.5 Computation of Φ and analysis

After computing $\underline{\phi}$ and $\underline{\tau}$, Φ can be computed using equation (1). One advantage of using the proposed performance degradation function framework is that $\underline{\phi}$ does not need to be recomputed or modified for different architectures. In fact, only $\underline{\tau}$ must be recomputed for each different architecture which is considered.

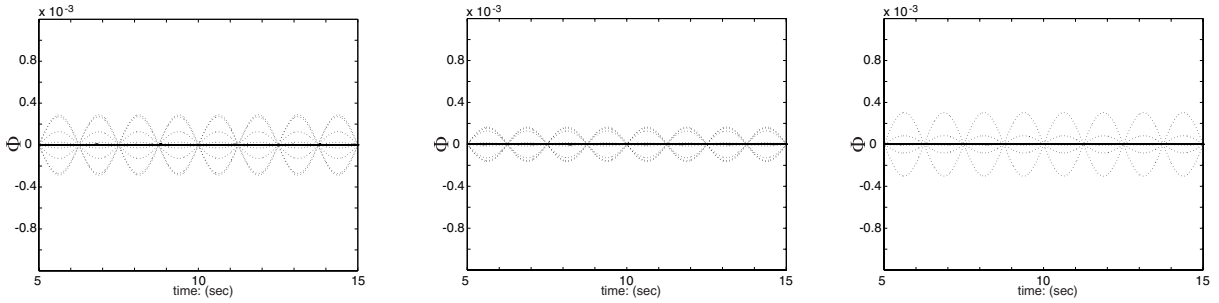
The performance degradation function for each architecture is computed and shown in Figure 17, along with the individual components $\Phi_{i,j}$. Note that Φ (solid line) in Figure 17 is of a similar magnitude for each architecture tested. This is due to the fact that the two-axis system is symmetric, and the communication delays are all equal, causing the terms $\Phi_{i,j}$ to largely cancel with each other. This phenomenon was noted in Section 4.1. If the delays are not equal, this cancellation will not occur. Thus, when choosing an architecture, both the overall performance degradation function Φ as well as its components $\Phi_{i,j}$ should be as small as possible. Thus, for the given assumptions on the magnitudes of the time delays, the lumped servo/motor architecture (b) would be the best implementation of the system.

If the reference trajectory is well-characterized (such as a step, ramp, or the circular contour considered here), then the Fourier transforms of the performance differential functions can be computed to extract more information about the best τ's to minimize the performance degradation function. For example, the steady state response region of ϕ's shown in Figure 14 is a collection of sinusoidal functions with different amplitudes and phases. The performance degradation function for the steady state can be expressed as a linear combination of these sinusoidal functions and the time delays. In this case, not only the impact of the magnitude of each delay to the performance of the system but also the interrelation between the delays can be seen algebraically. This information can be used to choose a communication protocol which can guarantee the desired relationships between different time delays.

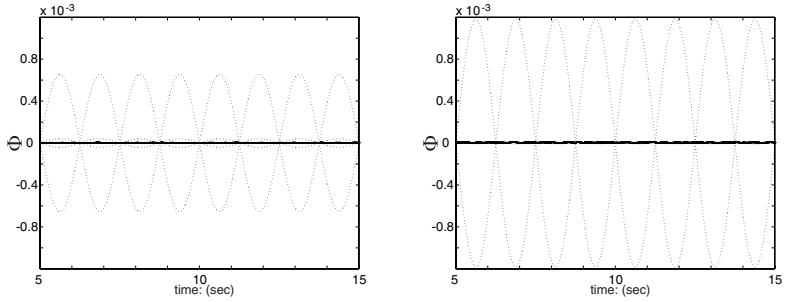
6 Conclusions and Future Work

Although more and more control systems are being implemented in a distributed fashion with networked communication, the unavoidable time delays in such systems impact the achievable performance. In this paper, we have presented a mathematical framework for analyzing the effect of time delays on the mechanical performance of distributed control systems. Once a control algorithm has been designed, the effect of different implementations of this algorithm can be determined using this framework. By defining the performance degradation function as the difference between the performance of the system with and without time delay, we are able to isolate the effect of the individual time delays from the effect of the controller. Once the individual performance differential functions have been computed, either analytically (for linear systems) or through simulation (for nonlinear systems), the overall effect of the time delays on a distributed system can be computed in a straightforward manner. Many different implementation architectures can be evaluated quickly using these techniques, allowing the system integrator to choose the best one. In addition, information gained in this analysis procedure can also be used to choose a control network protocol which can guarantee the desired relationship between different time delays in the system.

Much work remains to be done in this increasingly popular area of networked control systems. The work presented in this paper assumes that the time delays are well-characterized and constant. Although some network systems exhibit constant time delays, others have well-characterized time delays which may be nonconstant but bounded, or random and unbounded [19]. The analysis of control performance with bounded or random time delays remains an open problem. When the delays are random, the performance analysis will result in statistical performance guarantees. In this paper we also assume that the control algorithm for the centralized system has already been designed. New methodologies for designing control algorithms for distributed systems which can take advantage of the knowledge of the time delays in the



a. Completely distributed ($N = 8$) b. Lumped servo/motor ($N = 5$) c. Lumped axis control ($N = 3$)



d. Independent axes ($N = 2$) e. Centralized ($N = 1$)

Figure 17: The performance degradation function for each architecture is plotted. The dotted line indicates $\Phi_{i,j}$ and the thick solid line indicates Φ. Architecture b would be the best distributed architecture since the magnitude of individual performance degradation function is smaller than the other architectures even though the overall performance degradation function are same.

system must be developed. These methodologies should exploit the advantages of distributed systems but minimize the impact of the unavoidable time delays.

When implementing distributed control systems, a limiting factor is often the available communication bandwidth. Control methodologies which reduce the required bandwidth, especially in large-scale systems, will be required in the future. Some preliminary work in this direction, which trades increased computational resources for decreased communication, can be found in [36]. Estimators are implemented at each node, and the estimates are used to compute the feedback control law locally. The actual data is communicated only when the error between the estimate and the true value is greater than a predefined threshold. However, new control laws which only require some local data, rather than global state information, must be developed for true large-scale decentralized control systems to be successfully implemented. Integrating heterogeneous control structures, which may be supplied by different vendors or deployed at different times, into a unified distributed control architecture remains a challenge. Work also remains to be done on analyzing the reliability of distributed control systems.

References

[1] K. Arvind, K. Ramamritham, and J. Stankovic. A local area network architecture for communication in distributed real-time systems. *Journal of Real-Time Systems*, 3(2):115–147, May 1991.

[2] S. Cavalierei, A. Di Stefano, and O. Mirabella. Impact of fieldbus on communication in robotic systems. *IEEE Transactions on Robotics and Automation*, 13(1):30–48, 1997.

[3] H. Chan and Ü. Özgüner. Closed-loop control of systems over a communication network with queues. *International Journal of Control*, 62:493–510, 1995.

[4] J. B. Chen, B. S. R. Armstrong, R. S. Fearing, and J. W. Burdick. Satyr and the nymph: Software archetype for real-time robotics. In *Proceedings of the IEEE-ACM Joint Computer Conference*, Nov. 1988.

[5] J.-D. Decotignie, D. Auslander, and M. Moreaux. Fieldbus based integrated communication and control systems. In *Proceedings of the International Workshop on Advanced Motion Control*, pages 541–546, Tsu, Japan, 1996.

[6] L. Ferrarini, G. Ferretti, C. Maffezzoni, and G. A. Magnani. Hybrid modeling and simulation for the design of an advanced industrial robot controller. *IEEE Robotics and Automation Magazine*, 4(2):45–51, June 1997.

[7] P. Fiorini, H. Seraji, and M. Long. A PC-based configuration controller for dexterous 7-DOF arms. *IEEE Robotics and Automation Magazine*, 4(3):30–38, September 1997.

[8] G. F. Franklin, J. D. Powell, and A. Emani-Naeini. *Feedback Control of Dynamic Systems*. Addison-Wesley, Reading, Massachusetts, third edition, 1994.

[9] G. F. Franklin, J. D. Powell, and M. L. Workman. *Digital Control of Dynamic Systems*. Addison-Wesley, second edition, 1992.

[10] F. Göktas, J. M. Smith, and R. Bajcsy. μ-synthesis for distributed control systems with network-induced delays. In *Proceedings of the IEEE Conference on Decision and Control*, pages 813–814, 1996.

[11] Y. Halevi and A. Ray. Performance analysis of integrated communication and control system networks. *ASME Journal of Dynamic Systems, Measurement, and Control*, 112:365–371, September 1990.

[12] A. İftar and E. J. Davison. A decentralized discrete-time controller for dynamic routing. *International Journal of Control*, 69(5):599–632, March 1998.

[13] P. K. Khosla. Choosing sampling rates for robot control. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 1987.

[14] Y. H. Kim, W. H. Kwon, and H. S. Park. Stability and a scheduling method for network-based control systems. In *Proceedings of the IEEE Industrial Electronics Conference (IECON)*, volume 2, pages 934–939, 1996.

[15] K. Kobayashi, M. Kameyama, and T. Higuchi. Communication network protocol for real-time distributed control and its LSI implementation. *IEEE Transactions on Industrial Electronics*, 44(3):418–425, June 1997.

[16] H. Kopetz. A communication infrastructure for a fault-tolerant distributed real-time system. *Control Engineering Practice*, 3(8):1139–1146, August 1995.

[17] Y. Koren. Cross-coupled biaxial computer control for manufacturing systems. *ASME Journal of Dynamic Systems, Measurement, and Control*, 102(4):265–272, 1980.

[18] F.-L. Lian, J. M. Moyne, and D. M. Tilbury. Performance evaluation of control networks: Ethernet, ControlNet, and DeviceNet. Technical Report UM-MEAM-99-02, University of Michigan, Mechanical Engineering and Applied Mechanics, 1999.

[19] F.-L. Lian, J. M. Moyne, and D. M. Tilbury. Performance evaluation of control networks for manufacturing systems. In *Proceedings of the ASME International Mechanical Engineering Congress and Exposition (Dynamic Systems and Control Division)*, volume 67, 1999.

[20] M. Malek-Zavarei and M. Jamshidi. *Time-Delay Systems Analysis, Optimization, and Application*. North-Holland, 1987.

[21] Ü. Özgüner. Decentralized and distributed control approaches and algorithms. In *Proceedings of the IEEE Conference on Decision and Control*, pages 1289–1294, December 1989.

[22] F. Paganini. A recursive information flow system for distributed control arrays. In *Proceedings of the American Control Conference*, volume 6, pages 3821–3825, June 1999.

[23] A. Ray. Output feedback control under randomly varying distributed delays. *AIAA Journal of Guidance, Control, and Dynamics*, 17(4):702–711, 1994.

[24] A. A. Rizzi, L. L. Whitcomb, and D. E. Koditschek. Distributed real-time control of a spatial robot juggler. *IEEE Computer*, pages 12–24, May 1992.

[25] K. G. Shin and C. M. Krishna. New performance measures for design and evaluation of real-time multiprocessors. *Computer Systems Science and Engineering*, 1(4):179–192, October 1986.

[26] K. G. Shin, C. M. Krishna, and Y.-H. Lee. A unified method for evaluating real-time computer controllers and its application. *IEEE Transactions on Automatic Control*, 30(4):357–366, April 1985.

[27] D. Simon, E. Castaneda, and P. Freedman. Design and analysis of synchronization for real-time closed-loop control in robotics. *IEEE Transactions on Control Systems Technology*, 6:445–461, July 1998.

[28] D. Song, T. Divoux, and F. Lepage. Design of the distributed architecture of a machine-tool using FIP fieldbus. In *Proceedings of the IEEE Conference on Application-Specific Systems, Architectures, and Processors*, pages 250–260, 1996.

[29] K. Srinivasan and P. K. Kulkarni. Cross-coupled control of biaxial feed drive mechanisms. *ASME Journal of Dynamic Systems, Measurement, and Control*, 112:225–232, 1990.

[30] M. Tayara, N. Soparkar, J. Yook, and D. Tilbury. Real-time data and coordination control for reconfigurable manufacturing systems. In A. Bestavros and V. Wolfe, editors, *Real-Time Database and Information Systems: Research Advances*. Kluwer Academic, Norwell, MA, 1997.

[31] M. Törngren. On the modelling of distributed real-time control applications. In *Proceedings of the IFAC Distributed Computer Control Systems Workshop*, pages 13–18, 1995.

[32] M. Törngren and J. Wikander. A decentralization methodology for real-time control applications. *Control Engineering Practice*, 4(2):219–228, February 1996.

[33] G. C. Walsh, H. Ye, and L. Bushnell. Stability analysis of networked control systems. In *Proceedings of the American Control Conference*, pages 2876–2880, June 1999.

[34] K. Watanabe, E. Nobuyama, and A. Kojima. Recent advances in control of time delay systems—a tutorial review. In *Proceedings of the IEEE Conference on Decision and Control*, pages 2083–2088, 1996.

[35] J. Yook, D. Tilbury, K. Chervela, and N. Soparkar. Decentralized, modular real-time control for machining applications. In *Proceedings of the American Control Conference*, pages 844–849, 1998.

[36] J. K. Yook, D. M. Tilbury, H. S. Wong, and N. R. Soparkar. Trading computation for bandwidth: State estimators for reduced communication in distributed control systems. In *Proceedings of the Japan–U.S.A. Symposium on Flexible Automation*, 2000. To appear.