

A Class of Networked Control Systems: Architecture, Design and Implementation

S. Hart, ** N. Vozdolsky, * T. E. Djaferis *

*Electrical and Computer Engineering

**Computer Science

University of Massachusetts

Amherst MA 01003

Abstract

This paper deals with a class of networked control systems whose controller architecture consists of two components, one “local” and the other “remote.” We first propose a framework for the analysis and design of controllers for this architecture and show how communication delays can be addressed. We discuss how the JINI Technology client/server environment can be used for controller implementation, and apply this methodology for the network control of CIMCAR (Computer Intelligent Model CAR) an experimental facility developed for educational use.

1. Introduction

The Internet and Local Area Networks are great resources for real-time control applications as they provide a flexible and powerful infrastructure for implementation. They naturally offer the ability to distribute sensing, control, and actuation, advantageous for many applications. At the same time, a number of factors such as communication delays (limited data rates) and data quantization have adverse effects on overall system operation and performance. Safe and reliable operation of feedback systems requires that data moves swiftly and accurately through a network, and limited network bandwidth and finite word-length are factors that adversely affect these requirements. If these factors are not properly addressed in the system structure and in the manner in which controllers are designed, instabilities and unacceptable system performance can result. Therefore, it is imperative that appropriate controller architectures be proposed and control design methodologies employed to address these issues.

Networked control systems can be approached from a variety of settings, employing a number of system models and focussing on several system characteristics. Issues such as limited data rates and delays [1, 10], data quantization [1, 6], have been addressed in several scenarios and a number of results have been reported. In cases where the controller architecture can be manipulated, this also can be used as an “extra degree of freedom.” Indeed, in this paper we argue that if we are allowed to manipulate the overall system structure then we may be able to suggest one model that has improved robustness properties. We work in the context of linear time invariant systems where the overall system is connected via a

network (e.g., Internet) making possible a number of control architectures. One may choose to centralize control computation (i.e., all sensor data is collected and actuator commands issued at a central location) or choose to distribute sensor data flow and control computation. For such a networked system we will propose a controller architecture that “splits” controller effort into two components, one “local” to the plant and the other “remote.” We argue that such a setting is appropriate in a number of applications.

Within the context of this split-controller architecture we proceed with the suggestion of a framework for controller design. We employ a frequency domain approach and approximate network delays by rational transfer functions where the delay appears as a parameter. We proceed in this fashion, as our primary objective is controller design and we want to exploit available frequency domain tools. This allows us to define closed loop system specifications and design controllers that meet these requirements. The design problem can then be cast in the frequency domain and we see that robust design tools for systems with parameter uncertainty [3] are very relevant.

Our investigation continues with the choice of a software environment for implementation. In this paper we choose the JINI Technology framework developed by Sun Microsystems. It is implemented as a layer on top of the Java programming API and is a distributed server/client system architecture. Other possibilities exist but our intention here is to use an “off-the-shelf” software product and evaluate its performance in a real-time feedback control application. In fact, we proceed one step further and implement the entire process on an experimental testbed. We chose to use CIMCAR (Computer Intelligent Model CAR) which is an experimental facility developed for educational purposes.

The paper is organized as follows. In section 2 we propose the “split-controller” architecture and in section 3 suggest a framework for controller design that pays special attention to communication delays. In section 4 we discuss the JINI software environment as well as the interface required for CIMCAR. In section 5 we present data from real-time experiments that allow us to characterize network delays for our application and show experimental results from the

network control of CIMCAR. In section 6 we give initial reactions on the use of JINI in network controlled systems.

2. A Controller Architecture

In this section we introduce the split-controller architecture depicted in Figure 1 (using a “position control” application). The controller architecture consists of two components, one labeled “local controller” that affects the plant directly, and another labeled “remote controller” which receives a delayed error signal, its output affecting the plant after some delay. One can envision a number of control applications for which this control architecture is appropriate. In particular, consider a system where the local controller may be a low-cost, analog component and the remote controller being a digital microcontroller which provides flexibility and may be shared by a number of feedback loops. The remote controller has access to the error signal over some network (e.g., LAN or Internet) and its output reaches the actuator via this same network.

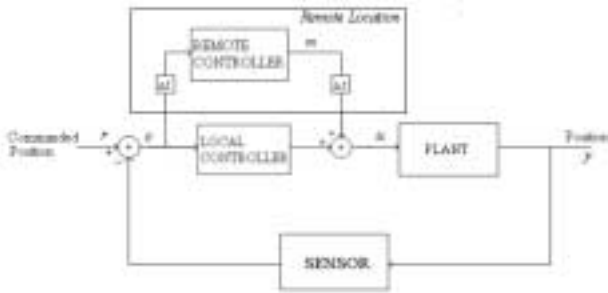


Figure 1: Split-Controller Architecture

Having a “shared” microcontroller might provide cost savings, as the alternative would be to have powerful devices for each feedback loop. Clearly, it is possible to implement the entire controller at the remote location as depicted in Figure 2. However, this implementation would have more stringent requirements on network delays and have reduced robustness properties compared to the earlier implementation. The split-controller architecture can be seen to intuitively have desirable properties. If the “local loop” is designed to ensure closed loop system stability and a certain level of basic performance, then even when the network “goes down,” we will still be able to maintain a certain level of performance. In fact, we see in section 5, that even though network delays are normally low, there are observed instances of extremely long delays. Clearly, it may be impractical or even impossible to implement network controllers that guarantee system performance in the presence of such very long delays. Having the local loop operational all the time provides greater reliability.

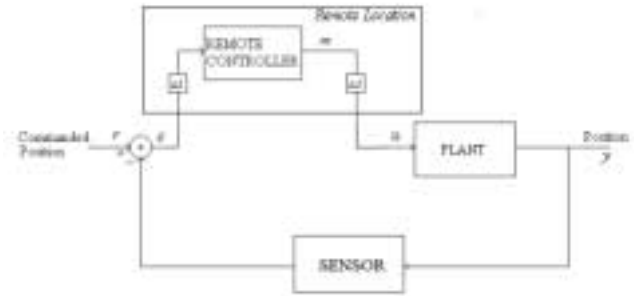


Figure 2: Network Controller Architecture

3. A Framework for Controller Design

We now turn our attention to the question of controller design. At this point we need to choose between a time-domain approach using state-space models or a frequency domain approach employing transfer functions. We choose the frequency domain and propose a framework for analysis and design. The frequency domain formulation will not allow us to study time-varying characteristics of the loop. Interconnected systems with time delays have been studied in the past. Our class of systems has special structure which can be exploited. Our approach will allow the formulation of a number of control problems where frequency domain machinery will be employed in their solution.

We replace the transfer function delay blocks with rational function approximations. This does introduce “inaccuracies” to the analysis but it nevertheless makes the design problem more tractable. Our major focus is dealing with network delays, which in the frequency domain can be modeled with rational function approximations. Consider the transport delay e^{-Ts} . Clearly, network delay will not be constant but rather time-varying. However, as we will see later one can identify a range of values for the delay and a piecewise constant approximation is quite valid. Such a term can be approximated by a rational function of some order [9] and a first order approximation is given by:

$$\frac{n_{\Delta}(s)}{d_{\Delta}(s)} = \frac{1 - \frac{T}{2}s}{1 + \frac{T}{2}s}$$

Let the plant, local controller and remote controller be described by their corresponding transfer functions:

$$P(s) = \frac{n_p(s)}{d_p(s)}, \quad C_l(s) = \frac{n_l(s)}{d_l(s)}, \quad C_r(s) = \frac{n_r(s)}{d_r(s)},$$

One can now evaluate the dynamic characteristics of the overall system by examining the closed loop behavior in the frequency domain. We proceed in this manner knowing full well that this linear, time-invariant model is only an

approximation of the actual networked control system. We do so for two reasons: 1) Controller design becomes much more transparent, and 2) In many cases this is a good approximation of the actual system. Once candidate controllers have been designed then one can employ other more accurate analysis tools (e.g., Lyapunov Theory) to verify and validate the design. This methodology has served us well in many other control applications and there is ample reason to believe that it will do so here as well. If we lump the two delay blocks into one block ($\alpha = 2T$) the closed loop characteristic polynomial $\phi(s, \alpha)$ is:

$$\phi(s, \alpha) = d_r(d_p d_l + n_p n_l) + n_r n_p d_l + \alpha s(d_r(d_p d_l + n_p n_l) - n_r n_p d_l)$$

One can begin to study the dynamic properties of the loop where α is a parameter. A very relevant question one can pose is the following: how should one design the controllers $C_b(s)$, $C_r(s)$ such that system stability is maintained for the largest possible value of α ? In other words, we would like to know how to “split” the controllers so that the overall system exhibits stable behavior for the largest possible delay. Some of these issues have been addressed in [4]. Many other design objectives having to do with transient and steady state response can also be addressed. The plant may also involve uncertainty, which would add to the complexity, but these problems can be investigated in this context. It is interesting to point out that this is not a “standard” design problem for which a solution can be immediately given. However, the setting is appropriate for the application of a number of robust control design tools (e.g., see [3]).

4. Overview of the JINI CIMCAR System

In this section, after giving a brief description of what JINI is and how it works, the JINI CIMCAR system will be described.

A. JINI Technology

JINI, implemented as a layer on top of the Java programming API, is a distributed server/client system architecture developed by Sun Microsystems in 1998. JINI is highly manageable, and requires practically no administrative support, while still being easily maintainable when components of the network fail or go down. JINI provides the capability of “network plug and play” to distributed systems [8]. In this way, devices can join a network, and become part of a JINI community of other services already present [8]. The devices connected through a JINI system can interact through real-time communication or by the transfer of complete programs written in any programming language. Although JINI is an extension of Java, it is still at its core pure Java [5]. Because of this, a JINI program can be imbedded into a Java Applet and published on the Internet (but they don’t have to be). This allows a JINI client to use a system with only a web-browser.

There are three main components of any JINI system: a service, a client, and a lookup service (or service locator). Each component must have a Java Virtual Machine (JVM) in which Java bytecode can be translated into machine code, or at least in direct communication with some proxy computer that does have a JVM. It also must be able to connect to a TCP/IP network, giving it an IP address and allowing it to send and receive multicast messages [5]. When a service comes on-line it must publish itself to the lookup service, specifying which Java language *interfaces* it implements. Clients can then gain control of a service by going to the service locator and inquiring about the availability of any services of a certain type – services that implement a particular interface. The lookup service can then grant or deny the client the control of any of these services. An example of a JINI enabled system in action might be a digital camera that wishes to print its pictures. The camera, with its JVM, connects to the JINI community and tells the service locator that it wants to access a printer service. If a printer service is available, the camera will be able to download code from that printer (the information it needs to know how to use this specific printer), and then print the pictures it wants.

B. The CIMCAR experiment

Our experimental testbed is CIMCAR (Computer Intelligent Model Car), shown in Figure 3. CIMCAR is a small electronic car with an 8051 microprocessor and sonar sensor capable of measuring distance away from an object.

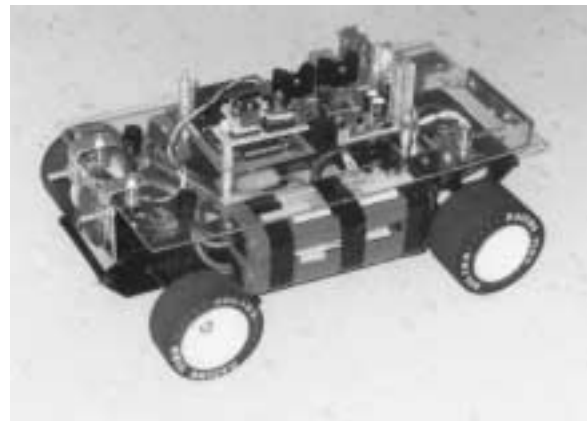


Figure 3: CIMCAR

Consider a collision avoidance experiment where CIMCAR starts a certain distance away from a wall and is commanded to move towards that wall, stopping three feet away. The move should be completed in less than 4 seconds and the response should not exhibit any overshoot. This is a “standard” tracking control problem that can be easily solved if the controller is to be implemented in a “standard” feedback control loop. Here, however, we want to use the “split controller” architecture and investigate the operation of a networked control system. We will assume that a “Lead” controller will be implemented with the local

controller being a pure gain and the remote controller a first-order component.

C. The JINI CIMCAR interface

The first issue that needed to be addressed in order to design a JINI system to control CIMCAR, was the development of an appropriate service interface that JINI clients could look for when wishing to perform a control task, maintaining a level of abstraction similar to [7]. The `controllableCarInterface`, was designed to represent any type of controllable car (CIMCAR included), with one method, `initCar()`, to provide initialization, and one method, `setPower()`, to perform the control computation. `initCar()` must be called before the control loop begins, specifying the reference signal for the control task, and the maximum amount of time the control loop is allowed to run before termination (a “time-out” parameter). The `setPower()` method, called by the client on the service, can be used to implement the remote controller in the split-controller system. When the client wishes to send a command signal to the service it calls this method, sending its control signal as a parameter, and is returned the error signal, which can in turn be used to compute the next control signal. The resulting service interface is shown below.

```
public interface controllableCarInterface
    extends java.io.Serializable {

    /* Init call to specify reference and time-out */

    public boolean initCar(double r, int maxruntime)
        throws java.rmi.RemoteException;

    /* Set the power level of the motors on the car
       and return the error signal */

    public double setPower(String value)
        throws java.rmi.RemoteException;

}

```

When the user starts the client program, a JINI browser appears, displaying to that user all of the available JINI lookup services available on the network. If the user selects a lookup service, all of the JINI services that implement the `controllableCarInterface` appear in the middle panel. When the user selects a service, certain attributes of that service, such as its name and physical location, can be seen. Future expansion will be to include the system model along with these attributes. The JINI browser can be seen in Figure 4.

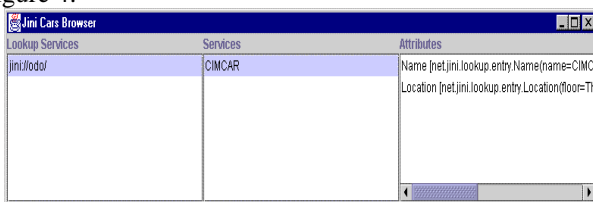


Figure 4: The JINI browser

When the user finds service that it wishes to gain control of, it can double-click on that service. If the lookup service grants the client control of that service, a “Controller Panel,” such as the one shown in Figure 5, appears that allows the client to enter parameters of a discrete first-order controller, and to begin running the split controller loop.

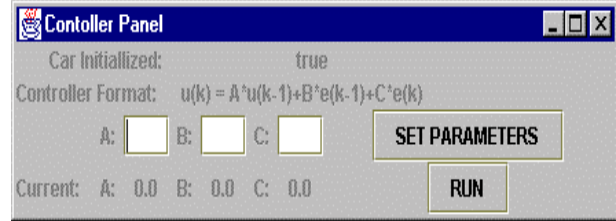


Figure 5: The Controller Panel

5. An Implementation

In this section we show experimental results from an implementation of the proposed split-controller methodology. Our experiment has CIMCAR start eight feet away from a wall and come to a complete stop three feet from the wall. Our design specifications are 1) the move should be completed in less than 4 seconds, 2) the motion should not exhibit overshoot, and 3) CIMCAR should not crash into the wall (instability). Our focus in this experiment is not so much to demonstrate the controller design process but rather the operation of the entire system. In particular, the design of the split-controller does not employ all the power of the tools available to us and neither do we strive to design one that can accommodate the largest possible network delay.

A. Characterization of Network Delay

In the controller design process it is important to “model” network delays. Here we explain an experiment that was performed using the JINI CIMCAR system set up to send signals from the client to the server continuously for an indefinite amount of time. Every time the service program received a signal from the client, an absolute time stamp was stored and eventually written to a log file. This program was run for two twenty-four periods, and one twelve-hour period. From the time stamps in the log file, delay intervals could be computed easily, and then plotted in MATLAB. Examining the results of the three tests, we have evidence that there are three types of delay intervals. The first delay type is the interval at which the system server and client communicate most of the time: an oscillating value of 50 and 60 milliseconds. This oscillation can be attributed to the fact that the Java `System.currentTimeMillis()` command used to generate the time stamp only returns values to then nearest 10 milliseconds, creating time stamps at intervals of both 50 and 60 milliseconds, where in fact the actual interval is somewhere between 50 and 60 milliseconds. We will approximate this interval with a value of 55 milliseconds. This was the smallest interval observed from all

experiments, and we will therefore assume that this is the fastest rate at which the communication between the service and the client can operate on our network. The second type of delay observed from our experiments contributed seemingly random spikes of intervals ranging from 100-300 milliseconds. The last type of delay, observed only in the first set of data, was that of two and a half seconds. From a real-time control perspective, this is no better than an infinite delay and will, as a result, be very hard to compensate for. Figure 6 shows the delay intervals over the first twenty-four hour period.

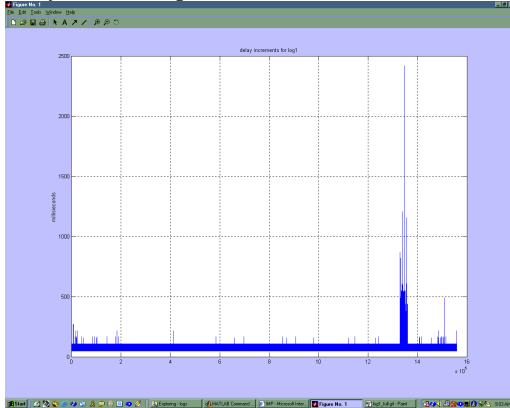


Figure 6: Network delay for a twenty-four hour period

B. Controller Design

In view of the experimental observations of network delay we will design a controller that can handle up to a 300 millisecond delay. The plant transfer function (see [2]) is given by:

$$P(s) = \frac{28.5}{s(s+3.8)}$$

and the two controllers are chosen to be:

$$C_l(s) = 0.13, \quad C_r(s) = \frac{s+1.1}{0.5s+10}$$

One can check that the “local controller” stabilizes the loop and provides the required performance. The combination of local and remote controllers improves the performance (in the absence of delay). Controller implementation for the remote controller is done in discrete-time using the controller (obtained from the continuous-time version via the matched-pole-zero method using a 0.1 second sampling interval):

$$C_r(z) = \frac{0.91z - 0.82}{z - 0.14}$$

Figure 7 shows the Simulink response of this controller with transport delays of 0 ms, 55 ms, and 300 ms, and compared with the response of the local gain controller alone. The response of the system with the local controller is the “slowest” response curve shown in Figure 7. In view of the fact that observed network delays could on occasion reach several seconds it is important to develop methods for addressing this problem. One possibility is to attempt the design of controllers that can tolerate such long delays.

Trying to design for such a delay may either be impossible or infeasible. Therefore, other methods should also be

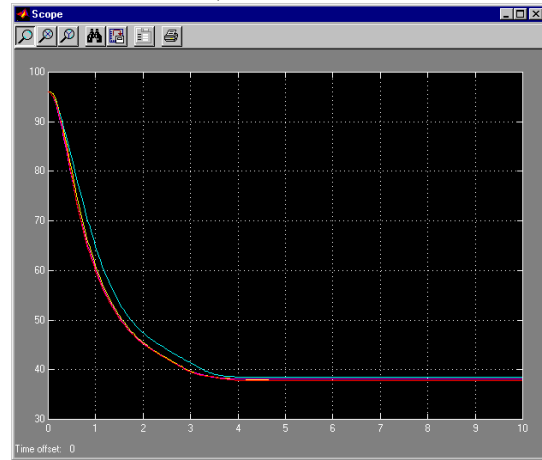


Figure 7: Simulation of system response with varying delays

considered. The concept of “time stamping” is an obvious one so we will proceed with such an implementation. We program the JINI software to implement the split-controller loop, and also to reject any control signal from the client (remote controller) that arrives after a specified time interval. This rejection protects the system from delays of an “infinite” type, and was implemented by hard-coding the control loop period at 100 milliseconds. At every iteration of this loop, the program waits for its period of 100 milliseconds to elapse. During this time the client has access to a variable on the service side in which it can store its control effort. When the client modifies this variable, it sets a flag indicating to the service that it has just produced a valid signal. This flag is initialized at the beginning of each control loop iteration, when the command to the client is issued. According to the JINI model, this is sufficient to guarantee that if the service receives a signal before the waiting period has finished, it is a valid signal. When remote methods are invoked in a JINI application, the system will either return with data, after however long it may take, or cause an exception to be thrown. If an exception is thrown, the service will know immediately that it has missed the data it was waiting for. This also implies, that our JINI client, therefore, automatically guarantees that the data will be received in order, and the control signal is valid for the current error signal. If this is the case, the service will add the client’s control effort to its own to determine the signal to send to the plant. If this flag is not set, the service will produce the control effort to the actuators from the effort made only by the local controller. This approach was taken to add simple “time stamps” to the design, where the remote signals are guaranteed to be received in the correct order that there were sent and that any particular signal is valid for the current state of the system, according to the controller design. More sophisticated time-stamps can be implemented, in which an indication of the absolute time (and order) that a signal was

sent is packaged with the control signal and is used in determining if that remote signal should be used. In so doing we are in essence introducing “controller switching” in the system operation. However, notice that the switching interval can be controlled so chattering may be avoided. Experimental results from the operation of this control algorithm are shown in Figure 8. The results of the split system are compared against the performance of the local gain controller only.

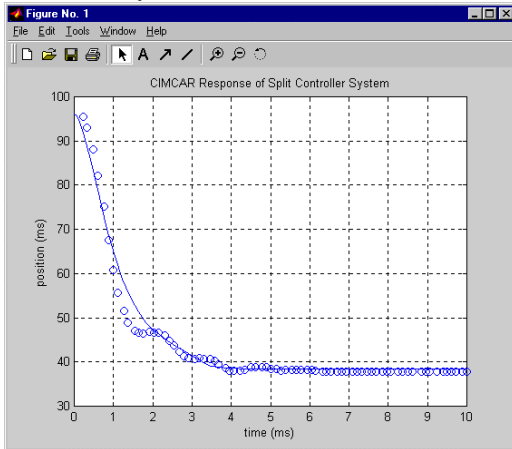


Figure 8: Performance of CIMCAR split-controller system

We can see from this plot that the response of the split system (as shown by the circles) was generally faster than that of the expected gain controller. The steady-state error also appears to be better by about an inch. The lack of a smooth response is most likely the result of controller switching. These are preliminary results and work along these lines is continuing.

6. Initial Observations

In the course of the development of our implementation certain limitations of the JINI framework have been observed. Although, initially promising an “easy-to-maintain” developing environment, actual implementation resulted in a number of complications. Most significantly from a real-time control point of view, the operation speed of the JINI programs was less than ideal. JINI, being “pure Java,” must deal with a Java Virtual Machine, thus adding an entire interpretive layer to each program and which could be avoided in other conventional programming languages. The 55 millisecond fastest (and approximate) delay observed in our split-controller implementation was the best that we could do with our JINI setup, even after a certain amount of optimization. Experiments pinging the client machine showed round-trip times of rarely more than ten milliseconds, demonstrating that the extra 45 milliseconds were a result of program execution, and not of network latency. These relatively long control periods would certainly limit our performance abilities in many control tasks. Another drawback of the JINI framework was a number of archaic and non-intuitive configuration

steps involved with developing both the client and the server programs. These steps do not represent the relative ease of programming that can be exhibited by Java at its best, and, it might be noted, are virtually non-existent in certain other client/server frameworks. For example, Real-Time Innovations’ NDDS implementation, designed for use in real-time control, automatically generates the code to handle the network communication, making this step almost transparent to the developer, while also providing more sophisticated time-stamping and ordering tools. One certain benefit, however, of JINI Technology is its low cost, the entire API being freely downloadable from Sun Microsystems.

7. Conclusion

In this paper we have presented a split-controller architecture for a class of networked control systems. We have demonstrated that within the context of this structure frequency domain design techniques can be employed to design controllers that guarantee closed loop performance in the presence of network delays. We suggested the JINI software framework for implementing the control architecture over the Internet and carried out experiments on the CIMCAR experimental facility. Initial results are encouraging and work is continuing in all aspects of the methodology.

References

- [1] R. W. Brockett, “Stabilization of Motor Networks,” *Proceedings, 34th IEEE CDC*, New Orleans, LA, 1995.
- [2] T. E. Djaferis, “*Automatic Control: The Power of Feedback Using MATLAB®*,” Boston, MA: PWS Publishing Company, 1998.
- [3] T. E. Djaferis, “*Robust Control Design: A Polynomial Approach*,” Kluwer, Boston, 1995.
- [4] T. E. Djaferis, “Controller Design for a Specific Distributed Architecture,” to be submitted, 2003 ACC.
- [5] W. K. Edwards, *Core JINI™*. Upper Saddle River, NJ: Prentice Hall, 1999.
- [6] N. Elia, S. K. Mitter, “Quantized Linear Systems,” *System Theory: Modeling, Analysis and Control*, T. E. Djaferis and I. Schick, Eds, Kluwer, Boston, 2000.
- [7] Holness, G., Karuppiah, D., Uppala, S., and Ravela, S. C., Grupen R. “A Service Paradigm for Reconfigurable Agents,” Proc. of the 2nd Workshop on Infrastructure for Agents, MAS, and Scalable MAS, Montreal, Canada, May 2001.
- [8] J. Newmarch, “JINI and Mindstorms,” [Online Document], 1999 Aug 2, [cited 2001 Dec 26], Available HTTP: <http://pandonia.canberra.edu.au/java/jini/paper.html>
- [9] B. A. Ogunnaik and W. H. Ray, “*Process Dynamics, Modeling, and Control*,” New York: Oxford University Press, 1994.
- [10] G. C. Walsh, H. Ye, L. Bushnell, “Stability Analysis of Networked Control Systems,” *Proceedings ACC*, 1999.