DISTRIBUTED REAL-TIME CONTROL VIA THE INTERNET

A Thesis

by

ABHINAV SRIVASTAVA

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

May 2003

Major Subject: Mechanical Engineering

DISTRIBUTED REAL-TIME CONTROL VIA THE INTERNET

A Thesis

by

ABHINAV SRIVASTAVA

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)	Suhada Jayasuriya (Member)
Richard A. Volz (Member)	John Weese (Interim Head of Department)

May 2003

Major Subject: Mechanical Engineering

ABSTRACT

Distributed Real-Time Control via the Internet. (May 2003)
Abhinav Srivastava, B.E., Motilal Nehru Regional Engineering College,
Allahabad, India
Chair of Advisory Committee: Dr. Won-jong Kim

The objective of this research is to demonstrate experimentally the feasibility of using the Internet for a Distributed Control System (DCS). An algorithm has been designed and implemented to ensure stability of the system in the presence of upper bounded time-varying delays. A single actuator magnetic ball levitation system has been used as a test bed to validate the proposed algorithm. Experiments were performed to obtain the round-trip time delay between the host PC and the client PC under varying network loads and at different times. A digital real-time lead-lag controller was implemented for the magnetic levitation system. Upper bounds for the artificial and experimental round-trip time delay that can be accommodated in the control loop for the maglev system were estimated. The artificial time delay was based on various probabilistic distributions and was generated through MATLAB. To accommodate sporadic surges in time delays that are more than these upper bounds, a timeout algorithm with sensor data prediction was developed. Experiments were performed to validate the satisfactory performance of this algorithm in the presence of the bonded sporadic excessive time delays.

To my father and mother.

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my thesis advisor, Professor Won-jong Kim, for his support and fruitful discussions during my course of study at Texas A&M University. It was a great opportunity for me to conduct the experiments. His ability to solve basic engineering problems and practical knowledge in dealing with the networked control systems motivated and guided me throughout my research activities.

I would also like to extend my thanks to my graduate committee members Dr. Suhada Jayasuriya and Dr. Richard A. Volz for their interest and valuable guidance during the research.

My thanks are due to Mr. S. C. Paschall, II for developing the single-actuator ball levitation system that was used as a test bed for the experiments. My thanks are also due to Mr. A. Ambike for his help in conducting the experiments for supervisory control via the Internet. I would also like to thank Mr. A. Bhattacharya and Mr. P. P. Harihara along with others for their cooperation throughout my research.

I am thankful to my parents, Mr. Arvind Kumar Srivastava and Mrs. Sadhna Srivastava for their immeasurable support throughout my life. Without their love, patience and support, I would have never come this far.

TABLE OF CONTENTS

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

FIGURE Page

FIGURE Page

CHAPTER I

INTRODUCTION

1.1 Introduction

During the last two decades of advancement in factory automation, the modern industry have grown tremendously in the size. Now a large number of processes run at various levels of operation. These processes interact with resources and provide the desired output. To control these processes, computers are used for the implementation of controller algorithms. With increase in industry size, geographic separation between these processes started increasing and their control became increasingly difficult. For convenient and cost effective control of the processes, the use of central controllers increased. These central controllers, located at convenient locations interacted with the distributed sensors and actuators. This mode of control required the industries to integrate communication and various aspects of controls into different levels of operation. Thus, computers communicated with the geographically distributed sensors and actuators via a communication network. Control system architecture communicating with geographically separated sensors, actuators and controllers via a communication network is called a distributed real-time control system.

Distributed real-time control offers the ability to locate the controllers for various processes at convenient locations that are easily accessible. For example, in deep-sea exploration and in nuclear plants the controller can be placed in an easy to access office environment whereas the sensors and actuators can be placed at the required locations. In an event of malfunctioning of any control system, this architecture of control also allows the observer to take corrective measures almost instantaneously. However, the success of this control scheme depends on the ability of the communication network in transferring data with stringent timing constraint and reliability requirements.

This thesis follows the style and format of *IEEE Transactions on Automatic Control.*

1.2 The Internet

The Internet being a free means of communication has gained a wide popularity over the last few years. It has virtually brought the whole world to our desktop. We can now access almost anything just at a mouse click, which was otherwise impossible in our home or office.

The smallest unit on the Internet is the individual user. Each individual has a user id within the local organization, and all the computers within the organizations may be connected with Local Area Network (LAN). The LAN makes communication between the members of the same organization much easier. LANs are attached to special computers called routers. When we send a message anywhere outside our LAN, we send it to the router, which distributes the message to the rest of the Internet. Each LAN thus cooperates with other networks to direct the Internet traffic so that information can pass among them. The networks are connected in a variety of ways. Together, these networks and organizations make up the wired world of the Internet. A variety of leased lines connect regional and local networks. The leased lines that connect networks can be as simple as a single telephone line or as complex as a fiber-optic cable with microwave links and satellite transmission.

Many researchers have tried to use the Internet for receiving and transmitting information over large distances. It is now possible to interact with people and participate in the events that are geographically separated by large distances. With the advances in the technology the above features can now be experienced in near real-time.

1.3 Internet Protocols

A very important aspect of communication via the Internet is the Internet Protocol (IP). The Internet Protocol determines the format of the data, known as the datagram, and the exact method by which data will be transmitted via the Internet. Most widely used protocols used for the Internet communication are:

1. Transmission Control Protocol / Internet Protocol (TCP/IP)

2. User Datagram Protocol (UDP)

TCP was specifically designed to provide a reliable end-to-end byte streams over an unreliable internetwork [1]. It is used because the Internet is a packet-switched network. In a packet-switched network, there is no single, unbroken connection between sender and receiver. Instead, the information is broken into small packets, sent over many different routes at the same time, and then reassembled at the receiving end. Each machine supporting TCP has a TCP transport entity, either a user process or a part of the kernel that manages TCP streams and interfaces to the IP layer. A TCP entity accepts user data streams from local processes, breaks them up into pieces not exceeding 64K bytes, and sends each piece as a separate IP datagram [2- 3]. When IP datagrams containing TCP data arrive at a machine, they are given to the TCP entity, which reconstructs the original byte streams.

The TCP provides reliable transmission of data in an IP environment. Among the services TCP provides are stream data transfer, reliability, efficient flow control, full-duplex operation, and multiplexing. TCP offers reliability by providing connection-oriented and confirmation-based packet delivery through the Internet. TCP does this by sequencing bytes with a forwarding acknowledgment number that indicates to the destination the next byte the source expects to receive. Bytes not acknowledged within a specified time period are retransmitted. Due to this confirmation based nature there is no loss of data during communication via the Internet. If the protocol detects a loss of data in the communication channel, it requests retransmission.

Thus, TCP/IP is the most reliable protocol for communication via the Internet. However, it also introduces additional amount of time delay in the Internet. So it becomes unsuitable for processes that are time critical. For a congested network the chances of losing the data are more and thus TCP has to initiate retransmission of the data frequently. Let us assume a time critical process uses this congested network with TCP protocol for communication. Due to frequent retransmission by TCP protocol there will be many vacant samplings. These vacant samplings will be detrimental to the stability of the system.

On the other hand UDP/IP is neither connection-oriented nor is confirmation based. UDP provides a way for applications to send encapsulated raw IP datagrams without having to establish a connection. Many client-server applications that have one request and one response use UDP [1]. A UDP segment consists of an 8-byte header followed by the data. The two ports serve the same function as they do in TCP. In other words the ports identify the end points within the source and destination machines. Thus, UDP is a connectionless protocol for applications that do not want TCP's sequencing of flow control. This makes UDP much faster compared to TCP/IP but at the same time is not a reliable protocol for communication via the Internet for data critical systems.

1.4 Real-Time Control via the Internet

If we can use the present form of the Internet as a communication medium for distributed real-time control, we can greatly enhance the advantages of this control architecture. Integration of the Internet will reduce the cost and the time involved in setting up the distributed real-time control system in large industries. Software applications can be used extensively to remotely monitor and control of the processes via the Internet. The incorporation of the Internet in the process control industry will also give engineers the freedom to distribute the control tasks, the sensors, and the actuators to optimal locations. Thus, Engineers will be able to monitor applications running in harsh environments that offer limited accessibility.

A real-time control system via the Internet may consist of various sensors, actuators and controllers connected with each other via the Internet. Controllers receive data from various sensors and send data to the actuators via the Internet. The two schemes of control system via the Internet, namely the supervisory control and the real-time feedback control offer the ability to process data from various sensors and actuators in real-time. With the Internet it will be possible to set up all the sensors and actuators on the same communication network. This will eliminate the need to setup a dedicated communication network for different closed-loop control processes in the industry.

For using the Internet as a communication medium for distributed real-time control, we have to adapt the system to stringent requirements for satisfactory

performance. Some of the key issues that need to be addressed for the satisfactory performance of a digital control implemented via the Internet as a communication medium include:

(1) The effect of the inevitable time delay associated with the Internet on the stability of the system. Due to the transmission of data over large distances between the sensor and the controller/actuator node, data packets undergo transmission delays. These transmission delays become critical for the stability of the system if they are more than the upper bound for the time delay.

(2) The unpredictability of the time delay associated with the Internet under different network protocols like TCP/IP and UDP. As discussed before, TCP/IP is the most widely used transport protocol over the Internet. It is the protocol used for applications that require reliable communication. TCP/IP is a confirmation-based protocol. In other words, it transmits the data and waits for the acknowledgement of the data receipt from the client before the next data are sent. This introduces more time delay in data transmission using TCP/IP. On the other hand, UDP is not a confirmation-based protocol. Thus, it eliminates the overhead of retransmission time. Since UDP is not confirmation based, data might get lost during the transmission. This fact affects the stability of data critical applications.

(3) Identification of fault when there is data loss in the system. During data transmission through the Internet, it is possible for the data to be lost. This data loss might be because either sensor is defective or the time delay is far more than the sampling period of the system. This may cause the Internet to lose the data. Thus, it will be difficult for the engineers to identify the exact reason for this data loss.

(4) The possibility of developing and implementing a novel control algorithm that can ensure satisfactory performance of the distributed real-time control via the Internet. Existing control algorithms will not be able to perform as desired in the presence of sporadic surge in the time delay that are more than the upper bound of the time delays that can be accommodated by the system.

1.5 Goal of the Thesis

In view of the above issues involved in implementation of distributed real-time control system via the Internet certain key areas have been identified that will be addressed in this thesis. Following are the goals of the thesis.

(1) Establishment of supervisory control of the test bed via the Internet. With this control scheme it will be possible to access the experimental test bed from anywhere with in the Texas A&M University campus.
(2) Determination of the stability regions for the controller with various sampling frequencies and time varying delay. These time delay data were both modeled with various probabilistic models and also as the results obtained from the experiments conducted over our LAN.
(3) Development of an algorithm that can guarantee stability of the test bed in the presence of time-varying delay with sporadic surge in the time delay. Thus, efforts will be made to stabilize the system for time delays that are little more than the upper bound of time delay for that system. The upper bound for the time-varying delay is the region in which the system stability is not compromised.

1.6 Thesis Overview

Chapter I deals with distributed real-time control via different communication medium. An introduction to a relatively new area of distributed real-time control via the Internet is also given. Various issues and advantages related to this field are discussed.

Chapter II explains the previous work by other researchers in the area of traditional distributed real-time control via various dedicated communication networks. It also gives a detailed study of the existing experimental setup used as test bed.

Chapter III gives a detailed description of the software architecture that I developed to achieve desired distributed real-time control via the Internet. It also explains various modes of distributed real-time control via the Internet. Various aspects of both

the modes are looked upon and schemes to achieve these kinds of architecture are discussed.

Chapter IV introduces the various mathematical models of the Internet that have been used to model the data traffic via the Internet. Analysis of the performance of the lead-lag controller for feedback control via the Internet has also been presented. To perform this analysis artificial time delay was introduced in the control code. Round-trip time delay measurements were made between PCs connected to same LAN and between PCs connected to different LANs.

Chapter V explains the methodology used for the development of the lead-lag controller designed directly in digital domain that runs at a sampling frequency of 333.3 Hz. It also deals with the development of the timeout algorithm with sensor data prediction that can ensure the stability of the test bed for the sporadic surges in the time delay.

Chapter VI concludes this thesis with the various experimental and emulation results obtained to test the algorithm developed in Chapter V. It also suggests ongoing work that is being carried out in the area of distributed real-time control via the Internet.

CHAPTER II

PRIOR WORK AND THE EXPERIMENTAL SETUP

2.1 Introduction

The main focus of this chapter is the study of previous work done by other researchers in the area of distributed real-time control using communication networks. Researchers have been working to make the distributed real-time control perform satisfactorily even in the presence of time-varying delay in the communication network. Most of the work has been developed for communication networks dedicated for distributed real-time control.

A detailed study of the single actuator magnetic ball levitation system to be used as a test bed is presented. Stephen C. Paschall, II developed the magnetic levitation (maglev) system and its preliminary control as his senior honors thesis [4].

2.2 Related Work

Researchers have been working in the feedback control via communication networks for long time. It is in the last decade, with the advancement in industry automation, research started in distributed real-time control via commercial communication networks. Research has mainly been focusing on the stability of the system in the presence of time-delay present in the communication channel.

Towards the end of the last decade, the Internet started to revolutionize the way of communication. New applications of the Internet in the areas of robotics, manufacturing, education, and so forth are currently hot topics for research. Research is also being carried out in the area of supervisory control of the telerobots.

Mercury project, developed by Goldberg, et al. was the first successful experiment in this field [5]. Their experiments allowed the general public to excavate objects from a sandbox, and thus demonstrated the feasibility of the use of the Internet for

teleoperation. Since then, other researchers have also conducted more experiments in the field of supervisory control of telerobots via the Internet. In September 1994, an industrial robot was remotely operated and programmed by remote users through the World Wide Web [6]. It demonstrated remote operation of complex machinery using minimal bandwidth data transmission. This opened up the door for a wide spectrum of applications of Internet-based telerobotics.

Backes, et al. used Web interface for telescience (WITS) an Internet-based tool for the Mars Pathfinder mission [7]. It enabled the scientists to collaboratively participate in planetary landing and rover missions. Along with the usual client/server architecture for the operation of the real spacecraft, it had a public system which the public could use for planning and simulating their own missions.

In engineering education, true distance learning was made possible only after the advent of the Internet. Laboratory courses, which are a vital part of engineering education, were considered impractical for distance learning. One such system has been successfully tested and applied in remote robotic education system in China [8]. The goal of the virtual laboratory was to create an experimental environment, suitable for remote robotic teleoperation. The system was also developed using the client server architecture. It allowed the users to conduct training exercises and perform experiments involving supervisory control.

Force feedback is another important topic in the area of teleoprations. In force feedback the user is provided with a feedback of the resistance to the task performed by the telerobot assigned by the user. With the help of force feedback, the task can be completed with much greater accuracy and speed compared to simple teleoperation. With the use of force feedback time delay is introduced that affects the stability of the system severely [9].

Munir showed that with the use of a predictor or an observer in the communication channel performance of the teleoperation system is greatly enhanced [10]. This is achieved in the presence of significant time delay and also variable time delays present over the Internet. The experimental set-up in the experiments of Munir, consisted of a two degree-of-freedom manipulator and a remote device that was a simulation of an identical manipulator. The two manipulators were connected via the

Internet connection between Atlanta, Georgia, USA and Tokyo, Japan. The problem of time delay present in the Internet for teleoperation was dealt with two different schemes. The first scheme used the wave variable approach and another approach used a time-forward observer. He concluded that the wave variable approach did not provide adequate performance results, but stability was guaranteed based on passivity for any amount of time delay.

The Internet also found its applications in the manufacturing industry. As we discussed before, manufacturing is no longer confined in a small region. With the Internet it is now possible to acquire data from the processes running on the shop floor at any time and take corrective measures if need be. Cybercut developed at the University of California at Berkeley focuses on rapid prototyping with a 3-axis computer numerical control (CNC) Machining via the Internet [11]. To provide CAD (Computer aided design) designers with the ability to perform real time manipulation of the part in progress, Cybercut uses Java's client side processing. This allows the designers to obtain structurally usable components with rapid turnaround times. Internet-based manufacturing was also researched by Renton, et al [12]. Center for Online Manufacturing Optimization (COMO) integrated advanced process simulation software, remote machine monitoring system and Internet technologies to realize virtual manufacturing. This was used to define, optimize, and execute a 3-axis milling operation.

2.2.1 Feedback Control via Communication Networks

Studies have been done to ensure the stability of the system in the presence of time-varying delay in the communication network. Several approaches have been suggested to deal with the issue.

Luck, et al. suggested a methodology to convert time-varying system into a time-invariant system with the use of buffers [13]. These buffers were introduced at the controller and actuator nodes. The proposed use of the buffers has been shown in the Fig.1.

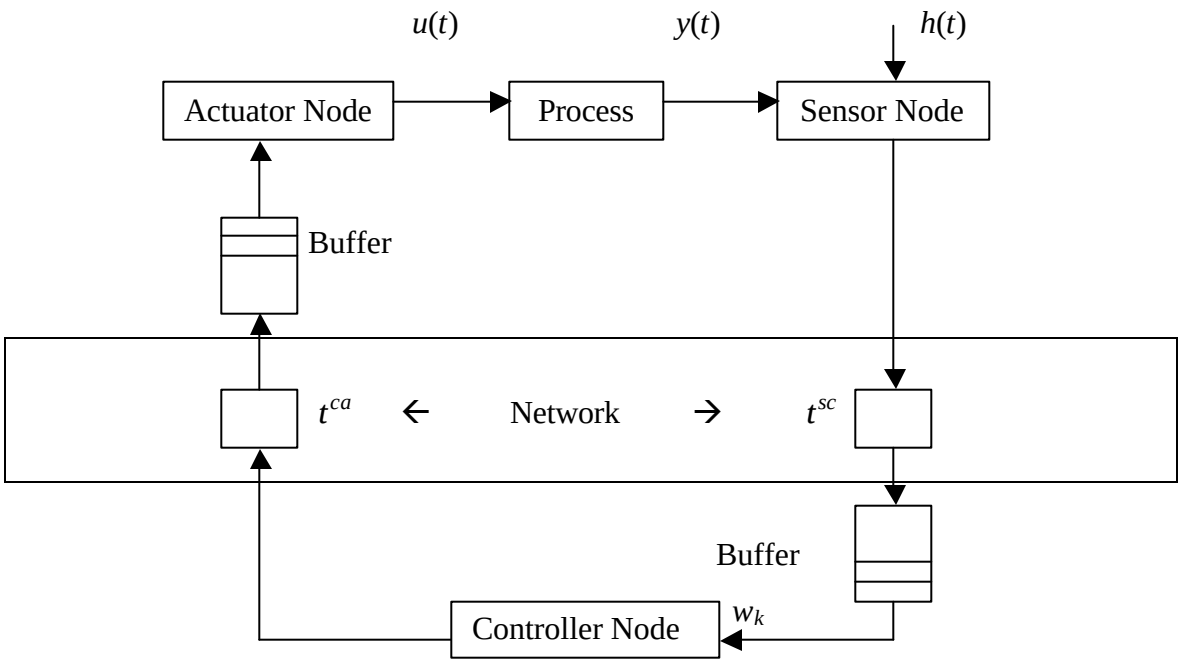

Fig.1. Schematic representation of use of buffers in control system via communication medium.

The nodes and the buffers were all time-driven and were synchronized in time. The buffers were used to store the data packets traveling through the communication channel. The buffers were made longer than the worst-case time delay present in the communication network. This allowed the data packets to be available at the controller and the actuator after fixed amounts of time making the system time invariant even in the presence of time-varying delay. Thus, the process was written as follows

$$x_{k+1} = Ax_k + Bu_{k-\Delta_1} \tag{2-1}$$

$$y_k = Cx_k \tag{2-2}$$

Where, *A*, *B*, *C* are system matrices, k denotes the particular k^{th} time stamp and $?_1$ is the length in samples of the buffer at actuator node. The problem was then formulated as a standard time-invariant sampled data system.

Liou, et al. proposed a time-driven sensor, a time-driven controller and an event-driven actuator [14]. The controller and the sensor have a time skew in their operation and the control value is calculated without the knowledge of new sensor signal. Due to an

event-driven actuator control signal is applied to the digital to analog (D/A) channel as soon as it arrives at the actuator node. These delayed signals were introduced in the discrete time augmented plant model. Thus, the augmented plant model was given by

$$x_{k+1} = A_k x_k + B_k u_k \tag{2-3}$$

where, A_k, B_k are stochastic matrices due to random communication delays. The augmented plant model was used to solve for (Linear Quadratic) LQ-optimal controller. They have also discussed construction of state estimator for the case when all the states are not measured.

Ray et al. in 1994 extended the concept of time-driven actuator and stochastic state estimator [15]. The estimator was used to estimate the sates that were unobservable. The state estimator is designed to minimize the variance of the state prediction error. The combination of the LQ-controller and the minimum variance estimator was introduced as DCLQG (Delay Compensated Linear Quadratic Gaussian) controller.

Krotolica et al. studied the random communication time delays [16]. The time delay from the sensor to the controller and from the controller to the actuator was modeled as being obtained from a Markov chain. All the nodes were assumed to be clock driven. Conditions necessary for the zero-state mean-square exponential stability were obtained.

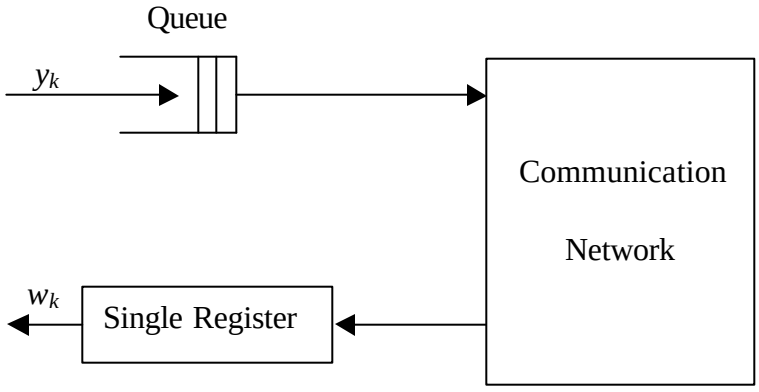

Fig.2. Block diagram of portion between sensor node and controller node.

Chan et al. studied the Ford SCP Multiplex Network hardware [17]. In this study a queue was placed at the sensor node and registers appended the controller node. The

data packets sent from the sensor were appended with a time stamp and the queue size at that instant. With this method the controller node could reduce the uncertainty about the order of the sensor signal that was being used to generate the control input. The schematic arrangement proposed by Chan et al. is depicted in Fig.2.

Nilsson addressed the problem of analysis and design of control systems when the communication delays vary in a random fashion [18]. The two delay models for the communication network were developed. The first model consisted of random delays that were independent from transfer to transfer. In the second model an underlying Markov chain was used to generate the probability distributions of the time delays.

For the case of independent delays a linear controller was used to show that closed-loop control could be written as

$$z_{k+1} = \Phi(t_k) z_k + \Gamma(t_k) e_k \tag{2-4}$$

where, the stochastic properties of t_k depend on the network model. The LQG-optimal controller was derived for the network where the time delays were independent from sample to sample. The controller thus derived used the knowledge of old time delays obtained by comparing the timestamps of the signals.

When the probability distributions of the delays are generated from a Markov chain the closed-loop system can be written as

$$z_{k+1} = \Phi(t_k, r_k) z_k + \Gamma(t_k, r_k) e_k \tag{2-5}$$

where t_k is the network delay and r_k is the state in Markov chain. A Lyapunov recursion for evaluation of covariance of signal in the system was found. A stability criterion for mean square stability followed directly from the Lyapunov recursion. The LQG-problem was solved by splitting it into a state feedback problem and an optimal state estimation problem.

2.2 Test Bed and Its Control

The single–actuator magnetic ball levitation system was developed by Stephen C. Paschall, II as his senior honors project [4]. The aim of this maglev system is to levitate a steel ball at a predetermined steady-state equilibrium position with an electromagnet. The

electromagnet is used to produce magnetic forces using current passing through the electromagnetic coil to support the ball's weight. Fig.3 depicts the maglev system.

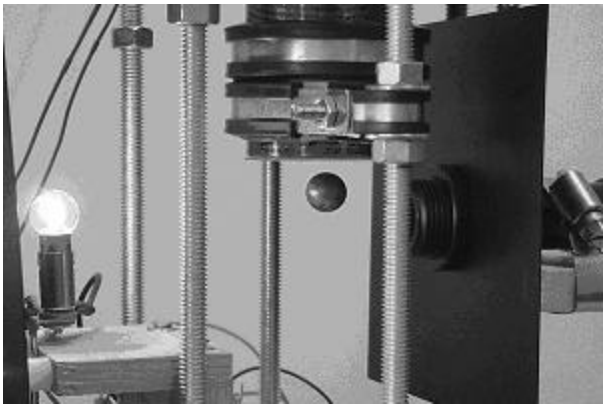

Fig.3. Single-actuator magnetic ball levitation system [4].

The system is subdivided into two main subsystems namely the force actuation subsystem and the position sensing subsystem. The force actuation subsystem consists of an electromagnet coil, a PWM (pulse-width modulation) amplifier and a 24-V DC power supply that supplies the power to the PWM amplifier.

The position sensor subsystem consists of a CdS photocell, an incandescent light source, and a 15-V DC power supply that supplies power to the photocell based sensor and the incandescent light bulb. The photocell in conjunction with the incandescent light bulb is used for sensing the position of the levitated ball. The vertical movement of the steel ball is measured by the sensor arrangement through gradual shield and exposure of photocell to the light from the incandescent light bulb. The tube is completely closed to external light except for a vertical slit so that ambient light does not interfere with the light from the light bulb. Thus, the amount of light exposed to the photocell is a function of the ball position in front of the photocell. This variable light exposure of the photocell results in the change of the electrical resistance of the photocell. The voltage across a resistance placed in series with the CdS photocell is used as the sensor signal for calculation of the control input by the controller board.

2.2.1 Modeling of the Test bed

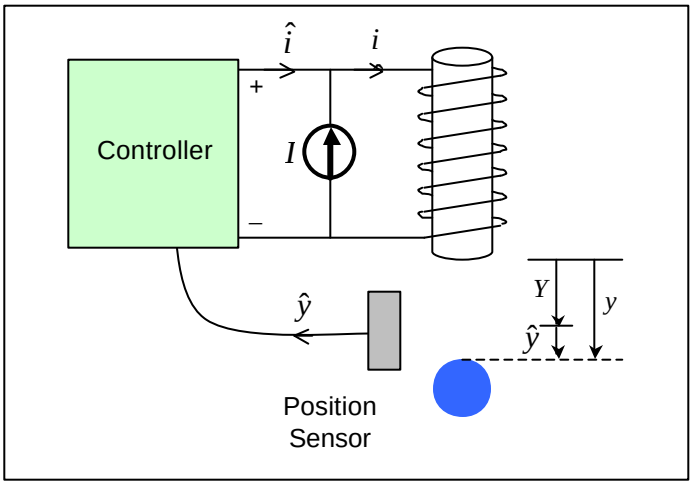

Fig.4. Basic setup for the magnetic levitation system.

Table I. Magnetic levitation theory nomenclature.

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 basic setup for the magnetic levitation system is depicted in Fig. 4. The setup consists of an electromagnetic actuator that creates magnetic field when an electric

current is passed through its wires. The position sensor detects the vertical position of the ball and passes the information to the controller board. The controller then adjusts the current to the electromagnet actuator based on the position of the ball.

The first step in creating the magnetic levitation system is to model the maglev plant. The following modeling process is by Woodson & Melcher [19]. For the test bed, the plant model yields the position of levitated ball as a function of the input current. The nomenclature used for the modeling is given in Table I. The electromagnetic force required to levitate the ball is given by the derivative of the coenergy. The coenergy (*W'*) is defined as

$$W'(i,y) = \frac{1}{2} i^2 L(y) \tag{2-6}$$

where, *L(y)* is given by

$$L(y) = L_1 + \frac{L_0}{1+\frac{y}{a}} \tag{2-7}$$

The substitution of electromagnetic inductance and coenergy into the force equation yields a relation for the electromagnetic force in terms of current and displacement.

$$f^e(i,y) = \frac{\partial W'}{\partial y} = \frac{-L_0 i^2}{2a\left(1+\frac{y}{a}\right)^2} \tag{2-8}$$

The forces acting on the ball along with their directions have been shown in Fig.5.

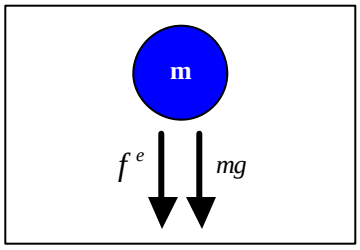

Fig.5. External forces on the levitated ball.

Thus the equation of motion is obtained as

$$m\ddot{y} = mg + f^e \tag{2-9}$$

From equations (2-8) and (2-9) at static equilibrium we obtain

$$mg = \frac{L_0 I^2}{2a\left(1+\frac{y}{a}\right)^2} \tag{2-10}$$

which implies, the steady state current is given by

$$I = \left(1+\frac{y}{a}\right)\sqrt{\frac{2mga}{L_0}} \tag{2-11}$$

With the non-linear equation of motion determined by equation (2-9), a Taylor series expansion is carried out to create a linearized equation of motion. Using the perturbation quantities for steady state position and current, we obtain the simplified open loop system model as

$$\ddot{\hat{y}} - \left(\frac{2g}{a+Y}\right)\hat{y} + \left(\frac{2g}{I}\right)\hat{i} = 0 \tag{2-12}$$

which on simplification yields the transfer function as

$$\frac{\hat{Y}(s)}{\hat{I}(s)} = \frac{-1}{As^2 - B}\left[\frac{\mathrm{m}}{\mathrm{A}}\right] \tag{2-13}$$

where,

$$\begin{aligned} A &= \frac{I}{2g}\left[\frac{\mathrm{As}^2}{\mathrm{m}}\right] \\ B &= \frac{I}{a+Y}\left[\frac{\mathrm{A}}{\mathrm{m}}\right] \end{aligned} \tag{2-14}$$

2.2.2 Modeling of the Plant

For the test bed the plant is the steel ball that is levitated using the electromagnetic coil. The steel ball used for the experiments is a 8.6 g ball with a 12.7 mm diameter. The time domain model of the plant is

$$F = m\ddot{y} \tag{2-15}$$

which after Laplace transform gives the transfer function as

$$\frac{Y(s)}{F(s)} = \frac{1}{ms^2} \tag{2-16}$$

2.2.3 Dynamics and Control of the Test Bed

After the modeling of the test bed and the plant a controller in continuos time domain was developed by Steve C Paschall, II [4]. The controller designed in the continuous time domain is

$$\frac{-33300s^2 - 2.564\times10^6 s - 1.632\times10^7}{s^2 + 700.7s + 490} \tag{2-17}$$

This controller maintained the following transient response requirements.

Settling Time (t_s) = 1s

Percentage Overshoot = 50%

The open-loop maglev system consisted of the power amplifier, electromagnet coil, and the steel ball. Analysis of the open-loop maglev system suggested it to be an unstable system. Thus, it was necessary to place the maglev system in a closed-loop control system. In the closed-loop system, the feedback from the position sensor in the form of voltage is used for the generation of control signal by the controller board. The control signal thus generated is continuously applied at the actuator thus making the maglev system stable at the equilibrium position.

For the purpose of implementing digital control algorithms, a DSP (digital signal processor) Controller Board model (DS1104) from dSPACE, Inc. is used. This controller board has a Texas Instruments' TMS320F240 DSP. One of the 16-bit analog-to-digital (A/D) converters of the controller board is used to read the optical sensor output. The voltage output signal from the 12 bit digital-to-analog (D/A) converter of the controller board is fed to the PWM amplifier (model 12A8K) manufactured by Advanced Motion Controls.

The continuous-time domain lead-lag controller was converted into the discrete domain using pole-zero mapping. Following is the implemented lead-lag controller with the sampling frequency of 2 kHz.

$$G_C(z) = 3.33\times10^4\left(\frac{z-0.9656}{z-0.7046}\right)\left(\frac{z-0.9965}{z-0.9996}\right) \tag{2-18}$$

CHAPTER III

MODES OF CONTROL VIA THE INTERNET

3.1 Introduction

In the last chapter, prior work done by other researchers in the area of distributed real-time control was discussed. Also the experimental test bed developed by Stephen C. Paschall II was described. In this chapter, the software architecture that has been used in the research is presented. Also various modes of control of the test bed via the Internet is presented. The two modes of control of the test bed via the Internet, namely supervisory control and feedback control are discussed in detail. The necessary architecture to achieve these modes is presented.

3.2 Software Architecture

The programs written in C and MATLAB form the skeleton of the software used for control algorithm implementation. Experiments were also conducted for the time delay emulation and simulation in the control loop. The codes for the implementation of the lead-lag controller were written in C programming language.

The compiler used for compilation and downloading of the control code on the DS1104 controller was provided by the dSPACE, Inc. The code test.c includes subroutines to initialize the analog to digital (A/D) and digital to analog (D/A) channels of DS1104 controller board. The test.c includes brtenv.h and io1104.h header files that are used for initializing the basic real-time environment and the input output for DS1104 board. The code test.c is included in the Appendix A. The skeleton for the code has been shown in Fig. 6.

For the generation of random time delays based on different probability distributions various commands of MATLAB were used. The commands of MATLAB

like poissrnd() and unidrnd() generated random numbers based on Poisson and Uniform distributions.

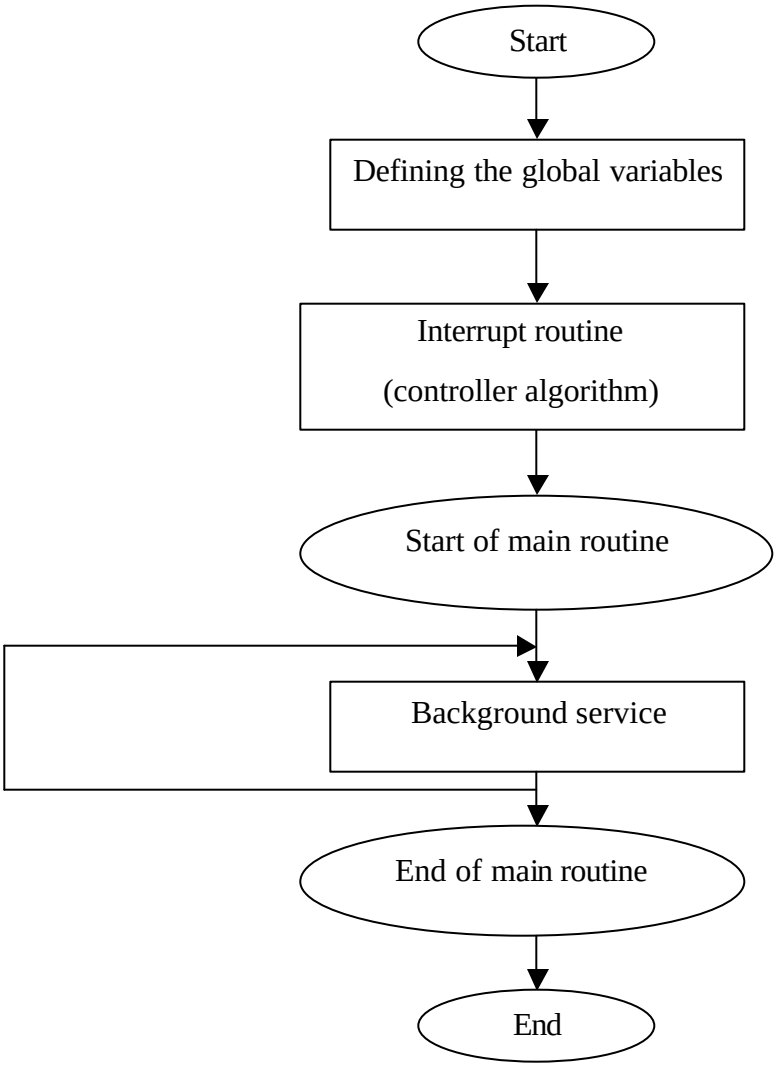

Fig.6. Software architecture for controller algorithm.

These random time delays were implemented in the control code for the simulation of time delay that the sensor data and control input will face while traveling through the communication network.

The communication between the PC and the controller board was achieved with the help of C code control_clib.c. The control_clib.c includes clib32.h along with other header files that helps in communicating between the PC and the controller board in real-time. This header file was provided by the dSPACE Inc.

For the implementation of the control via the Internet, Microsoft Visual Studio was used for development and compilation of the Common Gateway Interface (CGI) scripts. The CGI scripts are used for the communication between programs running on the host PC and the client requests coming via the Internet. These scripts are located on the host PC and are executed remotely by the client. This CGI script integrated with the finaltest.c program was used for the transfer of the commands via the Internet between the server PC and the client PC. The finaltest.c code is included in Appendix B for reference. To write the CGI script in C, a CGIC library developed by Thomas Boutell was used [20]. Since the CGI script resides on the server PC, it is executed as soon as the data comes from the client PC using web browser. The web browser opens the HTML (hypertext markup language) code test1.html for the user to input commands. Test1.html is listed in Appendix C. This methodology for communication via the Internet is not efficient for continuous real-time data transfer.

For the continuous transfer of data between the server PC and the client PC codes were written for the programming of sockets. Sockets are a way to communicate with other programs using standard file descriptors [21]. Socket programming can also be used to establish communication between the server PC and the client PC. The sockets present in host PC and client PC interact with each other using various protocols and thus provide continuous data transmission in real-time.

With the help of socket programming, the communication of the data from the client computer to the controller board was achieved. The communication through the sockets ensured transfer of the sensor data and the control input. The codes client.c and server.c were used to run on the client PC and the server PC. The server PC was connected to the maglev system and the client PC consisted of the controller board. The client.c was communicating with the controller board and was at the same time transferring the data using TCP/IP to the server.c on server PC. The server.c and client.c codes are presented in Appendices D and E, respectively.

3.3 Supervisory Control via the Internet

Supervisory control is a technique of remotely monitoring and enabling an outside computer to connect to an experiment and controlling it from a distance. With the supervisory control of the processes, the processes themselves adapt to changing circumstances and take corrective measures to retain the system stability. Such a methodology of control benefits engineers who need to monitor applications running in harsh conditions and offer limited access.

The advances in the communication technology have been applied to supervisory control of the processes in the process control industry. The supervisory control via a communication network is based on the client/server architecture. In this mode of control only the user-defined commands are sent via the communication network.

The control algorithm for the process runs on the controller board that is situated in the PC that controls the test bed. From the Fig. 7 it can be seen that the feedback loop is closed locally on the host side. The loop does not include time delay in the transmission of data between the sensor and the controller and between the controller and the actuator. The sensor data are sent to the controller and the controller generates the required control input after a finite computational time, $?_{c.}$ This computational time is deterministic and constant in nature and is usually less than the sampling period for the closed-loop system. Thus, the controller has the latest sensor data available for the computation of the control input to be applied to the actuator.

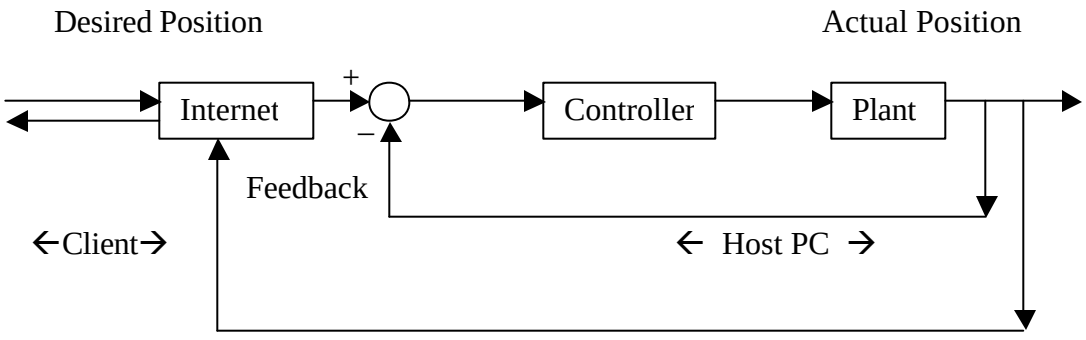

Fig.7. Block diagram for supervisory control via the Internet.

Supervisory control has been widely applied in the areas of telerobotics, under-sea environments hazardous environments and space programs. In these situations the controller is located along with the process to be controlled and the feedback loop is closed locally. Engineers can monitor these processes remotely and get feedback from the process in real time via the communication network. In an event of emergency where the controller is not performing efficiently or satisfactorily the engineers can take corrective measures. These corrective commands are transmitted via the communication network to the controller and are implemented in the next sampling cycle of the controller. The performance of the system due to these changes is sent through the communication channel from the process to the engineer in the same sampling cycle.

With the widespread popularity of the Internet and its ease of communication real-time supervisory control of the processes can be implemented via the Internet. In supervisory control of the process via the Internet, the host PC connected with the test bed runs a web server that can serve web pages related to the test bed. These Web pages written in HTML have the necessary arrangements for the user to input their commands and get the feedback from the test bed. Once the commands have been submitted to the Web page, the Web page transfers these commands via the Internet to the Web server running on the host PC. The transmission of these commands usually takes place by the TCP/IP protocol. The Web server running on the host PC passes these commands to the CGI (Common Gateway Interface) script present in the CGI bin of the Web server. The CGI script is executed in the CGI bin as soon as the request from the client is received. This CGI script is used to convert the encrypted data from the client into a format that is understandable to the host PC.

These commands from the CGI script are then transferred to the controller board in real time using C programs. The results of the performance of the test bed with user-defined commands are then sent via the Internet in the opposite direction using the same methodology.

For the implementation of supervisory control of the maglev system test bed, the test bed is connected to a Pentium IV PC that acts as host PC. The Pentium IV PC runs the Internet Information Services (IIS) 5.0 Web server on the Windows 2000 Professional operating system. The host PC interacts with the DS1104 controller board with the help

of C programs written using the CLIB header file clib32.h provided by dSPACE Inc. The header file gives the ability to transfer the commands from the PC to the controller board and from the controller board to the PC in real time. For the implementation of supervisory control of the maglev system via the Internet CGI script was integrated with the C code. The CGI script integrated with the C code control_clib.c provided the ability for directly capturing the incoming data from the user via the Internet and transferring it to the controller board in real time. The software architecture for the supervisory control of the maglev system via the Internet is depicted in the Fig. 8.

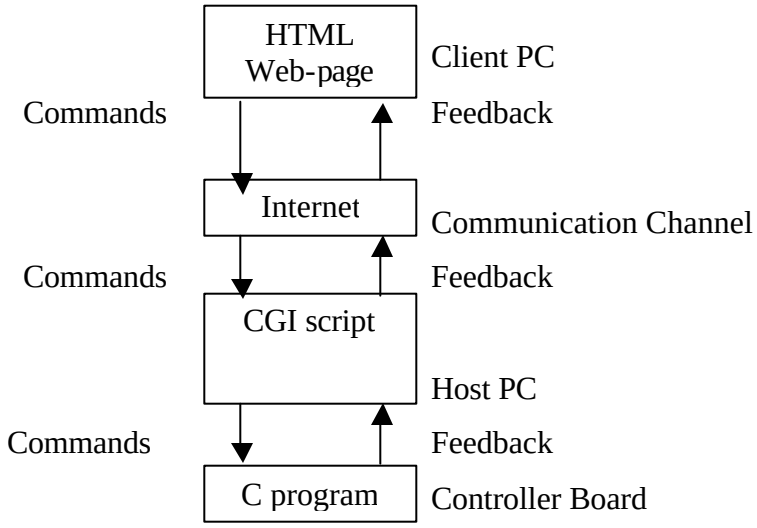

Fig.8. Software architecture for the supervisory control of the maglev system via the Internet.

In supervisory control of the maglev system, stability of the system was not affected. This was due to fact that no data from the sensor to the controller or from controller to the actuator traveled through a communication network. The sensor data collected at the k^{th} sampling period are applied to the controller at the $kT_s + \tau_c$ instant. Where τ_c, is the deterministic computational time taken by the controller board to generate control input using the sensor data. Thus the control input is applied to the actuator after every finite $kT_s + \tau_c$ time. In this mode of control $\tau_c < T_s$ where, T_s is the sampling period for the maglev system.

The communication network delay was introduced only in the communication path for the user inputs from the client PC to the host PC. With the supervisory control of the maglev system via the Internet the user will have a delayed response of the system to its input commands. In addition to this delayed response, the user inputs applied to the maglev system will also be delayed.

Thus, the success of the supervisory control of the maglev system via the Internet provides the freedom of accessing the maglev system from anywhere within Texas A&M University. The clients can now access the Web page of the maglev system by typing in the URL (uniform resource locator) of the maglev Web page in their Web browsers. A snapshot of the Web page for the maglev system has been shown in the Fig.9.

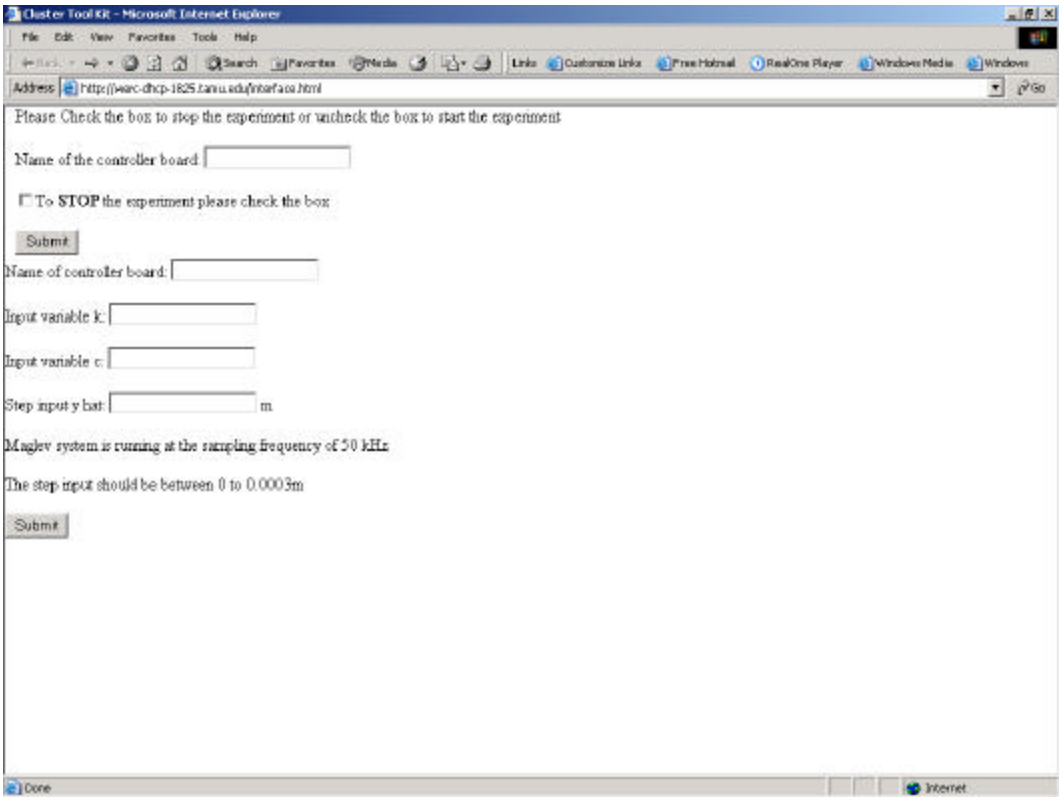

Fig.9. HTML Web page of the maglev system for the user interface.

This maglev test bed Web page opens in the Web browser. The Web page has the provisions for the users to change gain parameter k, the controller constant, c and the step size y. The transfer function used with this mode of control is given as:

$$Gc(z) = \frac{33300z^2 - 33300(e^{-100cT_s} + e^{-10cT_s}) + 33300e^{-110cT_s}}{z^2 - (e^{-1000cT_s} + e^{-cT_s})z + e^{-1001cT_s})} \qquad (3\text{-}1)$$

where, T_s is the sampling period of the controller and is 20 µs in this case.

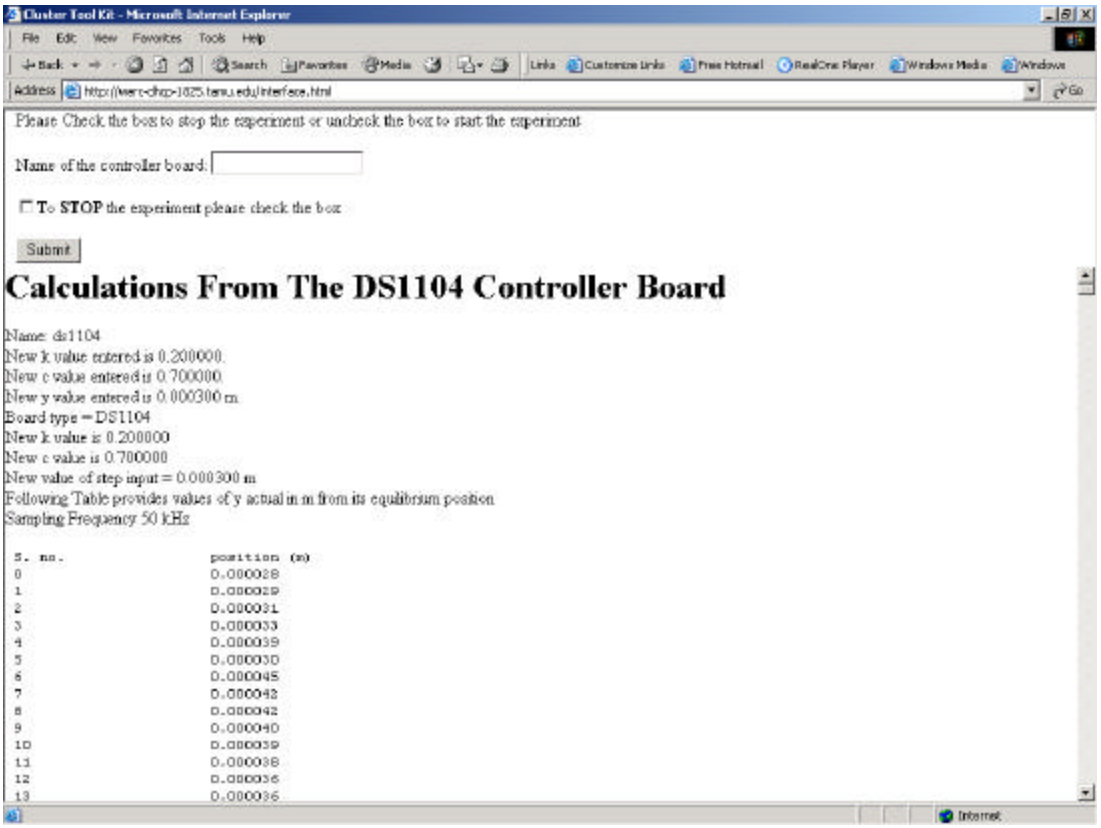

Fig.10. HTML output Web page of the maglev system.

This Web page also has the provision for stopping the system when the experiment has been completed via the Internet. By checking the appropriate box the users can stop the experiment remotely using their Web browser. The feedback of the maglev system to the user input is collected by the CGI program final.c presented in Appendix B and sent via the Internet to the client. The results are thus displayed on the client PC in their Web browsers in the form of numerical data. This numerical data can

be used to plot the desired graph off-line, on the client PC and the user can analyze the system performance for the inputs provided. A snapshot of the output Web page for the maglev system has been shown in the Fig.10. The plot obtained from the data in output Web page is shown in Fig.11.

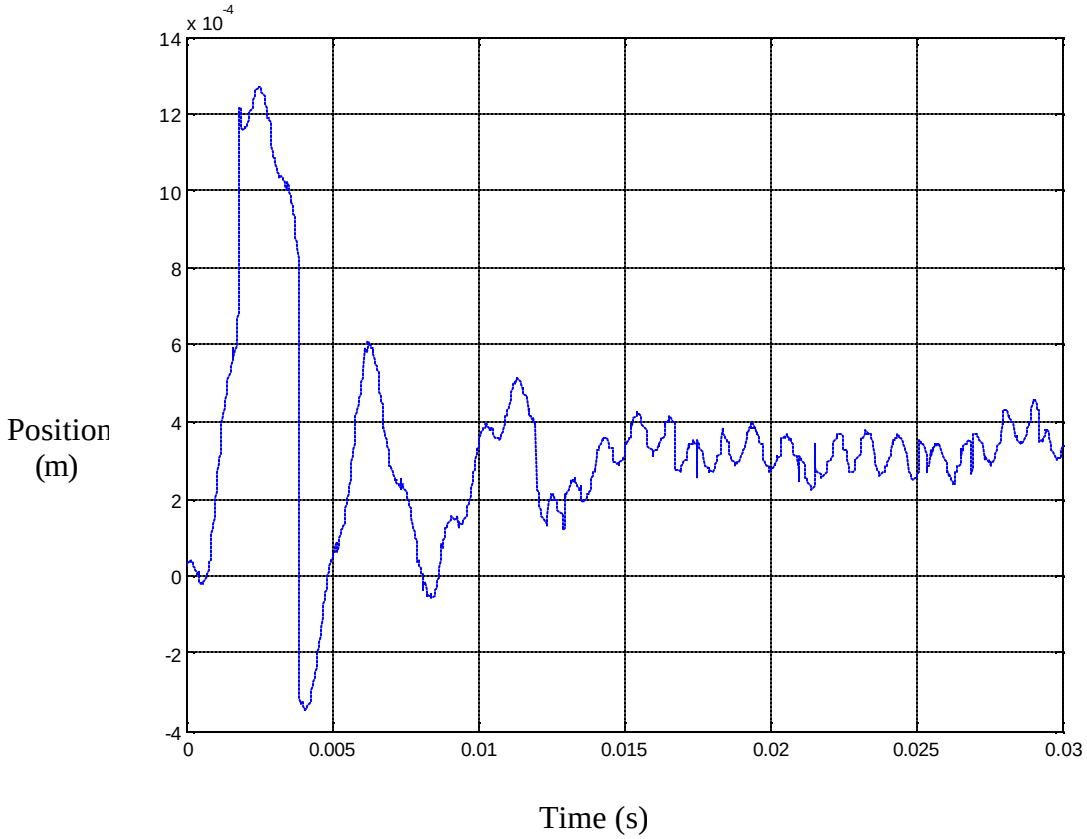

Fig.11. Maglev system response to a step input of 300 μm provided via the Internet.

3.4 Feedback Control via the Internet

Feedback control of processes has mostly been tested and implemented on dedicated communication networks. The defining feature of an NCS is that information (reference input, plant output, control input, etc.) is exchanged using a network among control system components (sensor, controller, actuator, etc.) [22]. Thus in NCS, real-time sensor data and control inputs are transmitted through a communication network, and these sensor and actuator nodes have to work closely to achieve the desired objective.

Feedback control via a communication network is based on client/server architecture. This mode of control allows the clients to run their own control algorithm for the processes as opposed to the supervisory mode of control. The clients connect to the test bed using programs running in the client PC that can communicate via a communication network using the sockets. These codes called socket programs aid in communicating between different PCs connected through a communication network that uses various protocols like TCP/IP and UDP.

With TCP/IP being a connection oriented and confirmation based protocol, the communication link between the host PC and the client PC remains active till one of the PCs terminates the session. This protocol ensures that the data being transferred between the two communicating PCs is transferred in a proper manner and there is no loss of data in the communication network. The actual sending and receiving of the data takes place between the sockets present in both the client and the server PC. Sockets are a combination of an IP address and a port. Each client has a unique port associated with it while a server can initiate multiple instances of sockets for a process. Thus, sockets act as the connection points between the communicating PCs. This strategy ensures that processes that take too much time in computation do not block other clients to connect to the server. These sockets collect the data from the process, and send them to the controller board or to the actuator node, with the help of socket program. The same socket on a client PC or on a server PC can be used for listening and sending data simultaneously. The schematic arrangement for the feedback control via the Internet has been shown in Fig. 12.

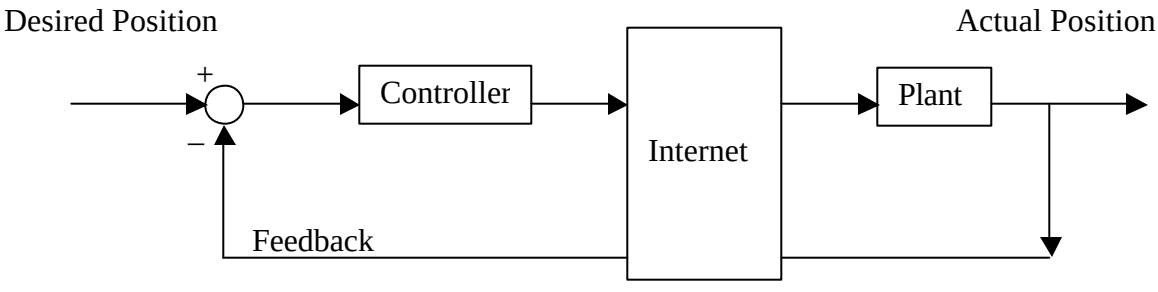

Fig.12. Block diagram for feedback control via the Internet.

The host PC connected to the test bed contains only the D/A (digital to analog) and A/D (analog to digital) channels. The D/A channel is used for the application of control inputs to the actuator whereas the A/D channel is used for collection of the data from the sensors. These channels operate once during each sampling period and thus control the process. The client PC has the controller board that collects the data from the sensor node of the host PC via the Internet. The Client PC then generates the control input using the sensor data and sends it to the D/A channel of the test bed via the Internet. Thus, the transfer of the data between the host PC and the client PC takes place continuously in real time via the Internet.

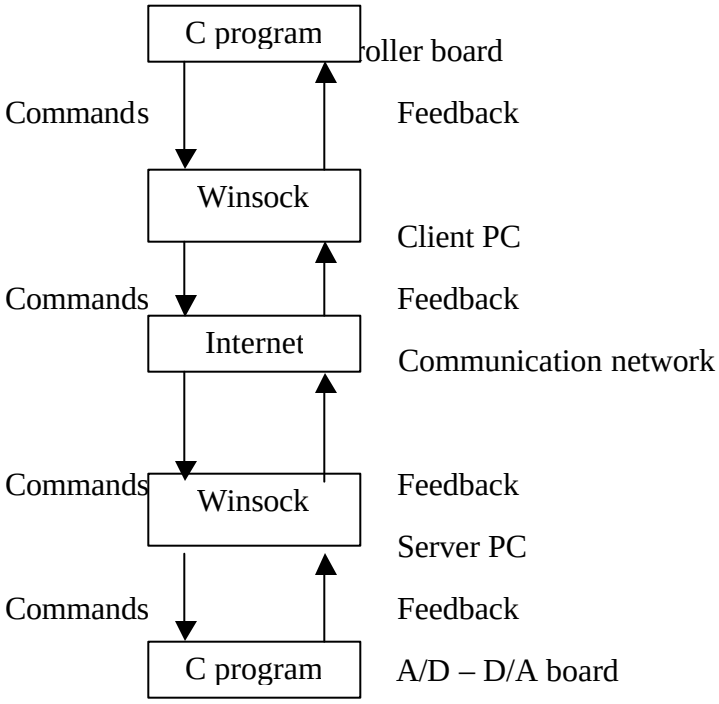

Fig.13. Software architecture for feedback control via the Internet.

For the implementation of feedback control of the maglev system, the test bed is connected to a Pentium IV PC that acts as the server PC. The server PC runs the server.c socket program on the Windows 2000 Professional operating system. The server.c code was written for handling data transfers between the sockets of the server PC and the

client PC. A Celeron PC running on Windows NT 4.0 operating system was fitted with the DS1104 controller board and acted as the client PC. The client.c code was developed for data transfer between the sockets of the client PC and the server PC. The server.c and client.c codes have been presented in Appendices D and E, respectively. The software architecture for the feedback control via the Internet is depicted in Fig. 13.

The server.c program was designed to collect the sensor data from the A/D channel of the test bed and sends it via the Internet using Windows Sockets (WinSocks). A WinSock augments the Berkeley socket implementation by adding Windows-specific extensions to support the message-driven nature of Windows operating system [21]. Till now the feedback control via the LAN has not been achieved. This might be due to the fact of improper time synchronization between the host PC and the DS1104 controller board. Also the sockets being used in Windows operating system might be buffering the data before sending it over the LAN and releasing it in a burst. Possible solutions to enable the feedback control via the LAN are:

1. Integration of the socket program along with the controller algorithm running on the controller board. This implies ability to communicate with the client PC directly using the DS1104, thus removing host PC altogether.
2. Proper usage of semaphores that can keep track of the time synchronization between the host PC and the DS1104 controller board.
3. Use of another operating system, such as UNIX or Linux, for the purpose of testing proper functioning of the sockets.

CHAPTER IV

ANALYSIS OF FEEDBACK CONTROL VIA THE INTERNET

In the previous chapter, two different modes of control namely the supervisory control and the feedback control via the Internet were introduced. Due to the presence of time delay in the communication channel the stability of the system is affected in feedback control via the Internet. In this chapter, an analysis of the performance of a simple lead-lag controller in the presence of time-varying delay is presented. An estimate of the time-varying delay is obtained that can be accommodated in the control without losing the system stability. In the later part of this chapter measurements of round-trip time delay between PCs connected to same LAN and between PCs connected to different LANs have been presented.

4.1 Introduction

In distributed real-time control via the Internet, a feedback control loop is closed via the Internet that multiplexes data from the sensor to the controller and from the controller to the actuator along with other data transfers. Due to sharing of the communication channel for the feedback loop with other data transfers, time-varying delays are introduced in the control loop. This time-varying delay in the communication channel degrades the performance of the control system and is potential cause of instability.

In the initial part of this chapter an upper bound for the time delay that does not affect the system stability is estimated. This upper bound has been established for various probabilistic distributions of delays. These delays were artificially generated and introduced in the actual control loop.

4.2 Modeling of the Data Traffic in the Internet

Modeling of the data traffic in the Internet is an important part of distributed real-time control via the Internet. These traffic models are employed in two fundamental ways: either as part of an analytical model, or to perform discrete-event simulation [23–24]. The simplest form of traffic consists of the arrival of data packets at the corresponding node. This process consists of arrival instants $T_1, T_2, T_3 \ldots T_{n\ldots}$ measured from the origin. By convention, the first time instant T_0 is assigned value of zero, i.e. $T_0 = 0$. This type of process is also known as a point process. The point process can equivalently be described as counting process and inter arrival time process.

A counting process is a continuous-time, non-negative, integer-valued stochastic process $\{N(t)\}$, where $N(t)$ is the number of data packet arrivals in $(0, t\ ]$ and is given by

$$N(t) = \max \{ n : T_n \leq t \} \tag{4.1}$$

An inter-arrival time process is a non-negative random sequence $\{A_n\}$ where A_n is the length of time interval separating the n^{th} arrival from the previous one and is given by

$$A_n = T_n - T_{n-1} \tag{4.2}$$

The process in which A_n are independent, identically distributed (IID), and their distribution is allowed to be general, is known as a renewal process. Poisson process and Bernoulli process are important and special cases of renewal process.

Poisson models are the oldest traffic models dating back to the advent of telephony [22]. A Poisson process is characterized as a renewal process whose inter arrival times $\{A_n\}$ are exponentially distributed with rate parameter l [25]:

$$P\{A_n \leq t\} = 1 - \exp(-lt) \tag{4.3}$$

Equivalently this is a counting process that satisfies

$$P\{N(t) = n\} = \exp(-lt)\,(lt)^n/n! \tag{4.4}$$

The number of arrivals in disjoint intervals is statically independent. Time-dependent Poisson process is obtained by letting the rate parameter l depend on time.

The process in which A_n is constant was employed for the modeling of the time delay in the Internet [13]. This constant time delay depicts the use of a buffer that delays the sensor signal or the control input by an amount that is more than the worst case time

delay. This converts analysis of the system with the time-varying delay into an analysis of time-invariant system. For the stability of the system this buffer should delay the signals by an amount that is always less than one sampling period for the system [13].

4.3 Stability Analysis of Existing Controller with Delays

The magnetic ball levitation system is open-loop unstable, and its closed-loop stability is lost with excessive time delay. To deal with the communication time delay that might be encountered in the Internet traffic, artificial time delays were incorporated in the control loop, and three sets of experiments were performed. The setup used for the experiments is shown in Fig.14.

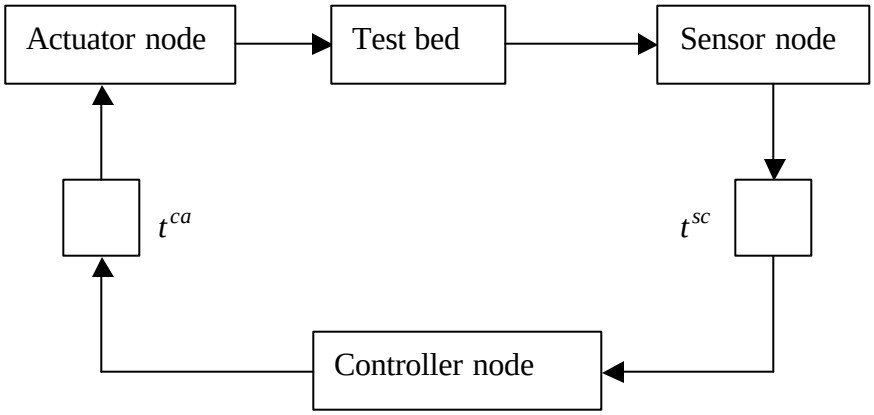

Fig.14. Digital control system with the induced delays.

Where, t^{sc} and t^{ca} are the artificially generated delays. It has been assumed that the delays between the sensor and the controller and between the controller and the actuator are the same and follow the same probability distribution. Along with the network delays, the controller takes finite computational time, t^{c}. As the delays have been artificially generated offline and incorporated in the control loop these time delays take computation time into consideration. This implies the effect of t^{c} is embedded in either t^{sc} or t^{ca}. In the experimental setup the sensor node is a time-driven sensor whereas the controller node and the actuator node are event-driven. In other words, the sensor node

collects the sensor data at a fixed sampling period, whereas the controller and the actuator function as soon as the data arrives at their respective nodes.

The first set of the experiments was performed with the introduction of a constant time delay. As discussed in Section 4.2 the constant time delay case represents the use of the buffer to make the system with time-varying delay appear time invariant. This time delay was accommodated on the simple lead-lag compensator designed in Chapter 2. The maximum amount of time delay for this model was obtained for various sampling frequencies. A plot between the maximum time delay and various sampling frequencies has been shown in Fig.15.

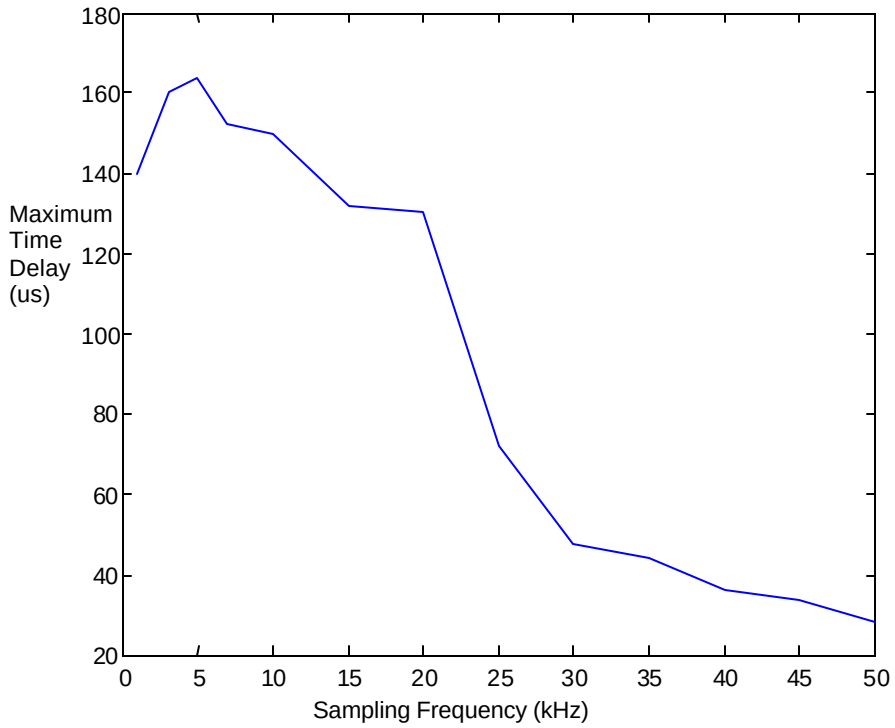

Fig.15. Maximum allowable constant time delay with respect to sampling frequencies.

This figure shows a general trend that the maximum time delay that can be accommodated at a higher sampling frequency is less than that at a lower sampling frequency. As the sampling period is increased by decreasing the sampling frequency, more time is available in the real-time control routine without frame overruns. Thus, the maximum allowable time delay keeps increasing with lowering of the sampling frequency. However, at a very low sampling frequency again the maximum allowable

time delay decreases. This trend can be accounted for the poorer stability of the system at a low sampling frequency. Thus, it can be concluded that there is an optimal sampling frequency for the system for which we can accommodate maximum allowable constant time-delay without affecting the system stability.

The second set of experiments was performed to deal with the randomness of the time delay that will be encountered while the control loop is closed via the Internet. A uniform probability density function (PDF) for this time delay (in the range of zero to 8 μs for 50-kHz sampling frequency) was used to model this random time delay in the control loop. The random time delays in the chosen range were generated using a MATLAB function, unidrnd(). The results from this set of experiments are presented in Fig. 16.

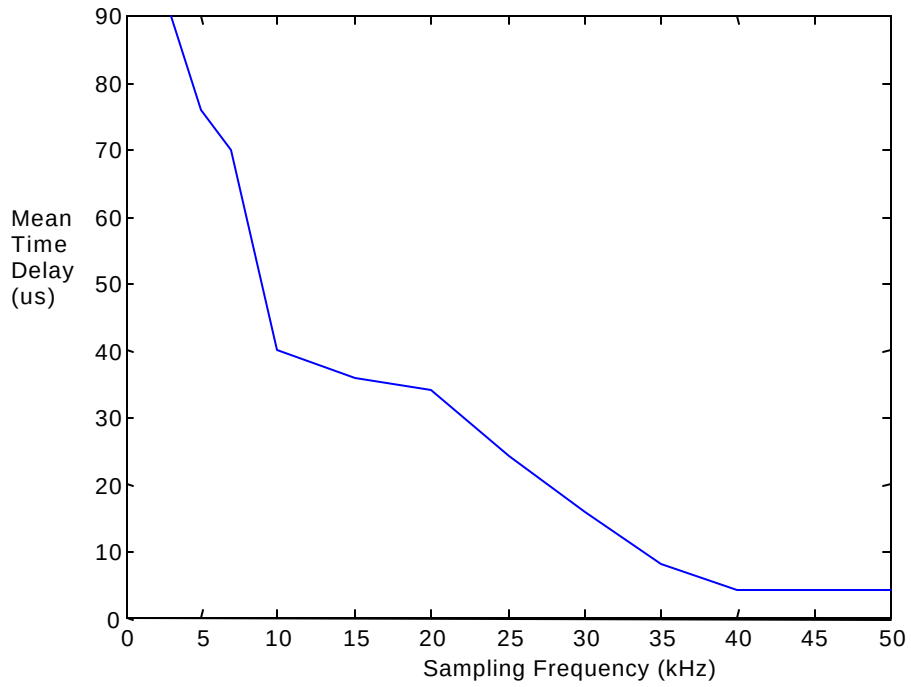

Fig.16. Mean allowable random time delay based on a uniform PDF with respect to sampling frequencies.

From Fig. 16, it can be concluded that the mean time delay that can be accommodated for different sampling periods is much less than that in the constant time

delay case due to the random nature of the delays. Due to these random delays, actual measured position information may be missing in some sampling periods. These vacant samples severely affect the stability of a closed-loop control system. Fig. 16 also indicates that reducing the sampling frequency down to a threshold allows for accommodation of more time delay in the control loop. Once the threshold sampling frequency is reached, reducing sampling frequency reduces the amount of random delay that can be accommodated in the control loop.

The third set of experiments used random delays following a Poisson PDF (with a mean of 8 μs was used for 50-kHz sampling frequency). These delays give more realistic characteristics of the random delays that might be encountered in the Internet. The results obtained for the third set of experiments have been presented in Fig. 17.

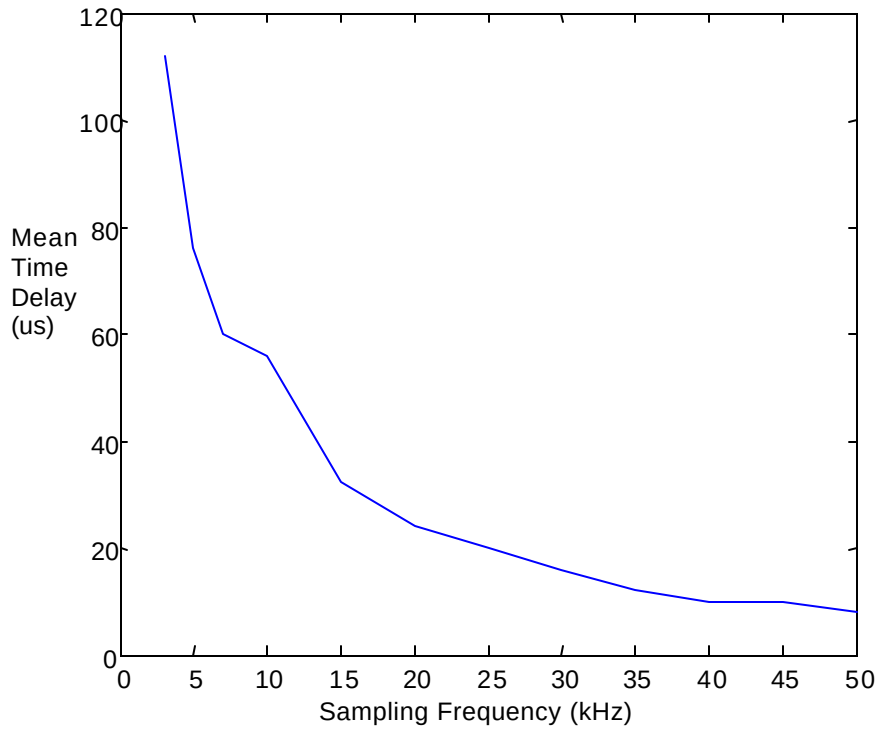

Fig.17. Mean allowable random time delay based on Poisson PDF with respect to sampling frequencies.

From the figure it can be concluded that the mean allowable random time delay, based on the Poisson distribution, is more than that based on the uniform distribution for

low frequencies. This is due to the fact that the random time delays generated according to a Poisson distribution function have time delays clustered around its mean.

4.3 Experimental Delay Measurement for LAN

The structure of a typical LAN consists of number of computers and network devices interconnected by a shared transmission medium, typically a cabling system. The cabling system is arranged in bus, ring or star topology and can use twisted-pair cable, coaxial cable, or optical fiber transmission media.

The computers and network devices are connected to the cabling system through a NIC (network interface card) or LAN adapter. The NIC card coordinates the transfer of information between the computer and the network. The NIC card transfers information in parallel format to and from main memory (RAM) in the computer. On the other hand, the NIC card transfers information in serial format to and from the network, so parallel-to-serial conversion is one of the NIC's function. Each NIC card is assigned a unique physical address that is burned into the ROM [26].

The Ethernet is the most widely used local area network (LAN) technology. The original and most popular version of Ethernet supports a data transmission rate of 10 Mb/s. Newer versions of the Ethernet called Fast Ethernet and Gigabit Ethernet support data rates of 100 Mb/s and 1Gb/s (1000Mb/s). An Ethernet LAN may use a coaxial cable, special grades of twisted pair wiring, or fiber optic cable. Bus and Star wiring configurations are supported. Ethernet devices compete for access to the network using a protocol called Carrier Sense Multiple Access with Collision Detection (CSMA/CD). In CSMA/CD a system with a frame to transmit waits until the channel is silent. When the channel goes silent, the station transmits but continues to listen for collisions that can occur if other stations also begin to transmit. If a collision occurs, the station aborts the transmission and schedules a later random time when it will reattempt to transmit its frame [26].

Round-trip time delay measurements have been done for PCs connected over the same LAN and on different LANs. Round-trip time delay data over the LAN were collected using the packet Internet groper (PING) utility. PING utility was used to

measure the round-trip time over the LAN because we wanted to keep the network overhead due to probe packets small. In addition to it we wanted to avoid the possibility of flooding the network with probe packets. The PING utility available with Windows operating system is run in an MS-DOS session and provides accuracy in milliseconds. Since the delay measurements were to be made over the LANs PING utility had to be modified to provide accuracy in microseconds. The modified PING utility providing this was obtained as freeware [27].

PING is a simple application used to determine whether a host is online and available. PING makes use of Internet Control Message Protocol (ICMP) messages. The purpose of ICMP is to inform sending hosts about errors encountered in IP datagram processing or other control information by destination hosts or by routers. ICMP is considered to be in the same layer as IP. ICMP has two advantages. First, there is no need to obtain an account on the hosts being pinged. Second, of the commonly available protocols, it places the least computational load on the hosts being pinged. PING sends one or more ICMP Echo messages to a specified host requesting a reply. PING is often used to measure the round-trip delay between two hosts [28–29].

The round-trip delay data were collected on several distinct days of the week and at various times of the day. The host was pinged and the round-trip delay statistics was collected over a period of approximately 5 hours each time. The results of these experiments are presented in Tables II and III.

Table II. Round-trip time delays between PCs on different LANs.

No.	Day	Date M/D/Y	Time	Round-Trip Time Delay (ms)			No. of Packets
				Min.	Mean	Max.	
1	Sat.	11/16/02	1:40 AM–2:40 AM	0.760	0.843	123.52	49985
2	Sun.	11/17/02	3:00 PM–8:00 PM	0.768	1.234	188.77	20939
3	Mon.	11/18/02	8:30 AM–1:00 PM	0.771	0.906	187.61	17814
4	Wed.	11/20/02	10:30 PM–3:30 AM	0.761	0.884	79.73	34348
5	Sun.	11/25/02	5:00 PM–2:40 AM	0.757	0.860	63.18	18293

Table II summarizes the measured round-trip times between the two PCs with IP addresses werc-dhcp-1825.tamu.edu and maglev7.tamu.edu. The two PCs were connected to different LANs. The minimum, maximum, mean and standard deviation of this time delay are also presented for each experiment. It can be observed that the maximum time delay was observed on Sunday Nov. 17, 2002, between 3:00 PM and 8:00 PM. This sudden surge in time delay on Sunday Nov. 17, 2003 can be accounted to sporadic congestion occurring in the LANs. The minimum round-trip time delay was on Sunday Nov. 25, 2003, between 5:00 PM and 2:40 AM. Typically, the LANs have the maximum traffic on weekdays during the daytime and have the lowest traffic in nights of weekends.

Table III summarizes the measured round-trip times between werc-dhcp-1825.tamu.edu and maglev1.tamu.edu. Both the PCs were connected to the same LAN. The minimum, maximum, mean and standard deviation of this time delay have also been presented for each experiment. It can be observed that the maximum time delay was observed on Sunday Dec. 08, 2002, between 2:00 PM and 10:00 PM. The minimum round-trip time delay was on Wednesday Dec. 04, 2003, between 12:00 PM and 8:30 PM.

Table III. Round-trip time delays between PCs on the same LAN

No.	Day	Date M/D/Y	Time	Round-Trip Time Delay (ms)			No. of Packets
				Min.	Mean	Max.	
1	Sat.	11/30/02	7:00 PM–4:00 AM	0.387	0.444	1.977	34758
2	Tue.	12/03/02	10:00 AM–4:00 PM	0.307	0.395	9.357	21340
3	Wed.	12/04/02	12:00 PM–8:30 PM	0.294	0.383	16.674	30000
4	Sat.	12/07/02	1:46 PM–7:40 PM	0.309	0.378	1.428	21212
5	Sun.	12/08/02	2:00 PM–10:00 PM	0.303	0.403	17.97	30707

Details of some of the experiments are presented in Appendix F. In each figure, the probability distribution of round-trip times are plotted. The following section discusses about the observations made from these experiments.

4.3.1 Observations and Discussions

In this section, some initial observations based on the experiments are presented. These are preliminary observations and the discussions reflect the current trend observed from the data. Following observations can be made from the experimental results obtained in the previous subsection.

1. The round-trip time delay is fairly uniform except for some sporadic sharp rise in the time delays. These sharp rises are considered to occur due to sudden congestion of the network by other users.
2. If we ignore the sporadic sharp rise in the time delays, our network is fairly smooth in the nights and over the weekends.
3. Most of the sharp rises in round-trip time delay occur one at a time. However, there are also examples of consecutive rise in round-trip time delay.
4. However, no loss of data packets was observed during the measurements.

But these observations might vary if other protocols had been employed for the round-trip time delay estimate. From the Tables II and III and the experiments conducted in the previous section, it can be inferred that the mean round-trip time delay is more than the upper bound for the time delay in the control loop. This is true for PCs connected to the same LAN as well as PCs connected to different LANs. Thus, to close actual control loop between two PCs using the existing communication network a new controller has to be designed. This controller should be able to accommodate time delays that are at least of the order of mean time delay observed in Table III. It also requires designing an algorithm that can handle sporadic rise of the round-trip time delay as observed from the graphs included in Appendix F.

In the next section a new lead-lag controller is designed and upper bound of time delay for this controller has been obtained. An algorithm is developed that can handle bounded sporadic surges in the round-trip time delay.

CHAPTER V

DIGITAL CONTROLLER DESIGN

5.1 Introduction

In the previous chapter, upper bound for the artificially generated time-varying delays were estimated for the existing controller. Experiments were also performed for the estimation of the time delay present between two PCs connected to the same LAN and between PCs connected to two different LANs. It was concluded that experimentally measured round-trip time delay to the same LAN and between two PCs connected to different LANs is more than the upper bound for the existing controller. This necessitated designing a new controller that could perform at a lower sampling frequency thus allowing more time in the sampling period. This helped in accommodating more time delay in the control-loop. For making the maglev system test bed to perform at lower sampling frequencies, a lead lag controller was designed directly in the digital domain.

5.2 Designing the Digital Controller

The first step in designing controller directly in z-domain is to transform the plant model to a discrete system. The plant model is transformed into a discrete system using the transformation

$$G(z) = (1 - z^{-1})Z\left(\frac{G(s)}{s}\right) \tag{5-1}$$

This discrete model for the plant is obtained by placing a ZOH (zero-order hold) before the plant and an A/D after the plant, as shown in the Fig 18 [30].

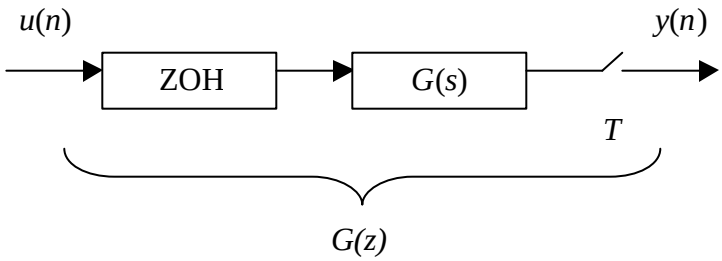

Fig.18. Discrete model of the plant.

where, $u(n)$ are the sampled signals from the computer and $y(n)$ are the samples of the outputs from the plant. By digitization of the plant model instead of the controller model, the approximate nature of the plant is eliminated. This is because the presence of the ZOH before the plant has an exact discrete equivalent that includes the lagging effect of the hold. The controller design iterations to achieve the desired system specifications are carried out using the z-domain analysis tools. The $D(z)$, thus found yields performance that are very close to the desired specifications for high and low sampling frequencies.

Thus, we find the discrete equivalent of the plant model for the maglev test bed. The discrete equivalent of the plant model developed by Steve C Paschall, II in Section 2.2.2 for a sampling period of 3 ms is given by

$$G_c(z) = \frac{7.7079 \times 10^{-6}(z+1)}{z(z-1)^2} \tag{5-2}$$

5.2.1 Root Locus Design

In the root-locus design, a unity feedback system is used with a proportional gain as a controller. The value of the gain is varied from negative infinity to positive infinity and the locations of the closed-loop poles are observed in the locus. From Fig. 19, it can be clearly seen that the poles located at 1 and are on the edge of the unit disk. These poles move out of the unit disk when the gain of the controller varies from negative infinity to positive infinity. Any gain value in the negative range moves the two closed-loop poles just outside the unit disk. Hence, the control of the plant by merely changing the proportional gain will not be satisfactory and the use of the other design methods such as lead-lag compensation should be considered.

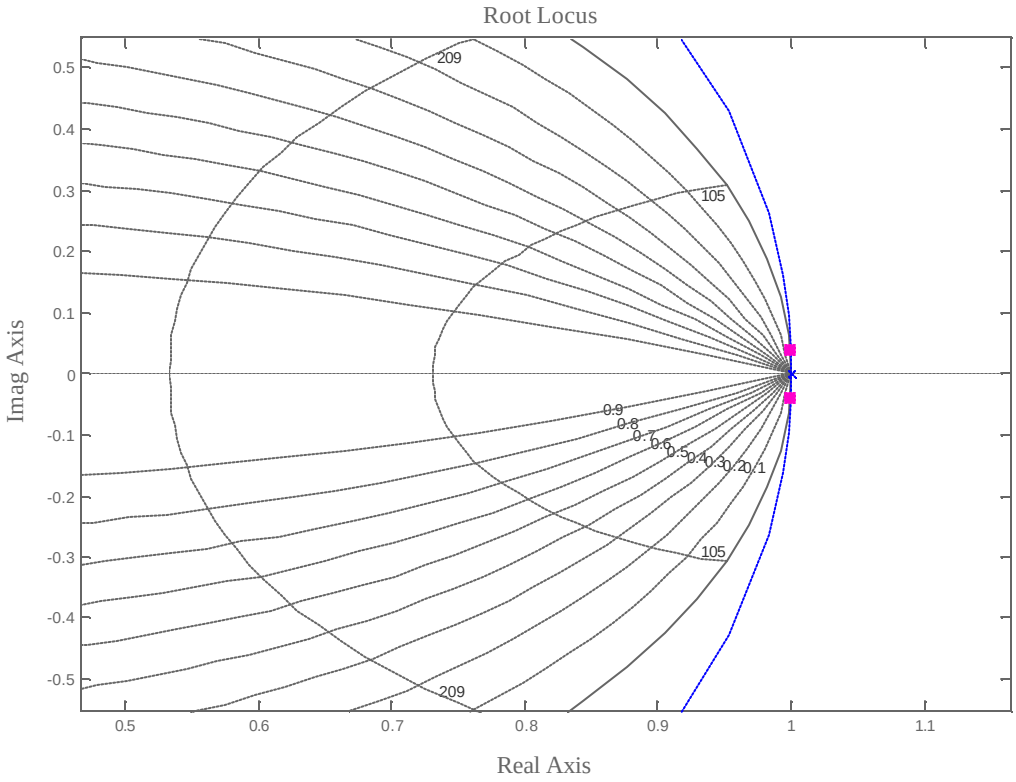

Fig.19. Root locus for feedback with a gain.

5.2.2 Lead-Lag Compensation

Since a satisfactory design was not obtained by a gain adjustment alone, lead-lag compensation was designed for the system. The lead compensation acts mainly to modify the dynamic response to raise the bandwidth and lower the rise time and also to decrease the transient overshoot. The lag compensation is used to raise the low frequency gain and thus to improve the steady-state accuracy of the system.

Equation (5-3) shows such a lead-lag compensator. The lead compensator has its zero placed at $z = 0.862$, which is close to the plant poles and pole placed at $z = -0.141$. The lag compensator has its zero placed at $z = 0.892$ and pole at $z = 0.923$.

$$D(z) = 4.15\times10^{4}\,\frac{z^{2}-1.754z+0.769}{z^{2}-0.782z-0.13} \qquad (5\text{-}3)$$

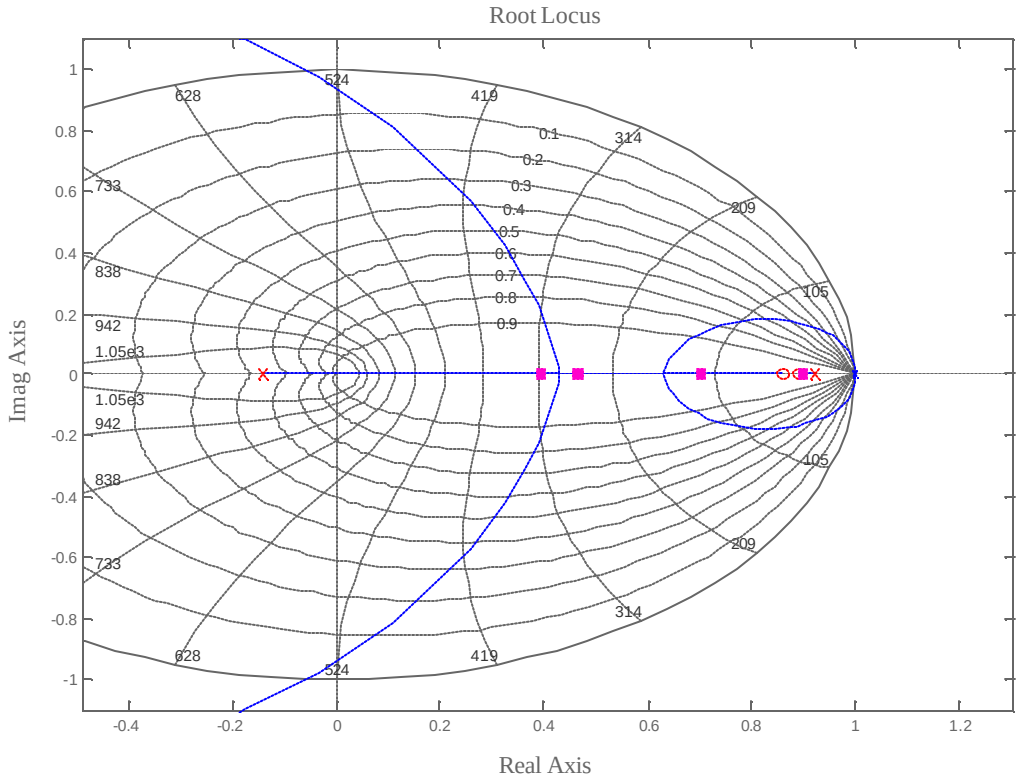

Fig.20. Modified root locus when the lead-lag compensator (5-3) is implemented.

Fig.20 shows the modified root locus, when the compensator (5-3) is implemented. The plant poles lying at the unit disk move towards the compensator zeros and thus make the whole system stable. It also provides with the scope of varying the compensator gain to achieve the desired performance of the system. The value of the gain at 4.15×10^{4} gives satisfactory performance of the system as desired.

Fig. 21 shows the transient response of the closed-loop system to a unit step input. It can be seen that the settling time for the system is about 45 ms and the percent overshoot is about 30%. The closed-loop bode plots is shown in Fig.22. The loop transmission figure shows the system crossover frequency to be approximately 28.3 Hz.

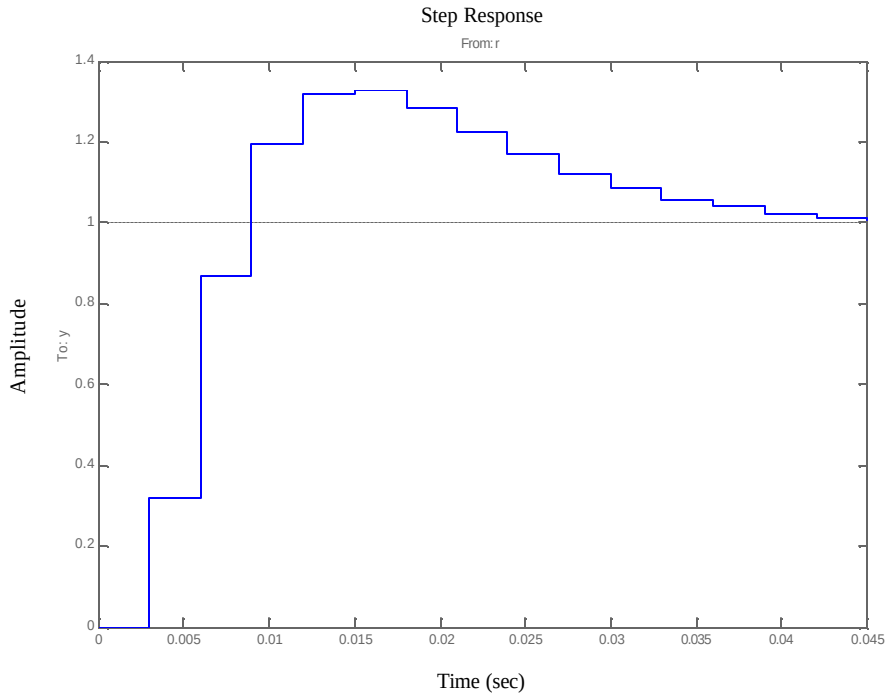

Fig.21. Closed-loop maglev system transient response for unit step.

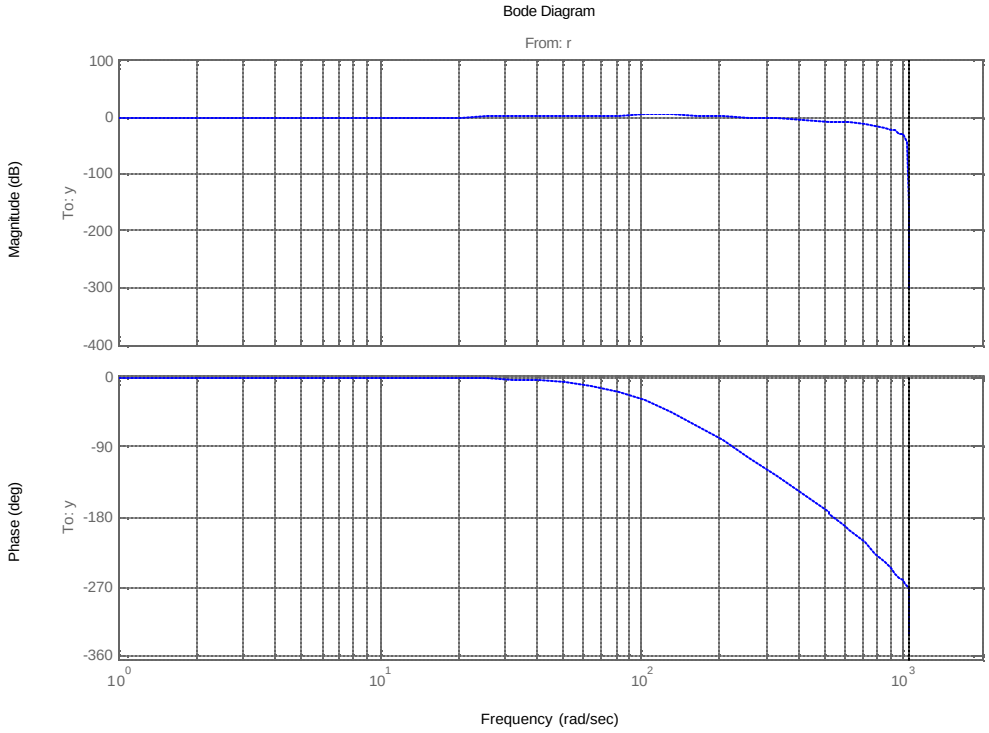

Fig.22. Maglev system closed-loop Bode plot.

5.2.3 Stability Analysis for the Digital Controller

The digital controller design technique was used iteratively to find out the lowest sampling frequency at which the controller can be run without making the system unstable. From the experiments, it was observed that the minimum frequency at which the digital controller can control the single-actuator magnetic levitation system test bed is 333.33 Hz.

Another digital lead-lag controller was designed to control the test bed at a sampling frequency of 500 Hz. This controller provided 2-ms sampling period in the control loop. The digital controller running at 500 Hz sampling frequency is

$$D(z) = 8.44 \times 10^4 \frac{z^2 - 1.762z + 0.7776}{z^2 - 0.761z - 0.133} \tag{5.4}$$

To calculate the upper bound for the time delay that can be accommodate in the control-loop for these two controllers, experiments similar to section 4.3 were conducted. The first set of experiments was conducted for the estimation of maximum amount of constant time delay that can be accommodated in the control-loop. The results for this set of experiments have been presented in Table IV.

Table IV. Maximum constant time delay for different sampling frequencies.

Sampling frequency (Hz)	Maximum constant delay in the control loop
500	650 μs
333.3	2 ms

From the Table IV, it can be observed that the maximum amount of constant delay that can be accommodated in the control loop is 2 ms for the 333.3 Hz sampling frequency. The maximum amount of round-trip time delay is more than the mean time delay that was observed in the experiments conducted in Section 4.3. The use of the constant round-trip time delay for calculating the upper bound of time delay suggests the use of buffers before the controller node and the D/A node.

The second set of experiments was conducted with the artificially generated round-trip time delay that followed the uniform distribution. The delays ranged from zero to 1 ms for the 500 Hz sampling frequency and 1.52 ms for 333.3 Hz. These random delays were generated using the MATLAB. The results for these experiments are presented in Table V. From the Table V, it can be observed that mean time delay that can be accommodated in the control loop is of the order of mean time delay observed from Table III.

Table V. Mean time delay based on a Uniform PDF with respect to sampling frequencies.

Sampling frequency (Hz)	Mean time delay in the control loop
500	500 µs
333.3	760 µs

The third set of experiments was conducted for estimation of upper bound for the round-trip time delays in the control-loop where the delays followed a Poisson probability distribution. These random round-trip time delays were generated using the MATLAB. The results of this set of experiments are presented in Table VI. From Table VI, it can be observed that mean time delay that can be accommodated in the control loop for 333.3 Hz sampling frequency is more than the mean time delay observed from Table II.

Table VI. Mean time delay based on a Poisson PDF with respect to sampling frequencies.

Sampling frequency (Hz)	Mean time delay in the control loop
500	600 µs
333.3	1.84 ms

Experiments were also conducted to test the stability of the controller running at the sampling frequency of 333.3 Hz in the presence of measured time delay between werc-dhcp-1825.tamu.edu and maglev1.tamu.edu and between werc-dhcp-1825.tamu.edu

and maglev7.tamu.edu. For conducting these sets of experiments the round-trip time delay between the two PCs presented in Appendix F was used offline by converting it to an array. This array had 20000 delay values and was downloaded on the controller board along with the control code. The delays in the array were with in the upper bound of the artificial time delays for the digital controller running at 333.3 Hz. The assumptions made for performing these experiments are that the same amount of time delay is present both in the feedback loop and the forward loop and no sporadic excessive delays are present in the actual data. These sporadic excessive time delays were several times as large as the value of sampling period for the controller running at a sampling frequency of 333.33 Hz, and thus led to instability of the system.

From the experiments we can conclude that if the network is free of the sporadic rise in the round-trip time delays, the digital controller running at the sampling frequency of 333.3 Hz is able to maintain the stability for the test bed. The round-trip time delay data that was used for these experiments included data measured between PCs connected on the same LAN and between PCs connected to different LANs. The system becomes unstable with the time delays exceeding the upper bounds because the effect of application of control input at the actuator is not sensed in the next sampling period by the sensor. The actuator takes a finite amount of time for implementation of the control input applied to it. If the time delays exceed the sampling period for the system, no control input is applied to the D/A channel and thus we have cases of vacant sampling. During the vacant sampling, no control input is applied to the D/A channel and thus the system loses its stability.

All the networks have a limited data carrying capacity. When the load is light, the round-trip time delay between the host PC and the client PC is almost uniform. Also the network load might have an important role in data transmission from the host PC using the sockets. When the network is congested the sockets might not be able to send the data and it gets stored in the buffer of the socket. When the network load again drops all the stored data in the socket might be released in a burst. This phenomenon is detrimental for the stability of the systems that are data critical. Some of the causes of the network congestion are:

1. Number of computers using the network capability for their usage.
2. High demand from networked applications, such as groupware (for scheduling and appointments) and e-mail with large attached files.
3. High demand from bandwidth-intensive applications, such as desktop publishing and multimedia.

5.3 Sensor Data Estimation and Timeout

It can be observed from discussions in the previous sections and chapters that the upper bound for the time delay in the control-loop is less than the sampling period for the system. As seen from the experiments included in Appendix F the time delays in the network are not always less than the sampling period of the system. If such a situation arises available digital controller can not perform satisfactorily and the stability of the system is lost. Thus for the existing controller to maintain the system stable the time delay in the control loop should always be less than the sampling period of the system.

In this section, a methodology to maintain the system stability in the presence of sporadic time delays that are more than the upper bound of time delays has been suggested. This methodology uses the concept of timeout from a control perspective. The concept of timeout has also been studied by Nilsson [18].

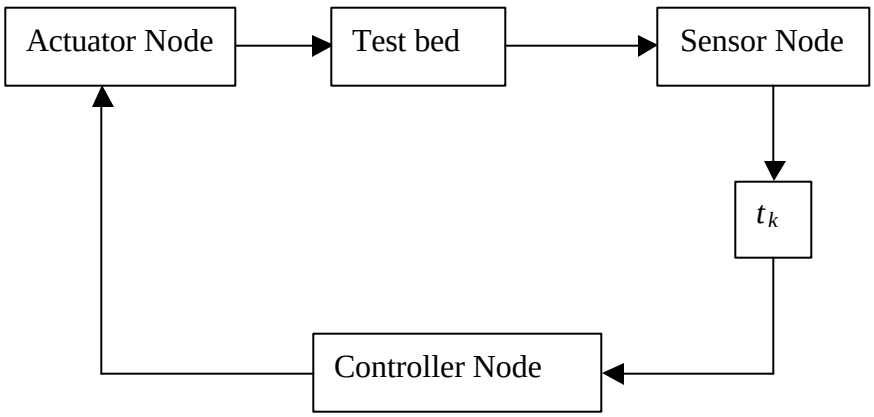

Fig.23. Setup for application of timeout methodology in control system with random time delays.

The setup for the application of the timeout methodology in the presence of time varying delays in the control loop is shown in Fig.23. In this setup t_k is a randomly varying delay which has a priori probability distribution. The timeout refers to the case where control input is calculated whenever a new sensor data arrives at the controller node with in the t^o time units for a given sampling period. Here, t^o is the amount of time for which the controller waits for the new data within a sampling period. With the experimental setup for the timeout methodology, selection of value of timeout is an important factor to ensure the stability of the system in the presence of sporadic high time delays. It can be observed from the results presented in Section 4.3, the timeout can be taken as the upper bound of the time delay for the test bed. If the new sensor data is not available till t^o time the control input is calculated based on the predicted sensor value for that sampling period.

Whenever the controller gets timeout, the controller generates control input based on an estimate of the sensor data at that instant. Thus, near accurate prediction of the sensor data is an important factor to guarantee the stability of the system during timeout cases. Various methodologies are available for the prediction of the data, having observed the previous values of the data.

5.3.1 AR Model

An AR (autoregressive) model of order *n* is given by the following model of the signal $\{y(t)\}$ [31]:

$$y(t) + a_1 y(t-1) + ... + a_n y(t-n) = v(t) \qquad (5\text{-}5)$$

we assume that, $\{v(t)\}$is a white gaussian noise with zero mean values in this thesis. The above model is obtained by assuming no input.

In ARX (autoregressive exogeneous) model we consider a dynamical system with input signal $\{u(t)\}$ and output signal $\{y(t)\}$ sampled in discrete time t = 1, 2, 3…. We assume that these signals are related thorough a difference equation

$$y(t) + a_1 y(t-1) + ... + a_n y(t-n) = b_1 u(t-1) + ... + b_m u(t-m) + v(t) \qquad (5\text{-}6)$$

For the sake of convenience, q^{-1} is chosen as the backward shift (or delay) operator implying:

$$q^{-1}y(t) = y(t-1) \tag{5-7}$$

Thus (5-6) can be rewritten as

$$A(q^{-1})y(t) = B(q^{-1})u(t) + v(t) \tag{5-8}$$

where,

$$\begin{aligned} A(q^{-1}) &= 1 + a_1 q^{-1} + ... + a_n q^{-n} \\ B(q^{-1}) &= 1 + b_1 q^{-1} + ... + b_m q^{-m} \end{aligned} \tag{5-9}$$

The dynamic relationship between the input and output signals is represented by (5-6) and (5-8) and is expressed in terms of the parameter as:

$$?^T = (a_1 \ldots a_n \quad b_1 \ldots b_m) \tag{5-10}$$

thus, (5-6) can be expressed as

$$y(t) = ?^T F(t) + \mathrm{v(t)} \tag{5-11}$$

The model thus derived describes the observed variable $y(t)$ as an unknown linear combination of the components of the observed vector $F(t)$ plus noise. The components of the *F(t)* are known as the regression variables and the model is called a linear regression model.

If the characteristic of the noise $\{v(t)\}$ is unknown, a prediction of the value of $y(t)$ having observed the previous values of $y(k)$, $u(k)$, where $k = t–1, t–2,\ldots$ is given by

$$y(t|?) = ?^T F(t) \tag{5-12}$$

In (5-12) the predicted value of the output depends on the model parameters *?*. if $\{v(t)\}$in (5-11) is a sequence of independent random variables with zero mean values *y* becomes a prediction of y [29].

Thus, for the timeout cases the sensor data can be predicted using (5-12). The model parameters *?* is estimated from the measurements of $y(t)$, $F(t)$ where $t = 1,2,3\ldots N$. The least-square estimate methodology can be employed for the estimation of model parameters *?*.

5.3.2 ARMAX Model

The model obtained by modeling the disturbance term $v(t)$ of ARX model is known as the ARMAX model [29]. The modeling of the disturbance term in (5-8) is described as a moving average (MA) of the white noise sequence $\{v(t)\}$:

$$v(t) = C(q^{-1})e(t) \tag{5-13}$$

where,

$$C(q^{-1}) = 1 + c_1 q^{-1} + \ldots + c_r q^{-r} \tag{5-14}$$

Thus the resulting model becomes:

$$A(q^{-1})y(t) = B(q^{-1})u(t) + C(q^{-1})v(t) \tag{5-15}$$

which, consists of a combination of an autoregressive (AR) part $A(q^{-1})y(t)$, a moving average (MA) part $C(q^{-1})v(t)$ and a control part $B(q^{-1})u(t)$. In this model the parameter vector is defined as:

$$?^T = (a_1 \ldots a_n \;\; b_1 \ldots b_m \;\; c_1 \ldots c_r) \tag{5-16}$$

On rearranging the terms in (5-15) we obtain:

$$y(t) = \left[1 - \frac{A(q^{-1})}{C(q^{-1})}\right] y(t) + \frac{B(q^{-1})}{C(q^{-1})} u(t) + v(t) \tag{5-17}$$

for some ?-dependent sequence $\{h_k(0)\}$ and $\{g_k(0)\}$

$$\begin{aligned} \sum_{k=1}^{\infty} h_k(0) q^{-k} &= 1 - \frac{A(q^{-1})}{C(q^{-1})} \\ \sum g_k(0) q^{-k} &= \frac{B(q^{-1})}{C(q^{-1})} \end{aligned} \tag{5-18}$$

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

$$C^*(z) = z^r + c_1 z^{r-1} + \ldots + c_r \tag{5-19}$$

are inside the unit circle.

From (5-17) it can be observed that the right-hand side is known at time t–1, except for the term $e(t)$. The term $e(t)$ is independent of everything that has happened up to time t–1. Therefore the predictor is given as

$$g_M(q; t, z^{t-1}) = \left[1 - \frac{A(q^{-1})}{C(q^{-1})}\right] y(t) + \frac{B(q^{-1})}{C(q^{-1})} u(t) \tag{5-20}$$

In (5-20) the predictor assumes that all the data from $t = -\infty$ is known. Usually the data collection is initialized at time 0. This is resolved in (5-20) by summing only up to $k = t$. Since h_k and g_k decay exponentially, this approximation is reasonable, except possibly for small t.

Thus we can organize the terms as

$$\hat{y}(t \mid q) = \left[1 - \frac{A(q^{-1})}{C(q^{-1})}\right] y(t) + \frac{B(q^{-1})}{C(q^{-1})} u(t) \qquad (5\text{-}21)$$

which on further rearrangement gives

$$C(q^{-1})\hat{y}(t \mid q) = [C(q^{-1}) - A(q^{-1})]y(t) + B(q^{-1})u(t) \qquad (5\text{-}22)$$

The above expression can be used for calculation of $\hat{y}(t \mid q)$.

If in (5-22) no input is available to the system then the model can be used as ARMA model. Thus the ARMA model is given by

$$C(q^{-1})\hat{y}(t \mid q) = [C(q^{-1}) - A(q^{-1})]y(t) \qquad (5\text{-}23)$$

This model can also be used to predict the output of the system at time *t*, when outputs till time *t*–1 are known to the system.

5.3.3 Model Selection

The different models explained earlier can be used for the estimation of sensor data in timeout cases. The estimated sensor data could be used by the controller to generate the control input whenever it does not receive the new data from the sensor node with in the specified time limit.

Experiments were conducted to select the best model to be used for sensor data estimation. The two sets of experiments for parameter estimation for different models were done offline using the batch process. In batch process the sensor data were collected for sampling frequencies of 500 Hz and 333.3 Hz. The sensor data were then arranged in the ascending order of the timestamps associated with each sensor data. The arranged sensor data was used for the estimation of the model parameter vector for both the models.

For the AR model experiments were conducted at sampling frequencies of 500 Hz and 333.3 Hz for the estimation of parameter vector for, where *?* is given by

$$?^{T} = (a_1 \ldots a_n) \qquad (5\text{-}24)$$

Various MATLAB commands such as iddata() and rarx() were used for the estimation of parameter vector using the sensor data arranged in the ascending order based on delays faced by them. The command iddata() is used for the creation of data objects that can be

used for identification techniques. The command rarx() is used to estimate the parameter vector iteratively.

A fifth-order AR model was selected for the prediction of the sensor data. The order of the AR model was selected by using the MATLAB. The AR models of the order of 10 and higher estimated sensor data with better accuracy as compared to lower order AR models. However, tenth order AR model or higher required the controller to maintain a large number of previous sensor data. Thus, an optimal fifth order AR model was chosen that estimated the sensor data with reasonable accuracy and required the controller to maintain lesser number of previous sensor data. The results of the experiments for the estimation of parameters for this model at sampling frequency of 333.3 Hz are presented from Fig.24 through Fig.28.

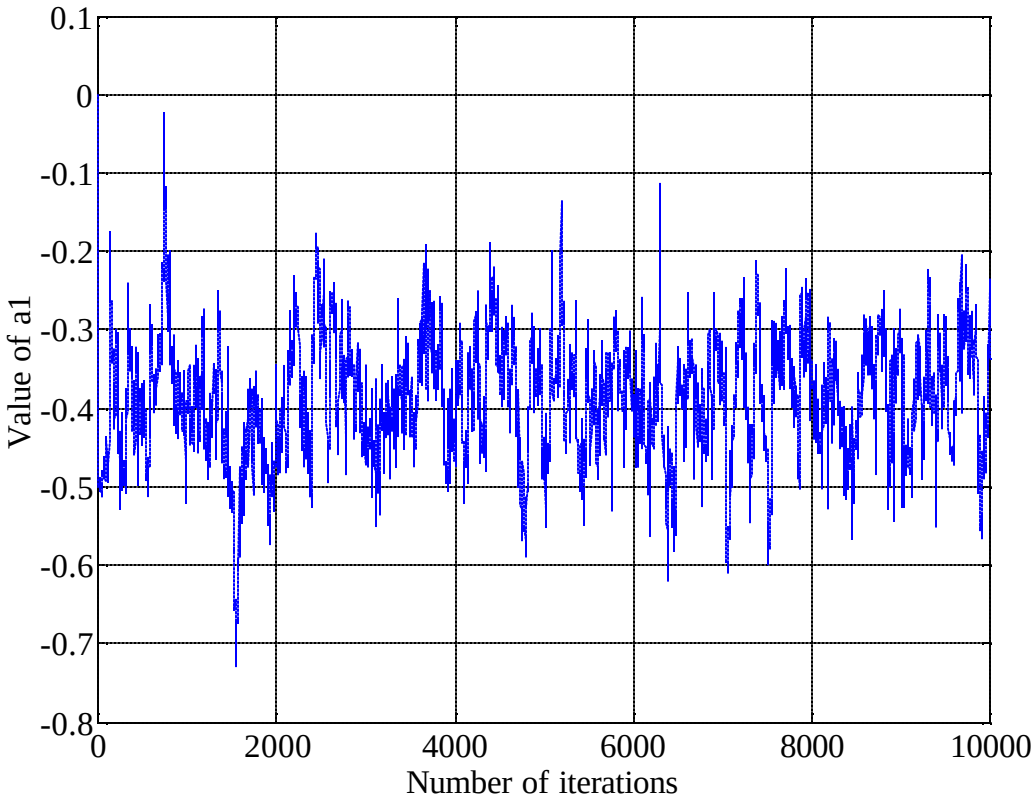

Fig.24. Plot between the value of a_1 and the number of iterations.

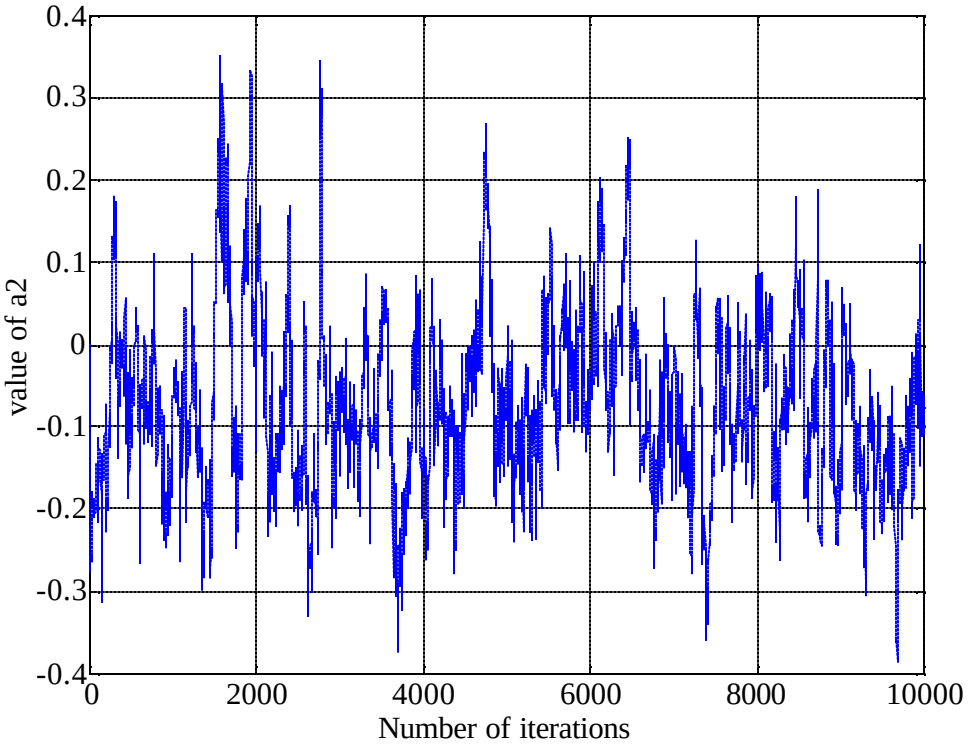

Fig.25. Plot between the value of a_2 and the number of iterations.

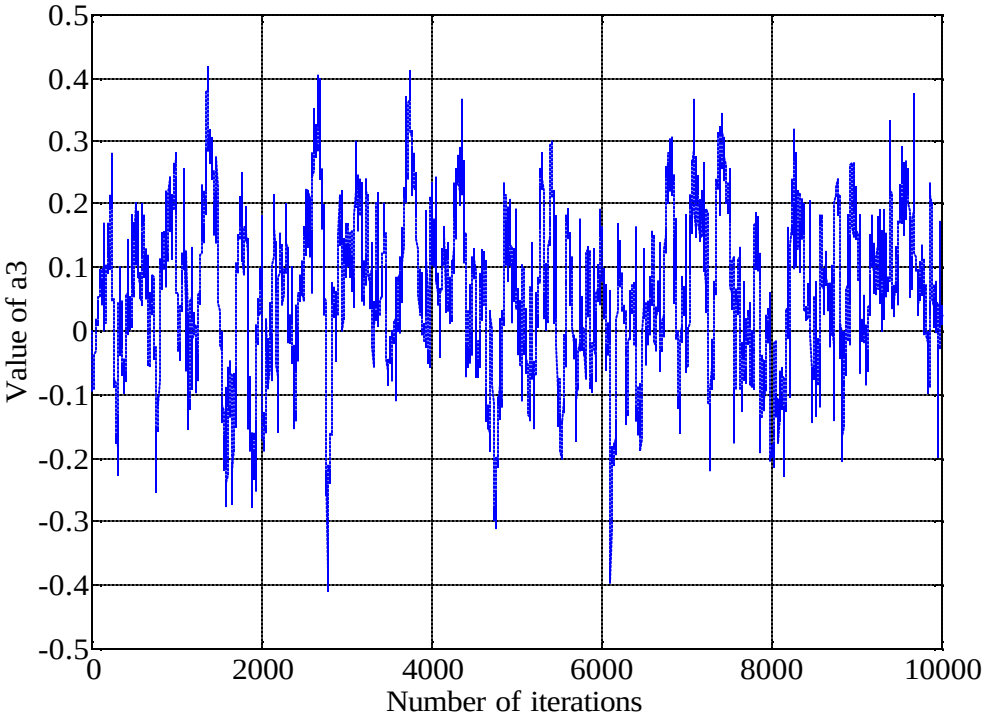

Fig.26. Plot between the value of a_3 and the number of iterations.

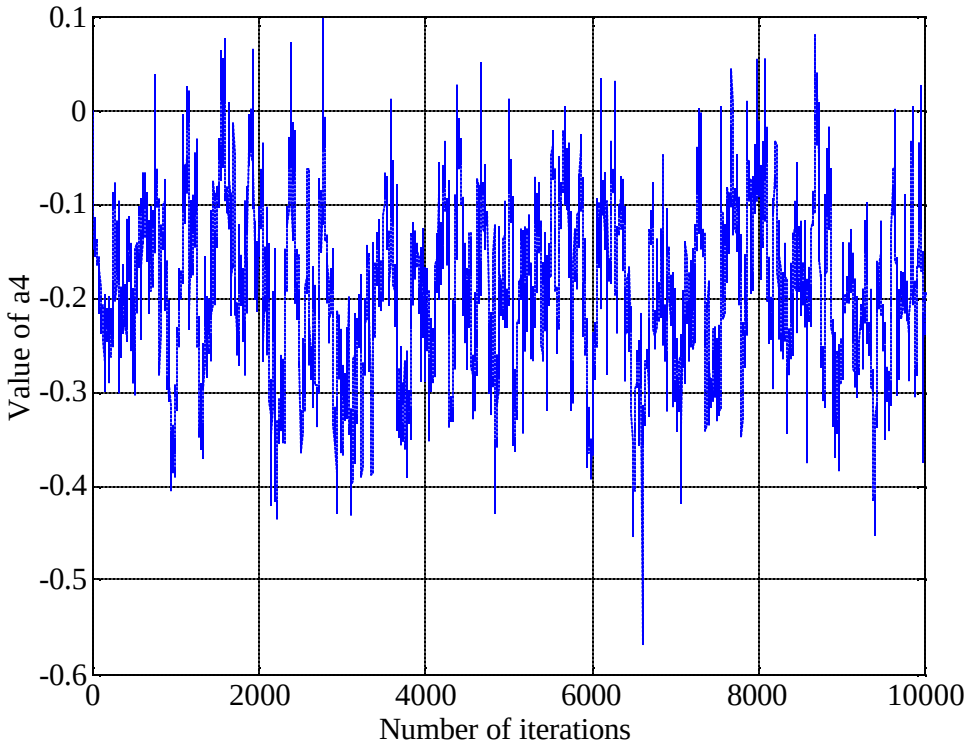

Fig.27. Plot between the value of a_4 and the number of iterations.

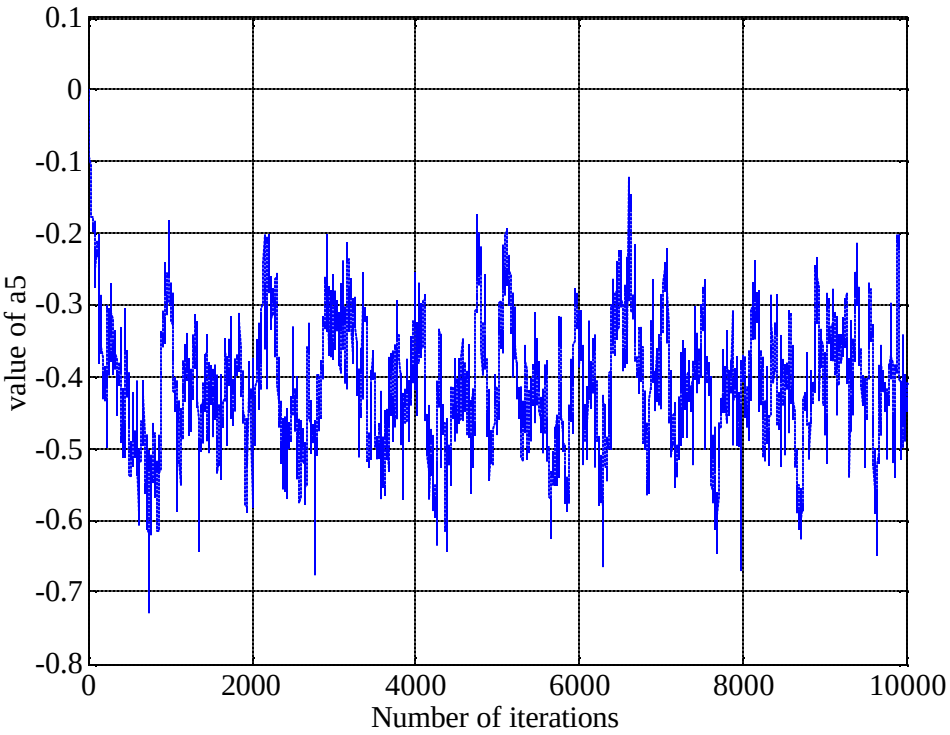

Fig.28. Plot between the value of a_5 and the number of iterations.

It can be observed that the values in the parameter model tend to be with in certain range even after approximately 26000 iterations. It can be observed from the experimental results presented in Fig. 29–33, the values tend to converge in a certain range. In these experiments the parameters were estimated using arbitrarily large initial values. It can be observed that the values of the parameters after certain number of iterations tend to converge to in a particular range. Thus, for building up the AR model for 333.3 Hz sampling period mean of these values can be taken as the coefficients in (5-12). The resulting AR model used for prediction of the sensor data for the timeout case is given by

$$\hat{y}(t \mid q) = 0.3195\,y(t-1) + 0.0669\,y(t-2) - 0.0622\,y(t-3) + 0.1960\,y(t-4) + 0.4064\,y(t-5) \quad (5\text{-}25)$$

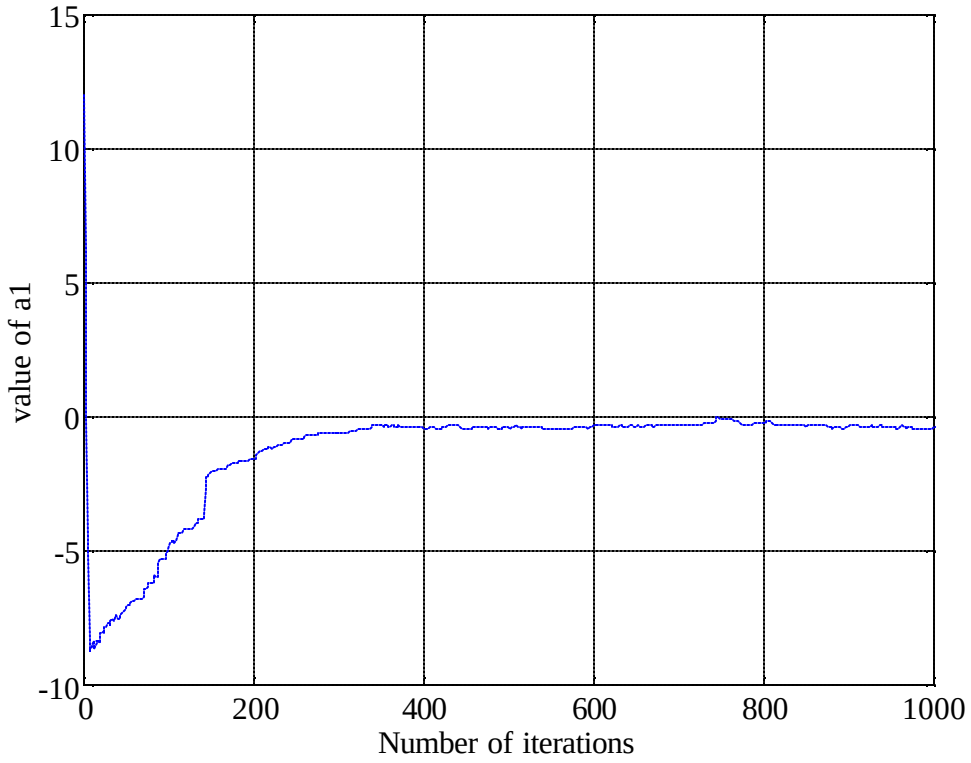

Fig.29. Plot between the value of a_1 and the number of iterations with arbitrary initial values of the parameters for AR model.

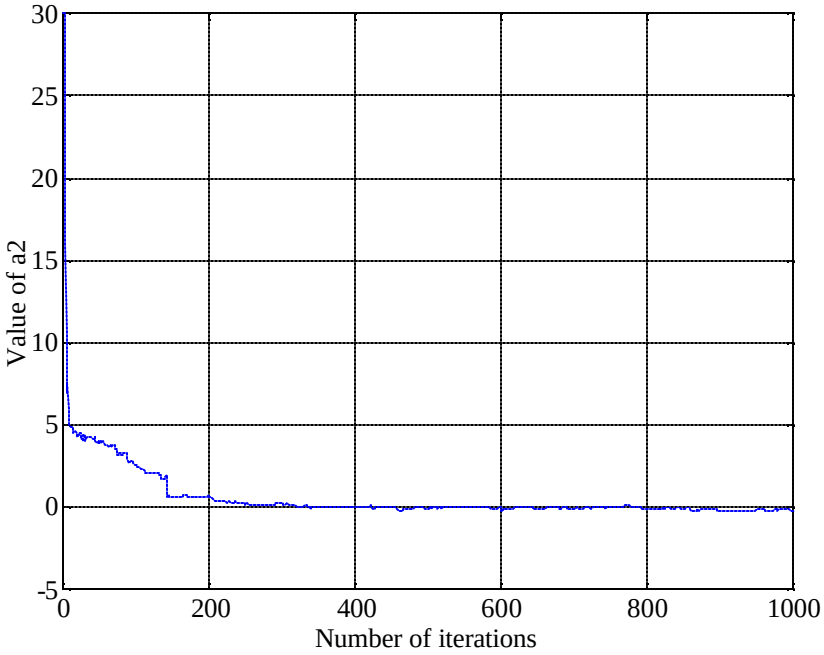

Fig.30. Plot between the value of a_2 and the number of iterations with arbitrary initial values of the parameters for AR model.

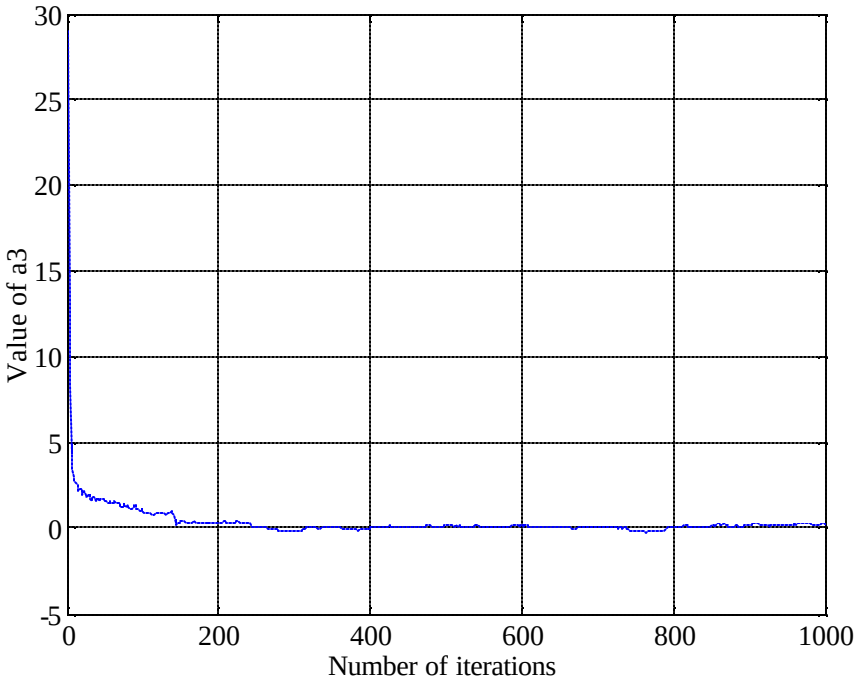

Fig.31. Plot between the value of a_3 and the number of iterations with arbitrary initial values of the parameters for AR model.

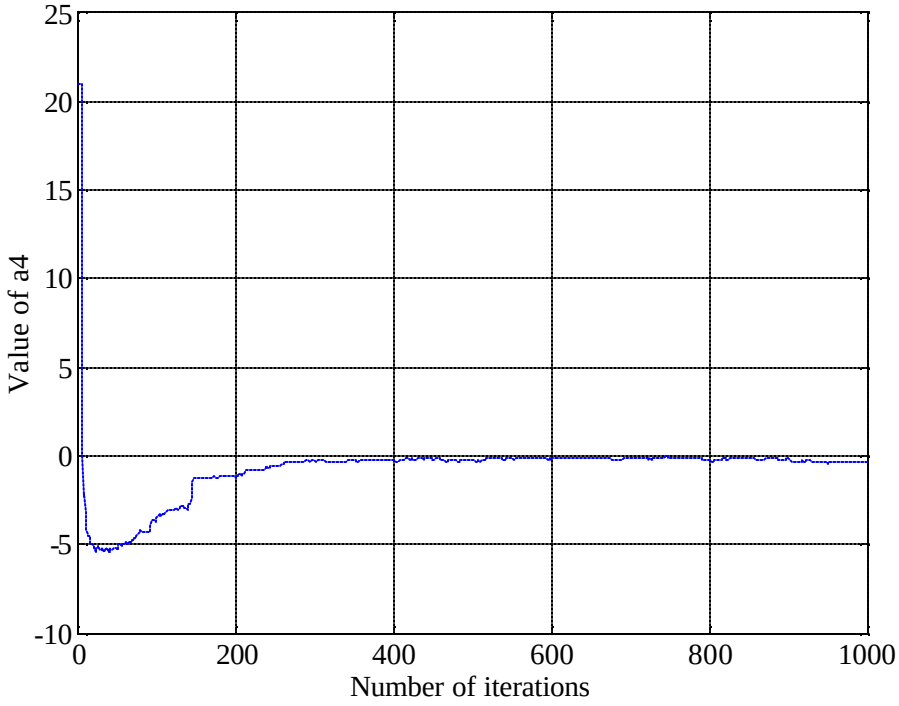

Fig.32. Plot between the value of a_4 and the number of iterations with arbitrary initial values of the parameters for AR model.

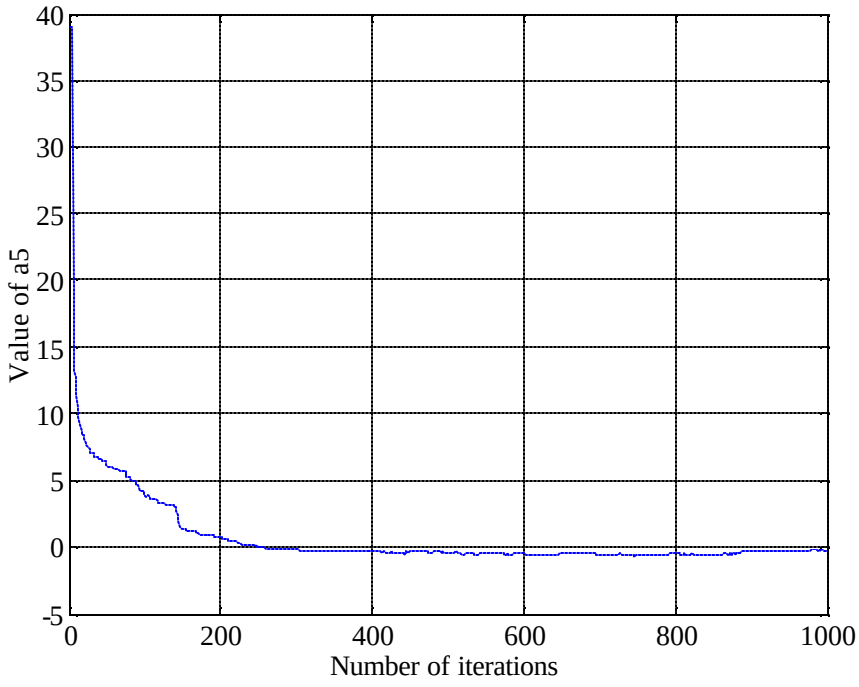

Fig.33. Plot between the value of a_5 and the number of iterations with arbitrary initial values of the parameters for AR model.

The model (5-25) was used for the prediction of the sensor data by the controller for the timeout case at sampling frequency of 333.3 Hz. In this model the controller uses the previous five values of the sensor data and computes the predicted sensor data. For this model it has been assumed that all the sensor data are time stamped and the controller is able to store the last 5 values of the sensor reading. It has also been assumed that there is no loss of data in the communication medium. The controller upgrades the predicted value of sensor reading with the actual reading whenever it is available before the next timeout occurs.

To verify the performance of this model, another set of sensor data was collected and was used for the validation purpose. A plot of the percentage error between the predicted sensor data and the actual sensor vs. the number of iterations is presented in Fig. 34.

It can be observed from Fig. 34 that the mean percentage error between the predicted value of sensor data using the AR model and the actual sensor data lies at 3 % approximately. Thus, the predicted value of the sensor data can be used for the calculation of control input in the event of timeouts.

The AR model given by (5-25) was implemented in the real-time control code running at 333.3 Hz sampling frequency and experiments were performed to find out the stability of the system for various artificial delays. The timeout for these experiments were taken as the upper bound of the delays at this sampling frequency using Tables V and VI. It was observed that with the use of the AR model and the timeout the mean of time delay based on uniform distribution was increased by approximately 10 %. With the time delay based on Poisson distribution the increase was not appreciable. This might be due to the limitation of using the predicted value of sensor data for frequent timeouts.

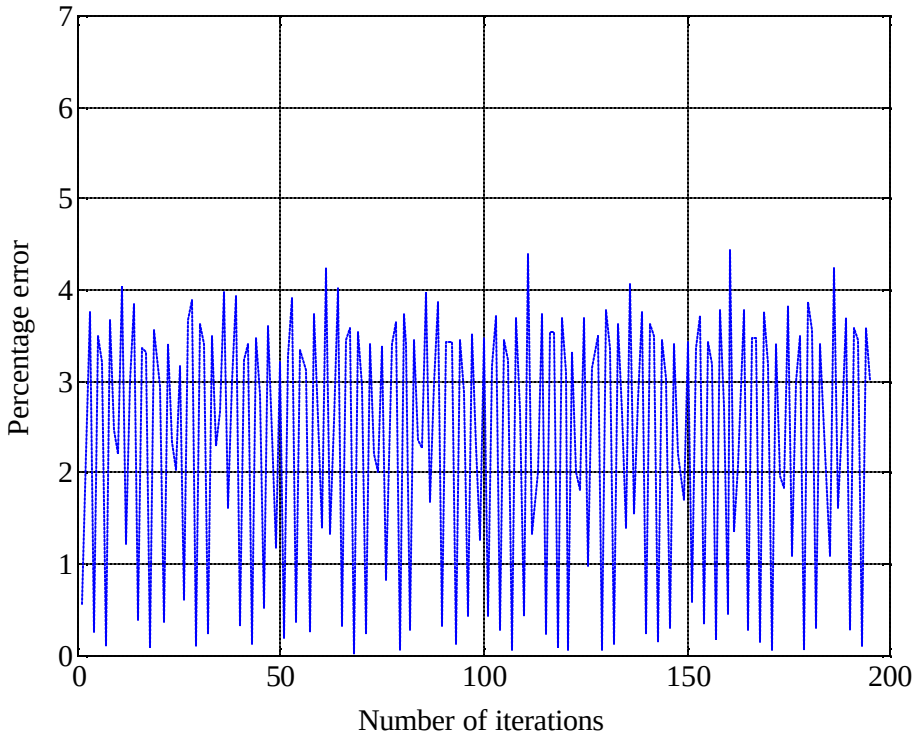

Fig.34. Plot between percentage error in using AR model for sampling frequency of 333.3 Hz and number of iterations.

Similar experiments were carried out with the AR model for the 500 Hz sampling frequency. A fifth-order AR model was selected for the 500 Hz sampling frequency. The values of the parameters for this model were also estimated iteratively using the same methodology for the AR model for the 333.3 Hz sampling frequency. They were estimated iteartively using the batch process. The sensor data were collected offline and arranged in the ascending order of the timestamps. The obtained AR model is given by:

$$\hat{y}(t \mid q) = 0.4292\,y(t-1) + 0.2169\,y(t-2) + 0.0956\,y(t-3) + 0.0604\,y(t-4) + 0.1964\,y(t-5) \quad (5\text{-}26)$$

To verify the performance of this model, another set of sensor data was collected and was used for the validation purpose. A plot between the error between the predicted sensor data and the actual sensor and the number of iterations is presented in Fig. 35.

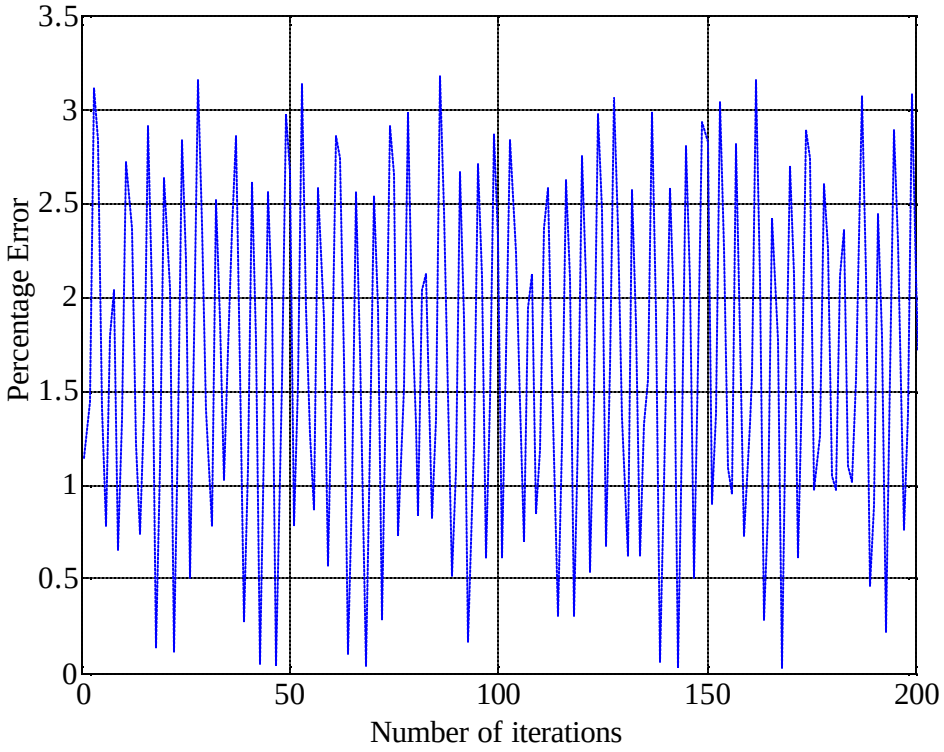

Fig.35. Plot between percentage error in using AR model for sampling frequency of 500 Hz and number of iterations.

It can be observed from Fig. 35 that the mean percentage errors between the predicted value of sensor data using the AR model and the actual sensor data is at 2 % approximately. The obtained AR model for 500 Hz sampling frequency predicts the sensor data with more error as compared with the AR model for the sampling frequency of 333.3 Hz.

The second set of experiments was conducted to estimate the parameter vector for the ARMA model. A [4, 1] order of the ARMA model was selected to predict the sensor data in the timeout case for both the 333.3 Hz and 500 Hz sampling frequency. The parameter estimation was done using the off-line batch process. The sensor data was collected and then arranged in an ascending order with respect to its time stamp. This sensor data was used iteratively for the estimation of the parameters of the [4, 1] ARMA model.

The results of the batch process for the sampling period of 333.3 Hz have been presented in Fig.36-40.

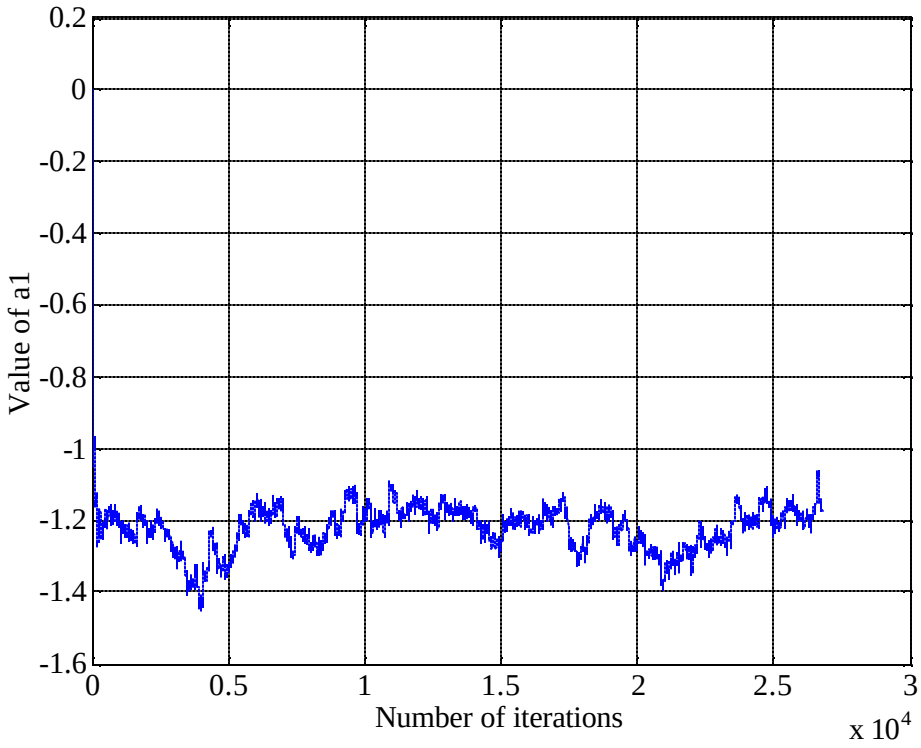

Fig.36. Plot between the value of a_1 and the number of iterations.

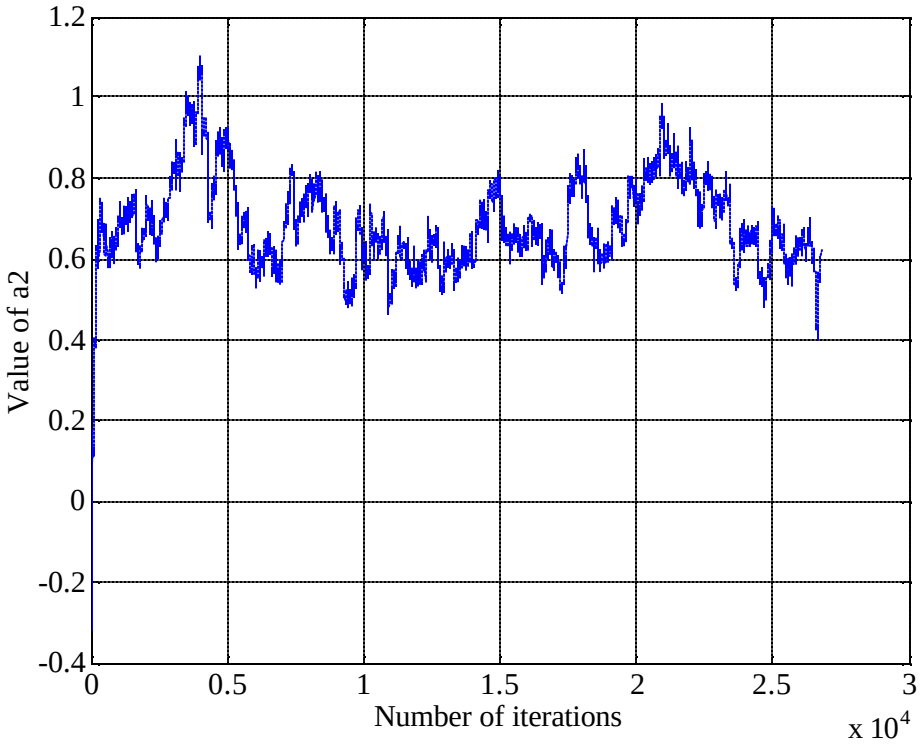

Fig.37. Plot between the value of a_2 and the number of iterations.

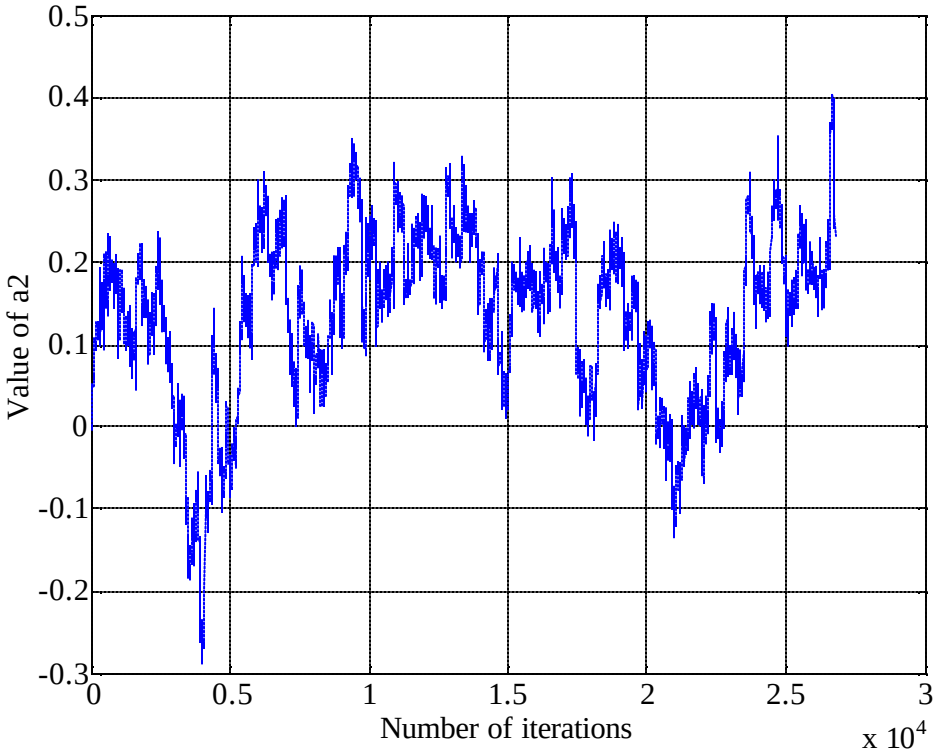

Fig.38. Plot between the value of a_3 and the number of iterations.

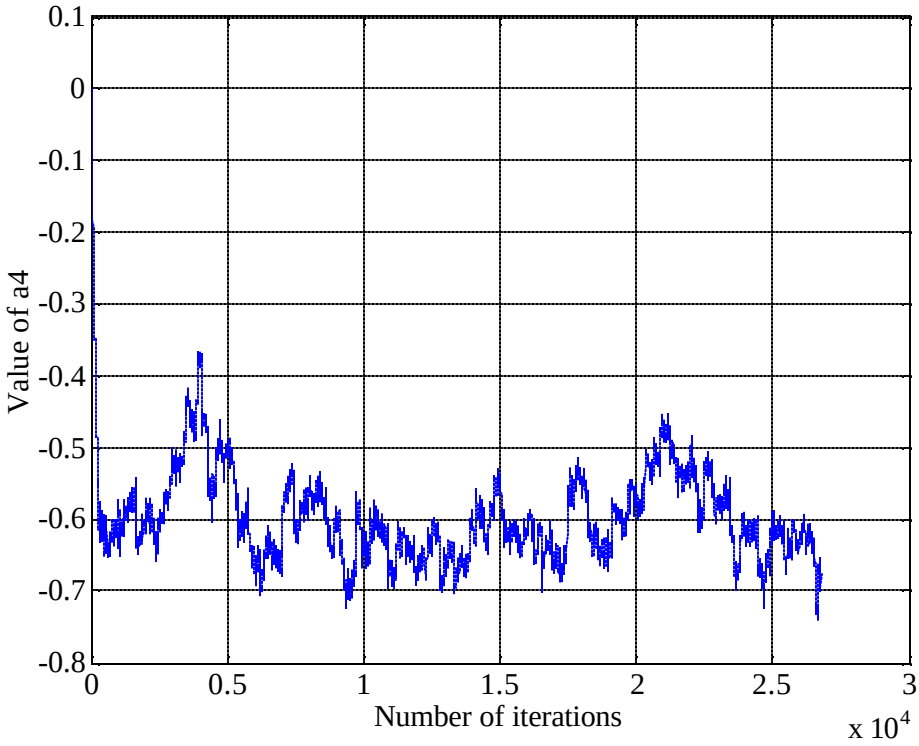

Fig.39. Plot between the value of a_4 and the number of iterations.

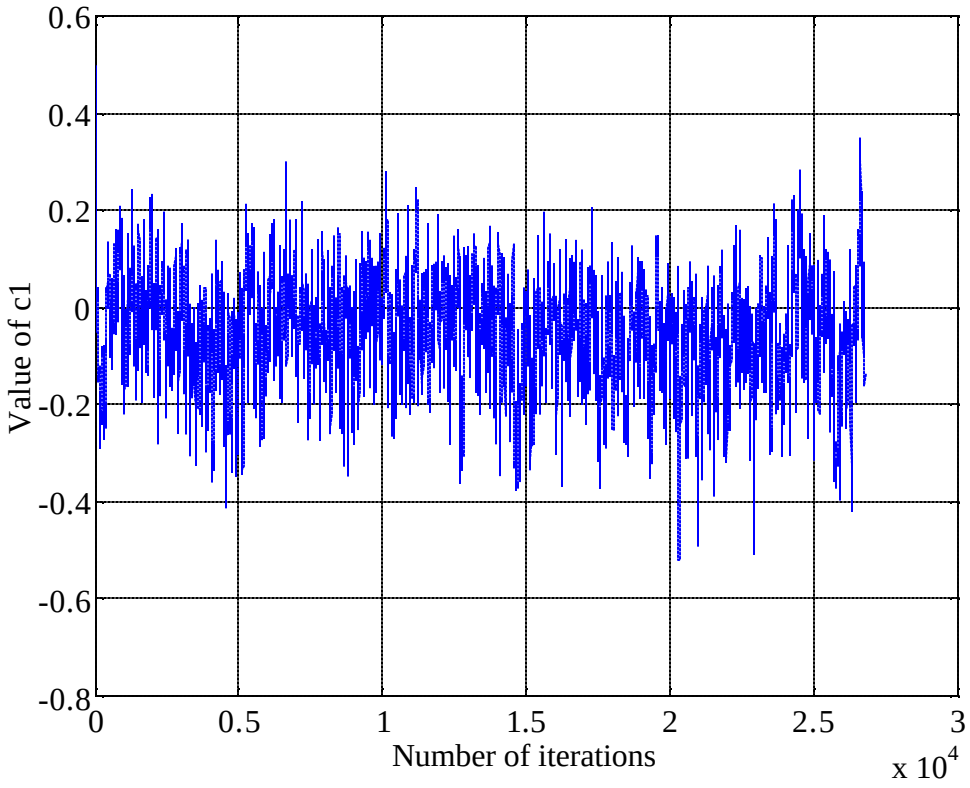

Fig.40. Plot between the value of c_1 and the number of iterations

It can be observed that the values in the parameter model tend to be within certain range after approximately 26000 iterations. This was confirmed by the experimental results for the estimation of parameters using arbitrary initial values. The results for this set of experiments are presented in Fig. 41-45. Thus for building up the ARMA model for 333.3 Hz sampling period mean of these values can be taken as the coefficients in (5-23). The resulting ARMA model used for prediction of the sensor data for the timeout case is given by

$$\hat{y}(t\,|\,q) = -0.0436\,\hat{y}(t-1\,|\,q) + 1.1777\,y(t-1) - 0.6859\,y(t-2) - 0.1315\,y(t-3) + 0.5960\,y(t-4) \qquad (5\text{-}27)$$

In this model the controller uses the previous four values of the sensor data and previous estimated value of sensor data to predict the current sensor data. It is assumed that there is no loss of data in the communication medium.

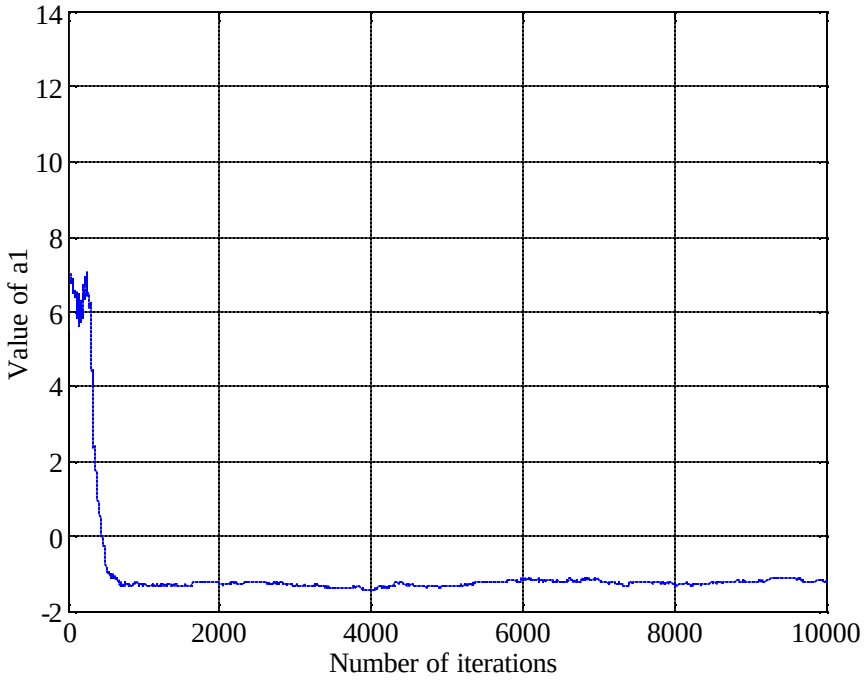

Fig.41. Plot between the value of a_1 and the number of iterations with arbitrary initial values of the parameters for ARMA model.

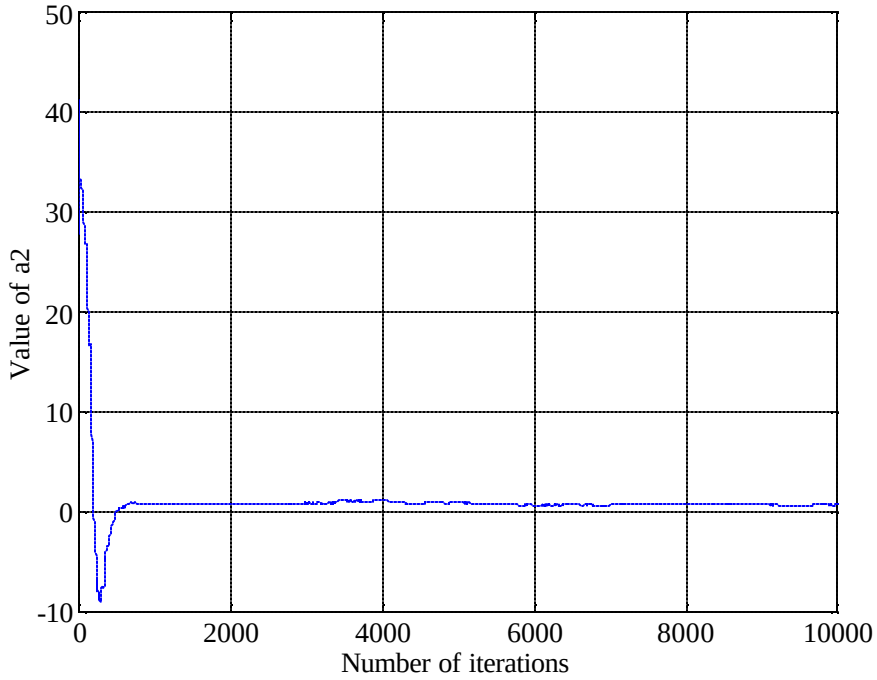

Fig.42. Plot between the value of a_2 and the number of iterations with arbitrary initial values of the parameters for ARMA model.

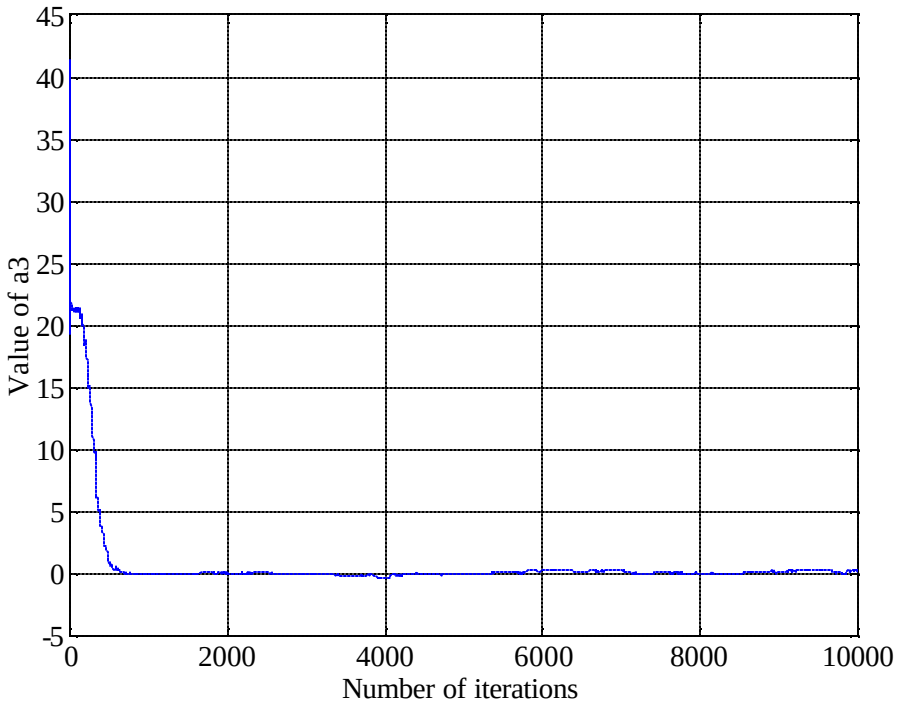

Fig.43. Plot between the value of a_3 and the number of iterations with arbitrary initial values of the parameters for ARMA model.

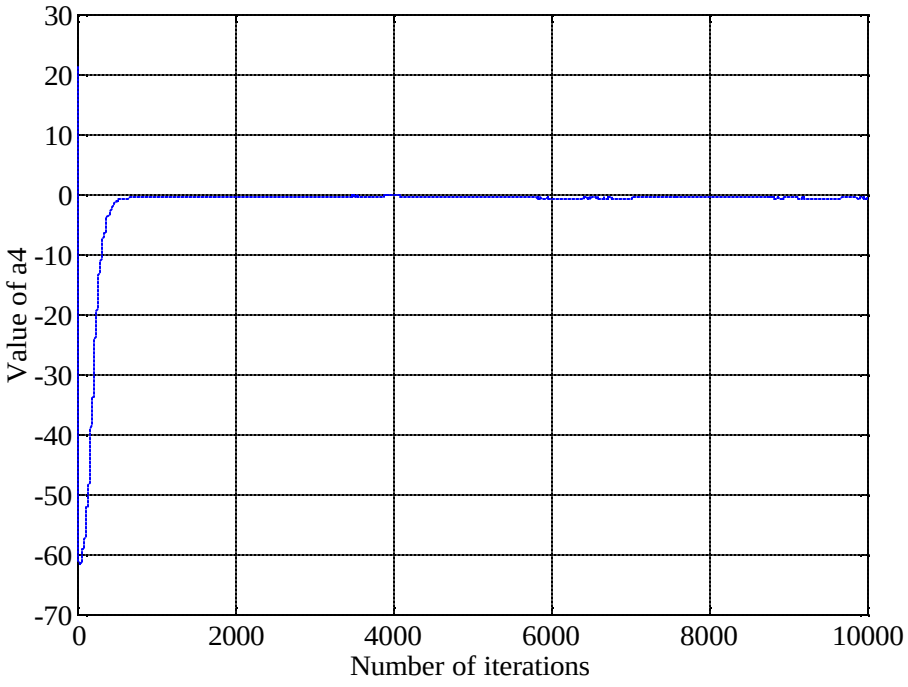

Fig.44. Plot between the value of a_4 and the number of iterations with arbitrary initial values of the parameters for ARMA model.

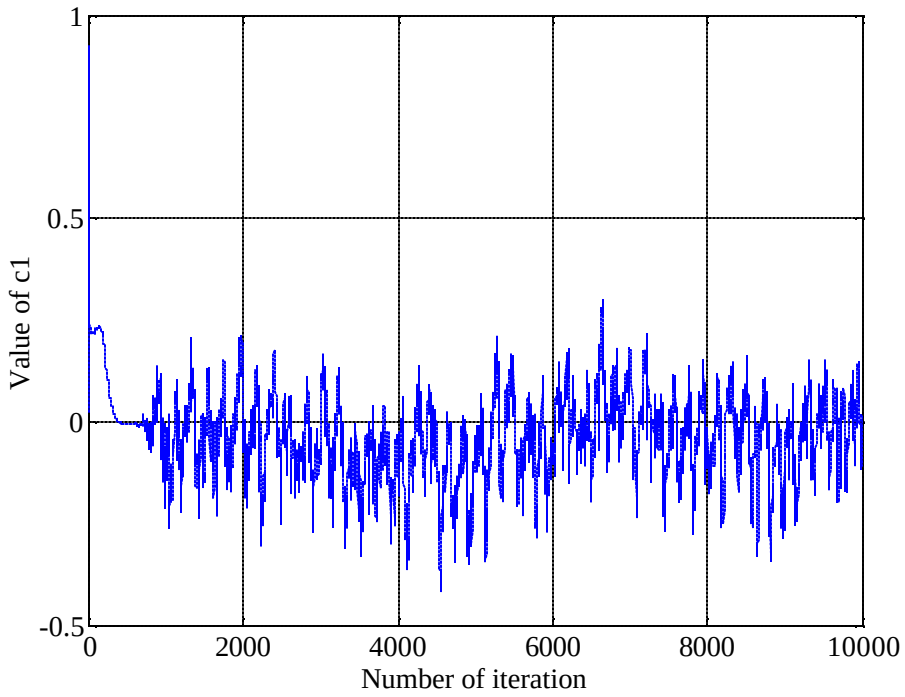

Fig.45. Plot between the value of c_1 and the number of iterations with arbitrary initial values of the parameters for ARMA model.

ARMA model given by (5-27) was implemented in the real-time control code running at the 333.3 Hz sampling frequency and experiments were performed to find out the stability of the system for various artificial delays. The timeout for these experiments were taken as the upper bound of the delays at this sampling frequency using Tables V and VI. It was observed that with the ARMA model and the timeout, the mean of time delay based on a uniform distribution was increased by approximately 6 %. Thus, the mean time delay that can be accommodated in the control code increased from 760 μs to approximately 805 μs. This can be attributed to the larger percentage error we are getting in predicting the sensor data as compared to AR model for the 333.3 Hz sampling frequency.

To accommodate more time delay in the control loop, the sampling frequency of 333.3 Hz is chosen. This facilitates us in having larger time duration before the timeout occurs in the control loop. On comparison between the AR model and the ARMA model for 333.3 Hz sampling frequency the AR model is chosen for the estimation of the sensor

data in the case of timeout. AR model is chose because the percentage error variation for this model is less than the percentage error variation for the ARMA model.

5.4 Performance Evaluation of the Selected Algorithm

The AR model selected was tested against the real round-trip time delay measured between werc-dhcp-1825tamu.edu and www.tamu.edu using the PING utility. The host werc-dhcp-1825.tamu.edu and the client www.tamu.edu were on different LANs. The probability distribution of the round-trip time delay is plotted in Fig. 46. The round-trip time delay was observed on Wednesday, Feb. 26, 2003 from 10:00 PM to 7:00 AM. It can be observed that mean round-trip time delay between the host and the client is about 851.8 μs. It can also be observed that there are sporadic rise in the round-trip time delay.

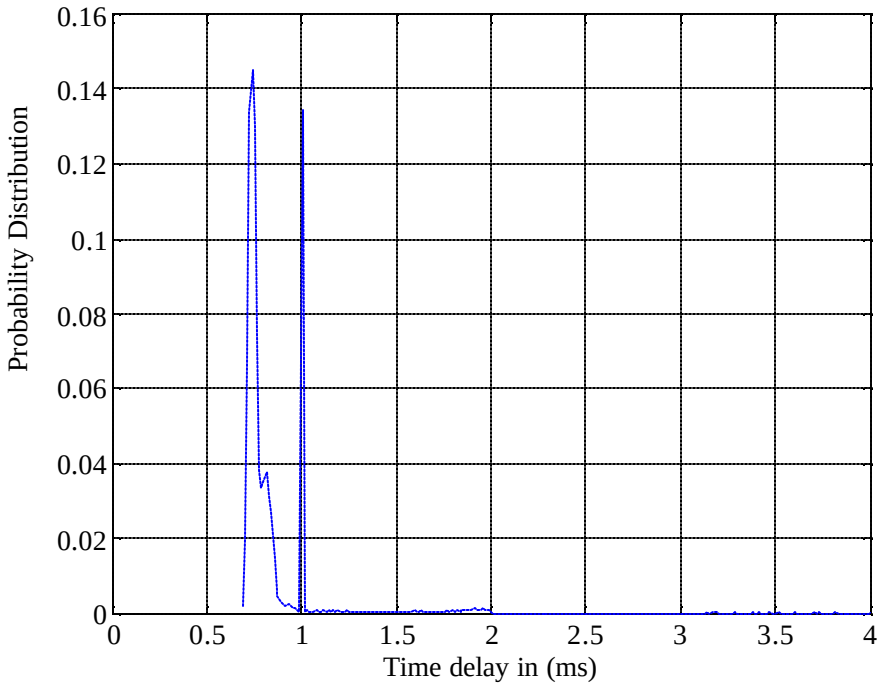

Fig.46. Probability distribution of round-trip time delay between werc-dhcp-1825.tamu.edu and www.tamu.edu on Wednesday Feb.26, 2003 from 10:00 PM to 7:00 AM.

For testing the performance of the AR model and timeout at a sampling period of 333.3 Hz, delays that were more than the sum of the sampling period and the timeout were removed from the actual data. The timeout for this experiment was taken as 1.3 ms. The modified data set was converted into an array and was downloaded on the controller board along with the control code.

The system was observed for its stability with the delays included in the control code. The test bed remained stable for approximately about 10 minutes. Thus, it can be concluded from the experiment that the sensor data estimation along with timeout can provide stability to the system under certain conditions. These conditions are:

1. The sensor data arriving at the controller node is all time stamped.
2. There is no loss of data in the network.
3. The mean time delay present in the network should be less than the upper bound for the time delay for the system.

The sensor data estimation technique for timeout cases maintains stability of the system for sporadic rise in the round-trip time delays. In addition to it the sporadic high time delays should always be less than the sum of one sampling period time and the timeout time. In other words the sensor data was unable to reach the controller node before the timeout should reach the controller node before the next time out occurs.

CHAPTER VI

CONCLUSIONS

6.1 Conclusions

The objective of this research is to demonstrate experimentally the feasibility of using the Internet for real-time control with single-actuator magnetic levitation (maglev) test bed. Two modes of control for the test bed via the Internet, namely supervisory control and feedback control have been studied. The supervisory control of the test bed via the Internet was successfully developed. Experiments were also conducted to test the stability of the existing controller in the presence of time-varying delays. Upper bounds of artificial time-varying delays that followed different probability distributions were estimated for the controller. Two sets of experiments were conducted to measure the round-trip time delay between PCs connected to the same LAN and PCs connected to different LAN. It is observed that the mean time delay in the two sets of experiments exceeds the upper bound of the time delay for the existing lead-lag controller. Thus, a new digital lead lag controller is designed directly in the digital domain to increase the upper bound of the time delay that can be accommodated in the control-loop. The upper bound of the time delay is measured for the digital controller running at the sampling frequencies of 500 Hz and 333.3 Hz. It is observed that the mean round-trip time delay is within the upper bound for the artificially generated time delay for the new digital controller.

However, the network had sporadic surges in round-trip time delay. These sporadic jitters in round-trip time delay are detrimental to the system stability. The system stability in this case was achieved by using a timeout for the controller. As soon as the timeout occurs for the controller, it predicts and uses the estimated sensor data to generate the control input. This control input is applied to the actuator within the same sampling period. The controller then uses the newest possible sensor data for the generation of the control input. This technique has been achieved using an AR model for the prediction of the sensor data for the timeout case. The estimation of the parameters of

the AR model for the sampling period is done using an off-line batch process. The sensor data are collected and arranged in the ascending order of their time stamp values. The data are then used iteratively for the estimation of the parameter vector for the AR model. It has been assumed that the sensor data arriving at the controller node are all time stamped.

The technique of using the timeout and the sensor data estimation based on an AR model was tested against the actual round-trip time delay data between two PCs. These two PCs were connected to different LANs and had a mean time delay less than the upper bound for the new digital controller. However, there were many time delays that were more than the wait time for timeout in the new digital controller. For maintaining the system stability time delays that were more than the sum of one sampling period and the timeout were removed from the data. With the implementation of timeout and sensor data estimation based on the AR model the mean time delay that can be accommodated in the control-loop increased from 760 μs to 836 μs. It was also observed that for the digital controller running at the sampling frequency of 333.3 Hz the system was stable with sporadic time delays in the order of 4.42 ms.

6.2 Ongoing Work

This methodology also assumes that there is no loss of data in the communication network when sensor data is traveling from the sensor node to the controller node.

Currently I am working on the software code to establish the real-time communication between the host PC and the client PC. It will help us verify the proposed methodology to accommodate sporadic surges in the round-trip time delay. The main challenge is to ensure there is no latency in transfer of data through the sockets. We are also working on the development of a general algorithm that can be used to maintain the system stability in the presence of sporadic surges in the time delay.

REFERENCES

[1] A. S. Tanenbaum, *Computer Networks.* Third edition. Prentice-Hall, Inc., Upper Saddle River, New Jersey, 2001.

[2] P. Gralla, *How The Internet Works.* sixth edition. QUE Publishing, Indianapolis, 2002.

[3] D. E. Comer, *Internetworking with TCP/IP*, third edition, vol. 1, Prentice-Hall, Inc., Upper Saddle River, New Jersey, 1995.

[4] 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.

[5] K. Goldeberg, S. Gentner, and J. Wiegley, *The Mercury Project: A Feasibility Study for Internet Robots,* Technical Report, UC Berkeley and University of Southern California, 1994.

[6] K. Taylor and J. Trevelyan, *A Telerobot on the World Wide Web*, Technical Report, Dept. of Mechanical and Materials Engineering, The University of Western Australia, W. Australia, 1995.

[7] P. G. Backes, K. S. Tso, and G. K. Tharp, "Mars Pathfinder Mission Internet based Operations Using WITS," in *Proc. IEEE Int. Conf. on Robotics and Automation*, vol. 1, May 1998, pp. 284–291.

[8] S. You, T. Wang, E. Roy, M. Cai, and Q. Zhang, "A low-cost Internet-based telerobotic system for access to remote laboratories," *Journal of Advanced Engineering Informatics*, vol. 15, no. 3, pp. 265–274, March 2001.

[9] R. J. Anderson and M. W. Spong, "Bilateral control of teleoperations with time delay," *IEEE Transactions on Automatic Control*, vol. 9, no. 5, 1989, pp. 494–501.

[10] S. Munir, *Internet-Based Teleoperation,* Ph.D. Dissertation, Department of Mechanical Engineering, Georgia Institute of Technology, Atlanta, GA., March 2001.

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

[12] P. Renton, P. Bender, S. Veldhuis, D. Renton, and M. A. Elbestawi, "Internet-Based Manufacturing Process optimization and Monitoring System," in *Proc. IEEE Int. Conf. on Robotics and Automation*, vol. 2, May 2002, pp. 1107–1112.

[13] R. Luck and A. Ray, "An observer-based compensator for distributed delays," *Automatica*, vol. 26, no. 5, pp. 903–908, September 1990.

[14] L.-W. Liou and A. Ray, "A stochastic regulator for integrated communication and control systems: Part I – formulation of control law," *ASME Journal of Dynamic Systems, Measurement and Control*, vol. 113, no. 4, pp. 604–611, December, 1991.

[15] A. Ray, "Output feedback control under randomly varying distributed delays," *Journal of Guidance, Control, and Dynamics*, vol.17, no. 4, pp. 701–711, August, 1994.

[16] R. Krotolica, U. Özgüner, H. Chan, H. Goktas, J. Winkelman, and M. Liubakka, "Stability of linear feedback systems with random communication delays," *International Journal of Control*, vol. 59, pp. 925–953, April 1994.

[17] H. Chan and U. Özgüner, "Closed-loop control of systems over a communications network with queues," *International Journal of Control*, vol. 62, no. 3, pp. 493–510, 1995.

[18] J. Nilsson, *Real-Time Control Systems with Delays*, Ph.D. Dissertation, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden, Jan. 1998.

[19] H. H. Woodson and J. R. Melcher, *Electromechanical Dynamics*, Malabar, Florida, Krieger Publishing Company, 1990.

[20] T. Boutell, *CGI Programming in C and Perl*, Addison-Wesley Developers Press, Reading, Massachusetts, 1995.

[21] A. Dumas, *Programming Winsock*, SAMS Publishing, Indianapolis, pp. 4–5,1995.

[22] W. Zhang, *Stability Analysis of Networked Control Systems*, Ph.D. Dissertation, Department of Electrical Engineering and Computer Science, Case Western Reserve University, Cleveland, Ohio, August 2001.

[23] V. S. Frost and B. Melamed, "Traffic modeling for telecommunications networks," *IEEE Communications Magazine*, vol. 32, no. 3, pp. 70–81, March 1994.

[24] D. Sanghi, A. K. Agrawala, Ó.Gunmundsson, and B. N. Jain, "Experimental assessment of end-to-end behavior on Internet," in *Proc. of IEEE Infocorn, '93*, vol. 3, March 1993, pp. 867–874.

[25] A. Papoulis, *Probability, Random Variables, and Stochastic Processes*, McGraw Hill Publication, New York, 1991.

[26] A. Leon-Garcia and I. Widjaja, *COMMUNICATION NETWORKS Fundamental Concepts and Key Architectures*, McGraw Hill Publication, Boston, Massachusetts, 2000.

[27] Modified Ping Utility [online]. Available, http://fgouget.free.fr/bing/index-en.shtml.

[28] A. Acharya and J. Saltz, *A Study of Internet Round-Trip Delay*, Technical Report CS-TR-3736, UMIACS and Department of Computer Science, University of Maryland, College Park, Dec.1996.

[29] H. Ohsaki, M. Morita, and M. Murata, "On modeling round-trip time dynamics of the Internet using system identification," *IEICE Transaction on Communication*, vol. E85-B, no. 1, pp. 1–9, January 2002.

[30] G. F. Franklin, D. J. Powell, and M. L. Workman, *Digital Control of Dynamic System*, second edition, Reading, Massachusetts, Addison-Wesley, 1990.

[31] L. Ljung and T. Söderström, *Theory and Practice of Recursive Identification*, The MIT Press, Cambridge, Massachusetts, 1983.

APPENDIX A

TEST.C

```
/*************************************************************************
#include <Brtenv.h>    /* Basic Real Time Environment*/
#include <math.h>
#include <io1104.h>

#define DT 3.0e-3        /* Interupt to execute every 2.0 miliseconds(500Hz)*/
#define INPUTS 4

   UInt16 scantable[4] = {1, 2, 3, 4};
   int i = 0;
   double v_hat_act[INPUTS];
   double y_hat_act = 0.0; /* From A/D convertor */
   double y_hat_desi = 0.0; /* user input (desired set point) */
   double v_hat_err = 0.0;
   double k=0.098;
   double v = 0.0;
   double tempv = 0.975;
  double u0 = 0.0;        /* Output from controller at time n */
  double u1 = 0.0;        /* Output from controller at time n-1 */
   double u2 = 0.0;        /* Output from controller at time n-2 */
   double er0 = 0.0;        /* Input to controller at time n */
   double er1 = 0.0;        /* Intput to controller at time n-1 */
   double er2 = 0.0;        /* Intput to controller at time n-2 */
   Float64 exec_time;   /* Execution Time */

void solve()
     {
          er0 = y_hat_desi-y_hat_act;  /* Error Calculation */
```

```
        er0=er0*k;
         u0 = 0.782*u1+0.13*u2-41500.0*er0+41500.0*1.754*er1-41500.0*0.769*er2;
                        /* Calculation of output from controller*/
      }

void result()
    {
        u2 = u1;
        u1 = u0;
        er2 = er1;
        er1 = er0;
              /* Book Keeping for next iteration */
      }

void isr_srt(void)     /* Timer Interrupt Service Routine */
    {
        RTLIB_SRT_ISR_BEGIN(); /*Overload Check has been removed*/
        ds1104_adc_read_mux(scantable, 4, v_hat_act);
         v_hat_act[0] = v_hat_act[0]*10;
         y_hat_act = -0.0004653*v_hat_act[0]+0.002525;
         solve();
         v_hat_err = u0 + v;        /* Vss= 1.018 */
         if(v_hat_err>10.0)
                {
                    v_hat_err = 10.0;
                 }
        if(v_hat_err<0.0)
                {
                     v_hat_err = 0.0;
                 }
         v_hat_err = v_hat_err*0.1;
```

```
        ds1104_dac_write(1, v_hat_err); /* set DACH1 to v_hat_err */
        result();
        ds1104_dac_strobe();
        RTLIB_SRT_ISR_END(); /*Overload Check has been removed */
    }

void main(void)
   {
       init();                  /* DS1104 and RTLib 1104 initilization */
       ds1104_adc_mux(1);
       ds1104_adc_delayed_start(DS1104_ADC1);
       ds1104_dac_init(DS1104_DACMODE_LATCHED);/* Init D/A in Latched mode
       */
       RTLIB_SRT_START(DT, isr_srt);
       while(1)
            {
               RTLIB_BACKGROUND_SERVICE();
                v = tempv;
            }
}
```

APPENDIX B

FINALTEST.C

```
/*****************************************************************************
#include <stdio.h>
#include <stdlib.h>

#include <cgic.h>
#include <clib32.h>

char board_name[DS_BOARD_NAME_LENGTH+1];
int cgiMain() {
        int result = cgiFormStringNoNewlines("board_name", board_name, 81);
        double  tempk, tempc, tempv;
        int     error, state;
        UInt32  value32;
        unsigned int  board_index;
        double  old_valuek64, new_valuek64, old_valueyha64;
        double  old_valuec64, new_valuec64;
        double  old_valuev64, new_valuev64;
        UInt32          k_add, y_hat_act_add, c_add, v_add;
        board_spec_tp board_spec;

#if DEBUG
    cgiReadEnvironment("/home/boutell/public_html/capcgi.dat");
#endif
    cgiHeaderContentType("text/html");

fprintf(cgiOut, "<HTML><HEAD>\n");
fprintf(cgiOut, "<TITLE>Controller Board</TITLE></HEAD>\n");
```

```
fprintf(cgiOut, "<BODY><H1>Calculations From The DS1104 Controller Board</H1>\n");

switch (result) {
        case cgiFormSuccess:
        fprintf(cgiOut, "Name fetched, result code: cgiFormSuccess<br>\n");
        break;
        case cgiFormTruncated:
        fprintf(cgiOut, "Name fetched, result code: cgiFormTruncated<br>\n");
        break;
        case cgiFormEmpty:
        fprintf(cgiOut, "Name fetched, result code: cgiFormEmpty<br>\n");
        break;
        case cgiFormNotFound:
        fprintf(cgiOut, "Name fetched, result code: cgiFormNotFound<br>\n");
        break;
        case cgiFormMemory:
        fprintf(cgiOut, "Name fetched, result code: cgiFormMemory<br>\n");
        break;
        default:
        fprintf(cgiOut, "Name fetched, unexpected result code: %s\n", result);
        break;
    }

    fprintf(cgiOut, "Name: %s<BR>\n", board_name);
    cgiFormDoubleBounded("tempk", &tempk, 0.0, 100, 0.0);
    fprintf(cgiOut, "New k value entered is %lf.<BR>\n", tempk);
    cgiFormDoubleBounded("tempc", &tempc, 0.0, 100, 0.0);
    fprintf(cgiOut, "New c value entered is %lf.<BR>\n", tempc);
    cgiFormDoubleBounded("tempv", &tempv, 0.0, 100, 0.0);
    fprintf(cgiOut, "New Vss value entered is %lf.<BR>\n", tempv);
```

```
/*****************************************************************/
error = DS_register_host_app("FINAL");
if(error != DS_NO_ERROR)
{
        printf("DS_register_host_app: error = %d\n",error);
        exit(1);
}
error = DS_board_index(board_name, &board_index);
if(error != DS_NO_ERROR)
{
        printf("A board named %s is not registered with the"
        " DSP Device Driver\n", board_name);
        DS_unregister_host_app();
        exit(1);
}
error = DS_board_spec(board_index, &board_spec);
if(error != DS_NO_ERROR)
{
        printf("DS_board_spec: error = %d\n",error);
        DS_unregister_host_app();
        exit(1);
}
printf("Board type = ");
switch(board_spec.board_type)
{
        case TYPE_DS1003:
                printf("DS1003<BR>\n");
                break;
        case TYPE_DS1005:
                printf("DS1005<BR>\n");
                break;
```

```
            case TYPE_DS1102:
                printf("DS1102<BR>\n");
                break;
            case TYPE_DS1103:
                printf("DS1103<BR>\n");
                break;
            case TYPE_DS1104:
                printf("DS1104<BR>\n");
                break;
            case TYPE_DS1401:
                printf("DS1401<BR>\n");
                break;
            default:
                printf("unknown.\n");
                DS_unregister_host_app();
                exit(1);
                break;
    }
    error = DS_is_reset(board_index, &state);
    if(error != DS_NO_ERROR)
    {
            printf("DS_is_reset: error = %d\n",error);
            DS_unregister_host_app();
            exit(1);
    }
if(state == DS_TRUE)
    {
            printf("No Application running on board %s.\n", board_name);
            DS_unregister_host_app();
            exit(1);
    }
```

```
/*****************************************************************/
error = DS_get_var_addr(board_index, "tempk", &k_add);
if(error != DS_NO_ERROR)
{
        printf("DS_get_var_addr: error = %d\n",error);
        DS_unregister_host_app();
        exit(1);
}
 switch(board_spec.board_type)
   {
        case TYPE_DS1003:
        case TYPE_DS1102:
                error = DS_read_32(board_index, k_add, 1, &value32);
                if(error != DS_NO_ERROR)
                {
                        printf("DS_read_32: error = %d\n",error);
                        DS_unregister_host_app();
                        exit(1);
                }
                old_valuek64 = (double)DS_cvt_ti_to_ieee(value32);
                break;
      case TYPE_DS1005:
      case TYPE_DS1103:
      case TYPE_DS1104:
      case TYPE_DS1401:
                error = DS_read_64(board_index, k_add, 1,
                (UInt64 *)&old_valuek64);
                if(error != DS_NO_ERROR)
                {
                        printf("DS_read_64: error = %d\n",error);
                        DS_unregister_host_app();
```

```
                    exit(1);
                }
                break;
}
 new_valuek64=tempk;
switch(board_spec.board_type)
{
        case TYPE_DS1003:
        case TYPE_DS1102:
                value32 = DS_cvt_ieee_to_ti((float)new_valuek64);
                error = DS_write_32(board_index, k_add, 1, &value32);
                if(error != DS_NO_ERROR)
                {
                        printf("DS_write_32: error = %d\n",error);
                        DS_unregister_host_app();
                        exit(1);
                }
                break;
        case TYPE_DS1005:
        case TYPE_DS1103:
        case TYPE_DS1104:
        case TYPE_DS1401:
                error = DS_write_64(board_index, k_add, 1,
                (UInt64 *)&new_valuek64);
                if(error != DS_NO_ERROR)
                {
                        printf("DS_write_64: error = %d\n",error);
                        DS_unregister_host_app();
                        exit(1);
                }
                break;
```

```
}
printf("New k value is  %f\n <Br>", new_valuek64);
/*****************************************************************/
error = DS_get_var_addr(board_index, "tempc", &c_add);
if(error != DS_NO_ERROR)
{
        printf("DS_get_var_addr: error = %d\n",error);
        DS_unregister_host_app();
        exit(1);
}
 switch(board_spec.board_type)
{
        case TYPE_DS1003:
        case TYPE_DS1102:
                error = DS_read_32(board_index, c_add, 1, &value32);
                if(error != DS_NO_ERROR)
                {
                        printf("DS_read_32: error = %d\n",error);
                        DS_unregister_host_app();
                        exit(1);
                }
                old_valuec64 = (double)DS_cvt_ti_to_ieee(value32);
                break;
          case TYPE_DS1005:
          case TYPE_DS1103:
          case TYPE_DS1104:
          case TYPE_DS1401:
                 error = DS_read_64(board_index, c_add, 1,
                 (UInt64 *)&old_valuec64);
                  if(error != DS_NO_ERROR)
                  {
```

```
                    printf("DS_read_64: error = %d\n",error);
                    DS_unregister_host_app();
                     exit(1);
                  }
                  break;
   }
   new_valuec64=tempc;
switch(board_spec.board_type)
   {
          case TYPE_DS1003:
          case TYPE_DS1102:
                value32 = DS_cvt_ieee_to_ti((float)new_valuec64);
                 error = DS_write_32(board_index, c_add, 1, &value32);
                  if(error != DS_NO_ERROR)
                {
                    printf("DS_write_32: error = %d\n",error);
                    DS_unregister_host_app();
                    exit(1);
                }
                break;
          case TYPE_DS1005:
          case TYPE_DS1103:
          case TYPE_DS1104:
          case TYPE_DS1401:
                error = DS_write_64(board_index, c_add, 1,
                (UInt64 *)&new_valuec64);
          if(error != DS_NO_ERROR)
          {
                printf("DS_write_64: error = %d\n",error);
                DS_unregister_host_app();
                exit(1);
```

```
                }
                break;
    }
    printf("New c value is  %f\n <Br>", new_valuec64);
 /*****************************************************************/
error = DS_get_var_addr(board_index, "tempv", &v_add);
if(error != DS_NO_ERROR)
    {
       printf("DS_get_var_addr: error = %d\n",error);
       DS_unregister_host_app();
       exit(1);
    }
 switch(board_spec.board_type)
    {
       case TYPE_DS1003:
       case TYPE_DS1102:
            error = DS_read_32(board_index, v_add, 1, &value32);
             if(error != DS_NO_ERROR)
                {
                     printf("DS_read_32: error = %d\n",error);
                     DS_unregister_ho st_app();
                     exit(1);
                }
                old_valuek64 = (double)DS_cvt_ti_to_ieee(value32);
                break;
       case TYPE_DS1005:
       case TYPE_DS1103:
       case TYPE_DS1104:
       case TYPE_DS1401:
              error = DS_read_64(board_index, v_add, 1,
                   (UInt64 *)&old_valuev64);
```

```
                if(error != DS_NO_ERROR)
                {
                        printf("DS_read_64: error = %d\n",error);
                        DS_unregister_host_app();
                        exit(1);
                }
                break;
  }
new_valuev64=tempv;
switch(board_spec.board_type)
  {
     case TYPE_DS1003:
     case TYPE_DS1102:
          value32 = DS_cvt_ieee_to_ti((float)new_valuev64);
          error = DS_write_32(board_index, v_add, 1, &value32);
          if(error != DS_NO_ERROR)
           {
               printf("DS_write_32: error = %d\n",error);
               DS_unregister_host_app();
               exit(1);
           }
           break;
     case TYPE_DS1005:
     case TYPE_DS1103:
     case TYPE_DS1104:
     case TYPE_DS1401:
          error = DS_write_64(board_index, v_add, 1,
                    (UInt64 *)&new_valuev64);
          if(error != DS_NO_ERROR)
               {
                    printf("DS_write_64: error = %d\n",error);
```

```
                    DS_unregister_host_app();
                    exit(1);
                }
                break;
    }


    printf("New Vss value is  %f\n <Br>", new_valuev64);
  /*****************************************************************/
error = DS_get_var_addr(board_index, "y_hat_act", &y_hat_act_add);
if(error != DS_NO_ERROR)
   {
      printf("DS_get_var_addr: error = %d\n",error);
      DS_unregister_host_app();
      exit(1);
    }
    switch(board_spec.board_type)
    {
       case TYPE_DS1003:  /* DSP based, 32 bit variable */
       case TYPE_DS1102:
          error = DS_read_32(board_index, y_hat_act_add, 1, &value32);
          if(error != DS_NO_ERROR)
          {
             printf("DS_read_32: error = %d\n",error);
             DS_unregister_host_app();
             exit(1);
          }
         old_valueyha64 = (double)DS_cvt_ti_to_ieee(value32);
          break;
       case TYPE_DS1005:  /* PPC based, 64 bit variable */
       case TYPE_DS1103:
```

```
        case TYPE_DS1104:
        case TYPE_DS1401:
            error = DS_read_64(board_index, y_hat_act_add, 1,
                        (UInt64 *)&old_valueyha64);
            if(error != DS_NO_ERROR)
            {
               printf("DS_read_64: error = %d\n",error);
               DS_unregister_host_app();
               exit(1);
            }
            break;
    }
printf("new value of y hat actual is %lf (m) from its equilibrium position\n<BR>",
old_valueyha64);
/*************************************************************************/
    DS_unregister_host_app();
   fprintf(cgiOut, "</BODY></HTML>\n");
          return 0;
}
```

APPENDIX C

TEST1.HTML

```
/*************************************************************************

<html>
<body>
<form action="/cgi-bin/finaltest.exe" method="POST">
<p>My name of controller board: <INPUT TYPE="text" NAME="board_name">
<p>My input variable k: <INPUT TYPE="text" NAME="tempk">
<p>My input variable c: <INPUT TYPE="text" NAME="tempc">
<p>My input variable for step input change: <INPUT TYPE="text" NAME="tempv">
<p><input type="SUBMIT" name="Submit" value="Submit"></p>
</form>
</body>
</html
```

APPENDIX D

SERVER.C

```
/*********************************************************************/
/* to be run on werc-dhcp-1825.tamu.edu*/
/*********************************************************************/
#include <windows.h>
#include <winsock.h>

#include <stdio.h>
#include <stdlib.h>
#include <clib32.h>                    /* prototypes of Device Driver CLIB functions */

#define PORTNUM              5000              // Port number
#define MAX_PENDING_CONNECTS  4        // Maximum length of the queue
                                                  // of pending connections

char board_name[DS_BOARD_NAME_LENGTH+1];
int main()
  {
        /*************************************************************/
        //for winscok
        /*************************************************************/
        int index = 0,                    // Integer index
        iReturn;                          // Return value of recv function
        float v_act[1];                   // the data in float type to be sent
        float v_err[1];                   // data to be recieved
        TCHAR szError[100];               //Error message string
        double vhatact=0.0,vhaterr=0.0;
        SOCKET WinSocket = INVALID_SOCKET,  // Window socket
        ClientSock = INVALID_SOCKET,                // Socket for communicating
```

```
                                                    // between the server and client
DuplicateSock = INVALID_SOCKET;
SOCKADDR_IN local_sin,                              // Local socket address
accept_sin;                                         // Receives the address of the
                                                    // connecting entity
int accept_sin_len;                                 // Length of accept_sin
WSADATA WSAData;                                    // Contains details of the Winsock
                                                    // implementation
/*****************************************************************/
//for the CLIB
/*****************************************************************/
int             error, state;
unsigned int  board_index;
UInt16          board_type;
UInt32          vhatact_add, vhatact_val32;
UInt32          vhaterr_add, vhaterr_val32;
UInt32          flag_add, flag_val32;
board_spec_tp board_spec;
 /*****************************************************************/
//For the Initialization of clib
/*****************************************************************/
error = DS_register_host_app("SERVER");
if(error != DS_NO_ERROR)
        {
                printf("DS_register_host_app: error = %d\n",error);
                exit(1);
        }
printf("Please enter a board name : ");
scanf("%s",&board_name);
/* Before accessing a processor board, the board index of that board   */
/* must be obtained from the DSP Device Driver. The board index is a   */
```

```
/* parameter for all CLIB functions accessing the board.          */
error = DS_board_index(board_name, &board_index);

if(error != DS_NO_ERROR)
        {
                printf("A board named %s is not registered with the DSP Device
                Driver\n", board_name);
                DS_unregister_host_app();
                exit(1);
        }
/* In order to get more information about the selected board call the  */
/* CLIB function DS_board_spec(). That function fills a board_spec     */
/* structure with information like board type, board name, memory size */
/* etc.                                                                */
error = DS_board_spec(board_index, &board_spec);
if(error != DS_NO_ERROR)
        {
                printf("DS_board_spec: error = %d\n",error);
                DS_unregister_host_app();
                exit(1);
        }
printf("Board type = ");
switch(board_spec.board_type)
        {
                case TYPE_DS1003:
                        printf("DS1003\n");
                        break;
                case TYPE_DS1005:
                        printf("DS1005\n");
                        break;
                case TYPE_DS1102:
```

```
                printf("DS1102\n");
                break;
        case TYPE_DS1103:
                printf("DS1103\n");
                break;
        case TYPE_DS1104:
                printf("DS1104\n");
                break;
        case TYPE_DS1401:
                printf("DS1401\n");
                break;
        default:
                printf("unknown.\n");
                DS_unregister_host_app();
                exit(1);
        break;
    }

/* Before accessing a board it is required to check if an application  */
/* is running on that board. A board access is based on command server,*/
/* which is a part of the real-time application background.           */

error = DS_is_reset(board_index, &state);
if(error != DS_NO_ERROR)
    {
        printf("DS_is_reset: error = %d\n",error);
        DS_unregister_host_app();
        exit(1);
    }
 if(state == DS_TRUE)
    {
```

```
        printf("No Application running on board %s.\n", board_name);
        DS_unregister_host_app();
        exit(1);
}

/*****************************************************************/
//For the winsock activation
/*****************************************************************/
// Initialize Winsock.
if (WSAStartup (MAKEWORD(1,1), &WSAData) != 0)
{
        wsprintf (szError, TEXT("WSAStartup failed. Error: %d"),
        WSAGetLastError ());
        MessageBox (NULL, szError, TEXT("Error"), MB_OK);
}
// Create a TCP/IP socket, WinSocket.
if ((WinSocket = socket (AF_INET, SOCK_STREAM, 0)) ==
INVALID_SOCKET)
{
        wsprintf (szError, TEXT("Allocating socket failed. Error:
        %d"),WSAGetLastError ());
        MessageBox (NULL, szError, TEXT("Error"), MB_OK);
}
// Fill out the local socket's address information.
local_sin.sin_family = AF_INET;
local_sin.sin_port = htons (PORTNUM);
local_sin.sin_addr.s_addr = htonl (INADDR_ANY);
// Associate the local address with WinSocket.
if (bind (WinSocket,(struct sockaddr *) &local_sin, sizeof (local_sin)) ==
SOCKET_ERROR)
{
```

```
        wsprintf (szError, TEXT("Binding socket failed. Error:
        %d"),WSAGetLastError ());
        MessageBox (NULL, szError, TEXT("Error"), MB_OK);
        closesocket (WinSocket);
}
// Establish a socket to listen for incoming connections.
if (listen (WinSocket, MAX_PENDING_CONNECTS) == SOCKET_ERROR)
{
        wsprintf (szError,TEXT("Listening to the client failed. Error:
        %d"),WSAGetLastError ());
        MessageBox (NULL, szError, TEXT("Error"), MB_OK);
        closesocket (WinSocket);
}
accept_sin_len = sizeof (accept_sin);
// Accept an incoming connection attempt on WinSocket.
ClientSock = accept (WinSocket,(struct sockaddr *) &accept_sin, (int *)
&accept_sin_len);
// Stop listening for connections from clients.
// closesocket (WinSocket);
if (ClientSock == INVALID_SOCKET)
{
         wsprintf (szError, TEXT("Accepting connection with client failed.")
        TEXT(" Error: %d"), WSAGetLastError ());
        MessageBox (NULL, szError, TEXT("Error"), MB_OK);
}
/*****************************************************************/
// infinte loop to collect data from the controller board and send to server
/*****************************************************************/
```

```
for (;;)
{
        /*************************************************************/
        //collecting the data from the controller board for the flag
        /*************************************************************/
        /* The function DS_get_var_addr returns the memory address of a     */
        /* variable given as parameter.To get another variable address, call*/
        /* the function with the name of that variable as second parameter. */
        error = DS_get_var_addr(board_index, "temp_flag", &flag_add);
        if(error != DS_NO_ERROR)
        {
                printf("DS_get_var_addr: error = %d\n",error);
                DS_unregister_host_app();
                exit(1);
        }
        /* The implementation of the variable depends on the board          */
        /* type. On DSP based real time processor boards it is implemented as*/
        /* 32-bit floating-point. On PPC based real time processor boards it */
        /* is implemented as 64-bit floating-point.                         */
        switch(board_spec.board_type)
        {
                case TYPE_DS1003:  /* DSP based, 32 bit variable */
                case TYPE_DS1102:
                        error = DS_read_32(board_index, flag_add, 1,
                        &flag_val32);
                        if(error != DS_NO_ERROR)
                         {
                                printf("DS_read_32: error = %d\n",error);
                                DS_unregister_host_app();
                                exit(1);
                         }
```

```
            /* convert form TMS320 single precision float to IEEE 754 */
            /* single precision float.                          */
                    flag = (double)DS_cvt_ti_to_ieee(flag_val32);
                     break;
            case TYPE_DS1005:  /* PPC based, 64 bit variable */
            case TYPE_DS1103:
            case TYPE_DS1104:
            case TYPE_DS1401:
                    error = DS_read_64(board_index, flag_add, 1,(UInt64
                    *)&flag);
                    if(error != DS_NO_ERROR)
                     {
                            printf("DS_read_64: error = %d\n",error);
                            DS_unregister_host_app();
                            exit(1);
                     }
                             break;
   }                // End of switch
if( flag = = 1 )            // Checking flag to see if the next sample period has
                            // started or not
{
   /*************************************************************/
   //recieving the data through the winsock
   /*************************************************************/
     // Receive data from the client.
     iReturn = recv (ClientSock, v_err, sizeof (v_err), 0);
     // Check if there is any data received. If there is, display it.
     if (iReturn == SOCKET_ERROR)
      {
            wsprintf (szError, TEXT("No data is received, recv failed.")
            TEXT(" Error: %d"),  WSAGetLastError ());
```

```
            MessageBox (NULL, szError, TEXT("Server"), MB_OK);
            break;
      }
     else if (iReturn == 0)
      {
            MessageBox (NULL, TEXT("Finished receiving data"),
            TEXT("Server"),MB_OK);
            break;
      }
    vhaterr=v_err[0];
/*************************************************************/
//transferring the data to the controller board
/*************************************************************/
error = DS_get_var_addr(board_index, "temp_vhaterr", &vhaterr_add);
if(error != DS_NO_ERROR)
{
            printf("DS_get_var_addr: error = %d\n",error);
            DS_unregister_host_app();
            exit(1);
}
/* The implementation of the variable depends on the board          */
/* type. On DSP based real time processor boards it is implemented as*/
/* 32-bit floating-point. On PPC based real time processor boards it */
/* is implemented as 64-bit floating-point.                         */
switch(board_spec.board_type)
{
case TYPE_DS1003:
case TYPE_DS1102:
            /* convert from IEEE 754 single precision float to TMS320 */
            /* single precision float.                              */
            vhaterr_val32 = DS_cvt_ieee_to_ti((float)vhaterr);
```

```
                error = DS_write_32(board_index, vhaterr_add, 1,
                &vhaterr_val32);
                if(error != DS_NO_ERROR)
                        {
                                printf("DS_write_32: error = %d\n",error);
                                DS_unregister_host_app();
                                exit(1);
                        }
                break;
        case TYPE_DS1005:
        case TYPE_DS1103:
        case TYPE_DS1104:
        case TYPE_DS1401:
                error = DS_write_64(board_index, vhaterr_add, 1,(UInt64
                *)&vhaterr);
                if(error != DS_NO_ERROR)
                        {
                                printf("DS_write_64: error = %d\n",error);
                                DS_unregister_host_app();
                                exit(1);
                        }
                break;
        }               // End of Switch
/*****************************************************************/
//collecting the data from the controller board
/*****************************************************************/
/* The function DS_get_var_addr returns the memory address of a     */
/* variable given as parameter.To get another variable address, call*/
/* the function with the name of that variable as second parameter. */
error = DS_get_var_addr(board_index, "temp_vhatact", &vhatact_add);
if(error != DS_NO_ERROR)
```

```
        {
                printf("DS_get_var_addr: error = %d\n",error);
                DS_unregister_host_app();
                exit(1);
        }
/* The implementation of the variable depends on the board          */
/* type. On DSP based real time processor boards it is implemented as*/
/* 32-bit floating-point. On PPC based real time processor boards it */
/* is implemented as 64-bit floating-point.                         */
switch(board_spec.board_type)
        {
        case TYPE_DS1003:  /* DSP based, 32 bit variable */
        case TYPE_DS1102:
                error = DS_read_32(board_index, vhatact_add, 1,
                &vhatact_val32);
                        if(error != DS_NO_ERROR)
                         {
                                printf("DS_read_32: error = %d\n",error);
                                DS_unregister_host_app();
                                exit(1);
                         }
                /* convert form TMS320 single precision float to IEEE 754 */
                /* single precision float.                          */
                vhatact = (double)DS_cvt_ti_to_ieee(vhatact_val32);
                         break;
                case TYPE_DS1005:  /* PPC based, 64 bit variable */
                case TYPE_DS1103:
                case TYPE_DS1104:
                case TYPE_DS1401:
                        error = DS_read_64(board_index, vhatact_add, 1,(UInt64
                        *)&vhatact);
```

```
                        if(error != DS_NO_ERROR)
                        {
                                printf("DS_read_64: error = %d\n",error);
                                DS_unregister_host_app();
                                exit(1);
                        }
                                break;
                }               // End of Switch
        v_act[0]=vhatact;
        /*****************************************************************/
        //sending the data through the winsock
        /*****************************************************************/
        // Send the data from the server to the client.
        if (send (ClientSock, v_act, sizeof(v_act), 0)== SOCKET_ERROR)
          {
                wsprintf (szError, TEXT("Sending data to the client failed. Error: %d"),
                WSAGetLastError ());
                MessageBox (NULL, szError, TEXT("Error"), MB_OK);
          }
            }       // End of Flag
  }                 // End of while loop
shutdown (ClientSock, 0x02);    // Disable both sending and receiving on ClientSock.
closesocket (ClientSock);               // Close ClientSock.
WSACleanup ();
 return TRUE;
}
```

APPENDIX E

CLIENT.C

```
/*************************************************************************
#include <windows.h>
#include <winsock.h>

#include <stdio.h>
#include <stdlib.h>

#include <clib32.h>                    // prototypes of Device
                                       // Driver CLIB functions

#define PORTNUM        5000            // Port number
#define HOSTNAME  "werc-dhcp-1825.tamu.edu"    // Server name string
                                       // This should be changed
                                       // according to the server.

char board_name[DS_BOARD_NAME_LENGTH+1];
unsigned long int j = 0;
void main(void)
{
        /*************************************************************/
        //for winscok
        /*************************************************************/
        int index = 0,                 // Integer index
        iReturn;                               // Return value of recv function
        double v_act[1];                       // data to be sent through winsock
        double v_err[1];                  // data to be recieved through winsock
        TCHAR szError[100];                    // Error message string
        double vhatact=0.0,vhaterr=0.0;
```

```
SOCKET ServerSock = INVALID_SOCKET;   // Socket bound to the server
SOCKADDR_IN destination_sin;                    // of the server
WSADATA WSAData;                  // Contains details of the
                                  // Winsocket implementation
PHOSTENT phostent = NULL;                                   // Points to the
HOSTENT structure
                                                            // of the server

/*****************************************************************/
//for the CLIB
/*****************************************************************/
int          error, state;
unsigned int  board_index;
UInt16       board_type;
UInt32       vhatact_add, vhatact_val32;
UInt32       vhaterr_add, vhaterr_val32;
board_spec_tp board_spec;
/*****************************************************************/
//For the Initialization of clib
/*****************************************************************/
error = DS_register_host_app("CLIENT");
if(error != DS_NO_ERROR)
{
        printf("DS_register_host_app: error = %d\n",error);
        exit(1);
}
printf("Please enter a board name : ");
scanf("%s",&board_name);
/* Before accessing a processor board, the board index of that board   */
/* must be obtained from the DSP Device Driver. The board index is a   */
/* parameter for all CLIB functions accessing the board.               */
```

```
error = DS_board_index(board_name, &board_index);
if(error != DS_NO_ERROR)
{
        printf("A board named %s is not registered with the DSP Device
        Driver\n", board_name);
        DS_unregister_host_app();
        exit(1);
}

/* In order to get more information about the selected board call the  */
/* CLIB function DS_board_spec(). That function fills a board_spec     */
/* structure with information like board type, board name, memory size */
/* etc.                                                                */
error = DS_board_spec(board_index, &board_spec);
if(error != DS_NO_ERROR)
{
        printf("DS_board_spec: error = %d\n",error);
        DS_unregister_host_app();
        exit(1);
}
printf("Board type = ");
switch(board_spec.board_type)
{
        case TYPE_DS1003:
                printf("DS1003\n");
                break;
        case TYPE_DS1005:
                printf("DS1005\n");
                break;
        case TYPE_DS1102:
                printf("DS1102\n");
```

```
                break;
        case TYPE_DS1103:
                printf("DS1103\n");
                break;
        case TYPE_DS1104:
                printf("DS1104\n");
                break;
        case TYPE_DS1401:
                printf("DS1401\n");
                break;
        default:
                printf("unknown.\n");
                DS_unregister_host_app();
                exit(1);
        break;
}
/* Before accessing a board it is required to check if an application  */
/* is running on that board. A board access is based on command server,*/
/* which is a part of the real-time application background.          */
error = DS_is_reset(board_index, &state);
if(error != DS_NO_ERROR)
{
        printf("DS_is_reset: error = %d\n",error);
        DS_unregister_host_app();
        exit(1);
}
if(state == DS_TRUE)
{
        printf("No Application running on board %s.\n", board_name);
        DS_unregister_host_app();
        exit(1);
```

```
}


/*****************************************************************/
//For the winsock activation
/*****************************************************************/
// Initialize Winsocket.
if (WSAStartup (MAKEWORD(1,1), &WSAData) != 0)
{
        wsprintf (szError, TEXT("WSAStartup failed. Error:
        %d"),WSAGetLastError ());
        MessageBox (NULL, szError, TEXT("Error"), MB_OK);
}
// Create a TCP/IP socket that is bound to the server.
if ((ServerSock = socket (AF_INET, SOCK_STREAM, 0)) ==
INVALID_SOCKET)
{
        wsprintf (szError, TEXT("Allocating socket failed. Error:
        %d"),WSAGetLastError ());
        MessageBox (NULL, szError, TEXT("Error"), MB_OK);
}
// Fill out the server socket's address information.
destination_sin.sin_family = AF_INET;
// Retrieve the host information corresponding to the host name.
if ((phostent = gethostbyname (HOSTNAME)) == NULL)
{
        wsprintf (szError, TEXT("Unable to get the host name. Error:
        %d"),WSAGetLastError ());
        MessageBox (NULL, szError, TEXT("Error"), MB_OK);
        closesocket (ServerSock);
}
```

```
// Assign the socket IP address.
memcpy ((char FAR *)&(destination_sin.sin_addr),phostent->h_addr,phostent-
>h_length);
// Convert to network ordering.
destination_sin.sin_port = htons (PORTNUM);
// Establish a connection to the server socket.
if (connect(ServerSock,(PSOCKADDR) &destination_sin,sizeof
(destination_sin)) == SOCKET_ERROR)
{
        wsprintf (szError,TEXT("Connecting to the server failed. Error:
        %d"),WSAGetLastError ());
        MessageBox (NULL, szError, TEXT("Error"), MB_OK);
        closesocket (ServerSock);
 }
/*****************************************************************/
// infinte loop to collect datd from the controller board and send to server
/*****************************************************************/

for (;;)
{
        /*************************************************************/
        //collecting the data from the controller board
        /*************************************************************/
        error = DS_get_var_addr(board_index, "temp_vhaterr", &vhaterr_add);
        if(error != DS_NO_ERROR)
        {
                printf("DS_get_var_addr: error = %d\n",error);
                DS_unregister_host_app();
                exit(1);
        }
        printf(" step 1 \n");   // to be removed just for check
```

```
/* The implementation of the variable depends on the board          */
/* type. On DSP based real time processor boards it is implemented as*/
/* 32-bit floating-point. On PPC based real time processor boards it */
/* is implemented as 64-bit floating-point.                         */
switch(board_spec.board_type)
        {
                case TYPE_DS1003:  /* DSP based, 32 bit variable */
                case TYPE_DS1102:
                        error = DS_read_32(board_index, vhaterr_add, 1,
                        &vhaterr_val32);
                        if(error != DS_NO_ERROR)
                                {
                                        printf("DS_read_32: error =
                                        %d\n",error);
                                        DS_unregister_host_app();
                                        exit(1);
                                }
         /* convert form TMS320 single precision float to IEEE 754 */
        /* single precision float.                          */

                        vhaterr =
                        (double)DS_cvt_ti_to_ieee(vhaterr_val32);
                        break;
                case TYPE_DS1005:  /* PPC based, 64 bit variable */
                case TYPE_DS1103:
                case TYPE_DS1104:
                case TYPE_DS1401:
                                error = DS_read_64(board_index,
                                vhaterr_add, 1,(UInt64 *)&vhaterr);
                                if(error != DS_NO_ERROR)
                                {
```

```
                                        printf("DS_read_64: error =
                                        %d\n",error);
                                        DS_unregister_host_app();
                                        exit(1);
                                }
                                break;
                        }
                printf(" step 2 \n"); // to be removed just for check
                v_err[0]=vhaterr;
/*************************************************************/
//sending the data through the winsock
/*************************************************************/
// Send the data to the server.
if (send (ServerSock, v_err, sizeof (v_err), 0)== SOCKET_ERROR)
        {
                wsprintf (szError,TEXT("Sending data to the server failed.
                Error: %d"),WSAGetLastError ());
                MessageBox (NULL, szError, TEXT("Error"), MB_OK);
        }
/*************************************************************/
// The program stays at this point till it again recieves value from the
// server
// Receive data from the server socket.
/*************************************************************/
iReturn = recv (ServerSock, v_act, sizeof (v_act), 0);
vhatact = v_act[0];
// Check if there is any data received. If there is, display it.
if (iReturn == SOCKET_ERROR)
{
        wsprintf (szError, TEXT("No data is received, recv failed.")
        TEXT(" Error: %d"),WSAGetLastError ());
```

```
        MessageBox (NULL, szError, TEXT("Client"), MB_OK);
        break;
}

else if (iReturn == 0)
{
        MessageBox (NULL, TEXT("Finished receiving data"),
        TEXT("Client"),MB_OK);
        break;
}
/*************************************************************/
//transferring the data to the controller board
/*************************************************************/
/* The function DS_get_var_addr returns the memory address of a     */
/* variable given as parameter.To get another variable address, call*/
/* the function with the name of that variable as second parameter. */
error = DS_get_var_addr(board_index, "temp_vhatact", &vhatact_add);
if(error != DS_NO_ERROR)
        {
                printf("DS_get_var_addr: error = %d\n",error);
                DS_unregister_host_app();
                exit(1);
        }
/* The implementation of the variable depends on the board          */
/* type. On DSP based real time processor boards it is implemented as*/
/* 32-bit floating-point. On PPC based real time processor boards it */
/* is implemented as 64-bit floating-point.                         */
switch(board_spec.board_type)
{
        case TYPE_DS1003:
        case TYPE_DS1102:
```

```
/* convert from IEEE 754 single precision float to TMS320 */
/* single precision float.                          */
        vhatact_val32 = DS_cvt_ieee_to_ti((float)vhatact);
        error = DS_write_32(board_index, vhatact_add, 1,
        &vhatact_val32);
        if(error != DS_NO_ERROR)
                {
                        printf("DS_write_32: error = %d\n",error);
                        DS_unregister_host_app();
                        exit(1);
                }
                        break;
      case TYPE_DS1005:
      case TYPE_DS1103:
      case TYPE_DS1104:
      case TYPE_DS1401:
                error = DS_write_64(board_index, vhatact_add,
                1,(UInt64 *)&vhatact);
                if(error != DS_NO_ERROR)
                {
                        printf("DS_write_64: error = %d\n",error);
                        DS_unregister_host_app();
                        exit(1);
                }
                break;
        }
printf(" the value of v hat act is %f\n",vhatact);
 while (j<10000)
{
        j = j+1;
 }
```

```
                    j = 0;
        }           // End of While loop

  //shutdown (ServerSock, 0x00);    Disable receiving on  ServerSock.
  // Close the socket.
  closesocket (ServerSock);
 WSACleanup ();
  DS_unregister_host_app();
}
```

APPENDIX F

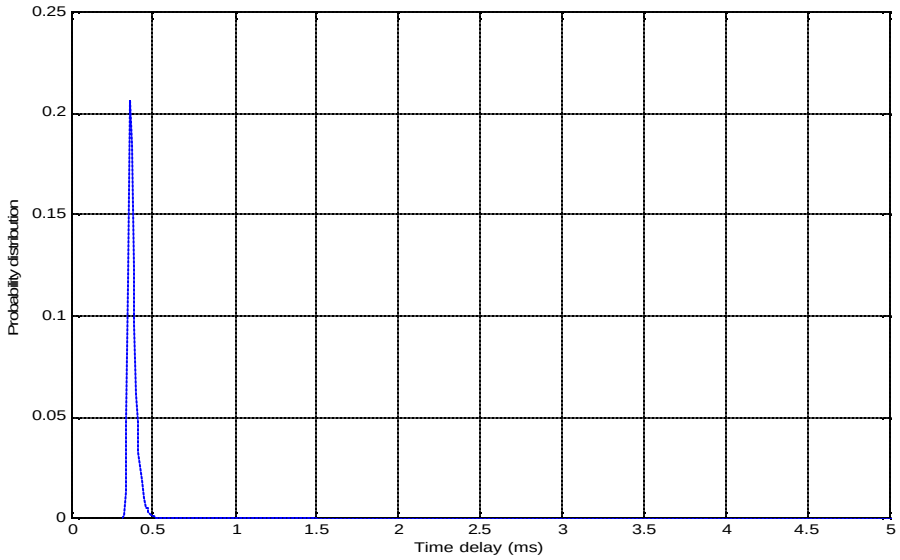

Fig.47. Probability distribution of round-trip time delay on Sat. 11/30/2002 from 10:00 AM to 4:00 PM between PCs on same LAN.

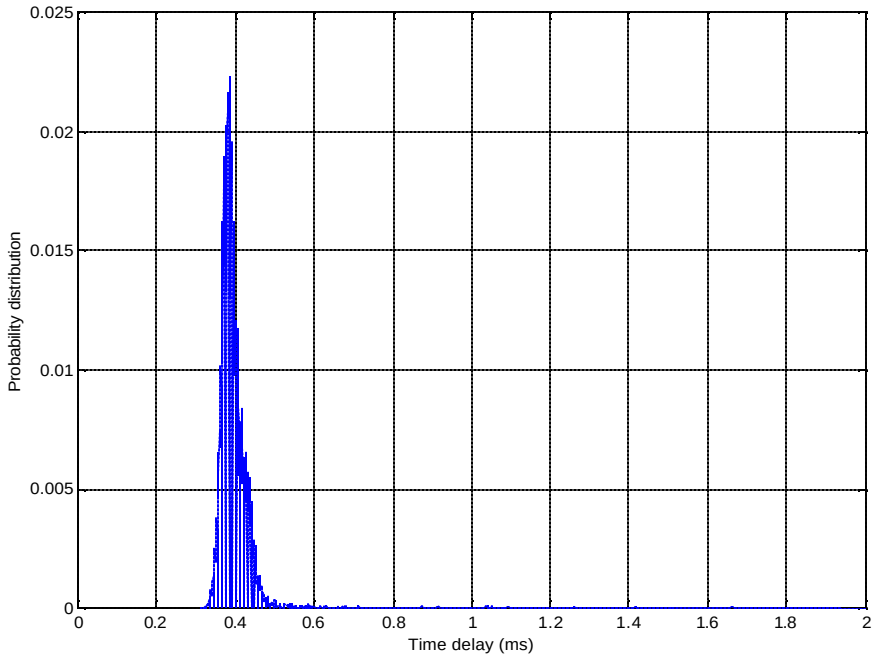

Fig.48. Probability distribution of round-trip time delay on Tue. 12/03/2002 from 7:00 PM to 4:00 AM between PCs on same LAN.

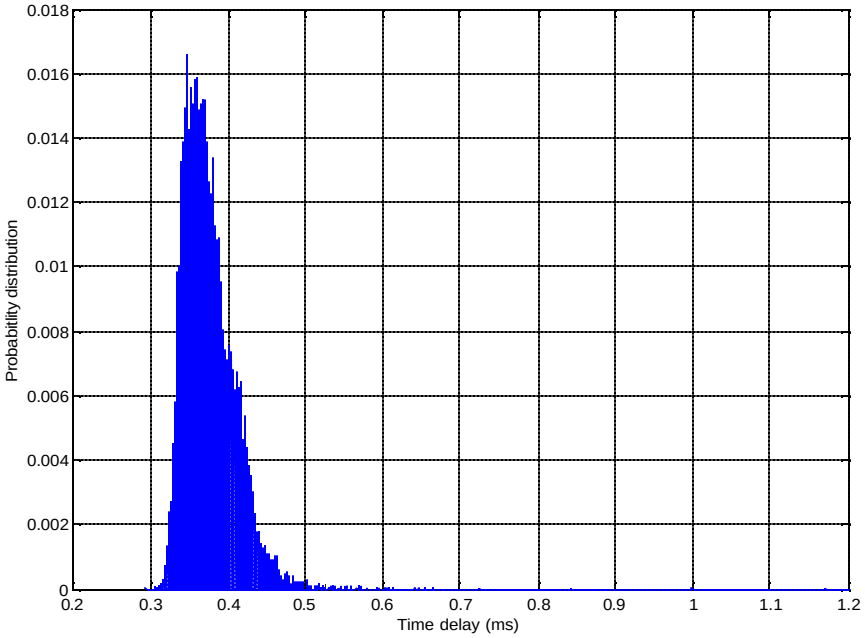

Fig.49. Probability distribution of round-trip time delay on Sat. 12/07/2002 from 1:46 PM to 7:40 PM between PCs on same LAN.

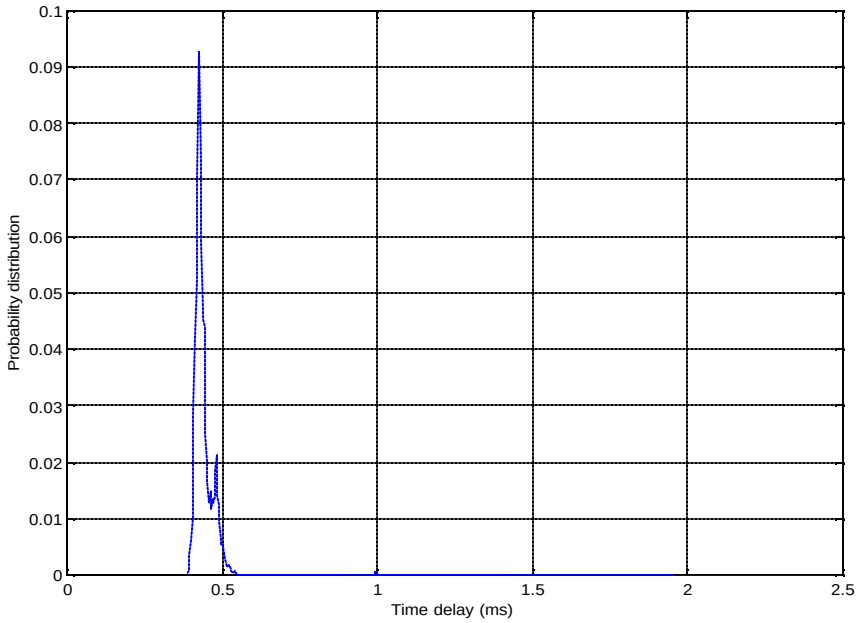

Fig.50. Probability distribution of round-trip time delay on Sun. 12/08/2002 from 2:00 PM to 10:00 AM between PCs on same LAN.

VITA

The author, Abhinav Srivastava, was born on December 3, 1979 in Allahabad, India. He graduated from Regional Engineering College, Allahabad, India with a Bachelor of Engineering degree (Mechanical Engineering) in May 2001. After that, he started his Master's degree at Texas A&M University in August 2001 in Mechanical Engineering.

His permanent address is:

Type V/1047 NH-4,

Faridabad 121001, Haryana, India.

His e-mail address is:

srivastava_abhinav@hotmail.com