CLOSED-LOOP REAL-TIME CONTROL ON DISTRIBUTED NETWORKS

A Thesis

by

AJIT AMBIKE

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

August 2004

Major Subject: Mechanical Engineering

CLOSED-LOOP REAL-TIME CONTROL ON DISTRIBUTED NETWORKS

A Thesis

by

AJIT AMBIKE

Submitted to Texas A&M University
in partial fulfillment of the requirements
for the degree of

MASTER OF SCIENCE

Approved as to style and content by:

Won-jong Kim
(Chair of Committee)

Richard Volz
(Member)

Dennis L. O'Neal
(Head of Department)

Suhada Jayasuriya
(Member)

August 2004

Major Subject: Mechanical Engineering

ABSTRACT

Closed-Loop Real-Time Control on Distributed Networks. (August 2004)
Ajit Ambike, B.E., Government College of Engineering, Karad, India
Chair of Advisory Committee: Dr. Won-jong Kim

This thesis is an effort to develop closed-loop control strategies on computer networks and study their stability in the presence of network delays and packet losses. An algorithm using predictors was designed to ensure the system stability in presence of network delays and packet losses. A single actuator magnetic ball levitation system was used as a test bed to validate the proposed algorithm. A brief study of real-time requirements of the networked control system is presented and a client-server architecture is developed using real-time operating environment to implement the proposed algorithm. Real-time performance of the communication on Ethernet based on user datagram protocol (UDP) was explored and UDP is presented as a suitable protocol for networked control systems. Predictors were designed based on parametric estimation models. Autoregressive (AR) and autoregressive moving average (ARMA) models of various orders were designed using MATLAB and an eighth order AR model was adopted based on the best-fit criterion. The system output was predicted several steps ahead using these predictors and control output was calculated using the predictions. This control output output was used in the events of excessive network delays to maintain system stability. Experiments employing simulations of consecutive packet losses and network delays were performed to validate the satisfactory performance of the predictor based algorithm. The current system compensates for up to 20 percent data losses in the network without loosing stability.

To my loving parents.

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my advisor, Professor Won-jong Kim; without his guidance this thesis would not have been accomplished. I appreciate his time and effort throughout the duration of my research.

I wish to extend my thanks to Professors Richard Volz and Suhada Jayasuriya for serving on my thesis committee. I am grateful to them for their valuable guidance.

I am fortunate to have the company of a talented and motivated bunch of colleagues in the Networked Intelligent Machine Lab, Texas A&M University. I wish to thank Stephen C. Paschall, II for developing the single-actuator ball levitation system that was used as a test bed for the experiments. I am thankful to Abhinav Srivastava for his valuable suggestions during the course of my research. I would also like to thank Aninda Bhattacharya, Yash Shukla, Kun Ji, Chirag Shah, and Shantur Tapar for their time and constructive suggestions. My special thanks to Arvind Ramarathinam for his help in setting up the real-time operating environment on lab computers.

My wholehearted gratitude to my parents, Shubhangi and Dilip Ambike, for their immeasurable support throughout my life. Without their unconditional love, encouragement and support, I could have never come so far.

TABLE OF CONTENTS

CHAPTER Page

LIST OF FIGURES

FIGURE Page

FIGURE Page

LIST OF TABLES

CHAPTER I

INTRODUCTION

A. Introduction

With the advancement in the automation industry, the need to perform complex remote operations has grown. Ever-increasing computational capabilities and advancements in the networking technology have aided researchers to develop architectures to implement control from a distance. In large-scale control applications of the modern industry, the functional agents such as sensors, actuators, and controllers are geographically distributed. For smooth working of a control application, all the agents have to exchange information through communication media. These control systems, where the functional agents are interconnected by communication networks are called networked control systems (NCSs) [1]. The success of a NCS depends on the efficient integration of computing resources, communication network, and control algorithms in different levels of the automation industry.

Communication network is the key component in such advanced automation environment. The easy and reliable exchange of information has to be ensured on these networks. An ideal networked architecture provides several attractive features such as easy installation, reconfiguration, and can reduce the setup and maintenance costs. In addition to these requirements, the ability of transferring data with stringent timing constraints on these networks is also crucial for real-time performance of networked control systems. Time delays are introduced in network communication due to the time sharing of communication media as well as additional functionality required for physical signal coding and communication processing. There is a need for designing

---

The journal model is *IEEE Transactions on Automatic Control.*

control architectures that ensure the stability in the presence of such time delays.

Various forms of distributed control systems have been put to use in the automation industry. Teleoperated control has been extensively studied and applied to a large variety of applications in hazardous nuclear environment, space explorations, and surgical operations. Supervisory control demonstrated its use in distance-learning environments, mining industry, and defence applications. The automation industry will soon reap the benefits of high-performance closed-loop control on distributed networks. Although this field is relatively new and still in its infancy, it has captured significant interest of many researchers worldwide.

B. Internet

The Internet is a very fast evolving global computer network. In today's world, it plays an important role in people's lives. It provides a convenient channel for receiving information, electronic communication, and conducting business. The world wide web (WWW) has extended the range of services which are available to the Internet users. One of the most important reasons for its popularity is that it is free. Some of its other advantages are versatility, wide availability, and ease of use. It provides information anywhere, anytime, and permits the presentation of information in various forms on various operating platforms.

With the rapid growth of the Internet, more and more intelligent devices and systems have been embedded into it for service, security, and entertainment. Some of the important services provided using WWW include distributed computer systems, surveillance cameras, telescopes, manipulators, remote manufacturing, and mobile robots. Streaming audio and video on WWW has completely revolutionized the world media. It is now possible to participate in events from a remote place and interact with

people at distant places in near real-time. Inspite of advances in Internet technology, it still remains an under-exploited resource for distributed control engineering.

The Internet is decentralized in design and each computer in the Internet, called the host, is independent. These individual computers can be organized to communicate with one another over a local shared medium called local area network (LAN). LANs are networks usually confined to a geographic area, such as a single building or a college campus. LANs can be small, linking as few as three computers, but often link hundreds of computers used by the employees of an organization or and industry. Wide area networking (WAN) combines multiple LANs that are geographically separate. These networks can be efficiently used to implement networked control systems in automation industry.

Standards that allow computers to communicate on the network are known as network protocols. A protocol defines how computers identify one another on a network, and the format of data they should use for data transfer. This protocol also defines how this information is processed once it reaches its final destination. The procedures for handling the lost or damaged data units called packets are also recognized by the network protocol. Some of the widely used network protocols are transmission control protocol / internet protocol (TCP/IP) for Unix, Windows, Linux, and other platforms, Internetwork Packet Exchange (IPX) for Novell Netware, Apple Talk for Macintosh computers, and Simple Mail Transfer Protocol (SMTP). As we will see later in this thesis, the choice of a particular protocol for communication in networked control systems plays an important role in their design.

Although the Internet provides a cheap and readily available communication channel for teleoperation, there are still many problems that need to be solved before successful real-world applications can be developed. These problems, including its restricted bandwidth and arbitrarily large transmission delay, influence the perfor-

mance of the Internet-based telerobotic systems. The reliability of the system should be guaranteed so that Internet users can access these systems 24 hours a day with minimum human interface.

C. Modes of control on the network

There are various modes of control via the network. The classification of these modes of control depends upon the communication architecture between the controlled system and the remote user. The various modes of control on the network can be roughly classified into teleoperation, supervisory control, and closed-loop control on the network. Following paragraphs cover their brief overview.

Teleoperation is an elementary domain of distributed control systems. In teleoperation, a human operator conducts a task using a remote manipulator to perform various tasks in hazardous environments like space, underwater, or an atomic power plant. The key characteristics of a teleoperated system are from those for general factory operations. First, the operator has to depend on the feedback provided by sensory feedback systems such as video cameras for performing subsequent actions. This results in the operator's poor perception of the environment, which in turn affects the operator's performance. Second, the time delay, limited signal bandwidth, signal noise, etc. hinder the smooth communication between the master and the slave. The performance of the operator is affected significantly by those factors. Third, there are uncertainties that the operator has to deal with. For example, if the environment is not accurately modeled, there is difference between the actual environment and the model and the user has to use his perception to deal with such differences. But researchers devised several algorithms to deal with the above mentioned issues. Figure 1 shows the schematic of a teleoperated system.

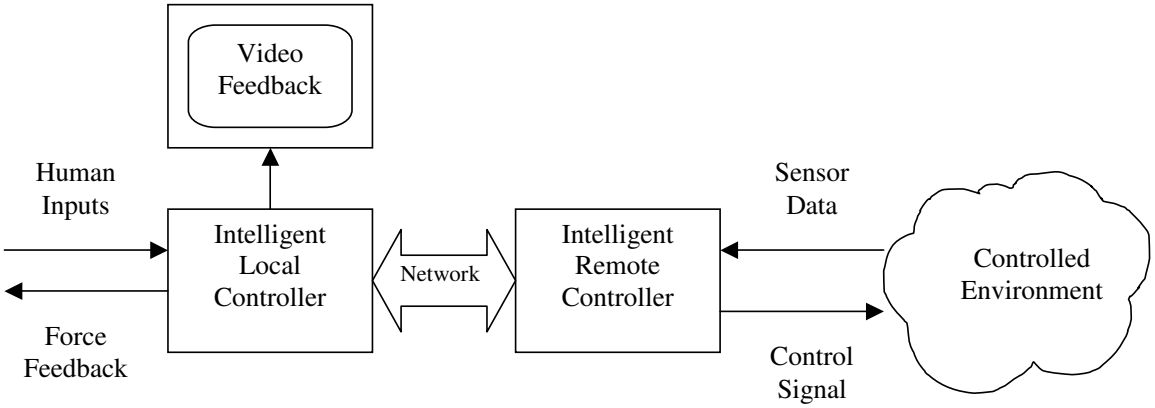

Fig. 1. Schematic of a teleoperated system.

In teleoperation, the force feedback strongly influences the system stability in the presence of time delays. Researchers have demonstrated that a trade-off exists between the transparency and stability of the teleoperated systems. To overcome the challenges posed by signal transmission delays, supervisory control proved to be an alternative. In supervisory control, an experiment can be monitored or supervised from a distance. Based on the client-server architecture, supervisory control uses user interface, instead of manipulator, to enable the user to interact with the controlled environment via the communication network. Srivastava [2] implemented supervisory control of the ball magnetic levitation setup via the Internet. Figure 2 shows the block diagram of this implementation.

In teleoperation and in supervisory control, access to the control system is provided to the user in the form of an interface. The actual controller is always implemented on the controlled environment side of the network. In the feedback control on distributed networks, the user side of the network always runs the control algorithm. The feedback loop of the control system is closed via the network using protocols to multiplex data from the sensor to controller and from the controller to the actua-

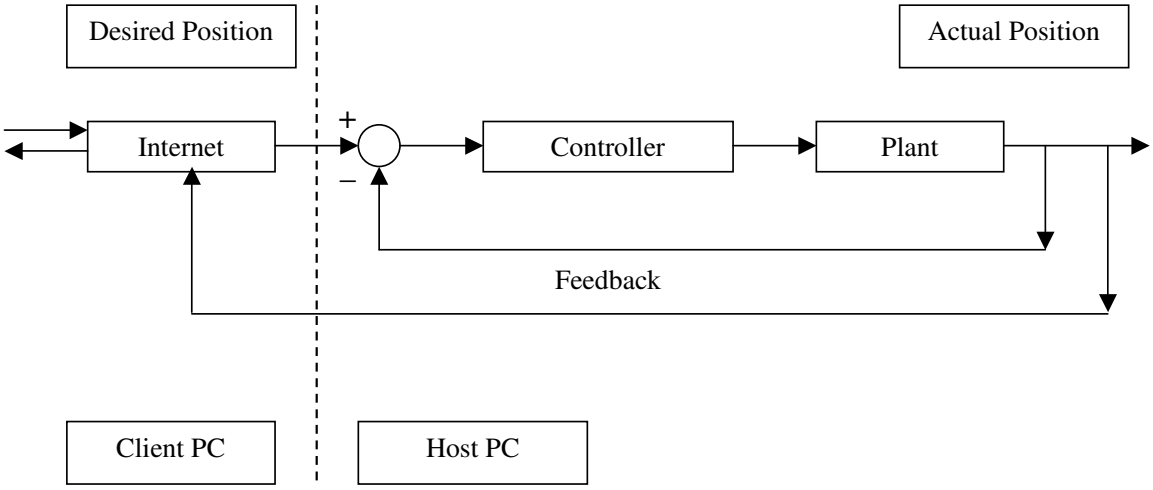

Fig. 2. Block diagram for supervisory control via the Internet [2].

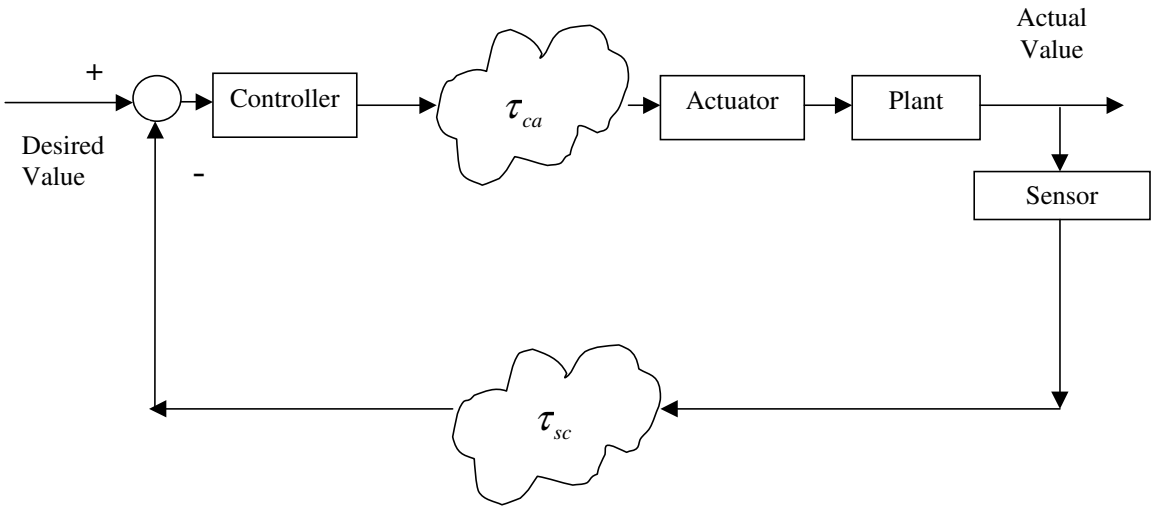

Fig. 3. Architecture of feedback control on distributed networks. T_{sc} is the network delay betweeen the sensor and controller and T_{ca} is the network delay between the controller and the actuator.

tor.The client-server based architecture of feedback control on distributed networks is demonstrated in Figure 3.

The feedback control loop shares the communication medium with other data transfer processes. Due to this sharing, time-varying delays are induced in the control loop. Current research focuses on stabilizing the feedback control of systems on distributed networks in the presence of these time delays. Figure 4 shows a block diagram of an example distributed control system of this type communicating over a local area network.

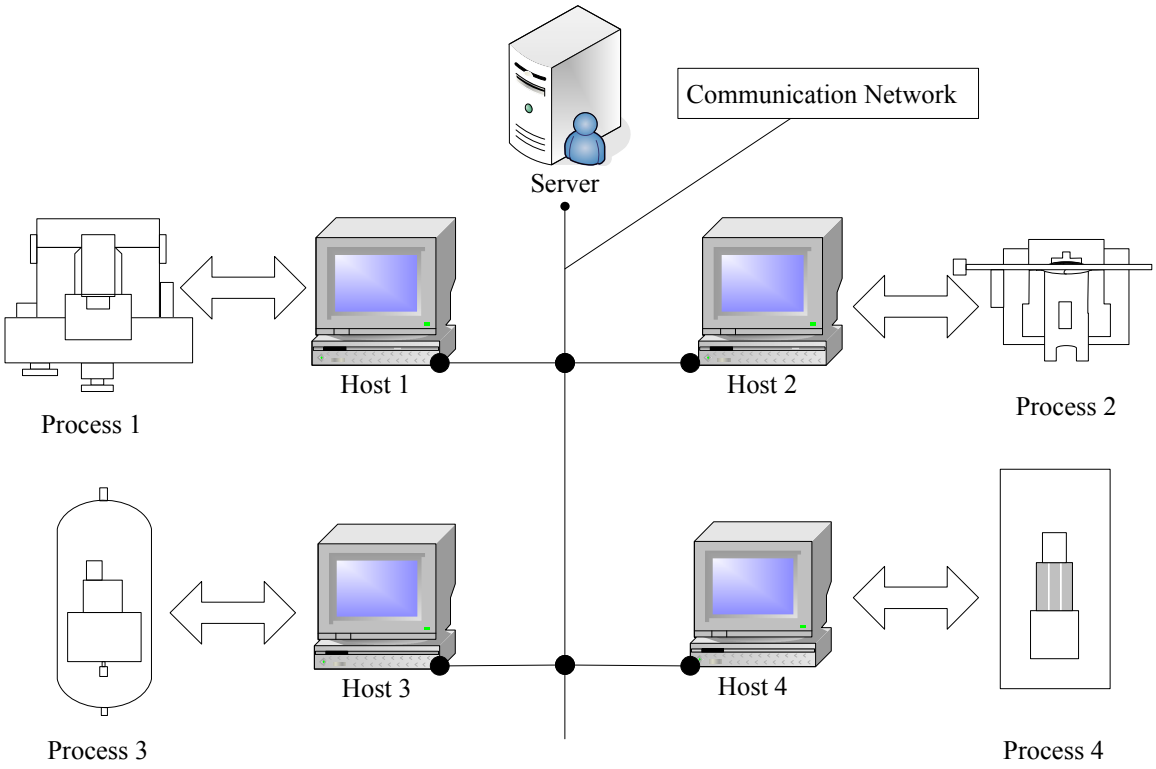

Fig. 4. Block diagram of an example distributed control system.

D. Objectives of thesis

A study of closed-loop real-time control on distributed networks is the main focus of the current research. Various key issues involved in the implementation of such systems have been identified and addressed in this thesis. Following are the main objectives of the thesis.

1. Establishment of closed-loop real-time control on computer network. With the proposed network based architecture, the components of feedback control system like actuator, controller, and sensor gain flexibility of spatial separation.

2. Determination of real-time requirements of the distributed control system and choice of an appropriate operating environment. An adequate control over the timing of various events in the time-sensitive networked control system can be achieved with the help of a real-time operating environment.

3. Development of an algorithm that can stabilize a control system in the event of sporadic network delays or packet losses in the network. Control signal based on predicted output data will be used in the event of data packet loss. With the proposed algorithm, several consecutive delays or data losses can be compensated for.

E. Contributions of the thesis

The contribution of this thesis is the implementation of closed-loop control on a LAN in a real-time operating environment and the design of a novel predictor based algorithm to stabilize this system in the event of consecutive network delays and packet drops.

Specifically, it includes (1) selection of a real-time operating environment, (2)

design and implementation of the client-server communication architecture, and (3) design of the predictor based algorithm to ensure system stability in the event of consecutive network delays. The development of the network communication architecture served as groundwork for the verification of underlying theory and fundamental working principle of the predictor based control algorithm.

F. Organization of thesis

Chapter I provides a brief introduction of NCSs and various modes of control using them. Advantages of and issues related to the NCSs are a part of the discussion. This chapter also describes the objectives and contributions of this thesis.

Chapter II throws light on the previous work done by researchers in the area of NCSs. It also gives a brief study of the existing experimental setup used as a test bed.

Chapter III describes the operating environment used to develop the software architecture of closed-loop real-time control on distributed networks. A justification for the selection of a particular operating environment is also given. At the end of this chapter the hardware interface is introduced.

Chapter IV explains the architecture of a NCS used for this research. The client-server design of the system, its communication model, and implementation details are presented. The mechanism of estimation and stability of NCS in the presence of time delays is discussed in detail.

Chapter V summarizes and concludes this thesis. The conclusions are based on the results presented in Chapter IV. The future work of the research in the area of NCSs is discussed at the end of this chapter.

CHAPTER II

BACKGROUND AND TEST-BED SETUP

A. Introduction

A substantial amount of work has been done in the area of teleoperation. Researchers have successfully developed several applications based on supervisory control. The effect of time-varying delays on the performance of distributed systems has also been studied. Most of the work done on closed-loop real-time control on distributed networks has been done with the help of dedicated networks. This chapter focuses on discussing the previous work done in the area of NCS. Stephen C. Paschall, II [3] developed the single-actuator magnetic ball levitation system. This system was used as the test-bed to implement the distributed control system for research. A brief study of this test bed is presented in the later part of this chapter.

B. Related work

Researchers have been working in the area of NCSs for long time. With the developments in the Internet technology in the last decade, distributed systems gained more importance in industry automation. Stability of such control systems in presence of time-delays has been an important area of research.

Teleoperation was the first form of a NCS that became popular. Internet-based teleoperation was used in telerobotics, remote manufacturing, tele-surgery, and distance education. The focus of research then shifted to the area of supervisory control using networks. Supervisory control found applications in various fields ranging from rapid prototyping to robotics. Researchers have recently started to delve into the field of closed-loop control over the networks. Some of the related previous work is

discussed in the following material.

1. Teleoperation

Master-slave systems have been in use since the development of first master-slave system in 1960's [4], [5]. Yokokohji and Yoshikawa studied the design of master-slave systems for superior maneuverability of robot manipulator [6]. A one degree-of-freedom (DOF) system including operator and object dynamics was analyzed. A quantitative index of maneuverability for master-slave system was derived for the system. A control scheme was also proposed to provide the ideal kinesthetic coupling. This scheme allowed the operator to maneuver the system as though he/she was directly manipulating the remote object himself/herself.

Industries find it difficult to train people to work with expensive, complex or potentially hazardous equipment, which is need for carrying out profitable tasks. Safaric et al. developed a teleoperation-based method of education and training that involved use of Virtual Environments for task planning [7]. Instead of allowing the trainees to interact with the resources directly, this method requires them to configure the experiments using a simulated representation of real-world apparatus. This configuration data can then be downloaded to the real work-cells. A verification of this data is done to safeguard the equipment from damage caused by the mistakes of the trainees during off-line programming.

Hu et al. discussed the use of co-operative internet robots having interactive human-machine interface [8]. In this scheme, using a web browser a remote operator can control the mobile robot to navigate a laboratory with visual feedback and a simulated map. Users can implement useful tasks in the laboratory using the highly intuitive user interface sitting at a remote location. With the use of cooperative learning control, better co-ordination in several robots and fully autonomous operation was

achieved.

Teleoperation is also used in the field of surgery. The Black Falcon developed by Madhani et al. implemented a teleoperated surgical instrument for minimally invasive surgery (MIS) [9]. MIS is a practice of performing surgery through small incisions using specialized surgical instruments. To feel the instrument-tissue interactions, the surgeon is given a force feedback. To demonstrate the usefulness of this eight degree-of-freedom and cable driven teleoperator, suturing along arbitrarily oriented suture lines in animal tissue was carried out. Mitsuishi et al. developed a master-slave type tele-micro-surgical system with an intelligent user interface [10]. An anti-shadow technique was used to present three dimensional (3D) relative information on a two dimensional (2D) display. Suturing of 1mm diameter artificial micro-blood-vessel was demonstrated.

The manufacturing industry gave ample opportunities to researchers to use teleoperation. Cybercut, developed at University of California at Berkeley provided a mechanical design and manufacturing service on the Internet [11]. It consisted of three components viz a design-for-manufacture computer aided design (CAD) interface written in Java, a choice between two computer-aided process planning(CAPP) systems and access to an open architecture machine tool for fabrication of mechanical parts. Rapid prototyping with a 3-axis computer numerical control was thus provided via the Internet.

Ho and Zhang argued the use of Java as an ideal implementation vehicle for internet-based tele-manipulation [12]. A teleoperation system was built to control a three-fingered hand. The use of Java made the implementation of system platform independent and object-oriented. The proposed system is shown in Figure 5, enabled users to be diverse.

As the system is Java-based, users have the liberty to use any operating system

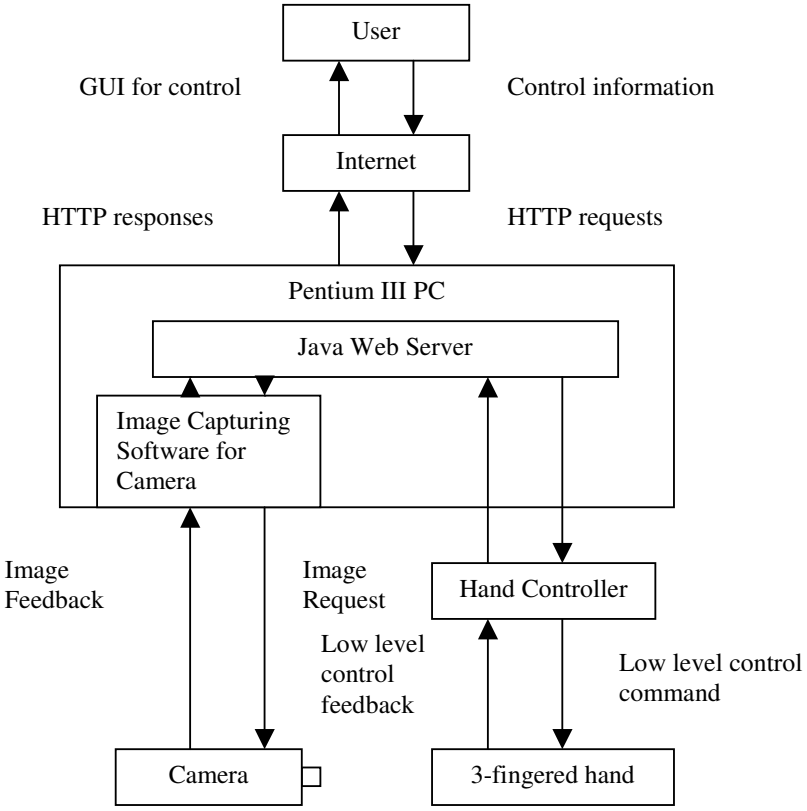

Fig. 5. System layout of a typical Internet-based teleoperation system [12].

(OS) on the computers at their end. The host machine can use any OS. The components within the system can be easily interchanged because the system is object-oriented and modular. One can use the single system to control different robots by rebuilding individual components that are specific to a robot. A virtual reality modelling language was used to construct graphical model to be displayed on the user interface.

Goldberg et al. developed a system that allowed a distributed group of users to simultaneously teleoperate an industrial robot arm via the Internet [13]. The mouse-motions of up to 30 remote users were used to produce a single control stream for

the robot. A digital camera mounted above the robot arm gave the users a visual feedback. Figure 6 describes the architecture for this system.

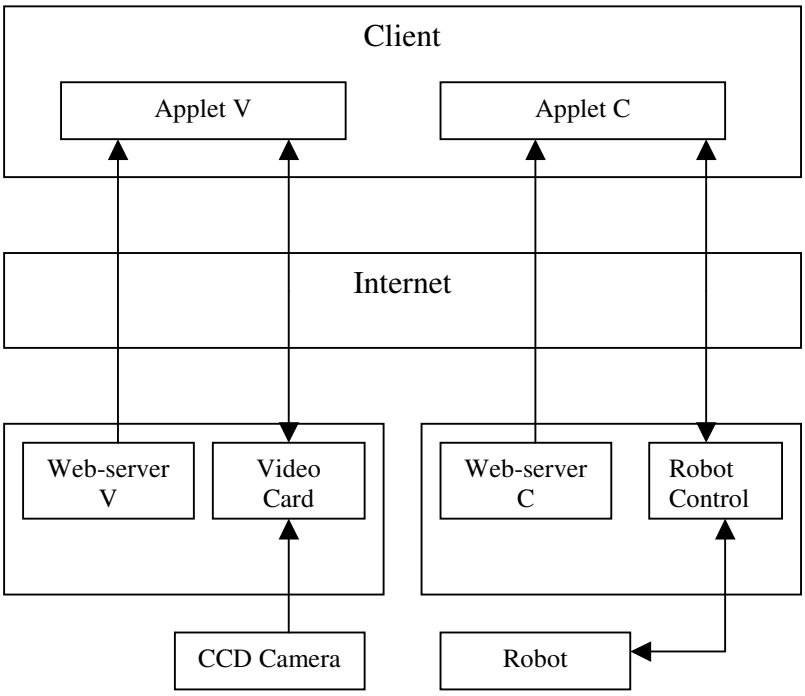

Fig. 6. System architecture: each client runs two applets, Applet V for video feedback and Applet C for control [13].

Ouija board game was selected as an application of collaborative control. An active email-address, user-id, and password are required for the new user to register. User's client-side gets two applets, each communicating with a unique server. Live streaming video was provided using applet and server V and the coordinate control of the robot was provided using applet and server C. This was the first collaboratively controlled robot on the Internet.

2. Supervisory control

In teleoperation, the operator controls a remote manipulator with the help of feedback he gets in the form of video or force feedback. The operator depends highly on the sensory feedback systems and his poor perception of the environment can result in the poor performance. So, researchers have been focusing their attention to supervisory control. With supervisory control, user can give symbolic or analogic instructions remotely to a computer attached to the manipulator instead of remotely guiding the telemanipulator.

Park and Sheridan described a system developed for supervisory control of a telemanipulator graphically simulated on an IRIS workstation [14]. The system had two modes of operation: manual mode and supervisory mode. In the manual mode, the operator guided the ball using six-dimension-sensor ball and in the supervisory mode, the operator used command menu and mouse to give symbolic instructions. In the supervisory mode, the operator can specify intermediate locations that the manipulator tip is to pass through. Operator can also select intermediate points that the hand tip should go through in the event of collision detection. Heuristics were used to avoid collision.

Luo et al. used the supervisory control technology to develop the desktop rapid-prototyping system [15]. The remote user can send a CAD model in the form of a standard template library (STL) file to a telecontrol server. This server transforms the model into a rapid prototyping liquid crystal diode (LCD) photo-mask display. User can monitor the part getting manufactured using Internet. The Figure 7 shows the overall system architecture.

The online visual inspections system allows user to control the part quality. A pattern-matching algorithm monitors the part building. The server informs the user

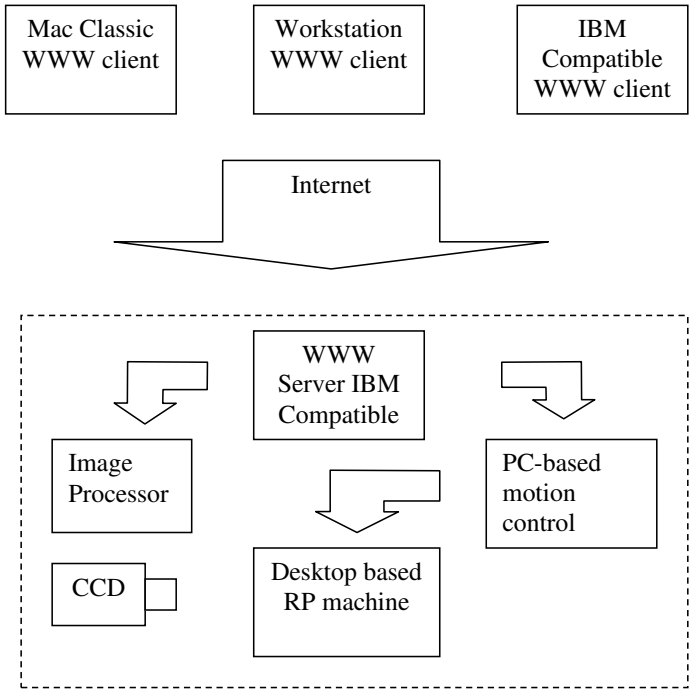

Fig. 7. Overall system architecture of automated telemanufacturing system [15].

if the server-grabbed image does not match the photo-mask.

Garcia et al. developed a telerobotic system using supervisory control based on a hybrid control approach [16]. To control the position and force interaction of a remote manipulator with its environment, a control structure was developed. Using hybrid systems theory, a supervisory control was designed to detect when force and position thresholds are overcome. In the occurrence of such events, the controller modified the references sent from the remote operator to improve the performance of the system and operate it safely.

Srivastava and Kim discussed the supervisory control via the Internet of a magnetic ball levitation system [17]. The setup was based on the client-server architecture with interface programmed with hyper text markup language (HTML) and common

gateway interface (CGI). Figure 8 explains the setup.

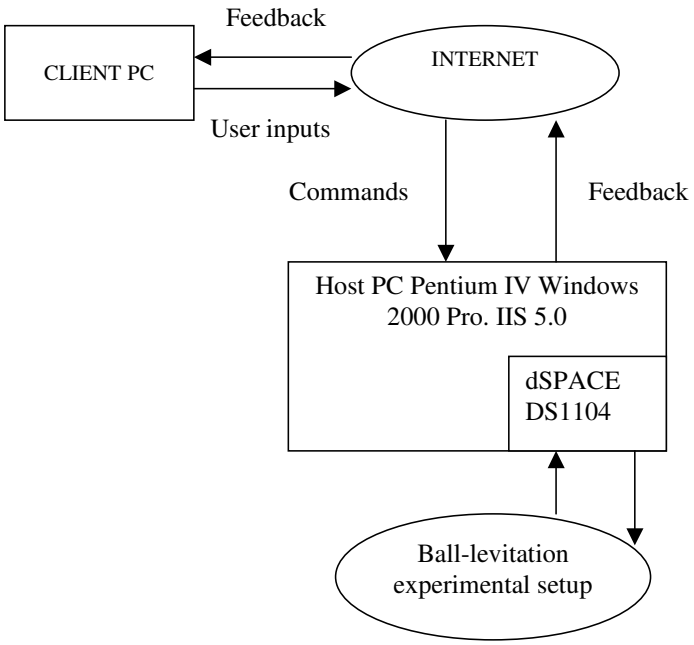

Fig. 8. Hardware architecture for the supervisory control of the maglev system over the Internet [17].

The client was able to change the position of the ball and change the control parameters remotely using the HTML interface. Stochastic nature of the Internet communication delays was also studied in reference to the various probabilistic models for the time delays.

3. Closed-loop control over network

Recently the researchers have started to delve into the closing the control loop over the network. In distributed control systems, the individual components like sensors, actuators, and controllers are specially displaced. The main advantage of this setup is flexibility. This allows placing the controller on the network, separated from the controlled process. But this setup also faces the potential problems introduced due

to network time delays.

Eker and Cervin developed distributed wireless control using the Bluetooth wireless technology [18]. A rotating inverted pendulum was used as a test bed for this research. Figure 9 describes the setup used for this research.

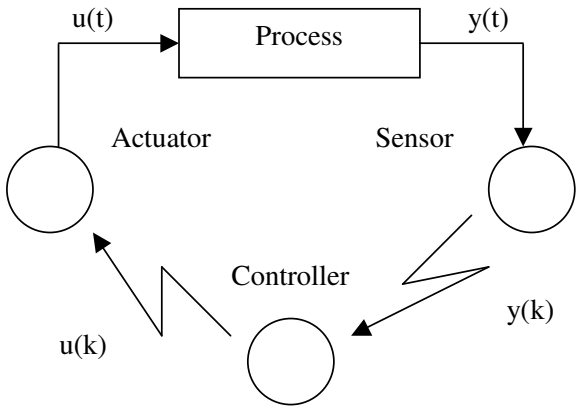

Fig. 9. The distributed system configuration used by Eker and Cervin [18]. The communication between the sensor, the actuator, and the controller is over a wireless link.

Setup consists of three communicating nodes, namely sensor node, actuator node, and controller node. The sensor node has the responsibility of sampling the data and sending it across the network so it is time driven. While the actuator and controller nodes receive and send the data and are event driven. In the case of constant time delay, the following control law was proposed:

$$u(k) = -L \begin{pmatrix} x(k) \\ u(k-1) \end{pmatrix} \tag{2.1}$$

where L is a constant feedback gain vector, $x(k)$ is the state vector and $u(k-1)$ is the old control signal.

In the case of varying delays, non-optimal solution was used and static delay

compensation was proposed for the mean delay. It was given by:

$$u(k) = -L(\tau_k) \begin{pmatrix} x(k) \\ u(k-1) \end{pmatrix} \tag{2.2}$$

Where the feedback gain vector L is now a function of the sensor-to-controller delay T_{sck} in the current sample. Simulation and experimental results showing the stability zone were demonstrated.

Ploplys and Alleyne achieved distributed control using UDP on stand-alone wireless network [19]. In this research, a Furuta pendulum was used as a test bed. UDP communication in a peer-to-peer network control loop was used and a communication range better than Bluetooth was achieved. The setup involved a simple timing scheme in which clock synchronization was not necessary. A delayed packet was considered to be lost and no compensation was used. The problem of data loss was addressed by looking at sampling rate variation to improve performance and reliability.

4. Time delays in networked control systems

The behavior of the communication medium plays vital a role in the working of NCSs. Various studies have been made in the past by researchers to study this behavior. Time-delay characteristic of the network is the most critical behavioral aspect of such control systems. The behavior of time delays is sporadic as observed by many researchers. Due to these sporadic time delays, stability of the system is adversely affected. Various studies have been made to calculate, estimate or compensate for such time delays to stabilize the NCSs.

Farrell [20] was the first to work on the problem of time delays in teleoperation. Later, Anderson and Spong studied the bilateral control of the teleoperated system with time delays [21]. The focus of this research was on maintaining the stability

in force-reflecting bilateral teleoperators in the presence of substantial time delays. Using the passivity and scattering theory, a criterion was developed to show the usefulness of the bilateral control law. Conway et al. [22] introduced new concepts called time and position clutches to deal with the time delays due to telemetry and signal propagation in teleoperations.

The next step to study the network traffic was studying the suitable models to describe their behavior. Frost and Melamed [23] provided an overview of computer simulation modeling for communication networks. Several models like Markov-renewal traffic models, Markov-modulated traffic models, fluid traffic models, and autoregressive traffic models were discussed. Various traffic streams, corresponding to different services like voice, video, file transfer, etc. could be superimposed to form a realistic heterogeneous mixture of traffic.

Lian et al. discussed the design considerations for NCSs [1]. Theoretical analysis included the study of network parameters, control parameters, and networked system performance. The study also identified the key components of the time delay given by,

$$T_{delay} = T_{pre} + T_{wait} + T_{tx} + T_{post} \tag{2.3}$$

where, T_{delay} is the total one way time delay, T_{pre} is preprocessing time at source nodes, T_{wait} is waiting time at source nodes, T_{tx} is the transmission time on the network channel, and T_{post} is the post-processing time at the destination node. It was concluded that the minimization of device processing time and improvement of network protocols to further guarantee the determinism of transmission time will be helpful to reduce end-to-end delays. The designer can then make improvements by choosing an appropriate sampling period and robust controller design.

The controllers, sensors, and actuators can be designed as either time-driven or

event-driven. In case of time-driven components, time synchronization has to be done so as the events occur at tightly controlled time instances. Zhang et al. studied this phenomenon and proposed several algorithms to estimate and remove the relative clock skews from delay measurements based on the computation of convex hull [24].

Before implementing a NCS, a decision has to be made about action to be taken in the event of excessive time delays. The easiest way to deal with such kind of situation is to discard the data as lost and take no action. Another approach is to predict the lost data as discussed by Beldiman et al [25]. Simulation results were used to prove that by adding predictors to estimate the plant output in between successive transmissions, the performance of a network control system is improved.

C. Test bed and its control

The single-actuator ball magnetic levitation system, developed by Stephen C. Paschall, II as his senior honors project [3] was used as the test bed for the research. In this system, a small steel ball is held in stable levitation at some steady state operating position. An electromagnet is used to produce forces to support the ball's weight. These forces are produced by controlling the currents in the coil of the electromagnet using a control system implemented on a personal computer (PC).

1. Magnetic levitation system

The main physical components of the magnetic levitation system like electromagnetic coil, optical sensor, levitated steel ball, and sensor light source are shown in the Figure 10. The system can be separated into two main sub-systems. They are the force actuation sub-system and the position sensing sub-system. The force actuation sub-system consists of an electromagnetic coil, a pulse-width modulator, and a 24-V DC

power supply whereas the position sensor sub-system consists of a cadmium sulfide (CdS) photocell, an incandescent light source, and a 15-V DC power supply.

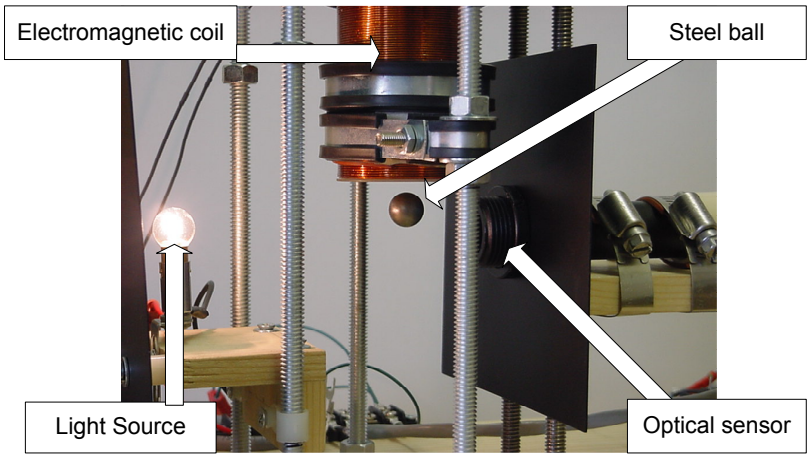

Fig. 10. Single-actuator magnetic ball levitation system [3].

The electromagnetic coil plays the role of actuator by producing magnetic field when an electric current is passed through its wires. This current is adjusted by the controller implemented in the form of software in the PC. The output of the controller depends on the position of the ball as sensed by the photocell used in conjunction with an incandescent light bulb. Gradual shielding and exposing of the photocell to light is used to measure the vertical movement of the ball. To ensure that the ambient light does not interfere with the light from the bulb, the tube is completely closed to external light except for the vertical slit located along the tube end near the ball. With this arrangement, the amount of light exposed to the photocell is the function of the position of the ball. The change in the light exposure to the photocell causes change in its resistance. This change is measured as voltage across a resistance placed in series of the photocell. This change in voltage is used to compute the control.

2. Modeling

A useful plant model is obtained with the levitated object position as a function of the input current. Figure 11 shows the schematic diagram of the magnetic levitation system and Figure 12 shows the free body diagram of the steel ball. The magnetic levitation theory nomenclature is shown in the Table I.

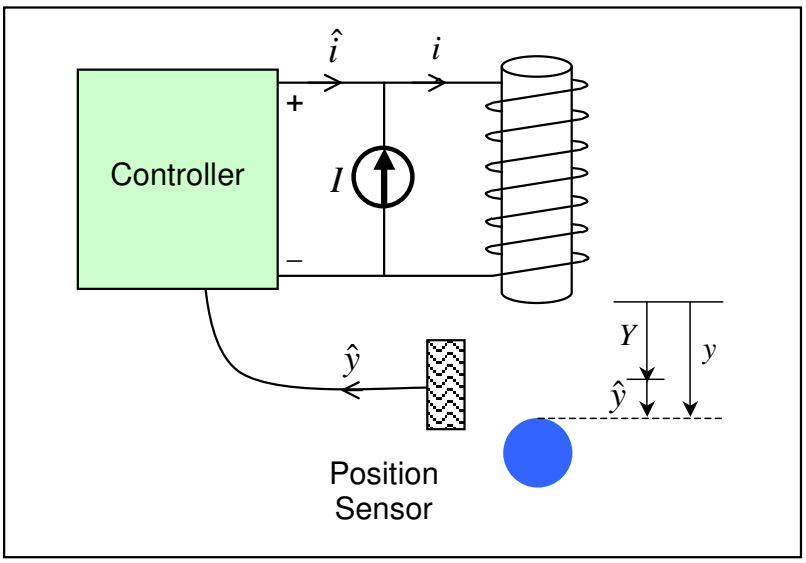

Fig. 11. Basic setup for the magnetic levitation system [3].

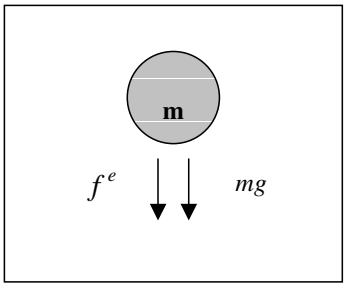

Fig. 12. External forces on the levitated ball [3].

Table I. Nomenclature for the magnetic levitation theory.

i	Electrical current through the electromagnet [A]
$\hat{i}$	Perturbation current [A]
I	Bias current at equilibrium [A]
y	Vertical displacement of object from electromagnet [m]
$\hat{y}$	Perturbation displacement [m]
Y	Steady-state displacement at equilibrium [m]
L	Electromagnet inductance [H]
L_0	Electromagnet inductance constant [H]
L_1	Electromagnet inductance constant [H]
a	Geometric Constant [m]
f^e	Electromagnet force [N]

The force produced by the electromagnet is given by the derivative of co-energy as

$$W'(i,y) = \frac{1}{2} i^2 L(y) \tag{2.4}$$

where, the inductance $L(y)$ is assumed to be of the form [26]

$$L(y) = L_1 + \frac{L_0}{1 + \frac{y}{a}} \tag{2.5}$$

The equation of motion is given by

$$m\ddot{y} = mg + f^e \tag{2.6}$$

where,

$$f^e(i,y) = \frac{\partial W'}{\partial y} = \frac{-L_0 i^2}{2a(1 + \frac{y}{a})^2} \tag{2.7}$$

From 2.6 2.7, we get

$$m\ddot{y} = mg - \frac{-L_0 i^2}{2a(1+\frac{y}{a})^2} \tag{2.8}$$

At the static equilibrium, it follows that

$$mg = \frac{L_0 I^2}{2a(1+\frac{y}{a})^2} \tag{2.9}$$

Rearranging the above equation, we get current given by

$$I = (1+\frac{y}{a})\sqrt{\frac{2mga}{L_0}} \tag{2.10}$$

Equation (2.10) is a nonlinear equation. Using Taylor's series expansion and simplifying it, transfer function of the system is obtained. It is given by

$$\frac{\hat{Y}(s)}{\hat{I}(s)} = \frac{-1}{As^2 - B}[\frac{m}{A}] \tag{2.11}$$

where,

$$A = \frac{I}{2g}[\frac{As^2}{m}] \tag{2.12}$$

$$B = \frac{I}{a+Y}[\frac{A}{m}] \tag{2.13}$$

3. Dynamics and control

The open-loop magnetic levitation system consists of the power amplifier, electromagnetic coil, and the steel ball. It was found that open-loop system is unstable. So, it is made stable by us placing the system in the closed-loop control. The position information provided by the photocell sensor is used as feedback. Figure 13 shows the feedback loop of the system.

The controller designed in continuous time domain is given by

$$\frac{-33300s^2 - 2.564 \times 10^6 s - 1.632 \times 10^7}{s^2 + 700.7s + 490} \tag{2.14}$$

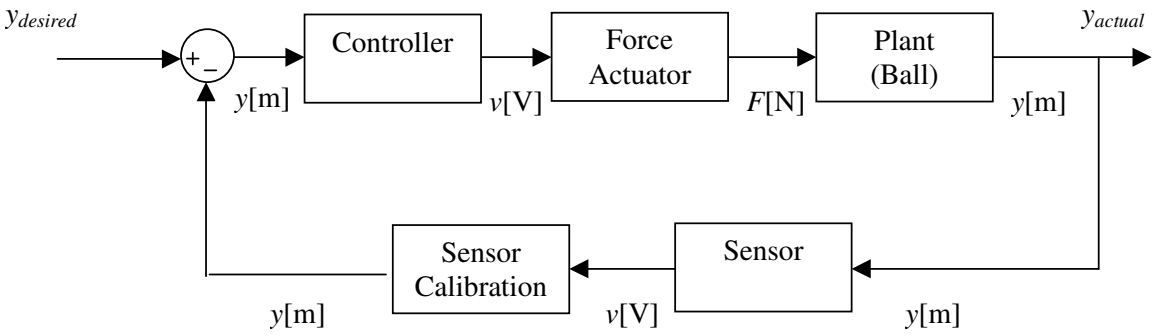

Fig. 13. Schematic of magnetic levitation system.

The digital controller designed by Abhinav Srivastava for 333 Hz sampling frequency is given by [2]

$$D(z) = 4.15 \times 10^4 \frac{z^2 - 1.754z + 0.769}{z^2 - 0.782z - 0.13}. \tag{2.15}$$

Stephen C. Paschall, II, implemented the digital control algorithm on the DS1104 digital signal processor (DSP) controller board. Windows 2000 was used as the OS. But for the current research, Windows 2000 was found to be inefficient and Linux with Real-Time Application Interface (RTAI) was used instead. The C programs running on the Linux workstation implemented the control. PCI-6025E card from National Instruments was used for data acquisition. One of the 12-bit analog-to-digital (A/D) converters was used to read the optical sensor output. The control signal was output using one of the 12-bit digital-to-analog (D/A) converters.

CHAPTER III

COMPUTING ENVIRONMENT

A. Introduction

In the last chapter, a brief literature review of the NCSs was presented, and the single-actuator ball magnetic levitation system was described. In this chapter, the operating environment for the software architecture will be discussed. First, the need of real-time OSs in NCSs will be investigated. Then, a brief introduction of the RTAI will be given. In the last part of this chapter, Linux control, and measurement device interface (Comedi) will be discussed. Comedi was used to develop an interface between the host PC and a National Instruments' PCI-6025 data acquisition card.

B. Need of real-time operating system

In the past, researchers used field-bus in the area of distributed control and automation [27]. It was easy to install and manipulate, had low latency, and used cheap and simple cabling. But the most important requirement was of real-time operation. Being a stand-alone network, the field bus performed efficiently in this domain.

Today several manufacturers demand Ethernet and Internet based solutions. Ethernet is widely used in the Information Technology (IT) industry and it allows integration to the world-wide internet networks. Various standard communication protocols like TCP/IP are developed and have become standard. A need has arisen for their use in the network based control systems and replace the stand alone networks. But, the use of Ethernet poses several challenges, dealing with network latencies being the most important.

To understand the concept of latency, let us consider a simple client-server com-

munication on Ethernet as shown in Figure 14. The client requests some information from the server and the server answers the request. The maximum delay in this communication - the latency - is a possible real-time constraint on the communication system. Table II explains the nomenclature.

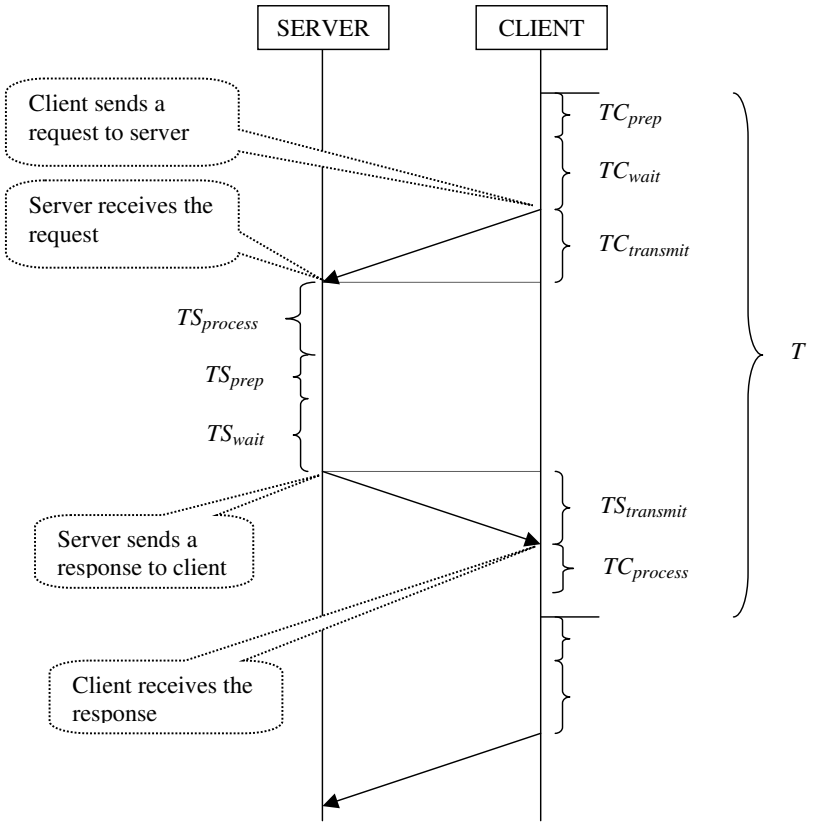

Fig. 14. Components of latency in a periodic client-server communication process.

As shown in the Figure 14, a typical client-server communication in NCS is periodic with a period T. The total communication is required to get completed in one period. The total communication time or latency is given by,

$$T_{total} = TC_{prep}+TC_{wait}+TC_{transmit}+TS_{process}+TS_{prep}+TS_{wait}+TS_{transmit}+TC_{process} \tag{3.1}$$

Table II. Nomenclature for time delay components.

TC_{prep}	Time taken by the client to prepare the request message
TC_{wait}	Time spent by the client waiting for network access
$TC_{transmit}$	Transmission time from the client to the server
$TS_{process}$	Time taken by the server to process the request
TS_{prep}	Time taken by the server to prepare the reply message
TS_{wait}	Time spent by the server waiting for network access
$TS_{transmit}$	Transmission time from the server to the client
$TC_{process}$	Time taken by the client to process the reply
T	Total period of the process on the client side

For all practical purposes, it can be assumed that

$$\begin{aligned} TC_{prep} &= TS_{prep} \\ TC_{wait} &= TS_{wait}. \end{aligned}$$

It can be also observed that the time TC_{prep} is deterministic where as TC_{wait} and $TC_{trasnmit}$ are indeterministic.

We can apply the client-server architecture to a closed-loop NCS with the architecture shown in Figure 15. The client side of the architecture hosts the test bed. Thus, in order to be consistent with the literature, the client side will be referred to as the host side in this thesis. The server side implements the controller. The request message send by the host to the server can carry the sensor data and the response message send by the server to the host can carry the control data. The processing of the message on the server side is the calculation of control data using the sensor data. The sampling is done at a certain fixed frequency, e.g 333.333 Hz for

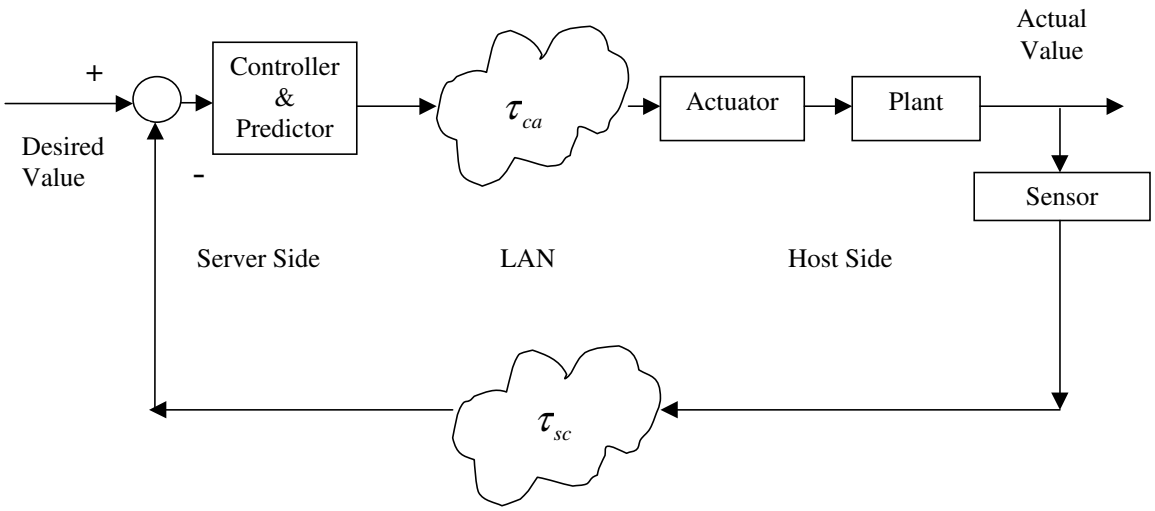

Fig. 15. Block diagram of a typical closed-loop NCS.

experiments done on single actuator magnetic levitation system discussed in Chapter II. After every 3 ms, a fresh sample of the sensor data is taken. This sensor data is then transferred to the server to calculate the control in the form of a message. The creation of a message to be sent to the server is dependant on this sampling. In other words, creation of message to be sent to the server is a time triggered process. So, this process of sampling the sensor data is a real-time process. It is the responsibility of the OS environment to ensure that a sample is taken every 3 millisecond.

On the other side, the server waits for the request message from the host. As soon as it receives a message from the host, it processes the request and creates a reply; in current research, the control data. The creation of the message to be sent to the host is dependent on the arrival of the request. In other words, it is an event triggered process. Thus, this process does not have strict real-time constraints. The server then sends back the reply message to the host. The host receives this reply and implements the control. At the start of the next period, another sample is taken

and the same type of communication takes place.

For this kind of closed-loop networked system to remain stable, the events have certain deadlines. If these deadlines are missed, the stability of the control system is affected. In the current research, the single-actuator ball magnetic levitation system used is open-loop unstable. It has strict time requirements. It was demonstrated by Srivastava [17] that for 333.333 Hz, the feedback loop should be completed in 1.4 ms. If this deadline is missed, the system becomes unstable. In order to ensure that the time constrained events happen at correct times, a real-time operating environment is needed.

A real-time system can be defined as a system that responds to externally generated stimuli within finite and specified period of time [28]. An efficient real-time system produces correct results at proper time. The real-time systems can be classified into hard real-time systems and soft real-time systems. The systems in which meeting all the time requirements is mandatory is known as a hard real-time system. In a soft real-time system, it is required that almost all the time requirements be met. But, the soft real-time system functions properly if a few deadlines are occasionally missed.

The real-time computing can be roughly classified into three types — clock-based computing, sensor-based computing, and interactive computing [29]. In clock-based real-time computing, the computer is able to maintain an accurate measure of the physical time. This clock is used to time the events. In sensor-based real-time computing, the computer utilizes an interface to measure the conditions or state of a physical process. Usually interrupts are used to indicate the occurrences of such events. In interactive computing, the computer is connected to external devices, process or other computers. If the computer communicates actively with the external units, the response of the on-line computer may need to be within certain time constraints.

C. Selection of operating environment

After the nature of the closed-loop NCS was studied, it was concluded that an appropriate operating system is required to successfully implement the distributed architecture. Following factors affected the selection of the OS.

- Periodic tasks: The OS should allow execution of periodic tasks.
- Time resolution: Time resolution can be defined as the smallest amount of time that can be measured accurately with the help of program. The length of a closed loop was expected to be not more than 1.4 milliseconds. So, it was expected that the time resolution be 14 microseconds which is 1% of the loop closing time.
- Threads: To implement the algorithms for closed-loop real-time control over the network, there was a need for multi-threaded programming. So, the OS was required to implement threads.

Commercially available OSs like Windows 2000, different flavors of Unix and Linux were some of the available choices. But it was found that these are not real-time OSs. Their performance with reference to above mentioned factors was not very satisfactory. Two simple timing tests, as stated by Volz [28] were performed to observe their performance.

The smallest amount of time that can be precisely measured on an OS is known as its clock resolution. It depends upon how the clock is implemented as hardware or software and the method used to access it. There are several low resolution and high resolution clock access commands in the above-mentioned OS. The time required to read a clock is typically much less than its resolution. For example, the time required to access the clock in Redhat Linux 7.3 was found to be approximately 0.01 μs where

as its clock resolution was found to be 10 μs. Both the tests were based on this principle.

Because the time for clock read is less than the resolution, many consecutive clock reads can be made before the value returned by the clock changes. This was used in the first test. The number of times the clock was accessed between the change in value returned by the clock was recorded in an array over several iterations. The values of this array were then plotted. Significant variations in these numbers indicated some OS activity. Figures 16 and 17 represent the results of the first test on Windows 2000 and Redhat Linux 7.3 OSs.

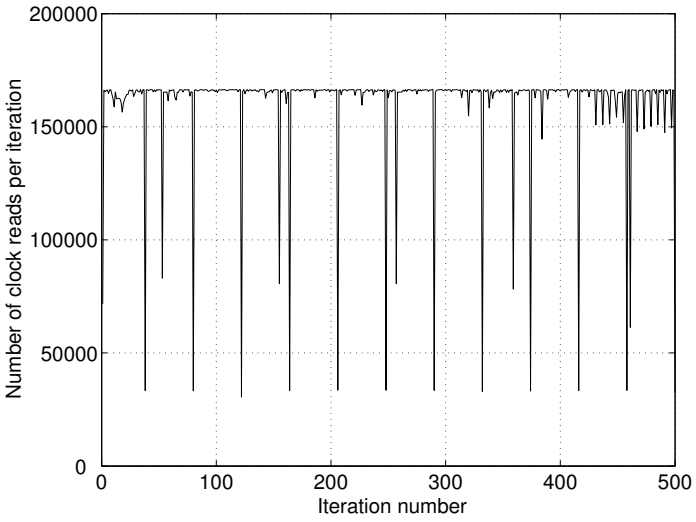

Fig. 16. Plot of number of clock reads per unit time for first timing test on Windows 2000.

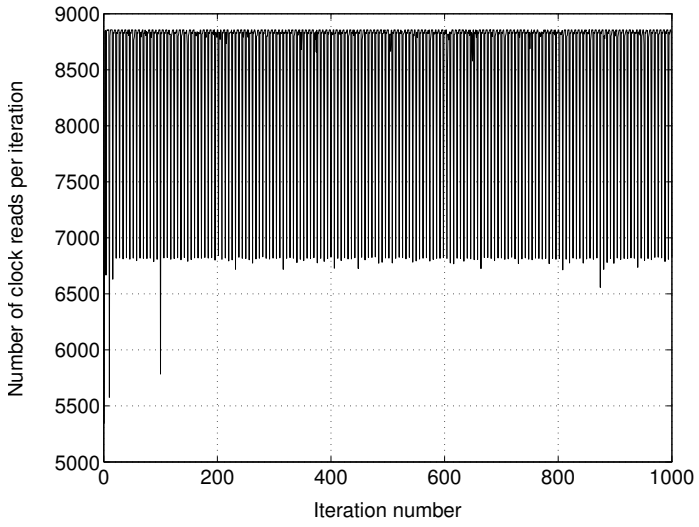

Fig. 17. Plot of number of clock reads per unit time for first timing test on Redhat Linux 7.3.

When the value returned by the clock changes, the difference between this value returned and the previous value returned, is the clock resolution. In the second test, the clock resolution was calculated over several iterations. These values of clock resolution were recorded in an array and plotted. It was found out that there was difference in the values returned as clock resolution. So, maximum value in microseconds returned is the clock resolution for the corresponding OS. Figures 18 and 19 represent the results of the second test on OSs Windows 2000 and Redhat Linux 7.3 respectively.

It can be seen from Figures 16 and 17 that between consecutive clock accesses, there is considerable amount of activity recorded. Although the clock resolution of Redhat Linux 7.3 was found to be uniform, the resolution of Windows 2000 was not. These simple tests demonstrated the non-realtime performance of the two OSs. So, a better OS environment was required. RTAI with Linux provided a competitive solution to the problem. The following section will give a brief introduction of RTAI.

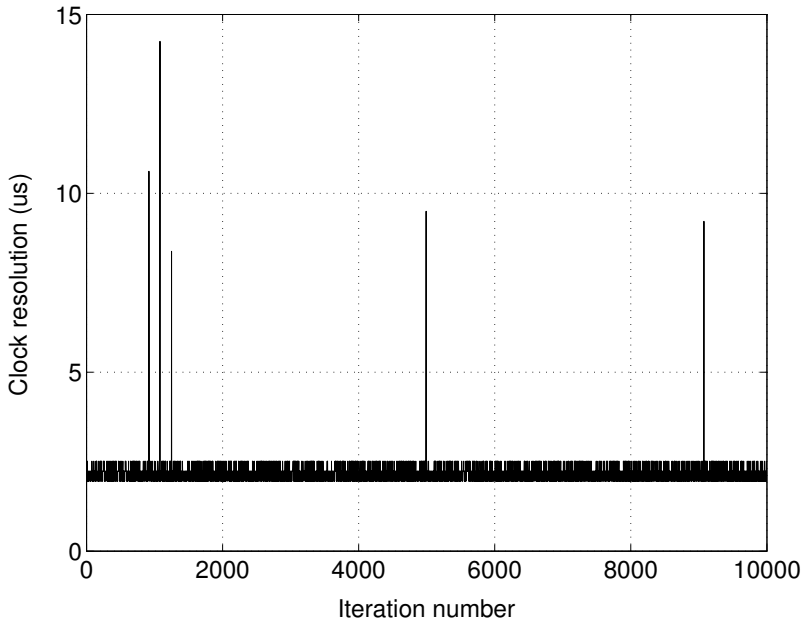

Fig. 18. Plot of clock resolutions obtained for second timing test on Windows 2000.

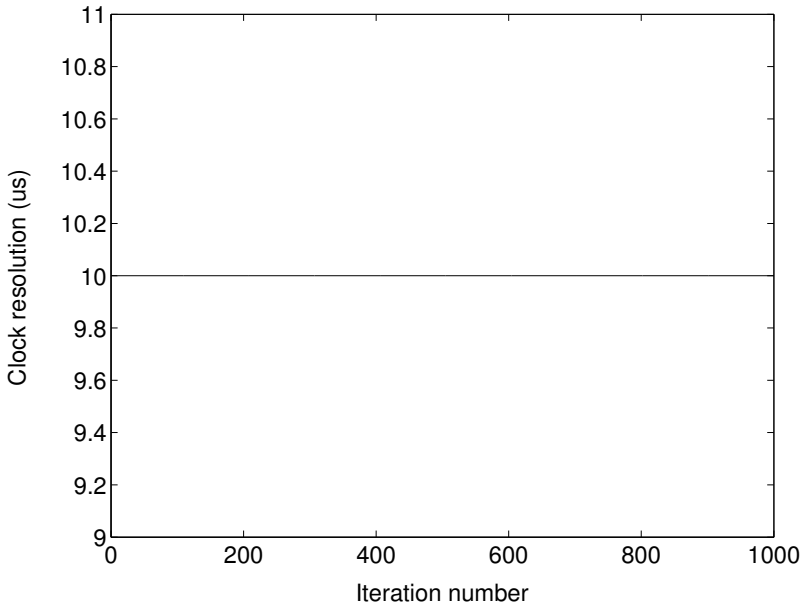

Fig. 19. Plot of clock resolutions obtained for second timing test on Redhat Linux 7.3.

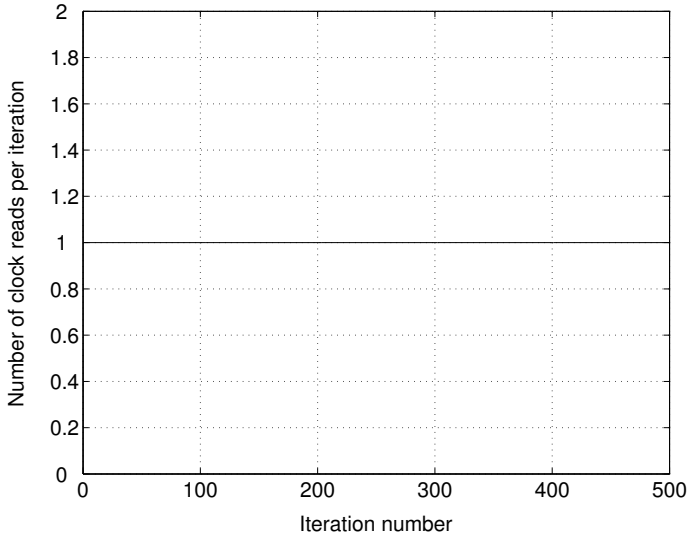

Fig. 20. Plot of number of clock reads per unit time for first timing test on RTAI 24.1.12 with Redhat Linux 7.3.

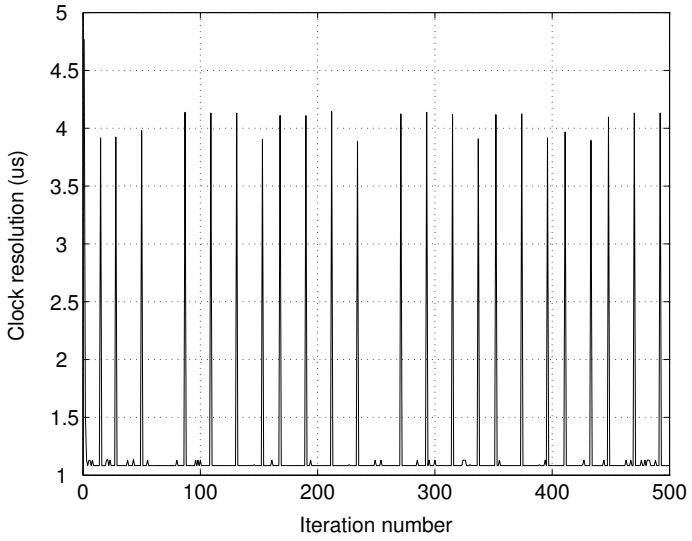

Fig. 21. Plot of clock resolutions obtained for second timing test on RTAI 24.1.12 with Redhat Linux 7.3.

Figures 20 and 21 show the results of first and second tests run on RTAI 24.1.12 with Redhat Linux 7.3.

D. Real-time application interface

The RTAI [30] was developed as a real-time operating environment solution at Dipartimento di Ingeneria Aerospaziale Politecnico di Milano (DIAPM). In strict terms, RTAI is not a real-time OS. Based on the Linux kernel, it provides the ability to make it fully preemptive. Linux OS provides several sophisticated services like hardware management, event polling, peripheral interrupts, scheduler classes dealing with process activation and priorities, interprocess communication, and implementation of network protocols like TCP/IP. But Linux alone as an OS lacks real-time support. It is necessary to make some changes in kernel behavior like interrupt handling and scheduling policies to make it a real-time platform with low latency and high predictability of timing performances. RTAI modifies Linux kernel to make it a real-time operating environment.

RTAI basically offers the same services as the Linux kernel core, adding the features of a real-time OS. The real-time scheduler treats the Linux OS kernel as an idle task. The real-time tasks are given higher priority than the non real-time tasks. Linux only executes when the real-time kernel is inactive. Linux cannot prevent itself from being preempted. The software emulation of interrupt control hardware make this possible.

The interrupt dispatcher of RTAI traps the peripheral interrupts and re-routes them to Linux when necessary. When an interrupt occurs, the real-time kernel intercepts the interrupt and takes a decision about what to dispatch. A proper real-time interrupt handler is invoked if available. In absence of a real-time interrupt handler,

the interrupt is marked pending. When Linux requests the enabling of interrupts, the pending interrupts are enabled and corresponding Linux interrupt handler is invoked. Thus, the real-time system is always able to respond to an interrupt.

There are several advantages of using RTAI. It relies on the Linux loadable module mechanism to install components of the real-time system. Linux can perform this non-real-time operation of loading and unloading modules. The task switch time for the real-time tasks is minimized because these real-time system modules use the kernel address space. If the scheduler is unsuitable for a particular application, the scheduler module can also be replaced by another module that meets the demands of the application. The real-time system is thus kept modular and extensible.

In addition to the above mentioned advantages, RTAI offers following computing features.

- POSIX 1003.1c compatible Pthreads including mutexs and condition variables.
- Traditional real-time inter process communications (IPCs) like semaphores, mailboxes, first-in-first-outs(FIFOs), shared memory, and remote procedure calls (RPCs).
- Dynamic memory allocation.
- Practical extraction and reporting language (PERL) bindings - which allow scheduling of soft real-time tasks from the PERL scripting language.
- (LXRT) - Allows the use of the RTAI system calls from within standard user space.
- UniProcessor, Multi-Processor and Symmetric Multi-Processor support.
- One-shot and periodic schedulers.

Compared to the commercially available real-time OSs, RTAI's performance is very competitive [30]. Table III summarized the typical performance of RTAI. Last but not the least, RTAI is open source. RTAI is free under the terms of the GNU (GNU is Not Unix) Lesser General Public License.

Table III. RTAI's typical performance.

Context switch time	4 μs
Interrupt response	20 μs
Periodic tasks	100 kHz
One-shot task rates	30 kHz

Linux Real-Time Extension (LXRT) Application Programming Interface (API) in RTAI was used to develop various programs to implement the distributed network control system. LXRT is a module that allows the symmetric use of all the services made available by RTAI and its schedulers in user space. Users can program both hard and soft real-time tasks with RTAI-LXRT. The most important advantage offered by LXRT is that the user can start developing a real-time application in the user space under the memory protection umbrella of standard Linux. After testing the application and getting the functionality working satisfactorily, it can be transitioned to a hard real-time task by compiling it as a kernel module. Another advantage in the distributed control perspective is that user can use TCP/IP to program network communication threads in LXRT.

LXRT requires the use of SCHED_FIFO scheduling policy with statically assigned process priorities to achieve soft real-time performance when executing in the user space. With simple additional function calls, this task can also be made to execute as hard real-time task. The interprocess communication interface is also made

available by LXRT. The only penalty associated with the use of LXRT is a slight increase in the context-switch times. Typical context-switch times in kernel modules of real-time tasks are under 40 μs while with LXRT, the context-switch times are under 100 μs.

E. Linux control and measurement device interface

Stephen C. Paschall, II [3] developed the single-actuator ball magnetic levitation system and implemented the control using the digital signal processor controller board DS1104 from dSPACE, Inc. Windows 2000 was used as the OS to interact with this hardware. For the current research, it was concluded that the RATI-Linux would be used as the operating environment. So, it was necessary to use a Linux-compatible hardware. Comedi proved to be the best available solution.

Comedi is a free software project that develops tools, libraries, and drivers for various forms of data acquisition. Comedi works with standard Linux kernels as well as the real-time extensions like RTLinux and RTAI. Comedi project is divided in two packages viz comedi and comedilib. Comedi is a collection of drivers for a variety of common data acquisition plug-in boards that use either PCI (Peripheral Component Interconnect) or PCMCIA (Peripheral Component Microchannel Interconnect Architecture) bus. The single core module called "comedi" provides the common functionality and the data acquisition board specific driver module provides the low-level functionality. Comedilib provides the developer friendly interface to the Comedi devices.

The single actuator ball magnetic levitation system is a single input single output system. For its closed-loop control a data acquisition card requires minimum of one analog/digital converter and one digital/analog converter. National Instruments PCI-

6025E card was selected for data acquisition. Following are some features of this data acquisition card.

- Two 12-bit analog outputs
- 32 digital I/O lines
- two 24-bit counters

CHAPTER IV

DEVELOPMENT OF ARCHITECTURE AND RESULTS

A. Introduction

In the previous chapter, the computing environment for the development of distributed control system was discussed. A brief review of the Linux control and measurement device interface was also done. A discussion on the software architecture used to develop the distributed control system will be conducted in this chapter. A network communication model using sockets will be developed. A 100 mega bits per second (MBPS) Ethernet LAN will be used as the communication network. A control strategy using estimators will be developed to deal with the time delays present in the closed-loop control over this network. With sensor-data estimators in the presence of time delays, the performance of the system is improved. An interrogation on the stability of closed-loop control on distributed networks will be done in the last section of this chapter.

B. Development of software architecture

1. Components

The distributed control system as shown in Figure 22 was developed. The choice of the hardware components in developing the distributed control system was based on their functionality and convenience. A desktop PC with a 600-MHz Celeron processor with the implemented controller will be referred to as the server PC. Another desktop PC with a 1.7-GHz Pentium IV processor was used for data acquisition and will be referred to as the host PC. As discussed in chapter III, RTAI 24.1.12 with Redhat Linux 7.3 was used as the real-time operating environment. National Instrument's

PCI-6025E card was used for data acquisition from the single actuator ball magnetic levitation system introduced in chapter II. This arrangement of has the advantage of using standard commercial hardware. On the other hand, it is not necessarily the best solution for any given specific problem.

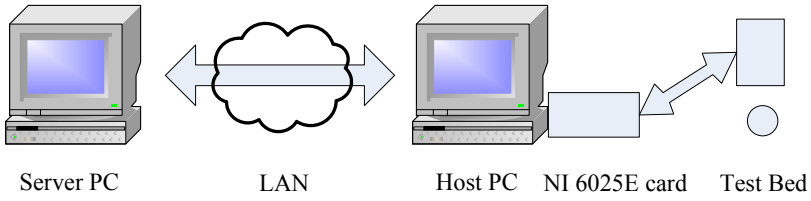

Fig. 22. Block diagram of the distributed system developed during current research.

2. Network communication protocols

The network communication was based on the TCP/IP protocol suite and the programs were developed in C programming language. Functions provided by the Sockets Application Programming Interface (API) were used for implementation. The TCP/IP suite provides various ways of data transfer. The selection of network protocol for communication was an important part of design. The two prominent choices were TCP and UDP.

TCP was specifically designed to provide a reliable end-to-end byte streams over any unreliable network [31]. TCP provides various services like stream data transfer, reliability, efficient flow control, full duplex operation, multiplexing, etc. The data transfer in TCP is connection oriented. TCP connections contain three phases: connection establishment, data transfer, and connection termination. Handshaking signals are used for making and breaking the connection. Parameters such as sequence numbers are initialized to help ensure ordered delivery and robustness. If time delay

is encountered, the data is retransmitted from the sender. Timers and acknowledgement messages are used to detect this time delay or data loss. Check sums are used to detect data corruptions that might occur during the transfer.

Although, TCP is a very reliable protocol for communication over computer networks, it has some disadvantages. Due to various services such as error checking and ordered and reliable data delivery, it has large overheads. In addition to consuming computing resources like CPU and memory, it also introduces time delay in the communication. This introduction of time delay is detrimental to smooth functioning of the time critical closed-loop control process on the network. In the event of congestion, the data are lost more frequently, so more retransmissions are done by TCP, increasing the overheads. For closed-loop control over the network the added reliability provided by TCP is not worth the cost of the network delays it introduces.

UDP is another alternative provided by the TCP/IP protocol suite. Data transfer with UDP is not connection oriented. UDP does not provide additional services such as ensuring ordered data delivery and robustness as provided by TCP. It is therefore known as a best-effort network protocol. UDP adds only application multiplexing and data checksum operations on top of an IP datagram. Although UDP is less reliable, it has fewer overheads, and induces less network delays. Reduced round-trip times justified the use of UDP for the present application.

A program can send and receive UDP messages by opening a socket and reading and writing data to and from the socket. By default, sockets are blocking in nature. This means that a process cannot complete the socket calls immediately. It sleeps and waits till a condition becomes true, then the call is completed. For example, in case of input operation like reading data from a socket, the process sleeps and waits if there is no data available in the socket receive buffer. The process will be awaken only when some data arrive at the receive buffer. This behavior introduces additional

time delays in the communication. To avoid this behavior, the sockets are made non-blocking. In non-blocking socket calls, the process makes an effort to complete the call. An error code is returned if the function fails to complete and an appropriate compensating action can be taken. This ensures that the control loop is executed in timely manner.

3. Timing of events

In addition to keeping the network delays to minimum, it is also important to coordinate the events in the network control loop. Prior work was done with the help of synchronization of clocks for the systems separated by the network [24]. Synchronization of clocks is a very complex process and it depends on the measurement of round trip delay measurement. It requires continuous adjustment to retain this synchronization once generated. Some network traffic is generated with this process. A comparatively simpler approach was followed in the current research. Networked control architecture in current research does not involve clock synchronization and is based on a combination of time-based and event-based communicating processes.

Figure 23 shows a sample timing diagram of network communication between the host and the server. The communications labeled y denote the sensor data transferred from the host to the server and the communications labeled u denote the control action data transferred from the server to the host. The subscripts of these labels denote the sampling period index and indicate whether the data is an estimate. For example, y_2 denotes the sample data of second sampling period, u_3 denotes the control for third sampling period and u_{2e} denotes the control estimate for the second sampling period. The arrows denote the transfer of data between the host and the server. An arrow with a cross in front of it denotes that the data was lost. In the current architecture, late arriving data are considered to be lost.

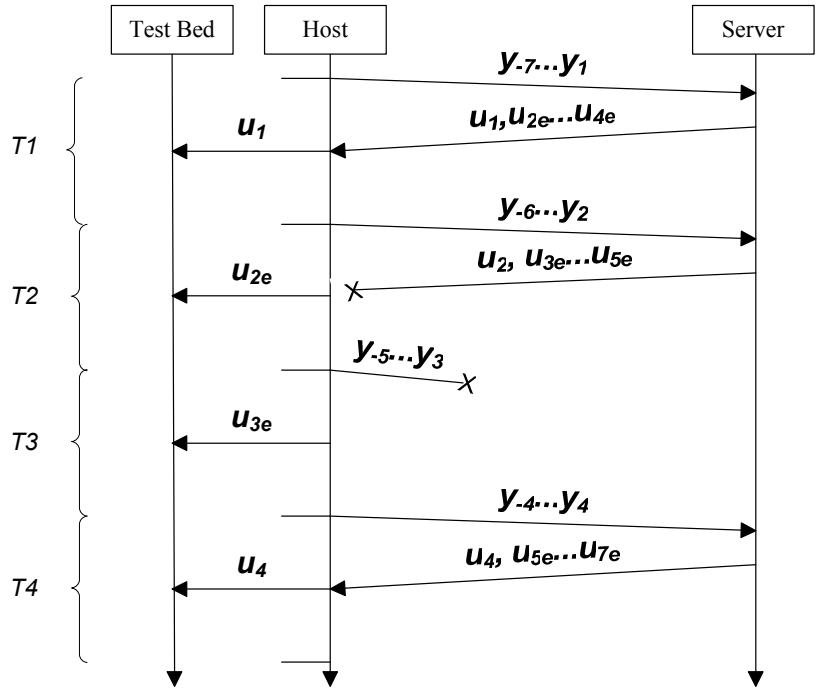

Fig. 23. Sample timing diagram of the system.

Figure 24 shows a pseudocode for the execution of the closed-loop control over the LAN. Sampling and actuation happen on the host side and are time driven, whereas the calculation of control is event driven. The magnetic levitation setup, used as the test bed, is open-loop unstable system. It was calculated that if the sampling period is 3 ms, the actuation has to occur within 1.4 ms after sampling for the system stability. Thus, for 333.33 Hz sampling frequency, the sampling and the actuation are offset by 1.4 ms on the host side. In the software, sampling and actuation are implemented in two different periodic threads. The sampling thread samples the data and sends them using a UDP socket. After sampling has occurred, the actuation thread waits for 1.4 ms for the arrival of the data on the same UDP socket. If the data arrive in time, they are used immediately for actuation, else, the estimate of data that was

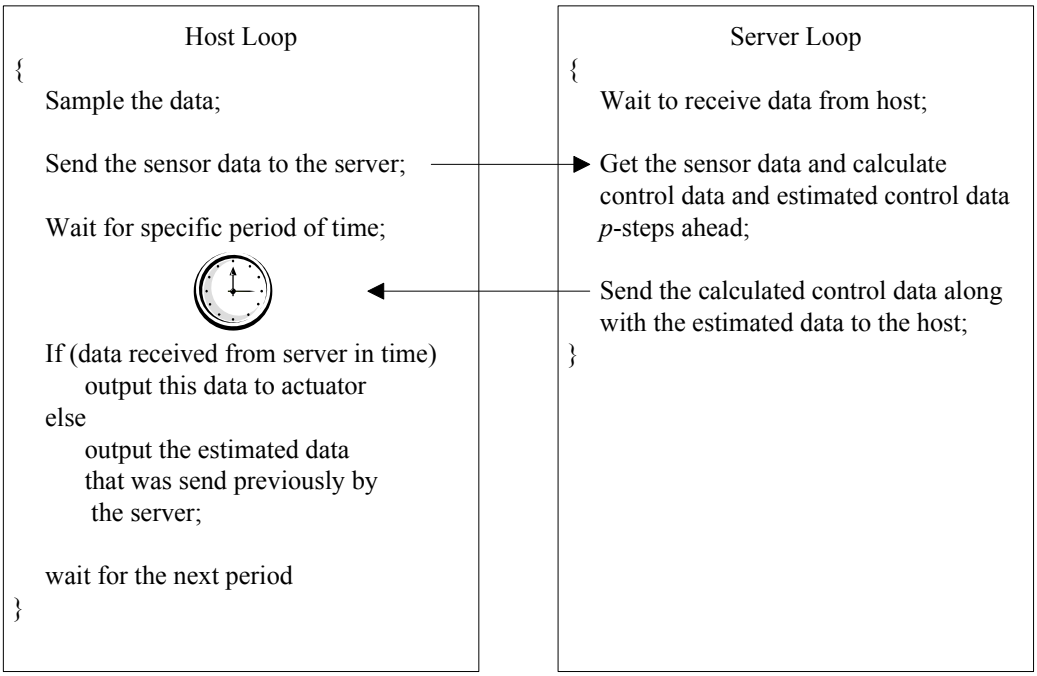

Fig. 24. Pseudocode for the host-server communication.

send in previous message by the server will be used. A discussion on this estimation of data follows shortly.

The control output is calculated on the server side and is event driven. The process on the server side waits for the arrival of sensor data from the host. As soon as the data arrives, appropriate control is calculated and is sent using a UDP socket. In addition to the current control data, predicted control data for next p sampling periods is send to the host. The criterion to select this p will be discussed in the following sections. To calculate these estimates, parametric predictors are used. These predictors require past sensor data which are transmitted to the server along with the current sensor data.

The host stores and refreshes these values of predicted control after each response from the server. In the event of time delays and data loss, these values of estimates are

used to stabilize the system. Another option is to output zero value when no control action is received by the actuator. Prior work has been done using this strategy [19]. But, with open-loop unstable plant like current test bed, the stability of the system is lost in the event of time delays. The use of p-step-ahead prediction is used in current research to maintain system stability in the event of delays. The host side program written in C language is included in Appendix A of this thesis, and the server side program is included in Appendix B.

C. Prediction

It was discussed in previous sections that the upper bound of the allowable time delay in the closed control loop is less than the sampling period of the system. If the time delay in the communication is more than this upper bound, control calculated based on predicted sensor data is used. It was shown in Figure 23 that the server calculated the estimated controls for the next p sampling periods in addition to the control for the current sampling period. Accurate estimation of control data is important to guarantee the stability of the system. In this section, the design of predictors to maintain system stability in the presence of sporadic time delays is explained.

To predict the data, a procedure called system identification is used. There are two types of identification procedures. In off-line identification, a batch of data is collected from the system and this data is used to construct a model. Off-line identification is used for systems which have minimum change in their behavior during operation. For some other systems, it is necessary to design a model in order to support decisions that have to be taken during their operation. Behaviors of these systems change considerably during operation. For these systems, data are updated continuously and it is necessary to infer the model at the same time as the data is

collected. This identification procedure is known as recursive system identification.

Techniques to infer a model from measured data typically contains two steps. A family of candidate models is first decided upon and then, the particular member of the family with satisfactory performance is selected using parameter estimation. In the following two subsections, we consider the development of two important parametric models [32].

1. AR model

For the development of an ARX (autoregressive exogeneous) model, consider a dynamic system represented by the following linear difference equation.

$$y(t) + a_1 y(t-1) + ... + a_n y(t-n) = b_1 u(t-1) + ... + b_m u(t-m) + v(t) \quad (4.1)$$

where, $\{u(t)\}$ is the input signal, $\{y(t)\}$ is the output signal, $v(t)$ is a disturbance of unspecified character and signals are sampled in discrete time $t = 1, 2, 3, ...$
To conveniently write the difference equations, a backward shift (or delay) operator q^{-1} is used. Equation (4.1) can be rewritten as

$$A(q^{-1})y(t) = B(q^{-1})u(t) + v(t) \quad (4.2)$$

where,

$$A(q^{-1}) = 1 + a_1 q^{-1} + ... + a_n q^{-n} \quad (4.3)$$

$$B(q^{-1}) = 1 + b_1 q^{-1} + ... + b_m q^{-m} \quad (4.4)$$

Equation (4.2) represents the dynamic relationship between the input and output signals and can me expressed in terms of a parameter vector and a vector of lagged input-output data. The parameter vector is given by

$$\theta^T = (a_1 ... a_n \; b_1 ... b_m) \quad (4.5)$$

and the vector of lagged input-output data is given by

$$\phi^T(t) = (-y(t-1)... - y(t-n)\ u(t-1)...u(t-m)). \tag{4.6}$$

Equation (4.2) can be rewritten as

$$y(t) = \theta^T \phi(t) + v(t). \tag{4.7}$$

Equation (4.7) is known as a linear regression model and the components of $\phi(t)$ are called as the regression variables or regressors. Assuming that the characteristic of the noise $v(t)$ is unknown, and previous values of $y(k)$ and $u(k)$ known, the prediction of $y(t)$ can be represented as,

$$\hat{y}(t|\theta) = \theta^T \phi(t) \tag{4.8}$$

where, $k = t-1, t-2, ...$ In the absence of input, the model of the signal $\{y(t)\}$ becomes,

$$y(t) + a_1 y(t-1) + ... + a_n y(t-n) = v(t). \tag{4.9}$$

Equation (4.9) is referred to as an autoregressive process of order n, or an AR(n) process.

2. ARMAX model

In (4.2), the behavior of disturbance term $v(t)$ is unspecified. If we model this term as a moving average (MA) of a white noise sequence $\{e(t)\}$,

$$v(t) = C(q_{-1})e(t) \tag{4.10}$$

where,

$$C(q^{-1}) = 1 + c_1 q^{-1} + ... + c_r q^{-r} \tag{4.11}$$

Substituting this in (4.2) we get,

$$A(q^{-1})y(t) = B(q^{-1})u(t) + C(q^{-1})v(t) \tag{4.12}$$

In (4.12), $A(q^{-1})y(t)$ is the autoregressive part (AR), $B(q^{-1})u(t)$ is the moving average part (MA) and $C(q^{-1})v(t)$ is the control part or the exogeneous part (X). So, the model is known as the ARMAX model. This model can be expressed in terms of the parameter vector

$$\theta^T = (a_1...a_n\ b_1...b_m\ c_1...c_r). \tag{4.13}$$

By rearranging the terms, (4.12) can be written as,

$$y(t) = \left[1 - \frac{A(q^{-1})}{C(q^{-1})}\right] y(t) + \frac{B(q^{-1})}{C(q^{-1})}u(t) + v(t) \tag{4.14}$$

Both the polynomials $A(q^{-1})$ and $C(q^{-1})$ are monic and can be represented as summations of θ dependent sequences as,

$$1 - \frac{A(q^{-1})}{C(q^{-1})} = \sum h_k(\theta)q^{-k} \tag{4.15}$$

$$\frac{B(q^{-1})}{C(q^{-1})} = \sum g_k(\theta)q^{-k} \tag{4.16}$$

These sequences will tend to zero exponentially if all the roots of the polynomial

$$C^*(z) = z^r + c_1 z^{r-1} + ... + c_r \tag{4.17}$$

are inside the unit circle. In (4.14), the values on the right hand side are known at time $t-1$ except for $e(t)$. But because $e(t)$ is independent of the events that have happened up to time $t-1$, the natural predictor can be stated as [32]

$$g_\mu(\theta; t, z^{t-1}) = \left[1 - \frac{A(q^{-1})}{C(q^{-1})}\right] y(t) + \frac{B(q^{-1})}{C(q^{-1})}u(t) \tag{4.18}$$

$$= \sum h_k(\theta)y(t-k) + \sum g_k(\theta)u(t-k) \tag{4.19}$$

This predictor assumes that all the data from $t = -\infty$ is known whereas, our assumption is that the data is available from $t = 0$. But, because h_k and g_k decay exponentially, summing up only to $k = t$ may be sufficient. For convenience, (4.19) can be reorganized as

$$\hat{y}(t \mid \theta) = \left[1 - \frac{A(q^{-1})}{C(q^{-1})}\right] y(t) + \frac{B(q^{-1})}{C(q^{-1})} u(t) \tag{4.20}$$

which gives,

$$C(q^{-1})\hat{y}(t \mid \theta) = [C(q^{-1}) - A(q^{-1})]y(t) + B(q^{-1})u(t) \tag{4.21}$$

Equation (4.21) is a convenient finite difference equation for calculating $\hat{y}(t \mid \theta)$. In this equation, if the input is not available, the system model is known as ARMA model and is represented as,

$$C(q^{-1})\hat{y}(t \mid \theta) = [C(q^{-1}) - A(q^{-1})]y(t) \tag{4.22}$$

D. Selection of predictors

The AR and ARMA models described above were used to design the predictors. It was also necessary to choose an appropriate order for the predictor. The accuracy of prediction and the number of computations required for prediction are the two important factors to consider in the decision of the order of predictors. It was observed that a tradeoff exists between accuracy of a predictor and number of computations done by the predictor for each prediction. Generally higher-order predictors have better accuracy of prediction than the lower-order predictors of the same type. On the other hand, higher-order predictors take more number of computations for a prediction than the lower-order predictors of the same type. A good predictor should give a reasonably accurate prediction with the minimum usage of computational resources.

Another factor to consider is the amount of past data required. With higher-order predictors, more past data are required for computation. But in the current communication architecture there is a limitation on this. In the current system, the storage of past output data and the control data is done on the host side. If the server had to store these data, in the event of packet drops from the host to the server, the data on the server side would be inaccurate. Therefore, each time the past output data are passed to the server by the host along with the current output data. It is necessary that these data travel in a single data packet. If these data are fragmented into multiple packets and any of these fragmented packets is lost, the past data required for prediction by the server would be inaccurate. Instead, it is acceptable if all the data are lost, because the predicted control data send in the last communication can be used.

IP provides packet delivery service for protocols like UDP and TCP. The version of IP used in current research is IPv4 (Internet Protocol, version 4). A minimum reassembly buffer size identified by IPv4 is 576 bytes [33]. This is the minimum datagram size that is guaranteed to be supported by any implementation of IPv4. In the event of congestion, the UDP packets are lost. Larger packets have higher probability of getting lost than the smaller packets. So, it was required to keep the size of packet to a minimum while sending adequate amount of data for better predictions.

MATLAB's system identification toolbox was used to develop predictors based on AR and ARMA models of various orders. Adequate amount of off-line output data of the stable system, was required for development of predictors. To collect these data, the test bed was operated with the feedback loop closed locally. The output data of the system for 17162 samples was collected. Figure 25 shows this collected data.

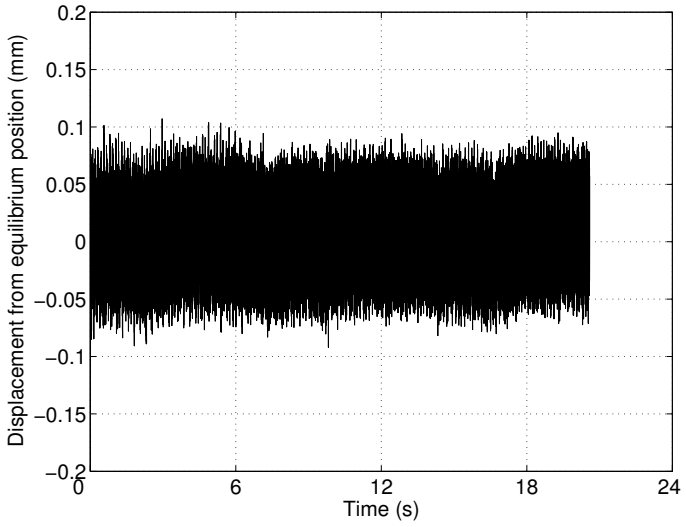

Fig. 25. Plot of displacement of ball from the equilibrium position vs. time. The plot shows the data collected for development of predictors.

MATLAB used half of this data for development of predictors and the rest half of the data for validation of the predictors. The percentage of the output variations that is reproduced by the predictors is given by a fit. Mathematically, a fit can be stated as [34]

$$fit = \left[\frac{1 - norm(Y - \hat{Y})}{norm(Y - mean(Y))}\right] * 100 \tag{4.23}$$

where, where Y is the measured output and $\hat{Y}$ is the simulated/predicted model output. Higher fit means better prediction. Table IV and Table V represent the best fit values for AR and ARMA models respectively for various step-ahead predictions.

Based on the above discussion and results, the 8th order AR model was selected to design the predictor. Predictors were designed for up to 4-step-ahead predictions. The parameter vectors for various predictors, calculated using MATLAB were used

Table IV. Best fits for AR models.

Order	1-step	2-step	3-step	4-step	5-step	6-step	7-step	8-step
8	74.43	67.02	65.40	65.60	64.78	63.98	64.86	64.39
7	72.63	64.79	63.76	63.98	63.99	60.96	60.36	60.04
6	72.05	62.68	61.35	62.16	63.27	59.67	57.34	55.92
5	71.95	62.86	61.74	62.05	63.43	59.61	57.64	56.97

Table V. Best fits for ARMA models.

Order	1-step	2-step	3-step	4-step	5-step	6-step	7-step	8-step
8	74.41	67.00	65.34	65.54	64.73	63.83	64.75	64.41
7	72.62	64.73	63.725	63.90	63.97	60.85	60.13	59.96
6	72.03	62.67	61.32	61.99	63.16	59.52	57.12	55.86
5	71.93	62.81	61.73	61.93	63.33	59.52	57.43	56.93

to develop the following equations for predictors.

$$\begin{aligned}\hat{y}(t+1) &= 0.8122y(t) - 0.3479y(t-1) - 0.0294y(t-2) + 0.4605y(t-3) \\ &\quad +0.0742y(t-4) + 0.1042y(t-5) + 0.1117y(t-6) - 0.3561y(t-7)\end{aligned} \tag{4.24}$$

$$\begin{aligned}\hat{y}(t+2) &= 0.3117y(t) - 0.3119y(t-1) + 0.4366y(t-2) + 0.4482y(t-3) \\ &\quad +0.1645y(t-4) + 0.1964y(t-5) - 0.2653y(t-6) - 0.2892y(t-7)\end{aligned} \tag{4.25}$$

$$\begin{aligned}\hat{y}(t+3) &= -0.0587y(t) + 0.3281y(t-1) + 0.4390y(t-2) + 0.3080y(t-3) \\ &\quad +0.2195y(t-4) - 0.2329y(t-5) - 0.2544y(t-6) - 0.1110y(t-7)\end{aligned} \tag{4.26}$$

$$\begin{aligned}\hat{y}(t+4) &= 0.2804y(t) + 0.4594y(t-1) + 0.3097y(t-2) + 0.1925y(t-3) \\ &\quad -0.2372y(t-4) - 0.2605y(t-5) - 0.1176y(t-6) + 0.0209y(t-7)\end{aligned} \tag{4.27}$$

The composition of a typical 76-bytes long IP packet going from the host to the server is shown in Figure 26. It consists of a 20-byte-long IP header, an 8-byte-long

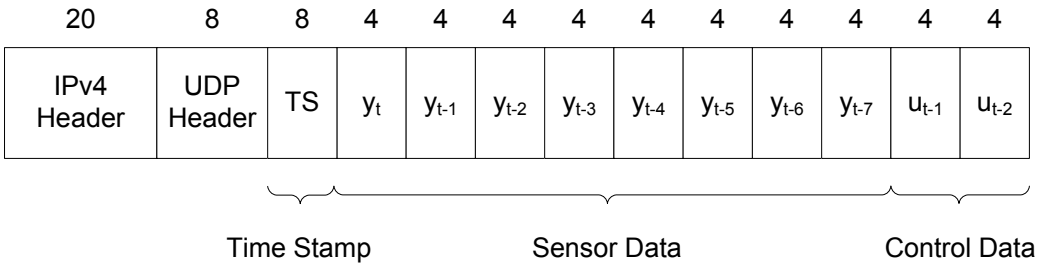

Fig. 26. Composition of IPv4 packet sent from the host to the server.

UDP header, an 8-byte-long time stamp, eight sensor data values 4-byte-long and two previous 4-byte-long control data values. A time stamp is taken on the host side at the time of sampling and is sent to the server. The server does not modify the time stamp but sends it back to the host along with the calculated control data. This time stamp is then used by the host to identify whether the arrived data packet is the expected packet or a delayed packet. If a packet is delayed, it is discarded in the current scheme. In future, improved schemes can be developed to use the data in these delayed packets effectively.

The composition of a typical 52-bytes long IP packet going from the server to the host is shown in Figure 27. It consists of a 20-byte-long IP header, an 8-byte-long UDP header, an 8-byte-long time stamp, one current control data value and four predicted control data values.

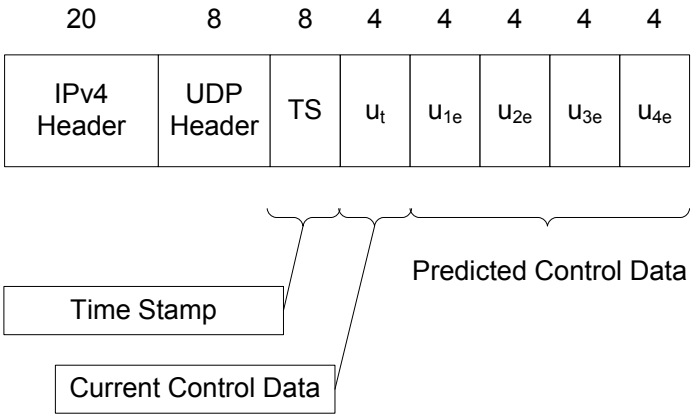

Fig. 27. Composition of IPv4 packet sent from the server to the host.

E. Performance of control strategy

To verify the working of the control strategy, various simulations were conducted. To have a precise control over the introduction of artificial delays, both the host and the server were run on the same workstation. The sampling was done at 333.333 Hz by the host side and the maximum allowable delay between sampling the data and application of control was kept 1.2 ms.

In the first set of simulations, the architecture was tested for the maximum number of allowable consecutive delays. In the first simulation of this set, for 3000th sample, a data packet was simulated to be lost and zero was output to the actuator. The system being open-loop unstable, the system lost stability as expected and the levitated ball could not maintain its equilibrium position. The output of the system is shown in Figure 28.

In the second simulation of this set, the control data based on predictions of the sensor data was used. Consecutive packet losses were simulated by rejecting the arrived data p successive times at the socket after every 3000 samples. In the case

of simulated packet losses, control calculated using predicted data was used. These values of control were send by the server to the host in the previous communication. The value of p was increased by one in every run of the experiment starting from zero. The maximum value of p was found out to be two with the current set of predictors. In other words, the system accommodated two consecutive packet drops without loosing stability. The response of the system is shown in Figure 29. It can be observed that every time these consecutive delays occur, the system performance is affected.

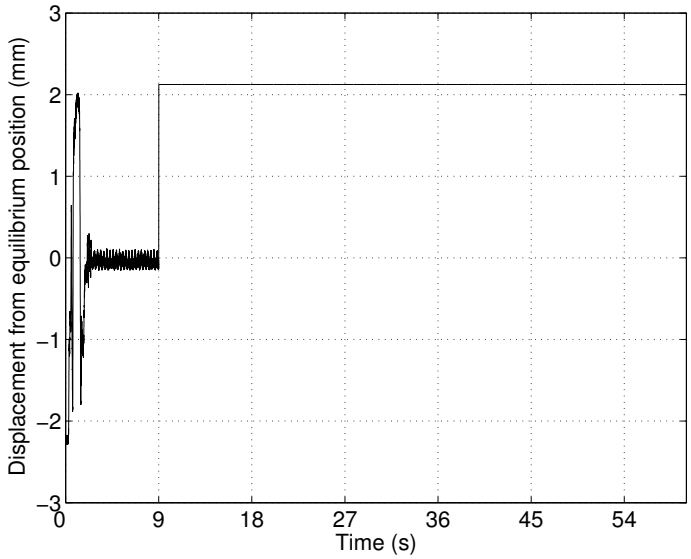

Fig. 28. Plot of displacement of ball from the equilibrium position vs. time. The plot is system response for the 1st simulation of the 1st set.

In the second set of simulations, the architecture was tested for 1-step-ahead prediction only. In the first experiment of this set, an artificial delay of 2 ms was introduced with real-time-sleep function on the server side at the 7000th time step. It was observed that as the data did not arrive in time at the host side, the system lost its stability and the ball could not maintain its equilibrium position. The response of

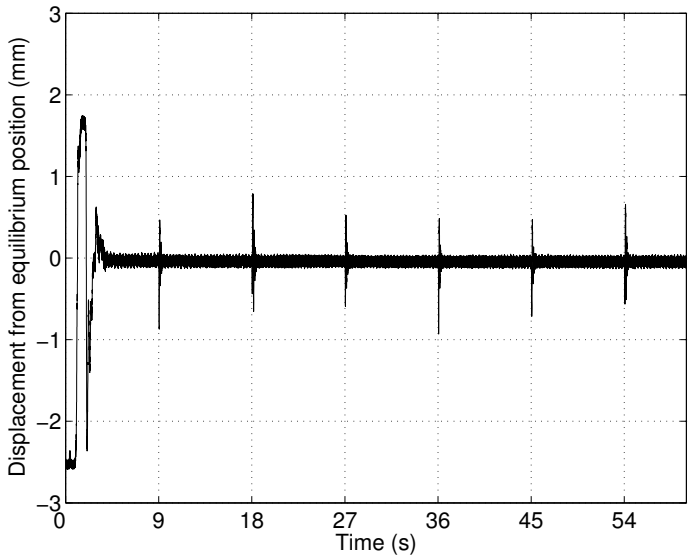

Fig. 29. Plot of displacement of ball from the equilibrium position vs. time. The plot is system response for the 2nd simulation of the 1st set.

the system is shown in Figure 30.

In the second experiment, the predictor was implemented using the control strategy discussed in Section D. Only one-step-ahead prediction was used. Instead of introducing a single artificial delay at the 7000th time step, as in the first experiment, an artificial time delay of 2 ms was introduced on the server side for every q_{th} time step starting from 6500th time step. The value of q was tested for 10 and the system was found to be stable. The value of q was then reduced by one for each subsequent run of the experiment and the system was checked for stability. The minimum value of q was found out to be 5. This represented the simulation of one long time delay of sporadic nature in five consecutive delays. Thus, system stability was achieved in the events of up to 20 percent loss of data in communication. The response of the system for q equal to five is shown in Figure 31. In the event of simulated network delays, the ball did not fall down from its equilibrium position.

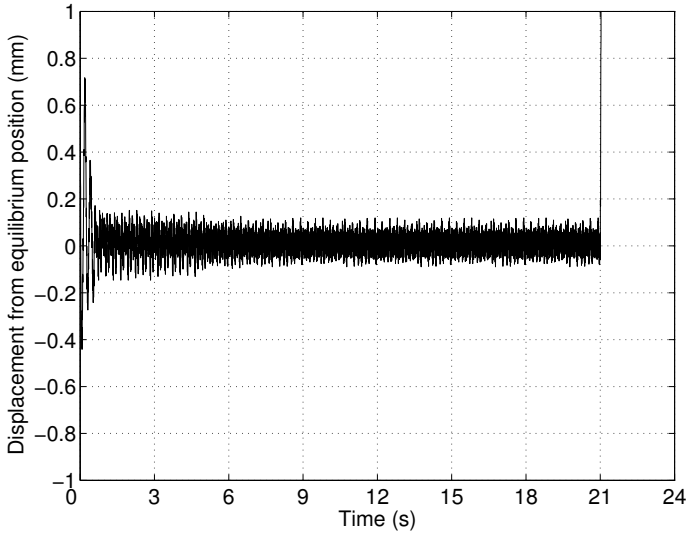

Fig. 30. Plot of displacement of ball from the equilibrium position vs. time. The plot is system response for the 1st simulation of the 2nd set.

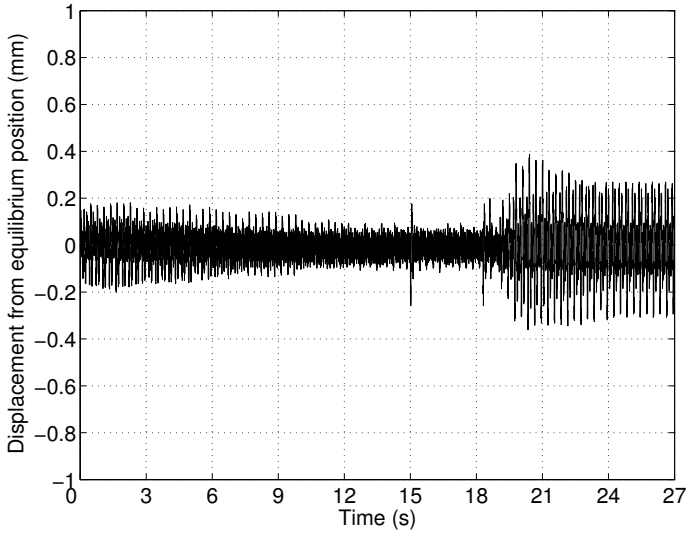

Fig. 31. Plot of displacement of ball from the equilibrium position vs. time. The plot is system response for the 2nd simulation of the 2nd set.

It was also observed that the movement of the ball about the equilibrium point increased after implementation of this control strategy. This degradation in the performance of the system was due to the use of control based upon the estimated sensor data instead of actual sensor data.

In order to observe the system stability on the network, the host and the server were executed on two different workstations. It was observed that the system remained stable for approximately about 15 minutes. Thus, the algorithm based on predicted sensor data was used to stabilize the network-based closed-loop control system in the event of sporadic time delays.

CHAPTER V

CONCLUSIONS

A. Conclusions

The closed-loop control of magnetic ball levitation system was achieved on the LAN. The closed-loop control on a network is a very time-sensitive system. For example, for the magnetic ball levitation system with the 3-ms sampling period, only 1.4 ms is available between taking an output sample and applying the control for that sampled data. In the network-based closed-loop architecture, one communication cycle consists of sending the sampled data from the host to the server, calculation of current control and predicted control on the server side, and getting this data to the host side. Completion of this communication cycle in 1.4 ms is critical for the stability of the system. To time the events properly, RTAI with Redhat Linux 7.3 as an operating environment was successfully used in the current research.

Although TCP communications are reliable, take longer as compared to UDP communications. Long communication delays are detrimental to the performance of closed-loop networked systems and additional reliability of TCP is not worth the cost of longer network delays. UDP was used in the communication architecture of closed-loop real-time networked system in the current research.

The delays in communication can exceed the upper bound of allowable time delay of the network-based closed-loop system. This upper bound in case of magnetic ball levitation system was 1.4 ms for the 3 ms sampling period. One strategy that was previously used by other researchers is to output zero to the actuator in the event of occurrence of such delays. Instead, control output calculated using predicted output is used to stabilize the system in the current research. These p-step-ahead predictions

were calculated on the server side and control data based on these predictions was send to the host. In the event of time delays exceeding the allowable upper bound, the predicted control data was used and the system stability was achieved.

AR and ARMA models were used to design the predictors. The best fit criterion was used to select an AR model of 8th order. The time required for calculations and increase in IP packet size limited the use of higher order predictors. Two sets of simulated experiments were conducted to test the performance of the network based architecture for closed-loop control. It was shown that the system stability was achieved in the events of up to 20 percent loss of data in communication. It was also demonstrated that the current system based on predictors can compensate for two consecutive packet losses in the network.

Some conclusions that can be made based on the results of this research include:

1. In order to time the events in a NCS properly, and have a complete control on their execution, use of real-time operating environment is essential.

2. UDP is more suitable than TCP for network-based closed-loop control systems.

3. Algorithm based on predictors stabilizes the network-based closed-loop control system in the event of sporadic network delays.

4. The number of consecutive network delays compensated for by the system depends on the accuracy of predictors.

B. Future work

The communication architecture used in current research does not use any kind of congestion control mechanism. In the event of network congestion, the probability of multiple consecutive network delays and packet losses increases. In the future, better

communication strategies like sending duplicate packets can be used to account for consecutive delays during congestion.

To maintain stability of the system in the event of larger number of packet losses and longer time delays, a deeper study of the architecture accompanied by rigorous mathematical treatment is necessary. The number of consecutive delays or packet losses that can be compensated for by the system can be increased by better design of predictors.

REFERENCES

[1] F. L. Lian, J. Moyne, and D. Tilbury, "Network design consideration for distributed control systems," *IEEE Transactions on Control Systems Technology*, vol. 10, no. 2, pp. 297–307, March 2002.

[2] A. Srivastava, "Distributed real-time control via the Internet," M.S. thesis, Texas A&M University, College Station, TX, May 2003.

[3] S. C. Paschall II, "Design, fabrication and control of a single actuator magnetic levitation system," Senior Honors Thesis, Texas A&M University, College Station, TX, May 2002.

[4] R. C. Geortz, "Manipulator systems developed at anl," in *Proc. of 12th Conference on Remote Systems Technology*, 1964, vol. 1, pp. 117–136.

[5] T. B. Sheridan, "Telerobotics," *Automatica*, vol. 25, no. 4, pp. 487–507, 1989.

[6] Y. Yokokohji and T. Yoshikawa, "Bilateral control of master-slave manipulators for ideal kinesthetic coupling - formulation and experiment," *IEEE Transactions on Robotics and Automation*, vol. 10, no. 5, pp. 605–620, October 1994.

[7] R. Safaric, M. Debevc, R. M. Parkin, and S. Uran, "Telerobotics experiments via Internet," *IEEE Transactions in Industrial Electronics*, vol. 48, no. 2, pp. 424–431, April 2001.

[8] H. Hu, L. Yu, P. W. Tsui, and Q. Zhou, "Internet-based robotic systems for teleoperation," *International Journal of Assembly Automation*, vol. 21, no. 2, pp. 143–151, May 2001.

[9] A. J. Madhani, G. Niemeyer, and J. K. Salisbury Jr., "The black falcon: A teleoperated surgical instrument for minimally invasive surgery," in *Proc. of the IEEE/RSJ Intl. Conference on Intelligent Robots and Systems*, Victoria, B.C., Canada, October 1998, vol. 2, pp. 936–944.

[10] M. Mitsuishi, S. Tomisaki, and T. Yoshidome, "Tele-micro-surgery system with intelligent user interface," in *Proc. of the IEEE International Conference on Robotics and Automation*, San Francisco, CA, April 2000, vol. 2, pp. 1607–1614.

[11] C. Smith and P. K. Wright, "Cybercut: A world wide web design-to-fabrication tool," *Journal of Manufacturing Systems*, vol. 16, pp. 281–286, January 1996.

[12] T. T. Ho and H. Zhang, "Internet-based tele-manipulation," in *Proc. of the IEEE Canadian Conference on Electrical and Computer Engineering*, Edmonton, Alberta, Canada, May 1999, vol. 3, pp. 1425–1430.

[13] K. Goldberg, B. Chen, R. Solomon, and S. Bui, "Collaborative teleoperation via the Internet," in *Proc. of IEEE International Conference on Robotics and Automation*, San Francisco, CA, April 2000, vol. 2, pp. 2019–2024.

[14] J. H. Park and T. B. Sheridan, "Supervisory teleoperation control using computer graphics," in *Proc. of the IEEE International Conference on Robotics and Automation*, Sacramento, CA, April 1991, vol. 1, pp. 493–498.

[15] R. C. Luo, J. H. Tzou, and Y. C. Chang, "Desktop rapid prototyping system with supervisory control and monitoring through Internet," *IEEE/ASME Transactions on Mecharonics*, vol. 6, no. 4, pp. 399–409, December 2001.

[16] C. E. Garcia, R. Carelli, J. F. Postigo, and C. Soria, "Supervisory control of a telerobotic system: A hybrid control approach," *Control Engineering Practice*,

vol. 11, no. 7, pp. 805–817, July 2003.

[17] A. Srivastava and W. -J. Kim, "Internet-based supervisory control with stochastic delay models," in *Proc. of the American Control Conference*, Denver, CO, June 2003, vol. 1, pp. 627–632.

[18] J. Eker and A. Cervin, "Distributed wireless control using Bluetooth," in *Proc. of IFAC Conference on New Technologies for Computer Control*, Hong Kong, P.R. China, November 2001.

[19] N. J. Ploplys and A. G. Alleyne, "UDP network communications for distributed wireless control," in *Proc. of the American Control Conference*, Denver, CO, June 2003, vol. 4, pp. 3335–3340.

[20] W. R. Ferrell, "Delayed force feedback," *IEEE Transactions on Human Factors in Electronics*, vol. 8, pp. 449–455, October 1967.

[21] R. J. Anderson and M. W. Spong, "Bilateral control of teleoperators with time delay," *IEEE Transactions on Automatic Control*, vol. 34, no. 5, pp. 494–501, May 1989.

[22] L. Conway, R. A. Volz, and M. W. Walker, "Teleautonomous systems: Projecting and coordinating intelligent action at a distance," *IEEE Transactions on Robotics and Automation*, vol. 6, no. 2, pp. 146–157, April 1990.

[23] V. S. Frost and B. Melamed, "Traffic modeling for telecommunications networks," *IEEE Communications Magazine*, vol. 32, no. 3, pp. 70–80, March 1994.

[24] L. Zhang, Z. Liu, and C. H. Xia, "Clock synchronization algorithms for network measurements," in *Proc. of IEEE INFOCOM*, 2002, vol. 1, pp. 160–169.

[25] O. Beldiman, G. C. Walsh, and L. Bushnell, "Predictors for networked control systems," in *Proc. of the American Control Conference*, Chicago, IL, June 2000, vol. 4, pp. 2347–2351.

[26] H. H. Woodson and J. R. Melcher, *Electromechanical Dynamics*, 3rd ed. Malabar, FL: Krieger Publishing Company, 1990.

[27] M. Kirk, "Time-critical communication systems," *Computing & Control Engineering Journal*, vol. 2, no. 1, pp. 35–42, January 1991.

[28] R. A. Volz, *Real-Time Computing*, Lecture notes for CPSC 456, Department of Computer Science, Texas A&M University, College Station, TX, 2003.

[29] D. A. Mellichamp, *Real-time Computing with Applications to Data Acquisition and Control*, 1st ed. Ontario, Canada: Van Nostrand Reinhold Company, 1983.

[30] P. Mantegazza, "DIAPM RTAI - realtime application," [Online]. Available: http://www.rtai.org, Accessed on March 2004.

[31] A. S. Tanenbaum, *Computer Networks*, 3rd ed. Upper Saddle River, NJ: Prentice-Hall, 2001.

[32] L. Ljung and T. Soderstrom, *Theory and Practice of Recursive Identification*, 1st ed. Cambridge, MA: The MIT Press, 1983.

[33] W. R. Stevens, *UNIX Network Programming*, 2nd ed. Upper Saddle River, NJ: Prentice-Hall., 1998.

[34] L. Ljung, "The system identification toolbox," [Online]. Available: http://www.mathworks.com/access/helpdesk/help/toolbox/ident, Accessed on May 2004.

APPENDIX A

HOST.C

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <pthread.h>
#include <signal.h>
#include <comedilib.h>
#include <sys/socket.h>
include<netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <sys/ioctl.h>
#include "defines.h"

#define KEEP_STATIC_INLINE #include<rtai_lxrt_user.h>
#include<rtai_lxrt.h>\
#define PERIOD 1000000 #define LOOPS 1000
#define NTASKS 2 #define taskname(x) (1000 + (x))
```

```
RTIME time_stamp;
double u0;
double u1e;
double u2e;
double u3e;
double u4e;
double u5e;
double u6e;
double u7e;
double u8e;

double y0, y_1, y_2, y_3, y_4;\
double y_5, y_6, y_7, u_1, u_2;
RTIME current_time_stamp;\
pthread_t task[NTASKS]; int ntasks =
NTASKS;\
 RT_TASK *mytask;

SEM *sem; static int cpus_allowed;

SEM *sock_sem;\
int sockid;\
RTIME start_instant;

int server_sock_size = 0;
struct sockaddr_in my_addr, server_addr;
```

```
comedi_t *it;
int in_subdev = 0;
int out_subdev = 1;

int in_chan = 0;
int out_chan = 0;
int in_range = 0;

int out_range = 0;
int aref = AREF_GROUND;
int i=0;

//comedi declarations
lsampl_t in_data;
lsampl_t out_data;
float volts = 0.0;

int in_maxdata = 0, out_maxdata = 0;
comedi_range *in_range_ptr,
*out_range_ptr;
int endme_int = 0;

void terminate_normally(int signo); void endme(int sig)
{
    printf("You want to kill me?\n");
```

```
    endme_int = 1;
    //rt_sem_delete(sem);
    //comedi_close(it);
    //stop_rt_timer();
    //rt_task_delete(mytask);
    //signal(SIGINT, SIG_DFL);
    exit(1);
}

void *send_thread_fun(void *arg)
{
    RTIME start_time, period, end_time, difference;
    RTIME t0;
    SEM *sem;
    RT_TASK *mytask;
    unsigned long mytask_name;
    int mytask_indx;
    double * buffer = NULL;
    int iRet = 0;
    struct recv_data *send_msg = NULL;
    int send_msg_size;
    FILE *fp = NULL;
    float print_data[20000];
    int loop_count = 0;

    fp = fopen("result.xls","w");
```

```
if(fp == NULL)
{
    printf("could not open file");
    exit(0);
}

pthread_setcanceltype(PTHREAD_CANCEL_ASYNCHRONOUS, NULL);
mytask_indx = 0;
mytask_name = taskname(mytask_indx);
cpus_allowed = 1 - cpus_allowed;
if (!(mytask = rt_task_init_schmod(mytask_name, 1, 0, 0,
    SCHED_FIFO, 1 << cpus_allowed)))
{
    printf("CANNOT INIT send_thread TASK\n");
    exit(1);
}
printf("send thread pid = %ld\t master pid = %ld\n",
        getpid(), getppid());
mlockall(MCL_CURRENT | MCL_FUTURE);

//rt_make_hard_real_time();

rt_receive(0, (unsigned int*)&sem);

send_msg_size = sizeof(struct recv_data);
if(( send_msg = (struct recv_data *)
```

```
    calloc(1, sizeof(struct recv_data))) == NULL)
{
        printf("cannot allocate message memory\n");
    exit(4);
}

period = nano2count(PERIOD);
start_time = rt_get_time() + nano2count(10000000);
t0 = start_instant;
rt_task_make_periodic(mytask, (t0 + nano2count(500000000)),
        nano2count(3000000));
//rt_task_make_periodic(mytask, rt_get_time(),
        nano2count(3000000));
start_time = rt_get_cpu_time_ns();
printf("starting the send_thread while loop\n");
for(;;)
{
    if(endme_int == 1)
    {
        break;
    }

    comedi_data_read(it, in_subdev, in_chan, in_range,
                    aref, &in_data);
    if(in_data > 4094)
    {
```

```
    in_data = 4094;
}
if(in_data < 2049)
{
    in_data = 2049;
}
current_time_stamp = rt_get_cpu_time_ns();

y0 = comedi_to_phys(in_data, in_range_ptr, in_maxdata);
print_data[loop_count] = y0;
send_msg->y0 = y0;
send_msg->y_1 = y_1;
send_msg->y_2 = y_2;
send_msg->y_3 = y_3;
send_msg->y_4 = y_4;
send_msg->y_5 = y_5;
send_msg->y_6 = y_6;
send_msg->y_7 = y_7;
send_msg->u_1 = u_1;
send_msg->u_2 = u_2;
send_msg->time_stamp = current_time_stamp;

rt_sem_wait(sock_sem);
iRet = sendto(sockid, (const void *)send_msg, send_msg_size,
              0,(struct sockaddr*)&server_addr,
              server_sock_size);
```

```
    rt_sem_signal(sock_sem);
    if(iRet <= -1)
    {
        perror("sendto() failed\n");
        break;
    }

    y_7 = y_6;
    y_6 = y_5;
    y_5 = y_4;
    y_4 = y_3;
    y_3 = y_2;
    y_2 = y_1;
    y_1 = y0;

    loop_count++;
    if(loop_count == 20000)
    {
        break;
    }

    rt_task_wait_period();
}
end_time = rt_get_cpu_time_ns();
difference = end_time - start_time;
printf("difference = %lld\n", difference);
```

```
    endme_int++;
    rt_sem_signal(sem);
    rt_make_soft_real_time();
    for(i=0;i<20000;i++)
    {
        fprintf(fp, "%f\n", print_data[i]);
    }
    fclose(fp);

    free(send_msg);
    rt_task_delete(mytask);
    printf("send_thread ENDS\n");
    return 0;
}

void *recv_thread_fun(void *arg) {
    RTIME start_time, period, end_time, difference;
    RTIME t0;
    SEM *sem;
    RT_TASK *mytask;
    unsigned long mytask_name;
    int mytask_indx;
    struct data *buffer = NULL;
    int iRet = 0;
    int recv_msg_size;
    struct send_data *recv_msg = NULL;
```

```
int loop_count = 0;

recv_msg_size = sizeof(struct send_data);
if(( recv_msg = (struct send_data *)
    calloc(1, sizeof(struct send_data))) == NULL)
{
        printf("cannot allocate message memory\n");
    exit(4);
}

pthread_setcanceltype(PTHREAD_CANCEL_ASYNCHRONOUS, NULL);
mytask_indx = 1;
mytask_name = taskname(mytask_indx);
cpus_allowed = 1 - cpus_allowed;
if (!(mytask = rt_task_init_schmod(mytask_name, 1,
    0, 0, SCHED_FIFO, 1 << cpus_allowed)))
{
    printf("CANNOT INIT recv_thread TASK\n");
    exit(1);
}
printf("recv thread pid = %ld\t master pid = %ld\n",
        getpid(), getppid());
mlockall(MCL_CURRENT | MCL_FUTURE);

//rt_make_hard_real_time();
```

```
rt_receive(0, (unsigned int*)&sem);

period = nano2count(PERIOD);
start_time = rt_get_time() + nano2count(10000000);
t0 = start_instant;
printf("recv: t0 = %lld\t", count2nano(t0));
printf("This period = %lld\t", count2nano(rt_get_time()));
printf("actual start = %lld\n", count2nano(t0
        + nano2count(500500000)));
rt_task_make_periodic(mytask, (t0 + nano2count(500500000)),
        nano2count(3000000));

start_time = rt_get_time();

printf("starting the recv_thread while loop\n");
for(;;)
{
    if(endme_int == 1)
    {
        break;
    }

    rt_sem_wait(sock_sem);
    iRet = recvfrom(sockid, (void *)recv_msg, recv_msg_size, 0,
            (struct sockaddr *)&server_addr, &server_sock_size);
    rt_sem_signal(sock_sem);
```

```
if(iRet <= -1)
{
    endme_int = 1;
    perror("recvfrom() failed\n");
    break;
}

if((loop_count < 3001))
{
u0 = recv_msg->u0;
u1e = recv_msg->u1e;
u2e = recv_msg->u2e;
u3e = recv_msg->u3e;
u4e = recv_msg->u4e;
u5e = recv_msg->u5e;
u6e = recv_msg->u6e;
u7e = recv_msg->u7e;
u8e = recv_msg->u8e;
}

if(loop_count < 3001)
{
    volts = u0;
}
else if(loop_count == 3001)
```

```
{
    volts = u1e;
}
else if(loop_count == 3002)
{
    volts = u2e;
}

if(volts > 10.0)
{
    volts = 9.99999;
}
if(volts < 0.0)
{
    volts = 0.0;
}

out_data = comedi_from_phys(volts, out_range_ptr,
                            out_maxdata);
comedi_data_write(it, out_subdev, out_chan, out_range,
                  aref, out_data);

u_2 = u_1;
u_1 = u0;
if(loop_count == 3001)
{
```

```
            loop_count = 0;
        }
        else
        {
            loop_count++;
        }
        rt_task_wait_period();
    }
    end_time = rt_get_cpu_time_ns();
    difference = end_time - start_time;
    //printf("difference = %lld\n", difference);
    endme_int++;
    rt_make_soft_real_time();

    free(recv_msg);
    rt_task_delete(mytask);
    printf("recv_thread ENDS\n");
    return 0;
}

int main(void) {
    int i;
    unsigned long mytask_name = nam2num("MASTER");
    struct sigaction sa;
```

```
char * server_ip = "165.91.214.188";
unsigned short my_port, server_port;

my_port = 4445;
server_port = 4444;

/* -- Create client side socket -- */
printf("creating socket\n");
if( (sockid = socket(AF_INET, SOCK_DGRAM, 0)) < 0)
{
  perror("socket() failed ");
  exit(2);
}

memset((void *) &my_addr, (char) 0, sizeof(my_addr));
my_addr.sin_family = AF_INET;
my_addr.sin_addr.s_addr = htonl(INADDR_ANY);
my_addr.sin_port = htons(my_port);
if ( (bind(sockid, (struct sockaddr *) &my_addr,
     sizeof(my_addr)) < 0) )
{
  perror("bind() failed ");
  exit(3);
}
```

```
/* -- Initialize server side socket address -- */
server_sock_size = sizeof(server_addr);
memset((void *) &server_addr, (char) 0, server_sock_size);
server_addr.sin_family = AF_INET;
server_addr.sin_addr.s_addr = inet_addr(server_ip);
server_addr.sin_port = htons(server_port);

 sa.sa_handler = endme;
 sa.sa_flags = 0;
 sigemptyset(&sa.sa_mask);

 if(sigaction(SIGINT, &sa, NULL))
 {
     perror("sigaction");
 }
 if(sigaction(SIGTERM, &sa, NULL))
 {
     perror("sigaction");
 }

 it = comedi_open("/dev/comedi0");
 if(it == NULL)
 {
     printf("Could not open comedi\n");
     exit(1);
 }
```

```
in_maxdata = comedi_get_maxdata(it, in_subdev, in_chan);
out_maxdata = comedi_get_maxdata(it, out_subdev, out_chan);

in_range_ptr = comedi_get_range(it, in_subdev, in_chan,
                                in_range);
out_range_ptr = comedi_get_range(it, out_subdev, out_chan,
                                out_range);

if (!(mytask = rt_task_init(mytask_name, 1, 0, 0))) {
    printf("CANNOT INIT main TASK \n");
    exit(1);
}
printf("MASTER INIT: name = %lu, address = %p.\n",
        mytask_name, mytask);

sem = rt_sem_init(10000, 0);
sock_sem = rt_sem_init(nam2num("SOCK"), 1);
//rt_set_oneshot_mode();
rt_set_periodic_mode();
start_rt_timer(nano2count(25000));

start_instant = rt_get_time();
printf("main: start_instant = %lld\n", start_instant);
if (pthread_create(&task[0], NULL, send_thread_fun,
    &start_instant))
```

```
{
    printf("ERROR IN CREATING send_thread\n");
    exit(1);
}

if (pthread_create(&task[1], NULL, recv_thread_fun,
    &start_instant))
{
    printf("ERROR IN CREATING recv_thread\n");
    exit(1);
}

for (i = 0; i < ntasks; i++)
{
    while (!rt_get_adr(taskname(i)))
    {
        rt_sleep(nano2count(20000000));
    }
}

for (i = 0; i < ntasks; i++)
{
    rt_send(rt_get_adr(taskname(i)), (unsigned int)sem);
}

printf("Start waiting for sem\n");
```

```
    while(endme_int == 0)
    {
        rt_sem_wait_timed(sem, nano2count(5000000000));
    }
    printf("Stop waiting for sem\n");

    for (i = 0; i < ntasks; i++)
    {
        while (rt_get_adr(taskname(i)))
        {
            rt_sleep(nano2count(20000000));
        }
    }

    rt_sem_delete(sem);
    rt_sem_delete(sock_sem);
    stop_rt_timer();
    comedi_close(it);
    rt_task_delete(mytask);
    printf("MASTER %lu %p ENDS\n", mytask_name, mytask);
    for (i = 0; i < ntasks; i++) {
        pthread_join(task[i], NULL);
    }
    return 0;
}
```

APPENDIX B

SERVER.C

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <string.h>
#include <asm/errno.h>
#include <sys/types.h>
#include <sys/user.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sched.h>
#include <comedilib.h>
#include <sys/socket.h>
 #include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <sys/ioctl.h>
#include <sys/time.h>
#include <errno.h>
#include <inttypes.h>
#include "defines.h"
```

```
#define KEEP_STATIC_INLINE
#include <rtai_lxrt_user.h>
 #include <rtai_lxrt.h>

RTIME time_stamp;
 double u0, u1e, u2e, u3e, u4e, u5e, u6e, u7e, u8e;

double y0, y_1, y_2, y_3, y_4, y_5, y_6, y_7;
 double u_1, u_2, y_hat_1, y_hat_2, y_hat_3;
 double y_hat_4, y_hat_5, y_hat_6, y_hat_7, y_hat_8;

double y_hat_desi = 0.0, v_hat_err = 0.0, k = 0.098, c = 0.0;
double v = 1.018, er0 = 0.0, er1 = 0.0, er2 = 0.0;
int i=0;

unsigned long mtsk_name;
RT_TASK *mtsk;
 struct sched_param mysched;

void terminate_normally(int signo)
{
   fflush(stdin);

   if(signo==SIGINT || signo==SIGTERM)
   {
```

```
      printf("Terminating the program normally\n");

      //make the process soft real time process
      rt_make_soft_real_time();

      printf("MASTER TASK YIELDS ITSELF\n");
      rt_task_yield();

      printf("MASTER TASK STOPS THE PERIODIC TIMER\n");
      stop_rt_timer();

      printf("MASTER TASK DELETES ITSELF\n");
      rt_task_delete(mtsk);

      printf("END MASTER TASK\n");
   }

   exit(0);
}

main(int argc, char *argv[]) {
   int sockid, nread, addrlen;
   struct sockaddr_in my_addr, client_addr;
   int nw, nr;
   int send_buffer_size, recv_buffer_size;
   unsigned short server_port = 0;
```

```
struct send_data *send_buffer = NULL;
struct recv_data *recv_buffer = NULL;

RTIME start_time = 0;
RTIME end_time = 0;
RTIME actual_period = 0;
RTIME difference = 0;
size_t iRet = 0;
int esti_count = 0;
double vhaterr_prev[5] = {0.0, 0.0, 0.0, 0.0, 0.0};
int j=0;

//signal handling
struct sigaction sa;

//Initialize the signal handling structure
sa.sa_handler = terminate_normally;
sa.sa_flags = 0;
sigemptyset(&sa.sa_mask);

if(sigaction(SIGINT, &sa, NULL))
{
   perror("sigaction");
}
if(sigaction(SIGTERM, &sa, NULL))
{
```

```
   perror("sigaction");
}

fprintf(stderr, "creating socket\n");
if ( (sockid = socket(AF_INET, SOCK_DGRAM, 0)) < 0)
{
   perror("socket() failed ");
   fprintf(stderr, "%s: socket error: %d\n", argv[0], errno);
   exit(2);
}

fprintf(stderr, "binding my local socket\n");
//sscanf(argv[1], "%hu", &server_port);
server_port = 4444;

memset((void *) &my_addr, (char) 0, sizeof(my_addr));
my_addr.sin_family = AF_INET;
my_addr.sin_addr.s_addr = htons(INADDR_ANY);
my_addr.sin_port = htons(server_port);

if ( (bind(sockid, (struct sockaddr *) &my_addr,
   sizeof(my_addr)) < 0) )
{
   perror("bind() failed ");
   fprintf(stderr, "bind() errno = %d\n", errno);
   exit(4);
```

```
}

recv_buffer_size = sizeof(struct recv_data);
if(( recv_buffer = (struct recv_data *)calloc(1,
   sizeof(struct recv_data))) ==NULL)
{
   fprintf(stderr, "cannot allocate memory for buffer!\n");
   exit(4);
}

send_buffer_size = sizeof(struct send_data);
if(( send_buffer = (struct send_data *)calloc(1,
   sizeof(struct send_data))) ==NULL)
{
   fprintf(stderr, "cannot allocate memory for buffer!\n");
   exit(4);
}

addrlen = sizeof(client_addr);

fprintf(stderr, "%s: starting blocking message read\n", argv[0]);

mysched.sched_priority = 99;

if( sched_setscheduler( 0, SCHED_FIFO, &mysched ) == -1 ) {
   puts(" ERROR IN SETTING THE SCHEDULER UP");
```

```
   perror( "errno" );
   exit( 0 );
}

mlockall(MCL_CURRENT | MCL_FUTURE);

mtsk_name = nam2num("MTSK");
if (!(mtsk = rt_task_init(mtsk_name, 0, 0, 0))) {
   printf("CANNOT INIT MASTER TASK\n");
   exit(1);
}

start_time = rt_get_cpu_time_ns();
printf("main: start_time = %lld\n", start_time);

printf("MASTER TASK STARTS THE ONESHOT TIMER\n");
//rt_set_oneshot_mode();
actual_period = start_rt_timer(nano2count(25000));

printf("actual_period = %lld\n", actual_period);

//make the process a hard real time process
//rt_make_hard_real_time();

printf("MASTER TASK MAKES ITSELF PERIODIC \n");
rt_task_make_periodic(mtsk, rt_get_time()+ nano2count(3000000),
```

```
   nano2count(3000000));

while( 1 )
{

   nr = recvfrom(sockid, (void *)recv_buffer, recv_buffer_size,
                 0, (struct sockaddr *) &client_addr, &addrlen);
   if( nr <= -1 )
   {
      fprintf(stderr, "recvfrom() errno = %d\n", errno);
      exit(10);
   }

   start_time = rt_get_cpu_time_ns();

   y0 = recv_buffer->y0;
   y_1 = recv_buffer->y_1;
   y_2 = recv_buffer->y_2;
   y_3 = recv_buffer->y_3;
   y_4 = recv_buffer->y_4;
   y_5 = recv_buffer->y_5;
   y_6 = recv_buffer->y_6;
   y_7 = recv_buffer->y_7;
   u_1 = recv_buffer->u_1;
   u_2 = recv_buffer->u_2;
```

```
er0 = (y_hat_desi - (-0.0004653*y0+0.002525))*k;
er1 = (y_hat_desi - (-0.0004653*y_1+0.002525))*k;
er2 = (y_hat_desi - (-0.0004653*y_2+0.002525))*k;
u0 = ((0.782*(u_1-v)) + (0.13*(u_2-v)) - (41500.0*er0) +
      (41500.0*1.754*er1) - (41500.0*0.769*er2)) + v;
  send_buffer->u0 = u0;

y_hat_1 = 0.8122*y0 - 0.3479*y_1 - 0.0294*y_2 + 0.4605*y_3
          + 0.0742*y_4 + 0.1042*y_5 + 0.1117*y_6 - 0.3561*y_7;
er0 = (y_hat_desi - (-0.0004653*y_hat_1+0.002525))*k;
er1 = (y_hat_desi - (-0.0004653*y0+0.002525))*k;
er2 = (y_hat_desi - (-0.0004653*y_1+0.002525))*k;
u1e = ((0.782*(u0-v)) + (0.13*(u_1-v)) - (41500.0*er0)
      + (41500.0*1.754*er1) - (41500.0*0.769*er2)) + v;
  send_buffer->u1e = u1e;

y_hat_2 = 0.3117*y0 - 0.3119*y_1 + 0.4366*y_2 + 0.4482*y_3
          + 0.1645*y_4 + 0.1964*y_5 - 0.2653*y_6 - 0.2892*y_7;
er0 = (y_hat_desi - (-0.0004653*y_hat_2+0.002525))*k;
er1 = (y_hat_desi - (-0.0004653*y_hat_1+0.002525))*k;
er2 = (y_hat_desi - (-0.0004653*y0+0.002525))*k;
u2e = ((0.782*(u1e-v)) + (0.13*(u0-v)) - (41500.0*er0)
      + (41500.0*1.754*er1) - (41500.0*0.769*er2)) + v;
  send_buffer->u2e = u2e;

y_hat_3 = -0.0587*y0 + 0.3281*y_1 + 0.4390*y_2 + 0.3080*y_3
```

```
          + 0.2195*y_4 - 0.2329*y_5 - 0.2544*y_6 - 0.1110*y_7;
er0 = (y_hat_desi - (-0.0004653*y_hat_3+0.002525))*k;
er1 = (y_hat_desi - (-0.0004653*y_hat_2+0.002525))*k;
er2 = (y_hat_desi - (-0.0004653*y_hat_1+0.002525))*k;
u3e = ((0.782*(u2e-v)) + (0.13*(u1e-v)) - (41500.0*er0)
      + (41500.0*1.754*er1) - (41500.0*0.769*er2)) + v;
  send_buffer->u3e = u3e;

y_hat_4 = 0.2804*y0 + 0.4594*y_1 + 0.3097*y_2 + 0.1925*y_3
          - 0.2372*y_4 - 0.2605*y_5 - 0.1176*y_6 + 0.0209*y_7;
er0 = (y_hat_desi - (-0.0004653*y_hat_4+0.002525))*k;
er1 = (y_hat_desi - (-0.0004653*y_hat_3+0.002525))*k;
er2 = (y_hat_desi - (-0.0004653*y_hat_2+0.002525))*k;
u4e = ((0.782*(u3e-v)) + (0.13*(u2e-v)) - (41500.0*er0)
      + (41500.0*1.754*er1) - (41500.0*0.769*er2)) + v;
  send_buffer->u4e = u4e;

y_hat_5 = 0.6872*y0 + 0.2122*y_1 + 0.1842*y_2 - 0.1081*y_3
          - 0.2397*y_4 - 0.0884*y_5 + 0.0523*y_6 - 0.0999*y_7;
er0 = (y_hat_desi - (-0.0004653*y_hat_5+0.002525))*k;
er1 = (y_hat_desi - (-0.0004653*y_hat_4+0.002525))*k;
er2 = (y_hat_desi - (-0.0004653*y_hat_3+0.002525))*k;
u5e = ((0.782*(u4e-v)) + (0.13*(u3e-v)) - (41500.0*er0)
      + (41500.0*1.754*er1) - (41500.0*0.769*er2)) + v;
  send_buffer->u5e = u5e;
```

```
y_hat_6 = 0.7703*y0 - 0.0548*y_1 - 0.1283*y_2 + 0.0767*y_3
          - 0.0374*y_4 + 0.1239*y_5 - 0.0231*y_6 - 0.2447*y_7;
er0 = (y_hat_desi - (-0.0004653*y_hat_6+0.002525))*k;
er1 = (y_hat_desi - (-0.0004653*y_hat_5+0.002525))*k;
er2 = (y_hat_desi - (-0.0004653*y_hat_4+0.002525))*k;
u6e = ((0.782*(u5e-v)) + (0.13*(u4e-v)) - (41500.0*er0)
      + (41500.0*1.754*er1) - (41500.0*0.769*er2)) + v;
  send_buffer->u6e = u6e;

y_hat_7 = 0.5708*y0 - 0.3963*y_1 + 0.0541*y_2 + 0.3173*y_3
          + 0.1810*y_4 + 0.0572*y_5 - 0.1586*y_6 - 0.2743*y_7;
er0 = (y_hat_desi - (-0.0004653*y_hat_7+0.002525))*k;
er1 = (y_hat_desi - (-0.0004653*y_hat_6+0.002525))*k;
er2 = (y_hat_desi - (-0.0004653*y_hat_5+0.002525))*k;
u7e = ((0.782*(u6e-v)) + (0.13*(u5e-v)) - (41500.0*er0)
      + (41500.0*1.754*er1) - (41500.0*0.769*er2)) + v;
  send_buffer->u7e = u7e;

y_hat_8 = 0.0673*y0 - 0.1445*y_1 + 0.3005*y_2 + 0.4438*y_3
          + 0.0995*y_4 - 0.0992*y_5 - 0.2105*y_6 - 0.2032*y_7;
er0 = (y_hat_desi - (-0.0004653*y_hat_8+0.002525))*k;
er1 = (y_hat_desi - (-0.0004653*y_hat_7+0.002525))*k;
er2 = (y_hat_desi - (-0.0004653*y_hat_6+0.002525))*k;
u8e = ((0.782*(u7e-v)) + (0.13*(u6e-v)) - (41500.0*er0)
      + (41500.0*1.754*er1) - (41500.0*0.769*er2)) + v;
  send_buffer->u8e = u8e;
```

```
    send_buffer->time_stamp = recv_buffer->time_stamp;

    nw = sendto(sockid, (const void *)send_buffer,
                send_buffer_size, 0, (struct sockaddr *)
                &client_addr, addrlen);
    if( nw <= -1 )
    {
       perror("sendto failed ");
       fprintf(stderr, "sendto() errno = %d \n", errno);
       exit(12);
    }
    end_time = rt_get_cpu_time_ns();
 }

 printf("MASTER TASK YIELDS ITSELF\n");
 rt_task_yield();
 printf("MASTER TASK STOPS THE PERIODIC TIMER\n");
 stop_rt_timer();
 printf("MASTER TASK DELETES ITSELF\n");
 rt_task_delete(mtsk);
 close(sockid);
 free(send_buffer);
 free(recv_buffer);
}
```

VITA

The author, Ajit Ambike was born on April 28, 1979 in Pusegaon, India. He received his Bachelor of Engineering degree in mechanical engineering from Government College of Engineering, Karad, India in May 2000. From Fall 2002 onwards he attended the Master of Science degree program in the Mechanical Engineering Department, Texas A&M University, College Station.

Permanent Address:

Ajit Ambike

9, Geetanjali, Shreeman Society,

Karvenagar, Pune, INDIA - 440008

Phone: (091) 20-25456939

Email: ajitambike@yahoo.com