REAL-TIME NETWORKED CONTROL WITH MULTIPLE CLIENTS

A Thesis

by

MINHYUNG LEE

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

MASTER OF SCIENCE

August 2009

Major Subject: Mechanical Engineering

REAL-TIME NETWORKED CONTROL WITH MULTIPLE CLIENTS

A Thesis

by

MINHYUNG LEE

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

MASTER OF SCIENCE

August 2009

Major Subject: Mechanical Engineering

ABSTRACT

Real-Time Networked Control With Multiple Clients.

(August 2009)

Minhyung Lee, B.S., Korean Military Academy, South Korea

Chair of Advisory Committee: Dr. Won-jong Kim

In this thesis closed-loop control strategies over a communication network with multiple clients are developed. To accomplish this objective, a steel-ball magnetic-levitation system, a DC motor speed-control system, and an autonomous wheelchair robot referred to as Clients 1, 2, and 3, respectively were used as Networked-Based-control (NCS) test beds to validate the proposed strategies. For real-time operation, Linux with Real-Time Application Interface (RTAI) and Control and Measurement Interface (Comedi) were used as the operating system for Clients 1 and 2. Client 3's software was written in Microsoft Visual Basic 6.0 on the Windows XP operating system (OS). User datagram protocol (UDP) was used as the communication network protocol in this research due to its better real-time performance instead of transmission control protocol (TCP). Although UDP has no guarantee for transferring data, it has smaller overheads and less time delay than TCP.

Since the robotic wheelchair and the server are run on different OSs, Samba was used to put both systems into the same LAN with a fast data-transmission speed. Using

Samba, the round-trip communication time between the robotic wheelchair and the server is only 11.2 ms whereas 30.8 ms is taken without using Samba.

When the server receives the sensor data from multiple clients at the same time, the NCS stability may be deteriorated due to the limitation of the system bandwidth. The NCS stability is affected by the sampling period of the system, and the reduction of the sampling period improves the control loop’s performance. However, a shorter sampling period requires more network bandwidth to transmit more sensor data or control data, which increases the network traffic load.

Using the PING test, the transmission time for each control loop was measured. The processing time for each system was also measured by a time-stamp function, and the operation time for each control loop was obtained. In order to maintain the NCS stable, several combinations of the sampling periods for each client are suggested and verified. The bandwidth utilization of Client 1 is set to be 43.5% and the range of the bandwidth utilization of Client 2 with guaranteed stability was found to be between 9.1% and 45.3%. Thus, the bandwidth utilization of Client 3 is from 11.8% to 46.8%. The multiple-client NCS test bed could maintain its stability within these ranges of the bandwidth utilizations of all clients.

To my parents

and

wife Hyekyung Yu

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my advisor Dr. Won-jong Kim for his time and effort throughout my study at Texas A&M University; without his guidance this thesis would not have been completed.

I wish to thank Drs. Langari and Datta for serving as my advisory committee members. I sincerely appreciate their valuable guidance.

I would like to thank Stephen C. Paschall, II and Pin-chun Hsieh for developing the ball levitation system and the autonomous wheelchair robot that were used as test beds for the experiments. Ajit Ambike and Kun Ji developed the real-time operating environment, which was crucial in this research. I would also like to give my special thanks to Youngchul Na and NaveenKumar Bibinagar for their help in setting up the real-time operating environment on lab computers and the DC motor speed control system.

Finally, many thanks to my parents for their immeasurable support through my life and to my wife for her patience and love. Without their unconditional love, encouragement, and support, I could have never come so far.

TABLE OF CONTENTS

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

CHAPTER I

INTRODUCTION

Recent advances in communication, computation, and embedded technologies have supported the development of NCSs. The NCS is defined as the combined system of controllers, actuators, sensors, and the communication network that interconnects them together. This NCS has the advantage of greater flexibility compared to traditional control systems. It also allows a lower installation cost with reduced wiring and permits greater agility in diagnosis and maintenance procedures [1].

1.1 Modes of Control on the Network

Many commercial companies and research institutes have shown interests in applying NCSs for remote industrial control and factory automation. As a result of extensive research and development in this area, various forms of NCSs exist in the automation industry. The classification of the modes of control depends on the communication architecture between the plant and the remote user. These various modes of control over the network can be classified into teleoperation, supervisory control, and closed-loop control over the network.

Teleoperation systems allow human operators to execute tasks in remote or hazardous

---

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

environment like in space, underwater, or nuclear applications [2]. In teleoperation systems, however, the operator must depend on the feedback provided by sensory feedback systems to perform subsequent actions as shown in Fig. 1-1. Therefore, the operator's limited perception of the environment could result in a poor performance. For this reason, researchers have been focusing their research attention on supervisory control.

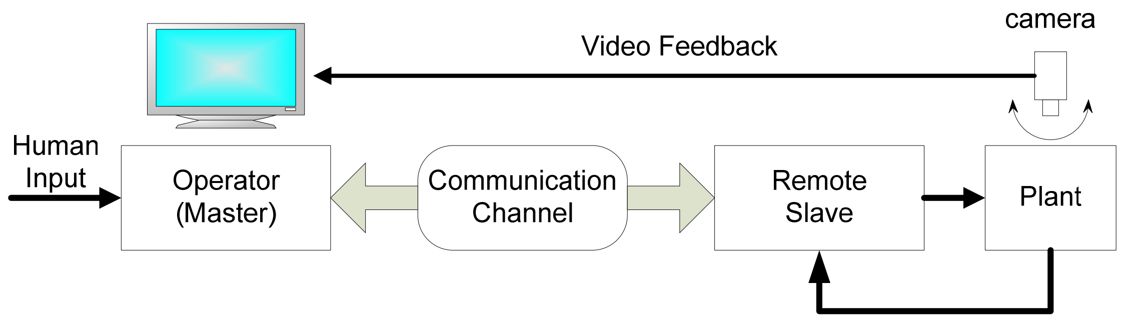


Fig. 1-1. Block diagram of a teleoperation system

Supervisory control is based on a client-server architecture. In supervisory control, the user on a client station can give symbolic or analogical instructions remotely to a server computer attached to the manipulator instead of remotely guiding the tele-manipulator as he does in the teleoperation systems. In supervisory control, the sensors, controllers, and actuators are located on the plant side as shown in Fig. 1-2. Like a teleoperation system, the control loop in supervisory control is also closed locally.

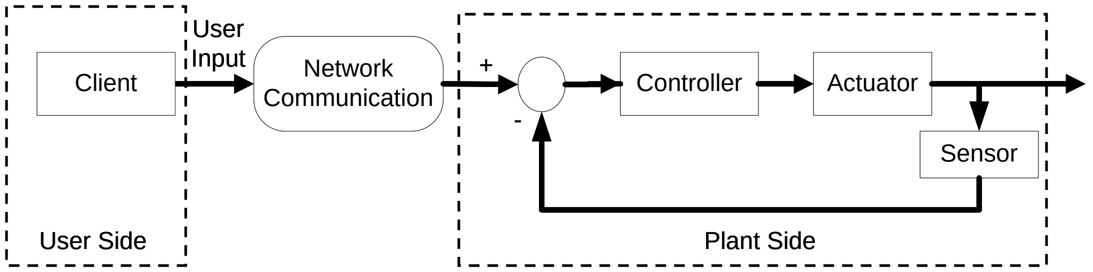


Fig. 1-2. Block diagram of a supervisory control system

In feedback control over the network, the control loop is closed over the network. Fig. 1-3 describes a block diagram representing feedback control over the Internet. The controller receives data from the sensors and sends the control data to the actuators over the network.

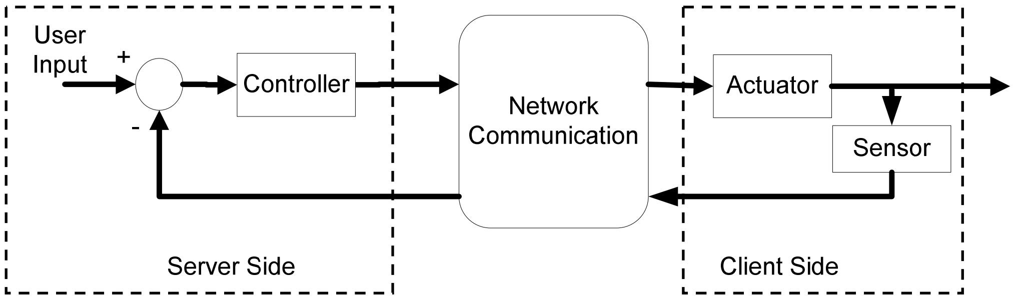


Fig. 1-3. Block diagram of a networked feedback control system

The multiple-client NCS used in this research can be one of the applications of feedback control over the network. A server controller and multiple clients share the network as a communication medium. A block diagram of the multiple-client NCS is shown in Fig. 1-4. Success of this multiple-client NCS relies on the system stability.

Thus, network scheduling and optimal bandwidth allocation for these clients are key issues in this system.

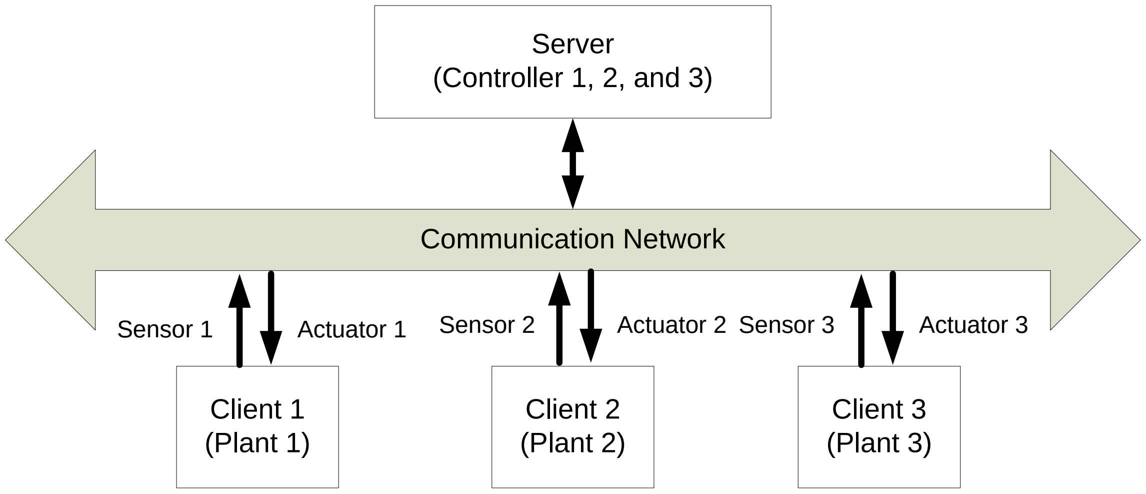


Fig. 1-4. Block diagram of the multiple-client NCS

1.2 Objectives

A study of closed-loop real-time control with multiple clients over a network is the main focus of the research. Following are its main objectives.

1. Establishment of real-time closed-loop control over a network with three multiple clients, which are a steel-ball maglev system, a DC motor speed-control system, and an autonomous wheelchair robot.
2. Development of an algorithm that can identify the clients on the server side by socket programming.

3. Suggestion of the optimal sampling periods for each client system to maintain the system stability.

1.3 Contributions

The main contribution of this thesis is the implementation of closed-loop control over a network with multiple clients by experiments. Major accomplishments include (1) establishment of the network connection of a Window-based personal computer (PC) with a Linux server using Samba, (2) design of a multiple-client-NCS architecture, (3) implementation of this NCS's communication architecture, and (4) suggestion of the clients' sampling periods for system stability.

To establish a multiple-client NCS, a steel-ball maglev system, a DC motor speed-control system, and an autonomous wheelchair robot are used as NCS test beds to validate the proposed strategy. The server can identify the clients with their identification numbers included in the sensor data packets from the clients.

The Windows-based robotic wheelchair is connected with the Linux server with Samba. The data-transmission time between the robotic wheelchair and the server is reduced significantly, and the robotic wheelchair can arrive at a destination without any collision with an obstacle with a certain bandwidth allocation.

For the network communication, socket programming is used. Since the robotic wheelchair uses a different type of the data packet from the server, Gateway is established between both systems to convert the data-packet type to another format.

With PING tests, the operation times for each client are obtained. Based on these operation times, the relation between the network bandwidth utilization and the sampling period is presented. Some possible combinations of the sampling periods are suggested and verified to maintain the stability of the multiple-client NCS.

1.4 Thesis Organization

This thesis is organized as follows

Chapter I provides a brief introduction of NCSs. The modes of NCSs are also given. This chapter also describes the objectives and contributions of this research.

Chapter II explains the previous work done by other researchers in the area of NCSs. The literature review is divided into the modes of control on the network and the network scheduling.

Chapter III describes in the detail the design of the multiple-client NCS which used in this research. It gives an overview of the existing experimental setup and its control scheme.

Chapter IV explains the software design for the NCS. It describes the computing environment and design of the software architecture.

Chapter V describes how each system can be operated and tested. With a series of experiments, the possible combinations of the sampling periods for each system are suggested and verified.

Chapter VI summarizes the achievements of this thesis. The future work towards further development of this NCS is also given.

CHAPTER II

LITERATURE REVIEW

In this chapter, previous work done by other researchers in the NCS field is described. Researchers have successfully developed several applications based on teleoperation and supervisory control. Real-time feedback control over networks also has been receiving increasing attention in the last few years.

In NCSs, time delays take place due to sharing a common network medium. These delays may make the system unstable. It is important that the data should be transmitted within a sampling period and the stability of control systems should be maintained.

2.1 Modes of Control on the Network

With the advancement in industry automation, the Internet technology led the shift of the emphasis from centralized to distributed control systems. As a result of extensive research and development, various forms of NCSs have been performed.

2.1.1 Teleoperation

Early developments on teleoperation system were carried out in the area of space, underwater, and nuclear applications with the common aim of reducing risk to human lives. Hirzinger *et al*. [3] developed a teleoperation-based method of a space flight with a multisensory gripper technology. This flight is teleoperated by astronauts using a control ball and a stereo-TV-monitor and can refine gross commands autonomously by local (shared autonomy) sensory feedback control concepts.

Lin *et al.* [4] introduced virtual telepresence operation approach of underwater robots. This virtual telepresence interface takes robot's position and orientation data from a sonar navigation system, and generates three dimensional (3D) synthetic images of the worksite based on its computer aided design (CAD) model using virtual reality technology. It gives the robot operators with a full perception of its spatial location, flexible options of viewpoints and functions for teleoperation of underwater robots.

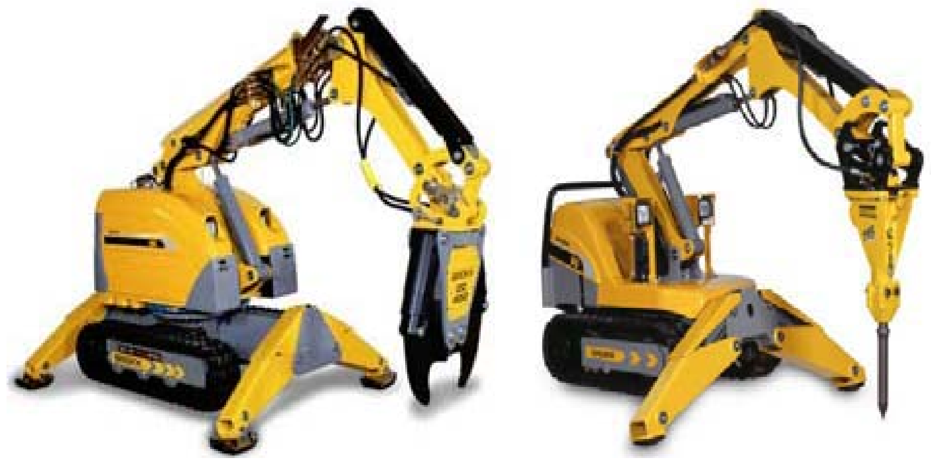


Fig. 2-1. Brokk – teleoperated robot for demolition [5]

The primary use of telerobotics in decommissioning applications is to reduce the radioactive dose levels to which workers are exposed. An example of teleoperated robot in this area is remote-control Brokk as shown in Fig. 2-1 [5]. A remote operation pendant allows the operator to be at a safe distance from high radiation areas and hazardous or falling debris. The Brokk is rugged enough for demolition work and small enough to work inside buildings. They are often electrically powered, through an umbilical cable, to make indoor working easier. A wide range of end-effector tools is available for most demolition tasks. Such teleoperated robots have become widely accepted throughout the decommissioning industry.

Teleoperation is also used in the field of surgery. Cohn *et al*. [6] implemented a tactile feedback system with tactile sensor and display into a simple force-reflecting teleoperator. This can be provided by linking a tactile sensor on the manipulator to a tactile display worn by the user. The display can convey both symbolic and representational stimuli and human subjects were able to discriminate very small displacements. Madhani *et al.* [7] developed Black Falcon, a teleoperated surgical instrument for minimally invasive surgery (MIS). MIS is the practice of performing surgery through small incisions using specialized surgical instruments. Using with eight-degree-of-freedom and cable-driven teleoperator, it allows motion scaling between the master and the slave. These advantages increase dexterity and enable tasks that were previously impossible such as suturing along arbitrarily oriented suture lines.

From advanced manufacturing to daily applications, Internet-based telerobotic systems have the potential to provide significant benefits. Wang *et al.* [8] developed an Internet-based multiple-telerobot system as shown in Fig. 2-2. This system includes a web server, browser client, and local distributed system. Three cameras are also used to provide online live video display. A browser client can operate, monitor, and simulate the local distributed system through the Hypertext Transport Protocol (HTTP) server. The HTTP server runs several threads to handle robot operations and obtain or deliver video information. Different functional software components are distributed on the host machine, target machines, and the robot controller.

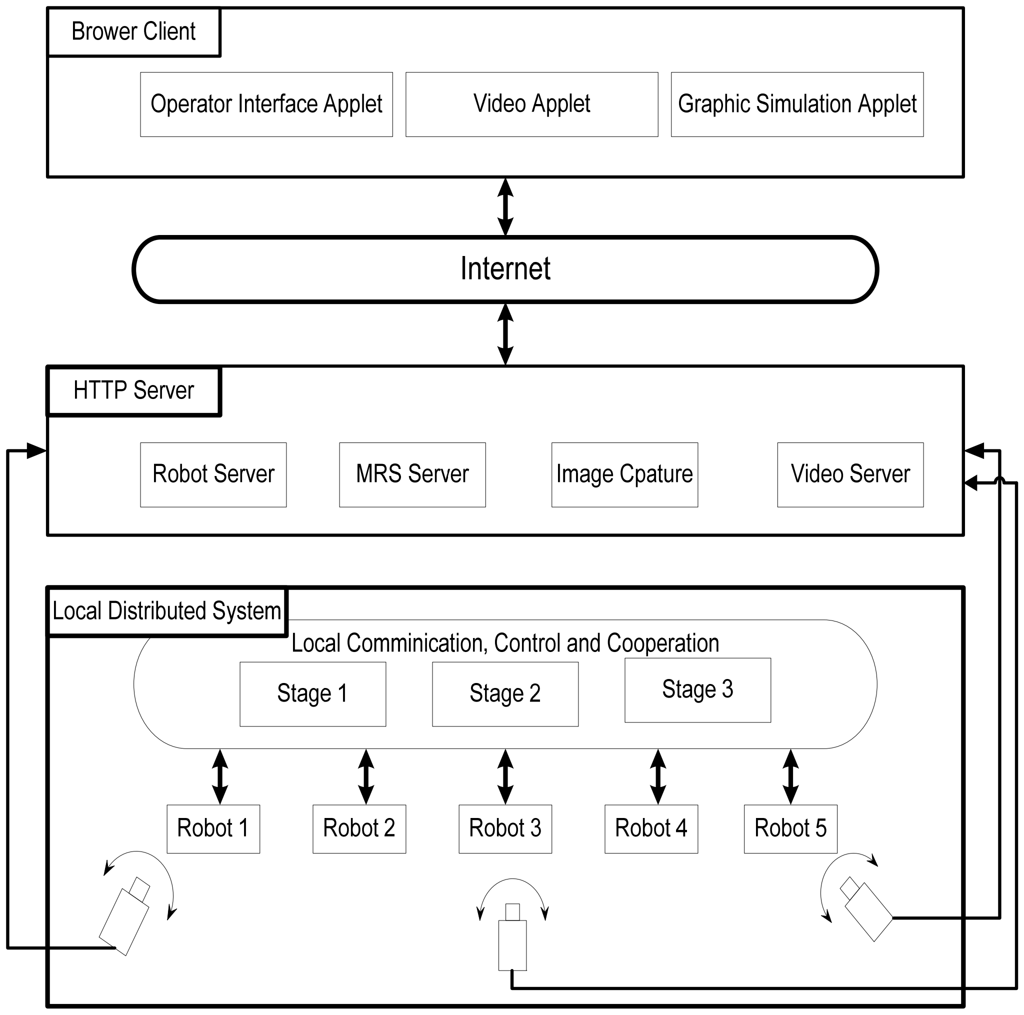


Fig. 2-2. Internet-based multiple-telerobot system architecture [8]

2.1.2 Supervisory Control

Recently researchers have shown interests in applying supervisory control for their telerobot and test beds. Luo *et al*. proposed a desktop rapid prototyping system with supervisory control and monitoring through the Internet [9]. The user sends a 3D Computer-Aided Design (CAD) model via the Internet to a telecontrol server. The telecontrol server transforms the CAD model into a rapid-prototyping (RP) liquid-crystal-diode (LCD) photomask display. The user can then direct the RP machine to build the RP part while watching a live image via the Internet. The overall system architecture of this telemanufacturing system is shown in Fig. 2-3. The online visual system allows inspection of RP part quality during manufacturing. A pattern matching algorithm which compares a grabbed image with the photomask monitors the part-building process.

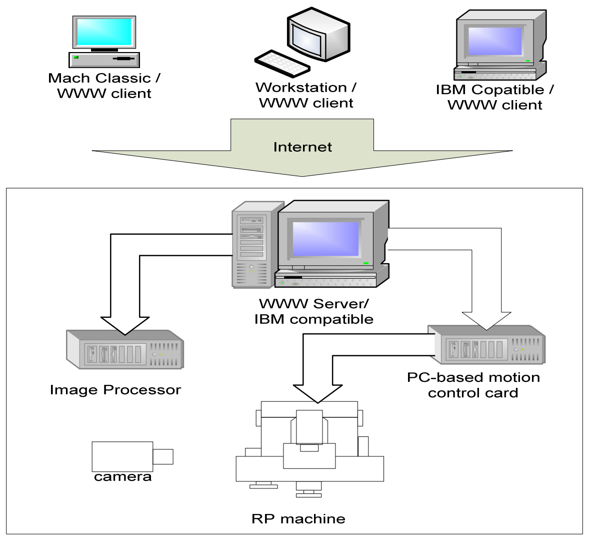

Mach Classic / WWW client
Workstation / WWW client
IBM Copatible / WWW client
Internet
WWW Server/ IBM compatible
Image Processor
PC-based motion control card
camera
RP machine

Fig. 2-3. Overall system architecture of automated telemanufacturing system [9].

Park and Sheridan [10] developed supervisory control of a six-degree-of-freedom telemaniuplator which was graphically simulated on an IRIS workstation. The system checks possible manipulator collisions with obstacles in the environment. The operator can specify intermediate locations that the manipulator tip is to pass through. If the system detects an impending collision, it uses heuristics to avoid it.

Due to recent developments in the communications technology in the last decades, discrete-event systems constituted by entities or processes that exchange messages or coordinate the execution of a task. When a system includes features of both discrete events and continuous signals, it is called a hybrid system [11].

Garcia *et al*. [12] developed a supervisory controller for a robotic teleoperation system with communication time delay, considering the hybrid systems theory. The discrete-event controller is based on discrete abstractions of the continuous dynamics. A supervisory control was designed to detect when force and position thresholds were overcome. The controller was developed to modify the reference sent from the local station to the remote station when a communication interruption arose.

Wanga *et al*. [13] developed a hybrid supervisory control system for the optimal temperature control of a reheat furnace based on expert knowledge and pyrology. The developed control model can replace the human operator by auto-searching the proper operation points for the reheating process under variations of boundary conditions. The preset module can calculate the optimal preset value of the furnace temperature set point of each zone under different operation modes.

Ji *et al*. [14] discussed the supervisory control via the Internet of a ball-maglev system. This supervisory control is based on the client-server architecture, which is used primarily for computer network communication. The overall system architecture for the supervisory control of the maglev system over the Internet is depicted in Fig. 2-4. The maglev system is controlled using a common gateway interface (CGI) and a hypertext markup language (HTML) interface where a client can operate the maglev system and move the ball within its travel range. Stochastic nature of the Internet communication delays was also studied in reference to the various probabilistic models for the time delays.

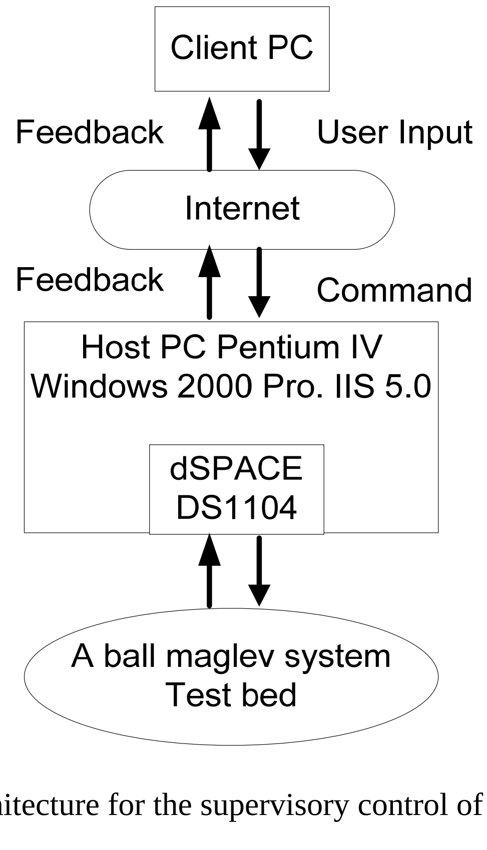


Fig. 2-4. Hardware architecture for the supervisory control of the ball-maglev system over the Internet [14].

2.1.3 Closed-Loop Control over Networks

While a teleoperation system and a supervisory control system are closed locally, feedback control allows the controller to be placed over the network, separated from the controlled process. Eker and Cervin [15] developed distributed wireless control using Bluetooth radio network. The distributed control configuration is shown in Fig. 2-5. A rotating inverted pendulum was used in the experiments. The sensor node is time-driven while the controller and actuator nodes are event-driven. The sensor samples the process periodically and sends the measurement values to the controller. Upon receipt, the controller calculates a new control signal and sends it to actuator node which outputs the value. A dynamic delay compensation scheme was simulated for random delays that might occur due to retransmissions.

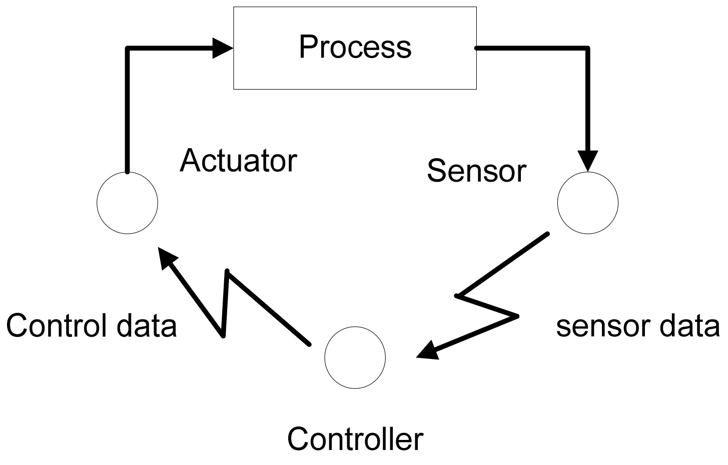


Fig. 2-5. Distributed wireless control system configuration using Bluetooth [15].

Because of some limitations of Bluetooth, such as relatively low throughput and very limited range, a recent study focuses on the use of TCP/IP and UDP protocol. Ploplys and Alleyne [16] developed UDP network communications for distributed wireless control. In this system, sampling and actuation are clock-driven while control is event-driven, making the controller node a slave to the actuator/sensor node. The actuator/sensor node has access to a real-time clock, which it uses to send feedback precisely at each sampling interval. The controller need only wait to receive feedback from the sensor and quickly reply with a control action for the actuator. The effects of delayed time and its performance were studied.

In feedback control over the network, there are two classes of time delays and packet losses in both the sensor feedback and control feedforward paths. Ambike [17] discussed the closed-loop real-time control over network of a ball-maglev system. An algorithm using predictors was designed to ensure the system stability in the presence of network delays and data packet losses. The system output was predicted several steps ahead and the control output was calculated using these predictions. This control output was used in the events of excessive network delay to maintain system stability.

Network scheduling and bandwidth allocation is an important issue in NCS design when a set of plants or processes connected to the same network and share the limited bandwidth resource. Cervin and Eker [18] proposed a feedback scheduler for real-time-control tasks. Linear quadratic cost functions are used as performance indicators. The feedback scheduler calculates an optimal resource-allocation pattern. The feedback scheduler is demonstrated on a three-control-loop system.

2.2 Scheduling for NCSs

To design an NCS, its control and communication aspects should be considered because the control performance is limited by the system bandwidth. Therefore, a network bandwidth allocation must be considered.

Park *et al.* [19] presented a scheduling method for networked-based control system with three types of data: periodic data, sporadic data, and messages. All periodic data have to be transmitted within the respective sampling period to guarantee the system stability, while guaranteeing real-time transferring of sporadic data and minimum transmission of messages.

Ji and Kim [20] developed a co-design methodology of dynamic optimal network bandwidth allocation. This dynamic bandwidth allocation algorithm makes scheduling decisions based on the quality of performance information of the control loop. With the consideration of time-varying sampling frequencies, the system can be performed efficiently.

CHAPTER III

TEST-BED SETUP AND ITS CONTROL

Based on the previous chapter, the multiple-client NCS is proposed in this research for closed-loop feedback control over the network. A server controller and multiple clients share the network as a communication medium. The overall system architecture of this multiple-client NCS is shown in Fig. 3-1.

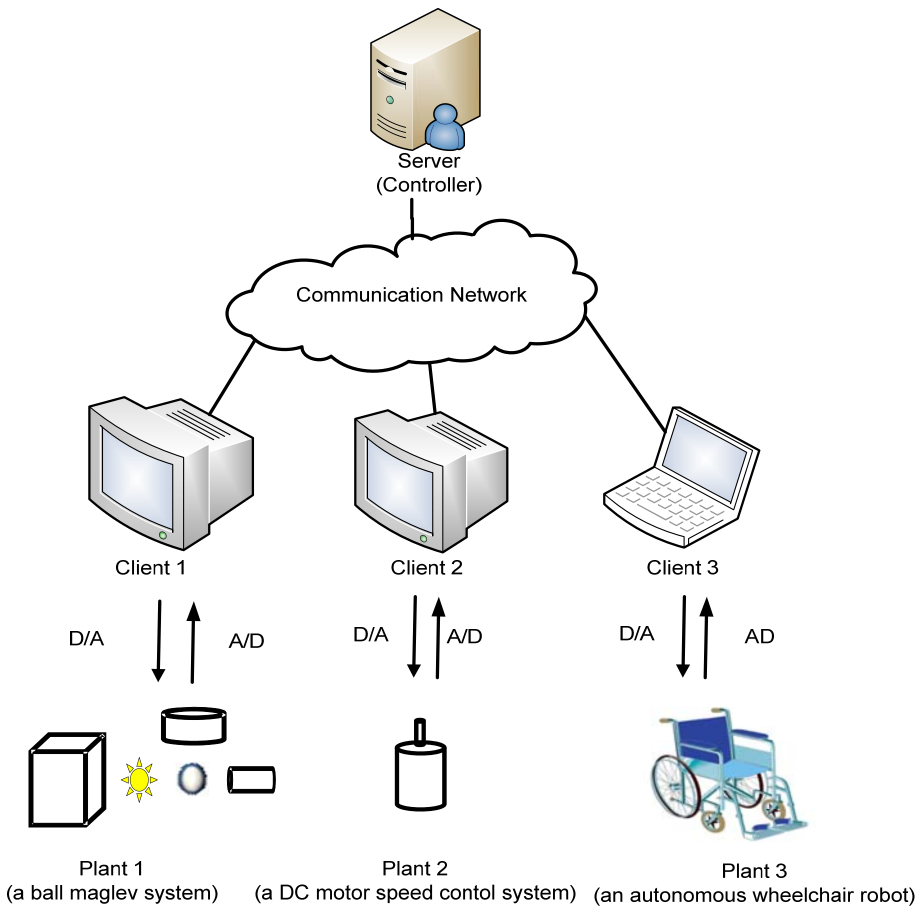


Fig. 3-1. Overall system architecture of the multiple-client NCS

Three client physical systems are used as test beds in this research: Client 1 - the single-actuator ball maglev system developed by Paschall [21], Client 2 - a DC motor closed-loop speed control system, Client 3 - an autonomous wheelchair robot developed by Hsieh [22]. Each test bed and designed control system is introduced in this chapter.

3.1 Client 1 - Ball-Maglev System

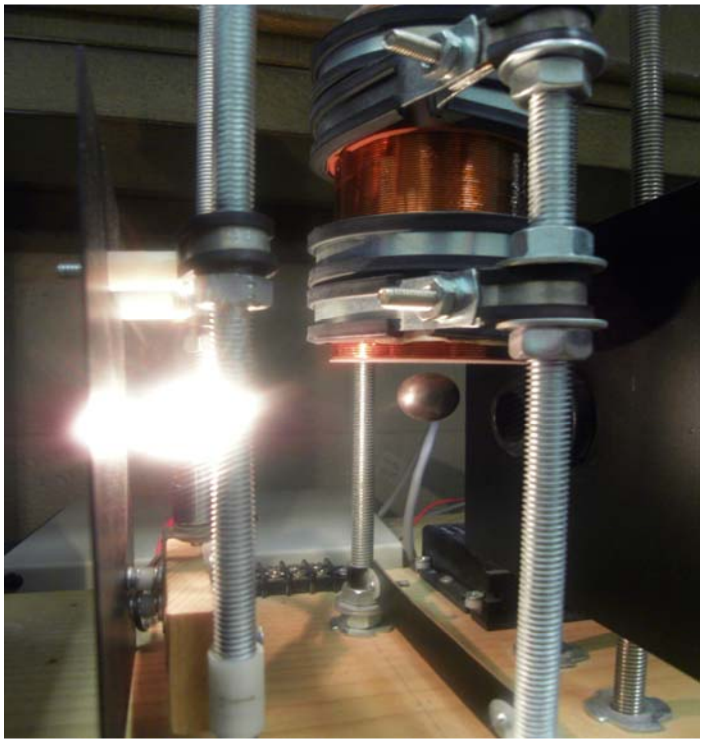


Fig. 3-2. Ball-maglev system used as Client 1

The single-actuator ball maglev system developed by Paschall [21] is used as Client 1. This test bed can levitate a small steel ball at a predetermined steady-state operating position with an electromagnet. The maglev system shown in Fig. 3-2 consists of essentially of a platform test bed and a Linux PC with a data-acquisition board. The test bed contains an electromagnetic actuator, an optical position sensor, a pulse-width-modulation (PWM) power amplifier, and power supplies.

This system contains two sub-parts. These are the position-sensing part and the force-actuating part. The optical position-sensing part consists of a photocell-based sensor, an incandescent light source, and a 15-V DC power supply. The unit allows the photocell to be exposed to the bulb light as a function of the ball position. This variable light exposure on the photocell results in a change of its electrical resistance. The force-actuating part consists of an electromagnet coil, a PWM amplifier, and a 24-V DC power supply. The control signal is given to the electromagnet coil through the PWM amplifier to control the levitated ball's position.

Fig. 3-3 shows a schematic diagram of the maglev system. The electromagnet was made by wrapping an iron core of high permeability with a copper wire. When an electrical current passes through the wire, the actuator creates an attractive force to the steel ball. A position sensor, the photocell, detects the vertical position of the ball and passes this information to the controller.

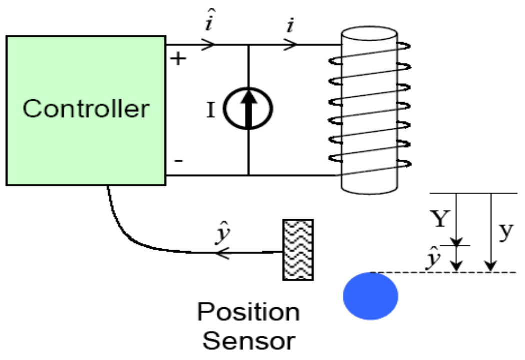


Fig. 3-3. The schematic diagram of the maglev system [21]

The small-sized open-loop plant transfer function developed by Paschall [21] is

$$\frac{\hat{Y}(s)}{\hat{I}(s)} = \frac{-1}{As^2 - B}\left[\frac{m}{A}\right] \tag{3.1}$$

with

$$A = \frac{I}{2g}\left[\frac{As^2}{m}\right] \tag{3.2}$$

$$B = \frac{I}{a+Y}\left[\frac{A}{m}\right], \tag{3.3}$$

where g = 9.81 m/s^2 is the gravitational constant, I = 0.5236 A and Y = 0.005 m are the operating points, and a = 2.49 mm is a geometric constant in the ball maglev system.

The position information provided by the photocell sensor is used as feedback to achieve closed-loop stability. Fig. 3-4 describes a block diagram for the feedback control of the ball maglev system.

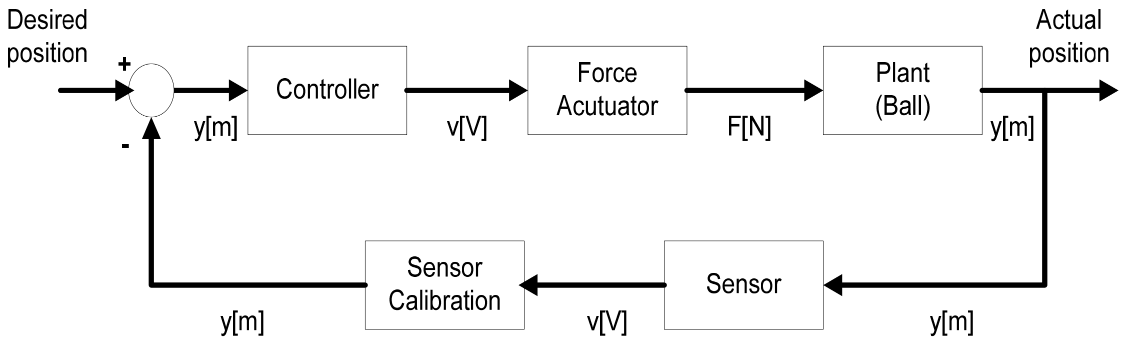


Fig. 3-4. Block diagram for the feedback control of the ball-maglev system [21]

In [21], the controller designed in continuous time domain is given by

$$\frac{-33300s^2 - 2.564\times10^6 s - 1.632\times10^7}{s^2 + 700.7s + 490} . \tag{3.4}$$

The controller maintained the following transient response requirements.

Settling Time(t_s) $\leq$ 1 s

Percentage Overshoot $\leq 50\%$

Linux with Real-Time Application Interface (RTAI) is used as the OS. A PCI-6025E card by National Instruments is used for data acquisition. One of the analog-to-digital (A/D) converters is used to read the optical sensor input. The voltage output signal from the digital-to-analog (D/A) converter is fed to the PWM amplifier.

Srivastava [23] designed the digital controller for the 333 Hz sampling frequency as

$$D(z) = 4.15\times10^4 \frac{z^2 - 1.754z + 0.769}{z^2 - 0.782z - 0.13} . \tag{3.5}$$

3.2 Client 2 - DC Motor Speed-Control System

A DC motor speed-control system is used as Client 2. A block diagram to illustrate DC motor closed-loop speed control is given in Fig. 3-5. The speed of a DC motor is directly proportional to the supply voltage. The speed command is a DC voltage, which is fed to the PWM amplifier. This drives the motor at a speed dependant on the commanded voltage. The shaft angular displacement per unit time of the motor is sampled using the encoder.

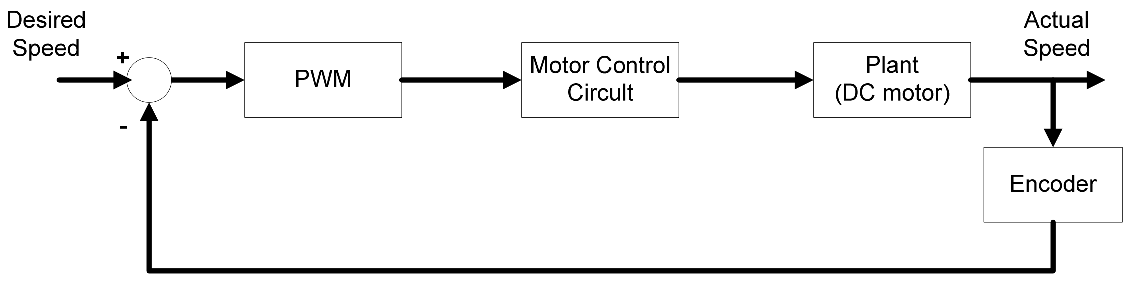


Fig. 3-5. Block diagram for DC motor speed control

The electric circuit of the armature and the free-body diagram of the rotor are shown in Fig. 3-6.

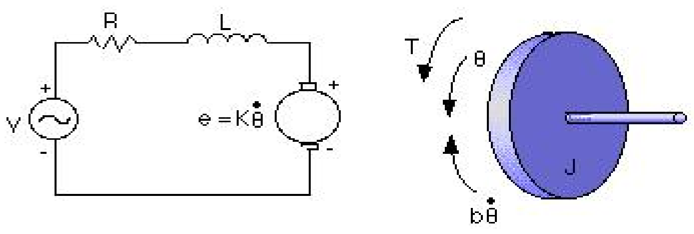


Fig. 3-6. The electric circuit of the armature and the free-body diagram of the rotor [24]

Using the parameters given the DC motor datasheet [25], the plant transfer function is obtained as [26]

$$G(s) = \frac{20.2}{9.92s + 2.57}\,. \tag{3.6}$$

A proportional-integral (PI) controller can be designed as [26]

$$G(s) = \frac{K_P s + K_I}{s} = \frac{1.5s + 5}{s} \quad . \tag{3.7}$$

From (3.4), we can get the discrete-time controller as

$$\begin{aligned} u(k) &= u(k-1) + [K_P + 0.5K_I h]e(k) + [0.5K_I h - K_P]e(k-1) \\ &= u(k-1) + [1.5 + 2.5h]e(k) + [2.5h - 1.5]e(k-1) \ , \end{aligned} \tag{3.8}$$

where h is the sampling period. Various sampling periods will be tested to operate Client 2. Refer to Section 5.2.2 for details.

3.3 Client 3 - Autonomous Wheelchair Robot

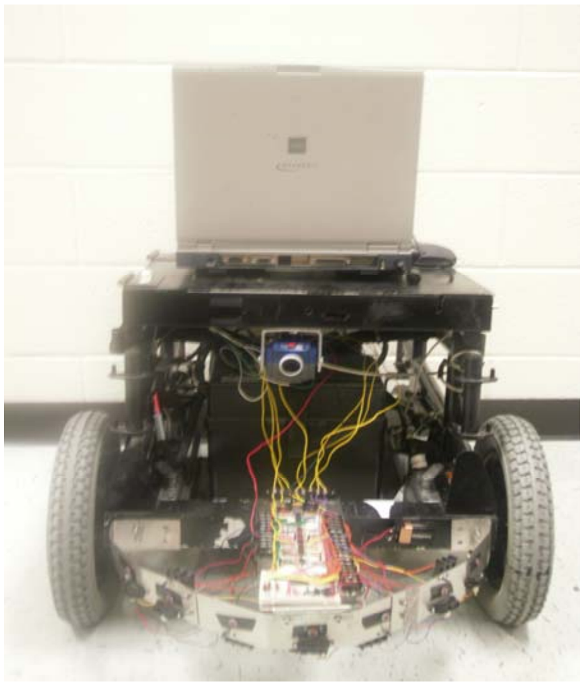
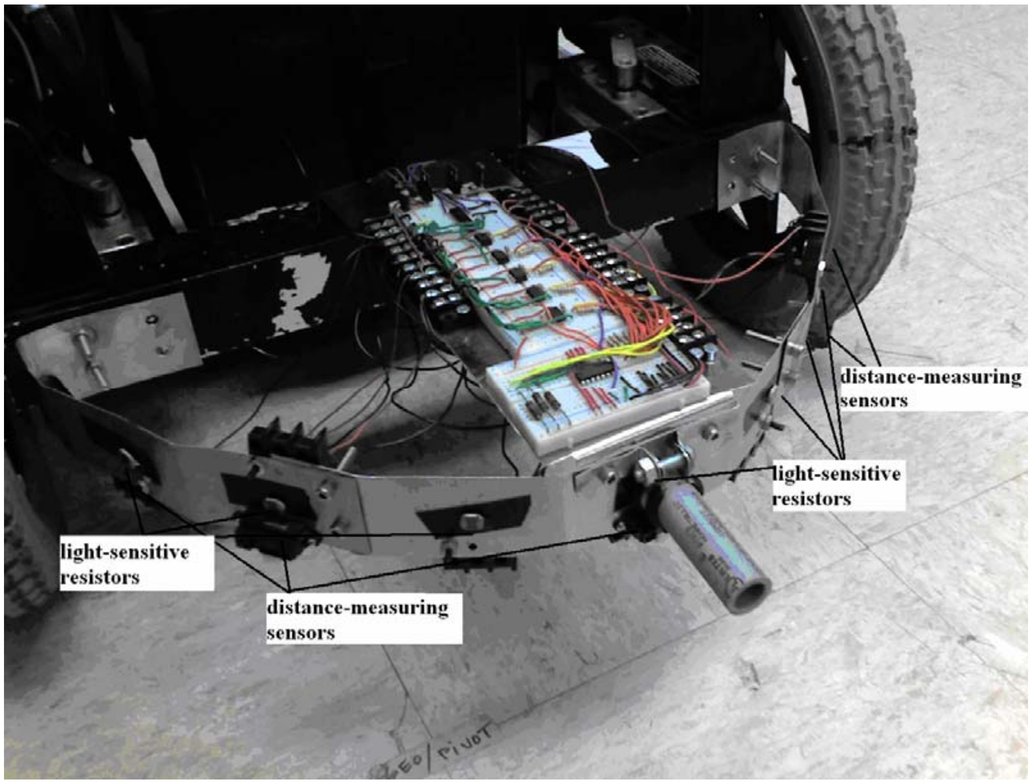


Fig. 3-7. Autonomous wheelchair robot used as Client 3

In this research, the autonomous wheelchair robot [22] is used as Client 3 as shown in Fig. 3-7. The robotic wheelchair is built upon the base frame of an Invacare Ranger II^{TM} electric powered wheelchair. The frame is 70-cm long and 48-cm wide with a height of 55cm. This wheelchair is driven by two independent 12-V DC motors for the front wheels with a diameter of 31.75 cm with built-in reduction gears that provide a maximum speed of 6 km/hr. Two MC-7 motor controllers are used for motion control. The stability of the platform is ensured by two 18-cm-diameter castors in rear.


distance-measuring sensors
light-sensitive resistors
light-sensitive resistors
distance-measuring sensors
Fig. 3-8. Sensor system of the autonomous wheelchair robot [22]

The sensor bracket is mounted at the front. The wheelchair robot has seven CdS light-sensitive resistors which are also known as photocells, five distance-measuring sensors, and two Hall-effect sensors as shown in Fig. 3-8 [22]. The wheelchair robot has the capability of tracking a motion trajectory defined with a light with the photocells. If any of the three photocells on the left bracket, then the wheelchair turns left until the front photocell detects the light. In the opposite way, if any of the three photocells on the right detects the light, the wheelchair turns right until the front photocell detects the light.

Table 3-1. Motion of the robotic wheelchair for each sensor signal [22]. L, H, F, TR, TL, and S are denoted low, high, forward, turn right, turn left, and stop respectively.

(a) The signal of the front GP2D15 is low

Left GP2D12	L	L	L	L	L	L	L	L	H	H	H	H	H	H	H	H
Left GP2D15	L	L	L	L	H	H	H	H	L	L	L	L	H	H	H	H
Right GP2D12	L	L	H	H	L	L	H	H	L	L	H	H	L	L	H	H
Right GP2D15	L	H	L	H	L	H	L	H	L	H	L	H	L	H	L	H
Movement of Robot	F	TL	TL	TL	TR	F	F	TL	TR	F	F	F	TR	F	TR	F

(b) The signal of the front GP2D15 is high

Left GP2D12	L	L	L	L	L	L	L	L	H	H	H	H	H	H	H	H
Left GP2D15	L	L	L	L	H	H	H	H	L	L	L	L	H	H	H	H
Right GP2D12	L	L	H	H	L	L	H	H	L	L	H	H	L	L	H	H
Right GP2D15	L	H	L	H	L	H	L	H	L	H	L	H	L	H	L	H
Movement of Robot	S	TL	TL	TL	TR	S	S	S	TR	S	S	S	TR	S	S	S

Three GP2D15 and two GP2D12 infrared distance-measuring sensors manufactured by Sharp are used to detect obstacles. The GP2D15 infrared sensor detects obstacles at a 25-cm range and the GP2D12 sensor, at 70 cm. The GP2D12 infrared sensors are assembled on the right and left side. One GP2D15 infrared sensor is mounted at the front. Two GP2D15 infrared sensors mounted on the right and left side just beside the GP2D15 infrared sensors. According to each condition of the infrared sensors' signal, we can set the movement of the robotic wheelchair as Table 3-1.

The sensors generate the output voltage signals fed to the A/D converters on the data-acquisition card which is installed on the laptop on the top. The laptop receives these sensor signals from the card, sends the data to the server over the wireless network. The server makes control data to avoid the obstacles and sends the data to the laptop over the network.

The control data from the laptop goes to MC-7 motor controller through the data-acquisition card. A SuperLogic PCMDIO 24-channel digital I/O type II Personal Computer Memory Card International Association (PCMCIA) card [27] is used as data-acquisition card to perform data-acquisition and control performance. Each MC-7 controller drives an electric motor in both the forward and backward directions. By this function the wheelchair can turn in a circle at an original point with one wheel moving forward and the other moving backward. Two decade-counter chips are used to count the pulses generated by the Hall-effect sensors. By this function, it is possible to measure the moving distance of the wheelchair robot.

CHAPTER IV

SOFTWARE DESIGN

In this chapter the computing environment and software architecture of the systems are discussed. First, the real-time OSs and the network protocol are described. The developed software architecture is also presented. The Windows-based wheelchair robot is integrated to the Linux server using Samba. The socket architectures provided in the last part of this chapter.

4.1 Computing Environment

The multiple-client NCS consists of one server and three clients. Software codes for Clients 1 and 2 are written in C language on Linux with RTAI and Comedi. Software code for Client 3 is written in Microsoft Visual Basic 6.0 on Windows XP and connected to the server over the wireless network.

4.1.1 Needs for Real-Time Operating System

Fig. 4-1 describes the client-server architecture in the closed-loop control system over the network. The client side implements the sensor and the actuator. The sampling is done at a certain fixed frequency. A new sample of the sensor data is taken for every sampling period. The sensor data by the client are sent to the server which implements the controller over the network. The server calculates the control data and sends the message to the client. Before receiving the sensor data from the client, the server side waits for the arrival of sensor data from the client. As soon as the sensor data arrive, the control data are calculated and sent by the server. Thus, the sensor and the actuator on the client side are time-driven whereas the controller is event-driven.

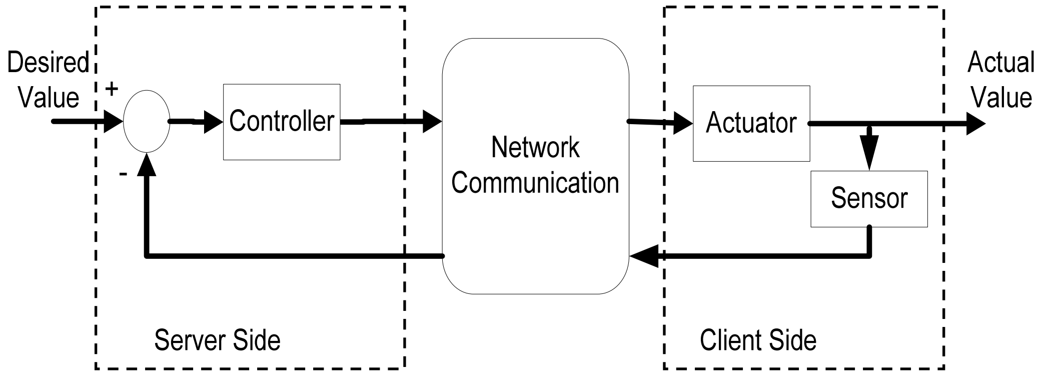


Fig. 4-1. Block diagram of closed-loop NCS

The events have certain deadlines. If these deadlines are missed, the control system would be lost the system stability. In other words, the feedback control loop should be completed in these deadlines. Client 1 needs very fast response from the server to maintain the stability of the system. Srivastava demonstrated the control data should be arrived from the server in 1.4 ms for 333.333 Hz in [23]. Client 2 also needs fast response for system stability. However, Client 3 does not need fast response to avoid an obstacle. In this research, the different sampling periods for each client are suggested and tested.

4.1.2 Operating System Environment

In order to ensure that Linux server, Clients 1, and 2 perform these tasks at correct time, real-time OS is needed. Linux with RTAI was found as a competitive OS environment for Clients 1 and 2 in [17]. The clock resolution of Linux is better than Windows-based OS. But Linux alone lacks real-time performance. RTAI modifies Linux kernel to make it a real-time operating environment. RTAI basically supports the same performances as the Linux kernel core, adding the features of a real-time OS [28].

For data acquisition, Comedi was installed and is used at Clients 1 and 2. Comedi is a free software project that develops tools, libraries, and drivers for various forms of data acquisition. Comedi works with standard Linux kernels like RTAI. Comedi consists of comedi and comedilib. Comedi is a collection of drivers for a variety of common data acquisition plug-in boards. Comedilib provides the developer-friendly interface to the Comedi devices [29].

The programs for Linux server, Clients 1, and 2 are written in C programming language. Client 3 is built on Windows XP. Although Windows XP is not real-time OS, it is fast enough to control Client 3. In addition, previous software programs of the robotic wheelchair were written in Microsoft Visual Basic 6.0 on Windows. In the current research, the software of Client 3 is modified in Microsoft Visual Basic 6.0. PCMDIO data acquisition-card and the PCMDRIVE data-acquisition software [30] were made for Windows-based OS only.

4.1.3 Communication Network Protocols

The TCP/IP protocol suite is used in this research. This suite contains two network protocols, TCP and UDP. The main difference is that TCP is slow, reliable, and connection-oriented whereas UDP is fast, unreliable, and connectionless.

TCP guarantees delivery of data and also guarantees that packets will be delivered in the same order in which they are sent. Being connection-oriented means that before transmitting data, the connection between the server and the client is opened. The data can be transferred in full duplex. When the transfer is completed, the connection is closed to free system resources. Handshaking signals are used for making and closing the connection. If time delay or data loss takes place, the client re-requests the packet to the server until the whole packet is completed. Although TCP is a reliable protocol for network communications, it is unsuitable for a client requiring a high sampling frequency such as Client 1. Due to various services like error checking and ensuring ordered data delivery, TCP has large overheads and may waste bandwidth. An additional time delay can be introduced to the system.

UDP can be an alternative network-protocol choice. UDP is connectionless protocol, and a datagram can be sent at any moment without any preparation. UDP does not guarantee that the datagram will be delivered to the destination host, and it can also be delivered in an incorrect order. Although UDP is unreliable protocol, it has fewer overheads. This makes UDP much faster compared to TCP; it introduces less time delay than TCP does.

Based on these considerations, UDP is used for Clients 1 and 2 in this research. Because Client 1 needs a fast response from the server, UDP is more suitable protocol for it. Since UDP has no guarantees for transferring data, predicted control data is needed to maintain the stability of the system. If data loss or time delay takes place, the client uses the predicted control data transmitted in previous data packets.

Ambike designed an 8th order 4-step-ahead predictor for Client 1 using with auto-regressive (AR) models. The following prediction equations were developed for Client 1 using MATLAB [17].

$$\begin{aligned}\hat{y}(t+1) = {}& 0.8122y(t) - 0.3479y(t-1) - 0.0294y(t-2) + 0.4605y(t-3) \\ & + 0.0742y(t-4) + 0.1042y(t-5) + 0.1117y(t-6) - 0.3561y(t-7)\end{aligned} \tag{4.1}$$

$$\begin{aligned}\hat{y}(t+2) = {}& 0.3117y(t) - 0.3119y(t-1) + 0.4366y(t-2) + 0.4482y(t-3) \\ & + 0.1645y(t-4) + 0.1964y(t-5) - 0.2653y(t-6) - 0.2892y(t-7)\end{aligned} \tag{4.2}$$

$$\begin{aligned}\hat{y}(t+3) = {}& -0.0587y(t) + 0.3281y(t-1) + 0.4390y(t-2) + 0.3080y(t-3) \\ & + 0.2195y(t-4) - 0.2329y(t-5) - 0.2544y(t-6) - 0.1110y(t-7)\end{aligned} \tag{4.3}$$

$$\begin{aligned}\hat{y}(t+4) = {}& 0.2804y(t) + 0.4594y(t-1) + 0.3097y(t-2) + 0.1925y(t-3) \\ & - 0.2372y(t-4) - 0.2605y(t-5) - 0.1176y(t-6) + 0.0209y(t-7)\end{aligned} \tag{4.4}$$

4.1.4 Integrating the Windows-based Robotic Wheelchair to the Linux Server Using Samba

Due to the OS mismatch between Linux server and Client 3, Samba [31] software is used to put both systems together on the same Local Area Network (LAN) as shown in Fig. 4-2. Samba allows the Linux PC to interact with the Windows-based PC.

Using Samba, the wheelchair robot is connected to the server on the same LAN with a fast data-transmission speed. With the Packet Internet groper (PING) utility, the round-trip time of interfacing between Linux server and Client 3 can be measured [32]. PING is a simple application used to check whether a host PC is online and available. PING makes Internet Control Message Protocol (ICMP) messages. The purpose of an ICMP is to inform sending hosts about errors encountered in the IP datagram processing or other control information by destination hosts. PING sends one or more ICMP Echo messages to a specified host, requesting a reply.

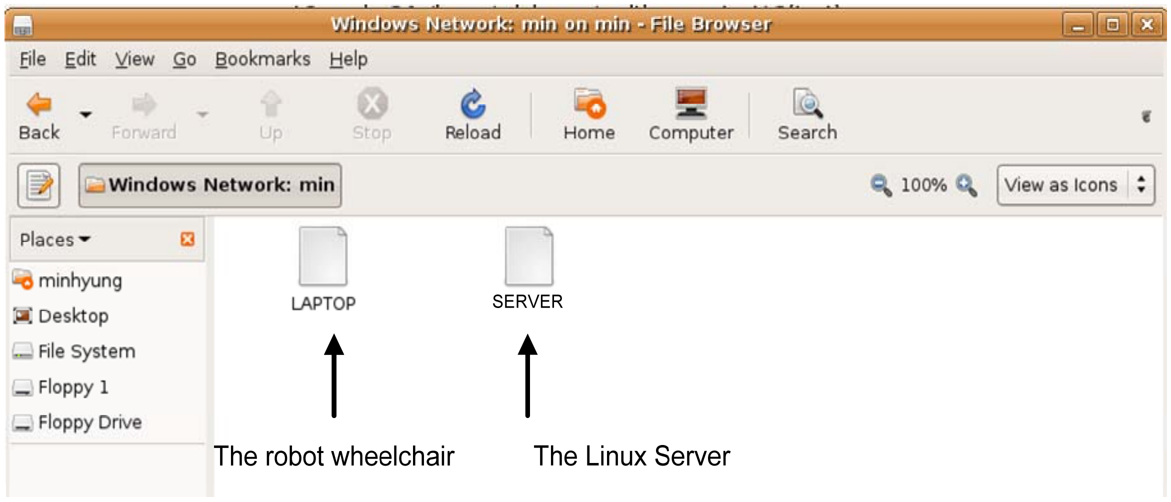


Fig. 4-2. Robotic wheelchair and the Linux server are on a same LAN using Samba

The round-trip times between Windows-based Client 3 and Linux server with Samba are compared to those without using Samba as shown in Table 4-1. It takes about 30.8 ms to process a round-trip task between the robotic wheelchair and the server without Samba whereas just 11.2 ms is taken using Samba. The round-trip time without

Samba is about three times as long as using Samba. This delayed time can have a negative effect on the system stability.

Table 4-1. Round-trip time between Window-Based Client 3 and Linux server

No.	with Samba			without Samba		
	Min.	Max.	Avg.	Min.	Max.	Avg.
1	1 ms	47 ms	10 ms	4 ms	156 ms	27 ms
2	2 ms	56 ms	12 ms	5 ms	100 ms	26 ms
3	1 ms	83 ms	14 ms	6 ms	167 ms	38 ms
4	1 ms	53 ms	11 ms	4 ms	162 ms	29 ms
5	2 ms	62 ms	9 ms	5 ms	169 ms	34 ms
Average	1.4 ms	60.2 ms	**11.2 ms**	4.8 ms	150.8 ms	**30.8 ms**

Using Samba has two advantages. First, the data-transmission time between the server and the client with Samba is faster than without it. The data-transmission time is an important factor for system stability. The wheelchair robot cannot avoid an obstacle unless it receives the control data from the server in time. The detailed data-transmission time between the wheelchair robot and the server is provided in Section 5.2.3. The second advantage of using Samba is that the data file of the wheelchair robot can be saved on the server side. This data file contains the output signals of the Hall-effect sensors. The wheelchair robot can read and write the data file in a shared folder on the Linux server. The user on the server side can access the data file easily as depicted in Fig. 4-3.

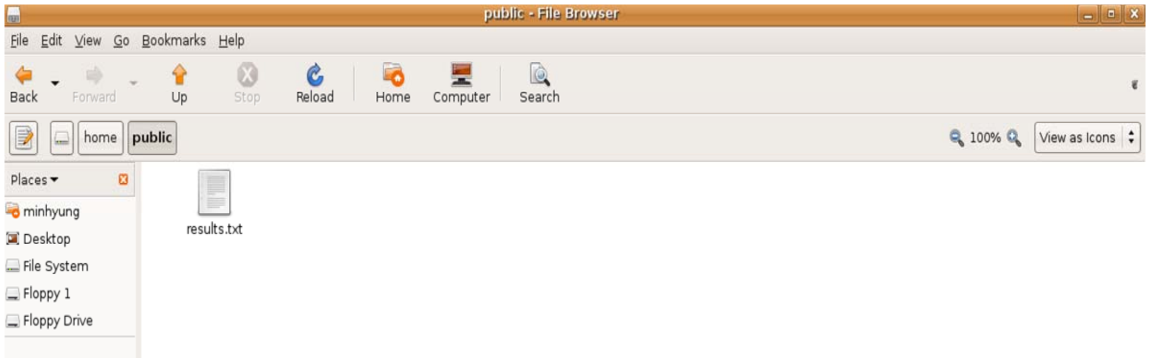


Fig. 4-3. Saved data file of Client 3 in a shared folder on the Linux server

4.2 Development of Software Architecture

In order to identify the individual clients, client identification numbers are needed for the server. Each client sends sensor data packets including its unique identification number to the server side. The server receives and reads the sensor data packets and then generates the control data packets for each client. The compositions of the sensor data packet and the control data packet for the multiple-client NCS are provided in this section. The developed software architecture is also given in the last part of this section. For all software codes presented in this section, refer to Appendix.

4.2.1 Identification of Clients

In order for the server to identify the client that sent a data packet, the identification number of each client is included in the UDP packet from the client side.

Once the sensor data arrive at the server side, the server reads the identification number and then calculates the control data for the specific client. The algorithm to identify the client on the sever side is shown in Fig. 4-4.

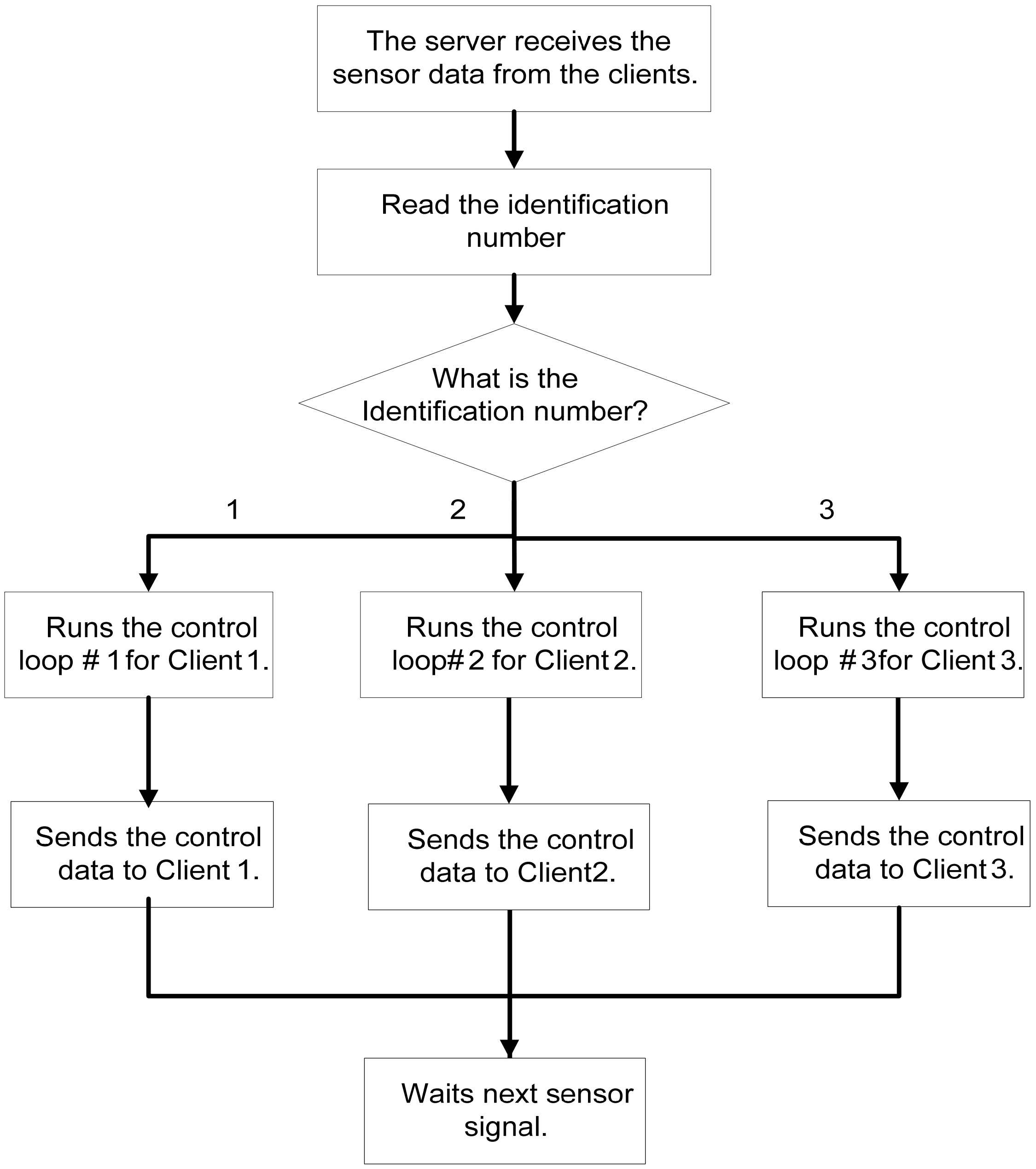

The server receives the sensor data from the clients.
Read the identification number
What is the Identification number?
1
2
3
Runs the control loop # 1 for Client 1.
Runs the control loop# 2 for Client 2.
Runs the control loop # 3for Client 3.
Sends the control data to Client 1.
Sends the control data to Client2.
Sends the control data to Client 3.
Waits next sensor signal.

Fig. 4-4. Algorithm to identify the client on the server side

4.2.2 UDP Packet Composition

With UDP, the composition of IP packet was developed. The composition of a typical 68-bytes-long sensor data packet going from the client to the server is shown in Fig. 4-5. It consists of a 20-byte-long IP header, an 8-byte-long UDP header, an 8-byte-long time stamp, a seven 4-byte-long sensor data values, and a 4-byte-long identification number. A time stamp is taken on the client side at sampling and the server sends it back to the client for identifying whether the arrived data packet is the expected data packet or a delayed one. Delayed data packets are discarded by the client. Using the identification number, the server can identify the client. Clients 1, 2, and 3 have their unique identification numbers as 1, 2, and 3, respectively. After sensor data arrive from the client, the server reads the last section of the data packet and identifies which client sent the data. Then the server calculates the control data for this client.

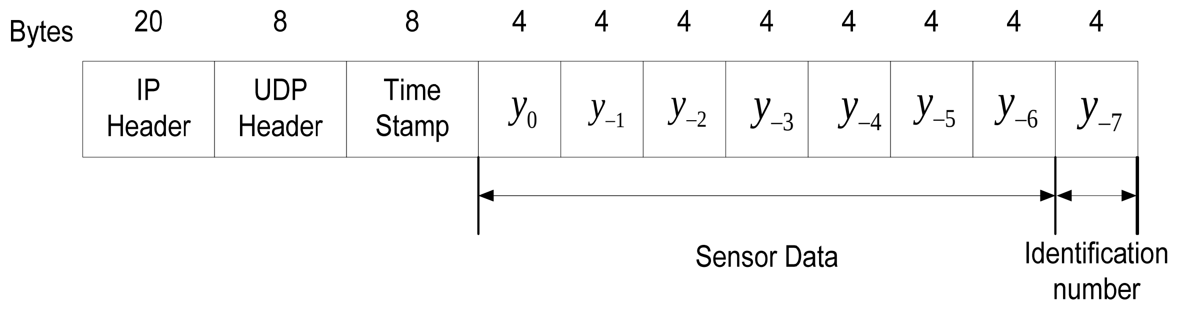


Fig. 4-5. Modified composition of a sensor data packet from the client to the server from [33]

The composition of a typical 56-byte-long control data packet transferred from the server to the client is shown in Fig. 4-6. It consists of a 20-byte-long IP header, an 8-byte-long UDP header, an 8-byte-long time stamp, one 4-byte-long current control data value, and four 4-byte-long predicted control data values.

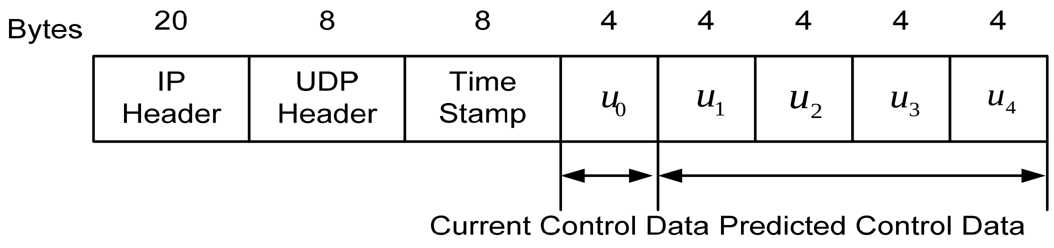


Fig. 4-6. Composition of a control data packet from the server to the client [33]

4.2.3 Network Interface for Client 3

For the network communication, socket programming is used. A client makes a socket consisting of a sensor signal and sends this socket to the server. After receiving the socket, the server reads it and makes socket of control data. Then, the server sends back the socket of control data to the client for its performance.

In this research, the data sockets that are transferred between the server and the client are of a user defined type (UDT). Client 3 with Microsoft Visual Basic 6.0 uses Winsock for data transmission. However using Winsock, we can transfer only string-data-type packets. In order to make network connection between Linux server and Client 3, an additional component that can convert data types is needed. This component is called as Gateway in this research. Gateway makes UDT sockets for the server using string-data-type packets from Client 3 and produces string-data-type packets for Client 3 using with UDT sockets from the server. Software code for Gateway is written in C programming language on Linux. A schematic of network interface between Linux server and Client 3 is shown in Fig. 4-7.

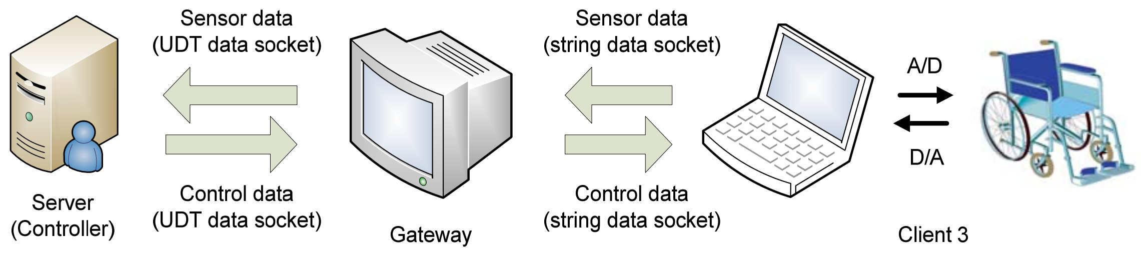


Fig. 4-7. Schematic of network interface between Linux server and Client 3

Client 3 is controlled by Linux server over a wireless network. Hsieh [22] developed the wireless connection using with the Tamulink Wi-Fi access as shown in Fig. 4-8 [34]. The user on the server side could control the robotic wheelchair with Winsock. Since the controller was placed on the client side as in supervisory control, however, the server could not receive any feedback signal from the client.

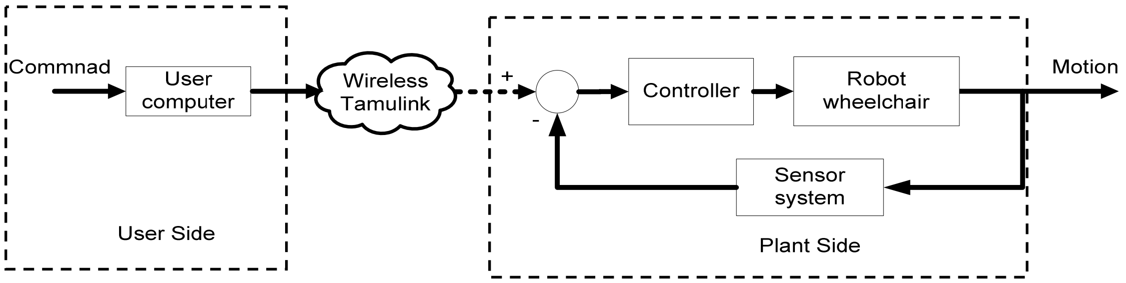


Fig. 4-8. Block diagram of supervisory control of the robotic wheelchair [22]

Fig. 4-9 depicts the feedback control over a wireless network realized in this thesis of Client 3. The sensor data from Client 3 are sent to Gateway and the control data from Gateway are transferred to Client 3 over the Tamulink wireless network. Gateway is connected with Linux server over Tamulink wired network. In this system, the controller is placed on the server side. Its interface with Client 3 is shown in Fig. 4-9. The dashed arrows between Gateway and Client 3 indicate the Tamulink wireless network.

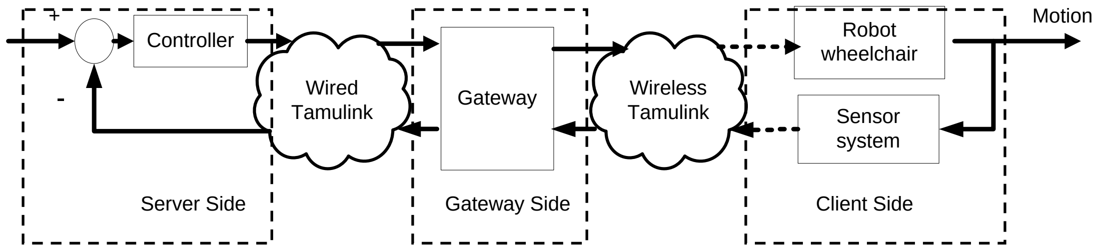
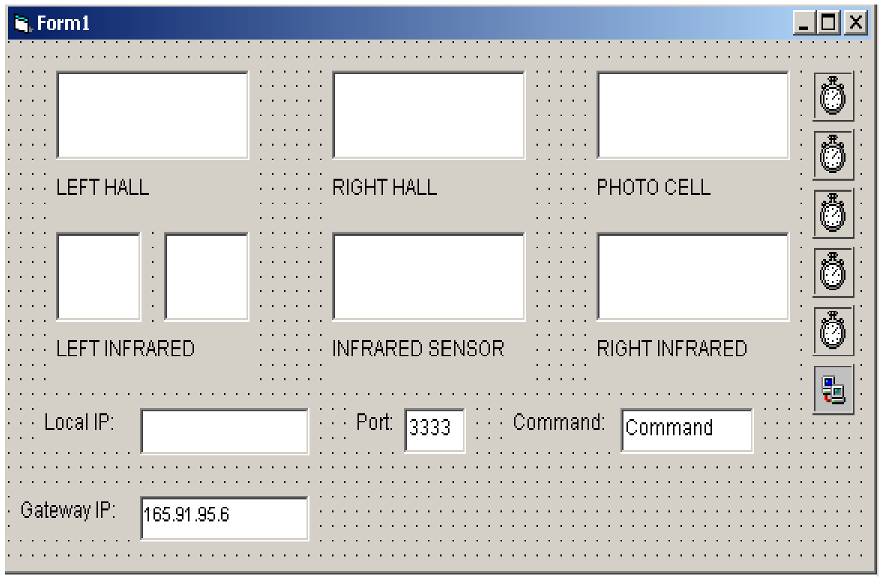


Fig. 4-9. Schematic of feedback control over the Tamulink wireless network of Client 3



Local IP:

Port: 3333

Command: Command

Gateway IP: 165.91.95.6

Fig. 4-10. Interface of the Client 3 program

The software program of Client 3 was written in Microsoft Visual Basic 6.0. Fig. 4-10 shows the user interface when the Client 3 program starts. All sensor signals are displayed in each window and are sent to Gateway.

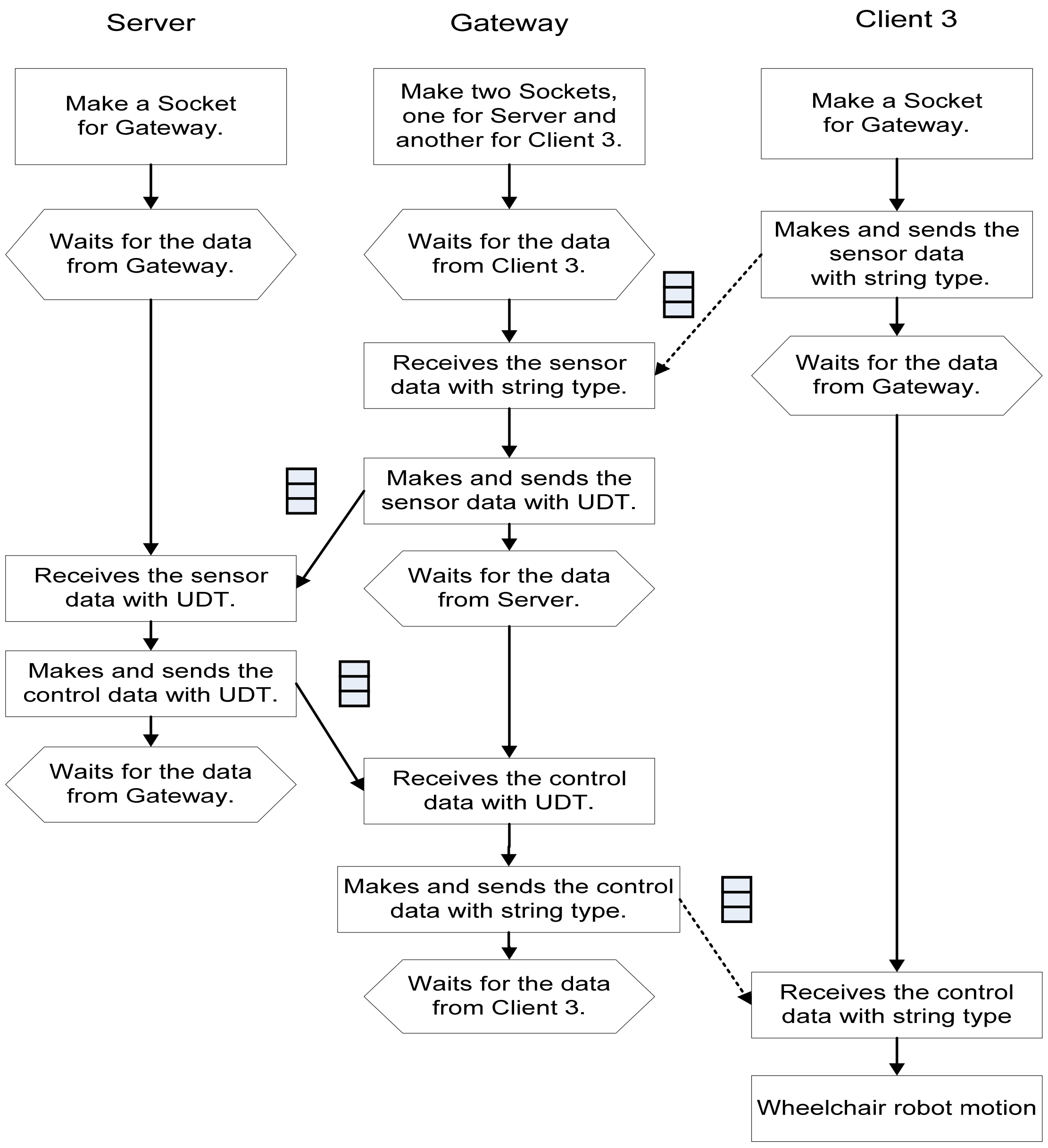

Server
Gateway
Client 3
Make a Socket for Gateway.
Make two Sockets, one for Server and another for Client 3.
Make a Socket for Gateway.
Waits for the data from Gateway.
Waits for the data from Client 3.
Makes and sends the sensor data with string type.
Receives the sensor data with string type.
Waits for the data from Gateway.
Makes and sends the sensor data with UDT.
Receives the sensor data with UDT.
Waits for the data from Server.
Makes and sends the control data with UDT.
Waits for the data from Gateway.
Receives the control data with UDT.
Makes and sends the control data with string type.
Waits for the data from Client 3.
Receives the control data with string type
Wheelchair robot motion

Fig. 4-11. Overall data-transmission architecture for Client 3

The overall data-transmission architecture for Client 3 is depicted in Fig. 4-11. The dashed lines between Gateway and Client 3 indicate actual data-packet transmissions over the Tamulink wireless network. Gateway creates two sockets; ‘sd’ for Client 3 and ‘sockid’ for Linux server. The source code to create sockets is given as

```
sd = socket(AF_INET, SOCK_DGRAM, 0)
sockid = socket(AF_INET, SOCK_DGRAM, 0).
```

When the sensor data socket arrives from Client 3, Gateway saves it with one socket and reads the data. The source code to receive packets is shown as

```
cr = recvfrom(sd, recv_msg, 10, 0, (struct sockaddr *)
&client_addr, &clilen).
```

The length of sensor data is fixed as 10 bytes which does not include header data. The UDT packets for Linux server are made and sent as shown in Fig. 4-12.

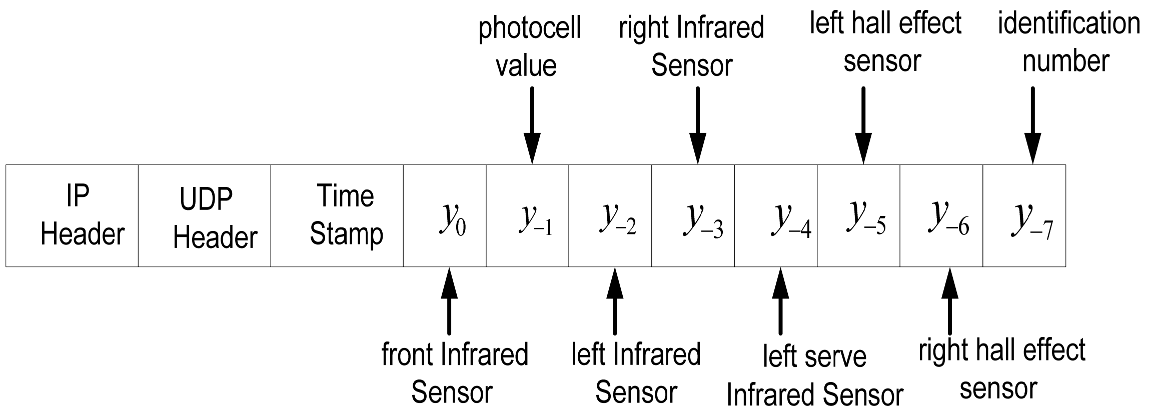


Fig. 4-12. UDT sensor data packet created by Gateway

The source code for sending UDT packets from Gateway to Linux server is given as follows.

```
nw=sendto(sockid, (const void *)send_buffer, send_buffer_size, 0,(struct sockaddr *) &server_addr, addrlen)
```

Linux server receives the UDT packets and makes the control data packets as shown in Fig. 4-13. The movement-commnad value u_0 can be 10, 7, 5, 2, and 0. The meanings of each value are described in Table 4-2. The UDT control data are converted to the string-data-type control data by Gateway.

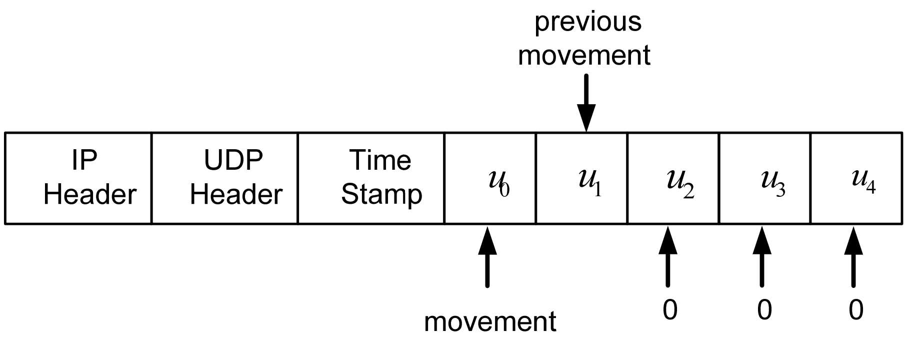


Fig. 4-13. UDT control data packet created by Linux server

Table 4-2. Meanings of UDT control data for movement

value	meaning	convert as
10	Moving Forward	fwd
7	Turn Right	right
5	Moving Back	back
2	Turn Left	left
0	Stop	stop

With the UDT control data packets from Linux server, Gateway makes another control data packets for Client 3. The source code for sending the control data is given as follows.

```
Switch(u0){
  Case(10):
    cw  =  sendto(sd,  fwd,  5,  0,  (struct  sockaddr  *)_
&client_addr, clilen);
    break;
  Case(7):
    cw  =  sendto(sd,  right,  5,  0,  (struct  sockaddr  *)_
&client_addr, clilen);
    break;
 Case(5):
    cw  =  sendto(sd,  stop,  5,  0,  (struct  sockaddr  *)_
&client_addr, clilen);
    break;
 Case(2):
    cw  =  sendto(sd,  left,  5,  0,  (struct  sockaddr  *)_
&client_addr, clilen);
    break;
 Case(0):
    cw = sendto(sd, back, 5, 0, (struct sockaddr *)_
&client_addr, clilen);
    break;
}
```

Client 3 receives and reads the control data with string-data-type. The source code for receiving and calling the motion function are given as follows.

```
Dim recData As String
Winsock.GetData recData  ‘receiving the data packet
Select Case recData
     Case “fwd”
          Call Front
     Case “right”
          Call TurnRight
     Case “left”
          Call TurnLeft
     Case “back”
          Call Back
     Case “stop”
          Call StopRobot
```

According to the called motion functions, the robotic wheelchair moves with collision-avoidance. The Gateway program is included in Appendix of this thesis.

CHAPTER V

OPERATION AND TESTING

In the previous chapter, the designed software programs for each client or server system are presented. This chapter describes how the sampling period of each system is determined and tested. In order to obtain the sampling period of each system, the operation time for each control loop is measured. Based on the relation between the bandwidth utilization and the sampling period for each control loop, the ranges of the sampling periods to ensure stability are presented. In the last part of this chapter, possible combinations of the sampling periods are suggested and experimented.

5.1 The Operation Time for Each Control Loop

From [19], the relation between the bandwidth and the sampling period for each control loop is given by

$$b_i = \frac{\tau_i}{h_i}, \quad 0 < b_i < 1, \tag{5.1}$$

where b_i is assignable partial bandwidth to the i th control loop, and τ_i is the operation time to perform each closed-loop operation. The operation time τ_i can be expressed as

$$\tau_i = T_{i,SC} + T_{i,CS} + T_{i,P}, \tag{5.2}$$

where $T_{i,SC}$ is the data-transmission time from the server to the client and $T_{i,CS}$ is the data-transmission time from the client to the server as shown in Fig. 5-1. Let $T_{i,P}$ be the time needed for data processing such as sensor sampling, actuator actuating, and controller calculating the control data for control loop i. The data processing time is given as

$$T_{i,P} = T_{i,PS} + T_{i,PC} \tag{5.3}$$

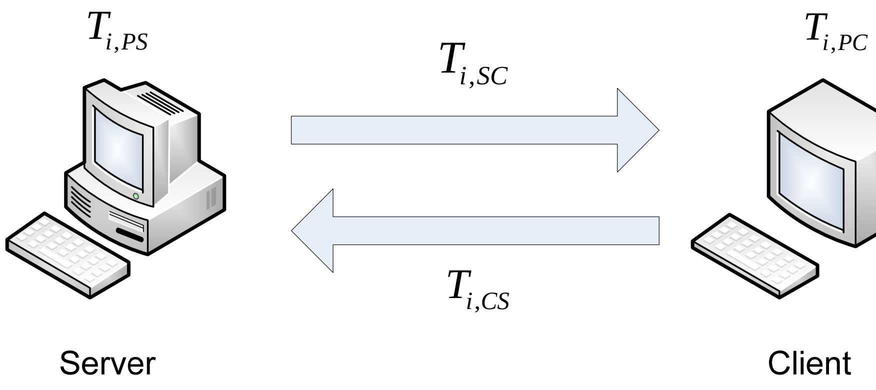


Fig. 5-1. Operation time to performed each control loop i.

The PING utility is also used to measure $T_{i,SC}$ and $T_{i,CS}$ for each system. Because the sizes of control data and sensor data are different, the PING test was performed separately. Since the PING measures the round-trip time, $T_{i,SC}$ and $T_{i,CS}$ can be obtained by dividing the round-trip time by 2.

In order to measure the data-processing time for each system, the functions `timeGetTime()` for Client 3 written in Microsoft Visual Basic 6.0 and `rt_get_cpu_time_ns()` for a Linux PC are used. The `timeGetTime()` function returns the time in milliseconds, and the `rt_get_cpu_time_ns()` function, in nanoseconds. The software architecture to measure the data-processing time is depicted in Fig. 5-2.

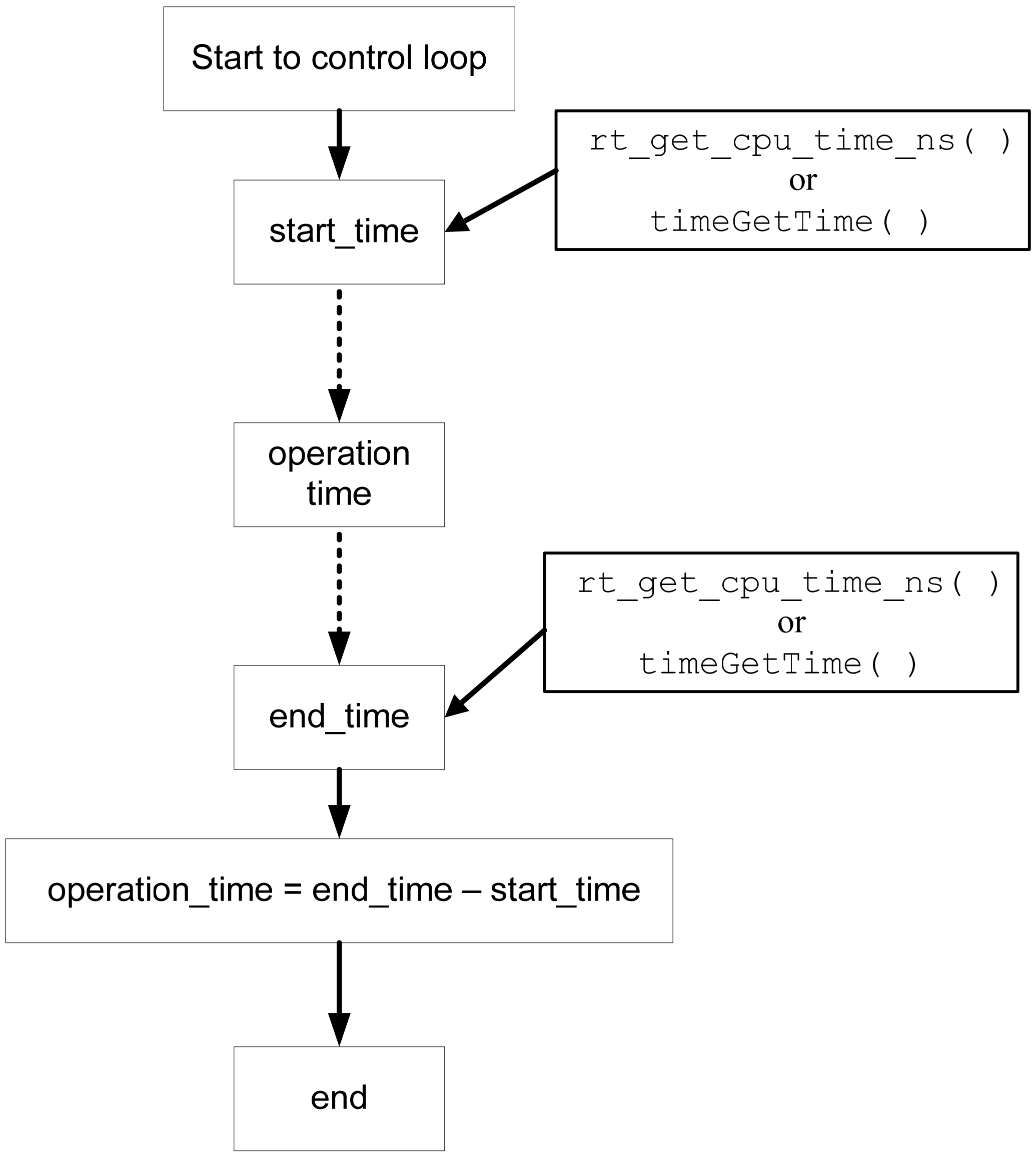

Start to control loop
rt_get_cpu_time_ns( ) or timeGetTime( )
start_time
operation time
rt_get_cpu_time_ns( ) or timeGetTime( )
end_time
operation_time = end_time – start_time
end

Fig. 5-2. Software architecture of measuring the data processing time

5.1.1 Client 1

The round-trip time between Linux server and Client 1 is measured by the PING test five times. Each PING test takes 100 round-trip times and gives the minimum, maximum, and average value. The results of these experiments are presented in Tables 5-1 and 5-2.

Table 5-1. Round-trip time from Linux server to Client 1, $T_{1,SC}$

Experiment No.	Bytes in packet	Round-trip times (ms)		
		Minimum	Maximum	Average
1	56	0.342	0.441	0.382
2	56	0.362	0.434	0.388
3	56	0.362	0.447	0.397
4	56	0.346	0.472	0.387
5	56	0.354	0.424	0.392
Average	56	0.353	0.443	**0.388**

Table 5-2. Round-trip time from Client 1 to Linux server, $T_{1,CS}$

Experiment No.	Bytes in packet	Round-trip times (ms)		
		Minimum	Maximum	Average
1	68	0.352	0.454	0.391
2	68	0.344	0.443	0.392
3	68	0.337	0.473	0.386
4	68	0.363	0.456	0.394
5	68	0.358	0.478	0.402
Average	68	0.351	0.461	**0.393**

The data-processing time for Client 1 can be obtained as shown in Table 5-3.

Table 5-3. Data processing time for Client 1

No.	Server (ms)	Client (ms)	Total (ms)
1	0.102	0.812	0.914
2	0.115	0.800	0.915
3	0.102	0.832	0.934
4	0.105	0.796	0.901
5	0.117	0.792	0.909
Avg.	0.108	0.806	**0.915**

Then, we can obtain the total operation time between Linux server and Client 1 as

$$\begin{aligned}\tau_1 &= T_{1,SC} + T_{1,CS} + T_{1,P} \\ &= \frac{(0.388 + 0.393)}{2} + 0.915 = 1.306\,\text{ms},\end{aligned} \tag{5.4}$$

which is the same result with [23].

5.1.2 Client 2

The round-trip time from Linux server to Client 2 is presented in Table 5-4, and that from Client 2 to Linux server is shown in Table 5-5. $T_{2,SC}$, the transmission time from Linux server to Client 2 is found to be 0.480 ms, and $T_{2,CS}$ the transmission time from Client 2 to Linux server, 0.489 ms.

Table 5-4. Round-trip time from Linux server to Client 2, $T_{2,SC}$

Experiment No.	Bytes in packet	Round-trip times (ms)		
		Minimum	Maximum	Average
1	56	0.402	0.976	0.512
2	56	0.432	0.554	0.476
3	56	0.378	0.506	0.454
4	56	0.408	0.542	0.478
5	56	0.408	0.512	0.478
Average	56	0.406	0.618	**0.480**

Table 5-5. Round-trip time from Client 2 to Linux server, $T_{2,CS}$

Experiment No.	Bytes in packet	Round-trip times (ms)		
		Minimum	Maximum	Average
1	68	0.412	0.532	0.484
2	68	0.432	0.524	0.484
3	68	0.418	0.532	0.492
4	68	0.434	0.522	0.492
5	68	0.410	0.532	0.492
Average	68	0.421	0.528	**0.489**

The transmission time between Linux server and Client 2 is longer than that of Client 2 because of the system capability. The PC of Client 1 is a 1.7-GHz Pentium IV processor whereas the PC for Client 2 is a 133-MHz Pentium III. This difference of system capacity may introduce the time delay between the server and the client.

The data-processing time of Client 2 was measured as shown in Table 5-6. On the server side, the processing time is smaller than that of Client 1. It means that the time to calculate the control data for a DC motor is shorter than that for the ball-maglev system.

Table 5-6. Data processing time for Client 2

No.	Server (ms)	Client (ms)	Total (ms)
1	0.096	0.771	0.867
2	0.092	0.771	0.863
3	0.088	0.775	0.863
4	0.097	0.772	0.869
5	0.099	0.771	0.87
Avg.	0.094	0.772	**0.866**

Then, we can obtain the operation time between Linux server and Client 2 as

$$\tau_2 = T_{2,SC} + T_{2,CS} + T_{2,P}$$

$$= \frac{(0.480 + 0.489)}{2} + 0.866 = 1.350 \text{ ms.} \tag{5.5}$$

5.1.3 Client 3

In order to measure the operation time for Client 3, we need to take Gateway into consideration. Thus, the operation time for Client 3 is given by

$$\tau_3 = T_{3,SG} + T_{3,GS} + T_{3,GC} + T_{3,CG} + T_{3,P} \quad , \tag{5.6}$$

where $T_{3,SG}$ is the data-transmission time from Linux server to Gateway, $T_{3,GS}$ is the data-transmission time from Gateway to Linux server, $T_{3,GC}$ is the data-transmission time from Client 3 to Gateway, and $T_{3,CG}$ is the data-transmission time from Client 3 to Gateway. $T_{3,P}$ is data-processing time as given

$$T_{3,P} = T_{3,PS} + T_{3,PG} + T_{3,PC} \quad . \tag{5.7}$$

Fig. 5-3 describes the operation time for Client 3 with Gateway.

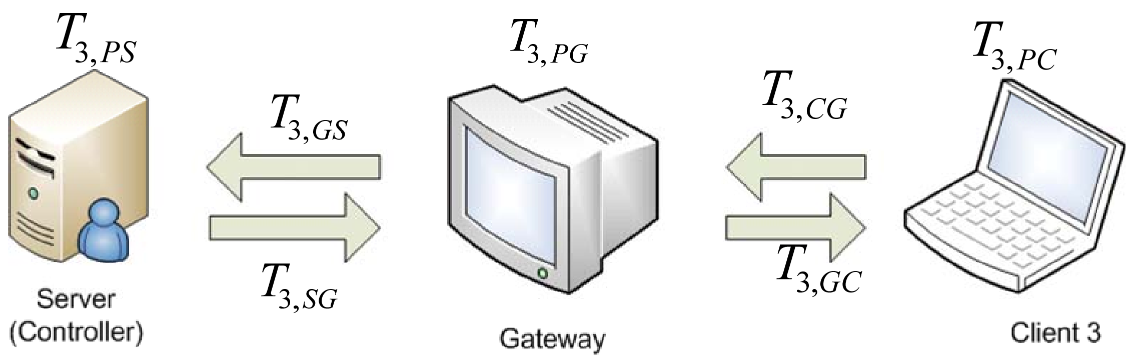


Fig. 5-3. Operation time for Client 3 with Gateway

The round-trip time between Linux server and Gateway is shown in Tables 5-7 and 5-8. The transmission time between Linux server and Gateway is longer than other clients since these two PCs are connected with different LANs. Fig. 5-4 describes the LAN connections for the multiple-client NCS developed in this research. Gateway and Client 3 are on the same LAN and connected to the default gateway 2 with the IP address 192.168.2.100 whereas Linux server, Clients 1, and 2 are connected to the default gateway 1 with the IP address 165.91.95.1. To transfer the data packet over different LANs, the data pass through these default gateways 1 and 2. That is why the transmission time between Linux server and Gateway is longer than that between Linux server and Client 1 or 2. In the case of the transferring the data packet in a same LAN, the data packet can be transmitted by passing through just its default gateway.

Table 5-7. Round-trip time from Linux server to Gateway, $T_{3,SG}$

Experiment No.	Bytes in packet	Round-trip times (ms)		
		Minimum	Maximum	Average
1	56	0.856	1.896	1.133
2	56	0.738	1.289	0.828
3	56	0.735	1.560	0.841
4	56	0.740	1.966	0.925
5	56	0.754	1.291	0.814
Average	56	0.765	1.600	**0.908**

Table 5-8. Round-trip time from Gateway to Linux server, $T_{3,GS}$

Experiment No.	Bytes in packet	Round-trip times (ms)		
		Minimum	Maximum	Average
1	68	0.747	1.744	0.962
2	68	0.764	2.680	0.967
3	68	0.752	2.682	0.945
4	68	0.770	1.700	0.934
5	68	0.748	2.503	0.883
Average	68	0.756	2.262	**0.938**

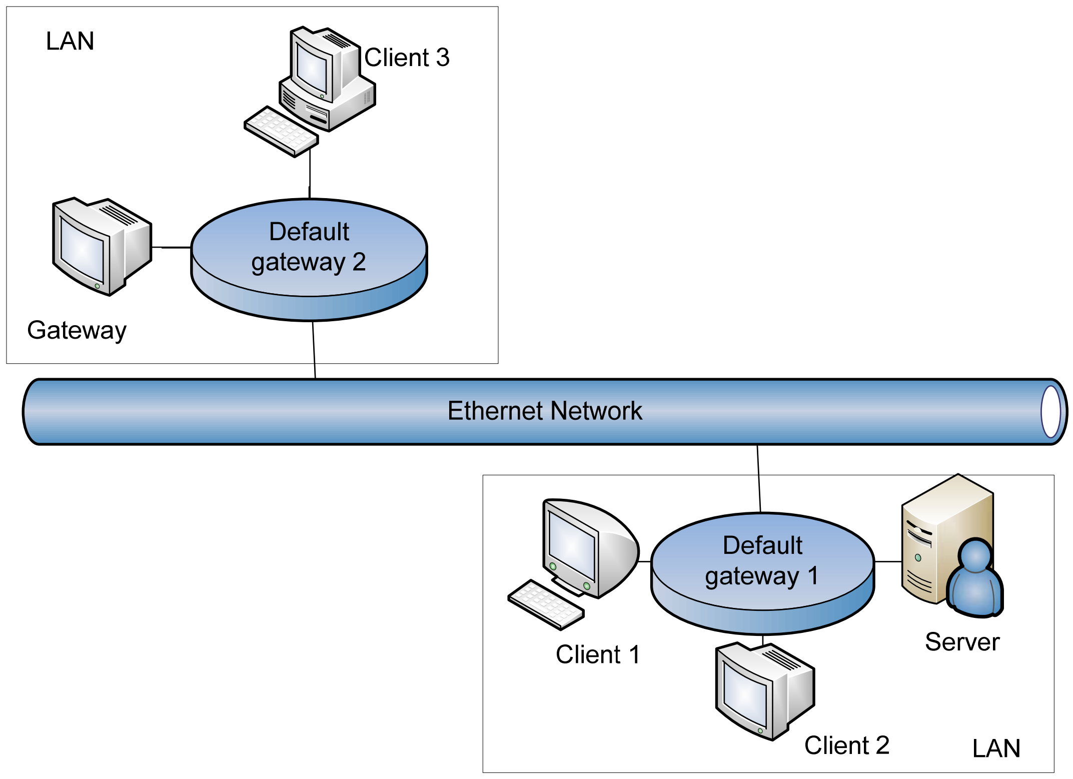


Fig. 5-4. LAN connections for the multiple-client NCS

The results of the transmission time between Client 3 and Gateway with Samba is shown in Table 5-9 and Table 5-10. Since Windows-based Client 3 is connected to Gateway with Tamulink wireless network, the transmission time between Client 3 and Gateway is very long. Since the wireless network is unstable and the round-trip times fluctuate widely, each PING test takes 1000 round-trip times in this experiment whereas other experiments over the wired network take just 100 round-trip times. In order to reduce the transmission time between Client 3 and Gateway, the both systems were put on the same LAN using Samba. $T_{3,CG}$ is measured by Window-based Client 3 with a 1-ms resolution whereas $T_{3,CG}$ is measured by Gateway with a 1-ns resolution.

Table 5-9. Round-trip time from Client 3 to Gateway, $T_{3,CG}$

Experiment No.	Bytes in packet	Round-trip times (ms)		
		Minimum	Maximum	Average
1	38	1	52	12
2	38	2	45	14
3	38	1	45	11
4	38	1	64	13
5	38	1	63	11
Average	38	1.2	55.8	**12.2**

Table 5-10. Round-trip time from Gateway to Client 3, $T_{3,GC}$

Experiment No.	Bytes in packet	Round-trip times (ms)		
		Minimum	Maximum	Average
1	33	1.063	13.117	10.571
2	33	1.734	19.448	11.646
3	33	1.973	25.723	13.408
4	33	1.104	36.587	12.460
5	33	1.287	38.367	11.763
Average	33	1.432	26.649	**11.970**

The processing time for Linux server, Gateway, and Client 3 is presented in Table 5-11. In Gateway, the processing time is the time for converting the data from Client 3 to Linux server or from Linux server to Client 3. Total processing time is 10.66 ms for the control loop. The data-processing time for Client 3 was measured by the Windows `timeGetTime()` with a 1-ms resolution.

Table 5-11. Data-processing time for Client 3

No.	Server (ms)	Gateway (ms)	Client (ms)	Total (ms)
1	0.107	1.312	9	10.419
2	0.110	1.125	10	11.253
3	0.110	1.115	10	11.225
4	0.101	1.112	9	10.213
5	0.112	1.114	9	10.226
Avg.	0.108	1.156	9.400	**10.660**

We can obtain the operation time between Linux server and Client 3 as

$$\tau_3 = T_{3,SG} + T_{3,GS} + T_{3,GC} + T_{3,CG} + T_{3,P} \tag{5.8}$$

$$= \left(\frac{0.908 + 0.938 + 12.2 + 11.970}{2} \right) + 10.660 = 23.668 \ . \tag{5.9}$$

5.1.4 Overall Operation Time for Each System

The overall round-trip operation time for each system is shown in Table 5-12. The operation time for Client 1 is similar with Client 2. It is because both systems are run on the same computational environment such as OS, programming language, and LAN. Although Client 3 is connected to Gateway with Samba, the operation time for Client 3 is about 19 times longer than Clients 1 and 2, Since Client 3 uses wireless network on Windows-based OS. Using Gateway between Linux server and Client 3 also can be one of the reasons for this result.

Table 5-12. Overall operation time for each client system

time	Client 1	Client 2	Client 3
transmission time (ms)	0.391	0.484	13.008
processing time (ms)	0.915	0.866	10.660
operation time (ms)	1.306	1.350	23.668

5.2 Operation of Each Client

To verify the working of the control strategy, each client is tested in this section. Based on the operation time, the bandwidth utilization is determined with respect to the sampling period for each client. With these considerations, the possible combinations of the sampling period for each client are suggested and tested.

5.2.1 Client 1

The digital controller for the 333.3 Hz sampling frequency was designed in [23] as

$$D(z) = 4.15 \times 10^4 \frac{z^2 - 1.754z + 0.769}{z^2 - 0.782z - 0.13} \quad . \tag{5.10}$$

The sampling frequency was 333.3 Hz, and the test of Client 1 is conducted with the 3-ms sampling period in the control loop. Fig. 5-5 shows the ball displacement from its equilibrium position. The system remained stable and the ball did not fall down from its equilibrium position. From [33], the controller with the 3-ms sampling period is found as the best sampling period through the simulation. Therefore, the controller with 3-ms sampling period is used for the possible combinations of the sampling period for Client 1. From (5.4), the bandwidth utilization for Client 1 is given by

$$b_1 = \frac{\tau_1}{h_1} = \frac{1.306\,\text{ms}}{3\,\text{ms}} = 0.435 \ . \tag{5.11}$$

Thus, 43.5% of the network bandwidth is used to operate Client 1 for the 3-ms sampling period. Since Client 1 with the 3-ms sampling period has a nice performance, the bandwidth utilization for Client 1 is fixed as 0.435 in this research.

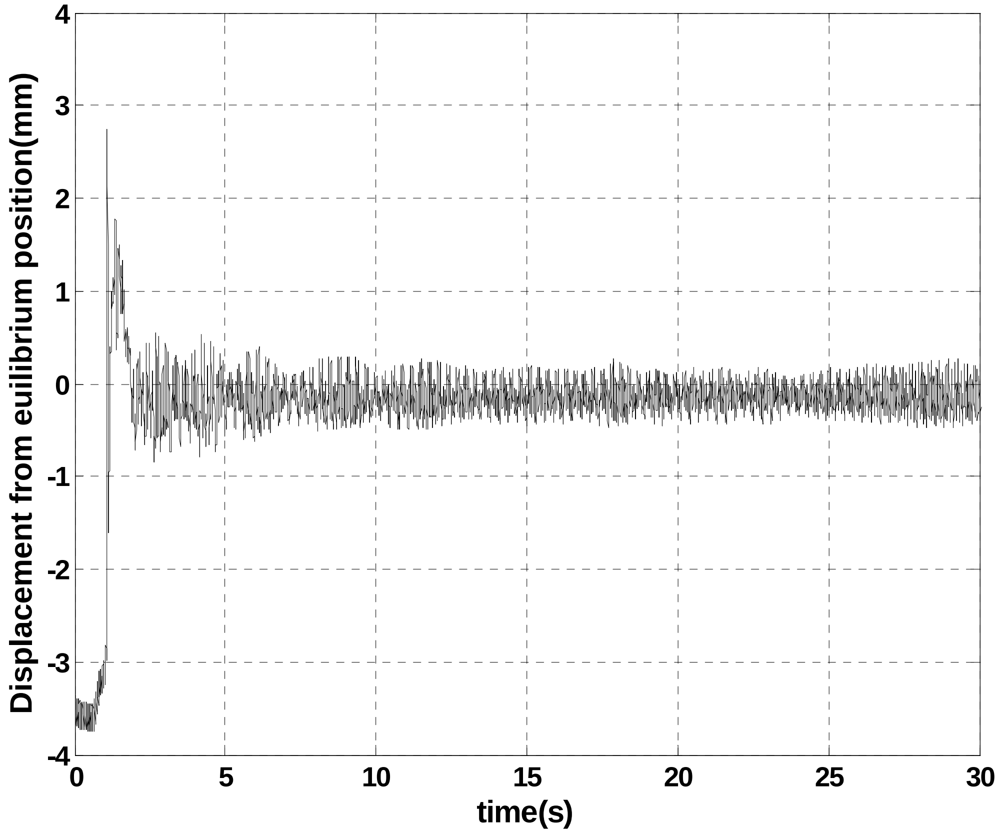


Fig. 5-5. Plot of the ball displacement from the equilibrium position with respect to time

5.2.2 Client 2

From (3.3), the discrete-time controller for Client 2 is developed as

$$u(k) = u(k-1) + [1.5 + 2.5h]e(k) - [2.5h - 1.5]e(k-1) \quad (5.12)$$

Assume that Client 3 needs at least 10% of the bandwidth utilization. Then the bandwidth utilization of Client 2 cannot exceed 0.465. With design the discrete-time controller, the sampling period for Client 2 can be controlled. In order to check the system stability, seven cases of experiments are designed and performed as depicted in Table 5-13.

Table 5-13. Cases of experiments for Client 2

No.	Sampling Period	Operation time	Bandwidth utilization	Discrete-time controller *u(k)* =
1	3 ms	1.36 ms	0.453	$u(k-1)+1.5075e(k)-1.4925e(k-1)$
2	5 ms	1.36 ms	0.272	$u(k-1)+1.5125e(k)-1.4875e(k-1)$
3	10 ms	1.36 ms	0.136	$u(k-1)+1.5250e(k)-1.4750e(k-1)$
4	15 ms	1.36 ms	0.091	$u(k-1)+1.5375e(k)-1.4625e(k-1)$
5	20 ms	1.36 ms	0.068	$u(k-1)+1.5500e(k)-1.4500e(k-1)$
6	25 ms	1.36 ms	0.054	$u(k-1)+1.5625e(k)-1.4375e(k-1)$
7	30 ms	1.36 ms	0.045	$u(k-1)+1.5750e(k)-1.4250e(k-1)$

The results of the experiments are described in Fig. 5-6. Every case of experiments is shown as stable system. However, Cases (e), (f), and (g) show a bad system performance and are eliminated from making combinations with other clients.

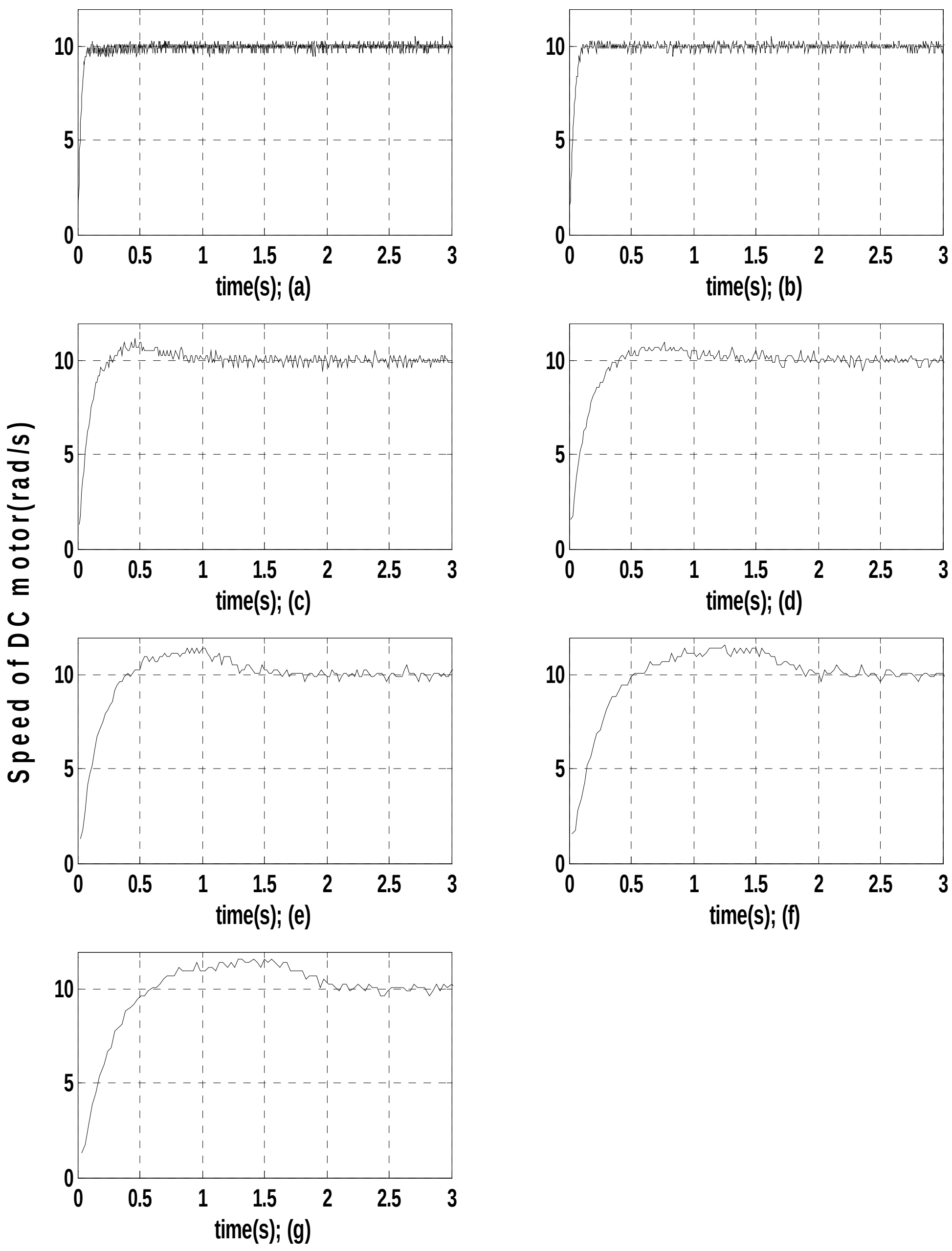


Fig. 5-6. Results of experiments of Client 2 with the sampling periods of (a) 3 ms, (b) 5 ms, (c) 10 ms, (d) 15 ms, (e) 20 ms, (f) 25 ms, and (g) 30 ms

5.2.3 Client 3

The motion of Client 3 can be represented in the two-dimensional coordinate system using by Hall-effect sensor readings [35]. Assume that Client 3 begins at (0, 0) point in the *xy*-plane. The Hall-effect sensors count the pulses every 100-ms sampling interval time. At a sampling time i, the distance d between start point and the position of Client 3 is defined as

$$d = \frac{LH_i + RH_i}{2} \quad , \tag{5.13}$$

where LH_i is the pulse counted by the left-side Hall-effect sensor and RH_i is the pulse counted by the right-side Hall-effect sensor at sampling time i. The point (x_i , y_i) represents the position of Client 3 in the *xy*-plane at sampling time i can be found as

$$x_i = \left(\frac{LH_i + RH_i}{2} \right) \sin \theta_i \tag{5.14}$$

$$y_i = \left(\frac{LH_i + RH_i}{2} \right) \cos \theta_i \ , \tag{5.15}$$

where θ_i is turning angle at sampling time i.

From (5.9), the suggested sampling periods are presented as shown in Table 5-14. According to the bandwidth utilization of Clients 1 and 2, the bandwidth utilization of Client 3 can be varies from 0.110 to 0.474. To check whether the assumption in the previous section is right or wrong, the 300-ms sampling period is also suggested to test in case 4.

Table 5-14. Cases of experiments for Client 3

No.	Sampling Period	Operation time	Bandwidth utilization
1	50 ms	23.668 ms	0.473
2	100 ms	23.668 ms	0.237
3	200 ms	23.668 ms	0.118
4	300 ms	23.668 ms	0.079

The experiments of Client 3 are performed in a ground-floor hallway and Precision Mechatronics Lab inside the Zachery Engineering Center of Texas A&M University as shown in Fig. 5.7. In order to test its collision-avoidance function, a trash can is used as an obstacle. The testing environment is represented in the *xy*-plane with MATLAB as shown in Fig. 5.8. With the `Timer.Interval()` function, the sampling period of robotic wheelchair can be controlled.

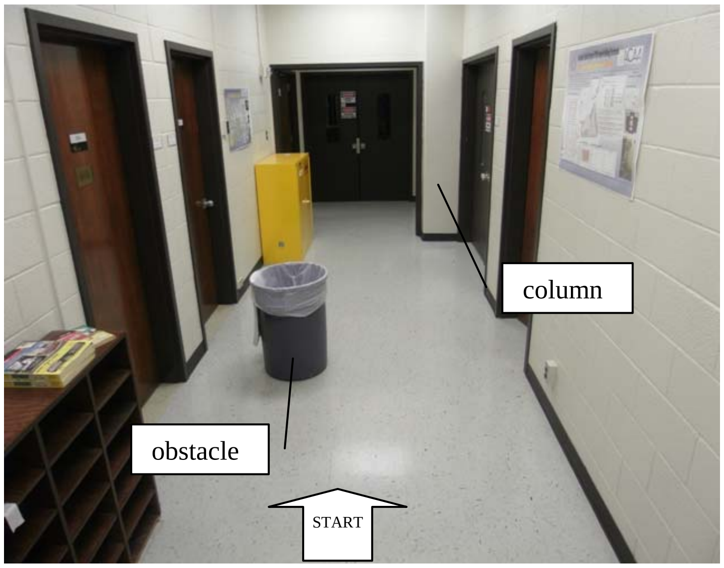


Fig. 5-7. Testing environment with obstacles for Client 3

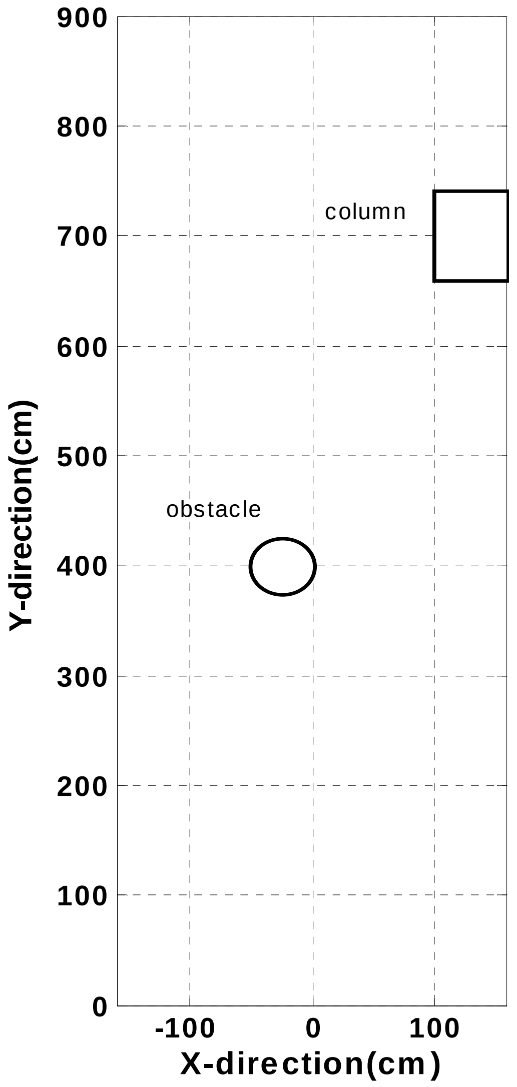


Fig. 5-8. Representing the testing environment in the the *xy*-plane with MATLAB

The results of the experiments are descibed in Fig. 5-9. Except for the case of 300-ms sampling period, the robotic wheelchair could arrive at the destination. Thus, the sampling period of Client 3 cannot exceed 300 ms.

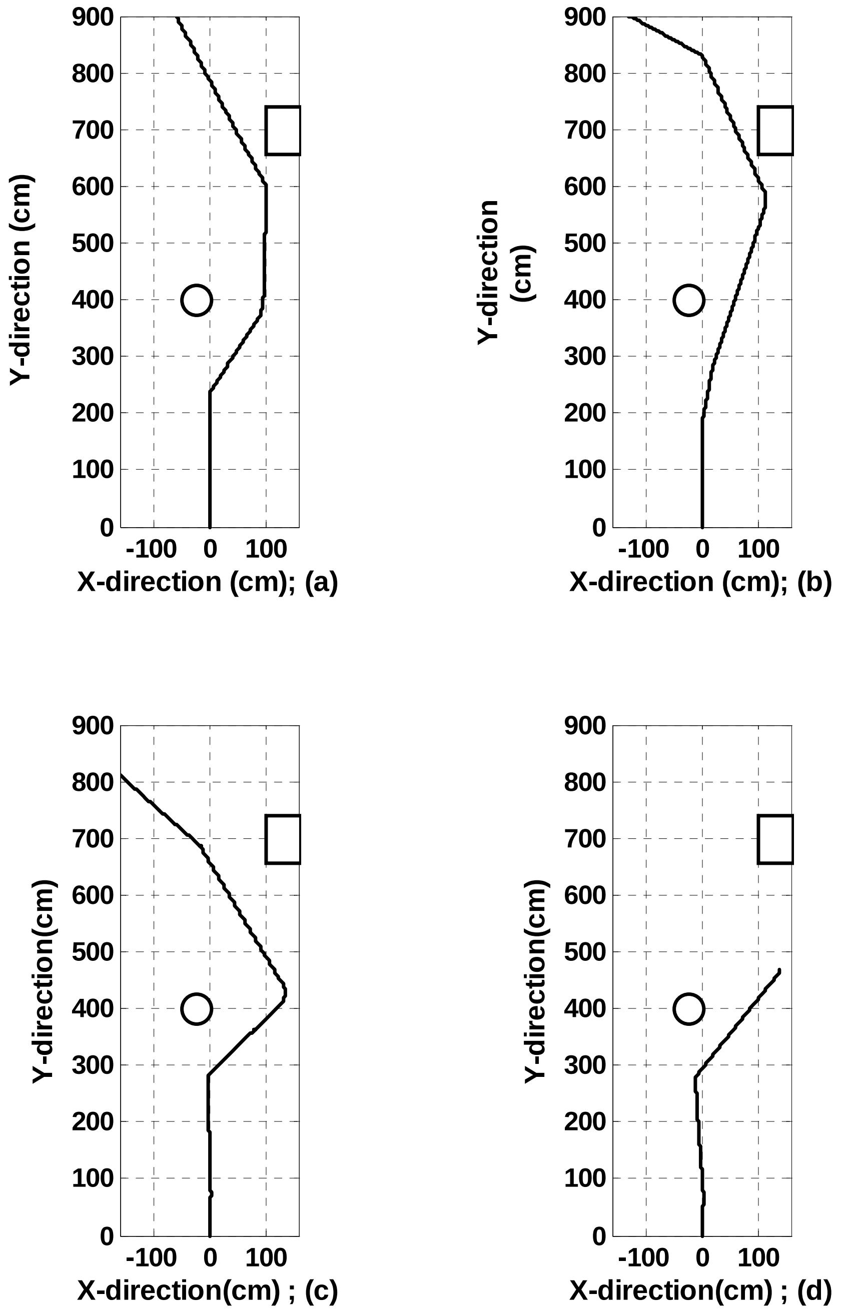


Fig. 5-9. Results of experiments of Client 3 with the sampling periods of (a) 50 ms, (b) 100 ms, (c) 200 ms, and (d) 300 ms

5.3 Suggestion for Possible Combinations of the Sampling Period

Based on the results in the previous section, possible combinations of the sampling period of each client are suggested and verified in this section. The sampling period for Client 1 is fixed and used as 3 ms. Client 2 needs the bandwidth utilization from 0.091 to 0.453 for its stability. The bandwidth utilization of Client 3 can be from 0.118 to 0.468. According to these considerations, the combination range of the sampling period for each client can be obtained. The conditions of the bandwidth utilization for each client is given as

$$b_2 + b_3 \leq 56.5 \tag{5.16}$$

$$9.1 \leq b_2 \leq 45.3 \tag{5.17}$$

$$11.8 \leq b_3 \leq 46.8\,. \tag{5.18}$$

Based on these conditions, the combinations of the sampling period for each client are suggested and tested. Table 5-14 describes the suggested combinations of the sampling period for each client. In order to check the failed case, Case 4 is added to the tests-set.

Table 5-15. Suggested combinations of the sampling period for each control loop

No.	Bandwidth Utilization			Sampling Period		
	Client 1	Client 2	Client 3	Client 1	Client 2	Client 3
1	43.5%	27.2%	28.9%	3 ms	5 ms	82 ms
2	43.5%	13.6%	43%	3 ms	10 ms	53 ms
3	43.5%	9.1%	47.3%	3 ms	15 ms	50 ms
4	43.5%	45.3%	47.3%	3 ms	3 ms	50 ms

Case 1 is performed as shown in Fig. 5-10. Client 1 maintains the stability of system and the ball did not fall down from its equilibrium position. The performance of Client 2 with the 5-ms sampling period is observed as the stable system. Client 3 also can arrive at destination without any collision.

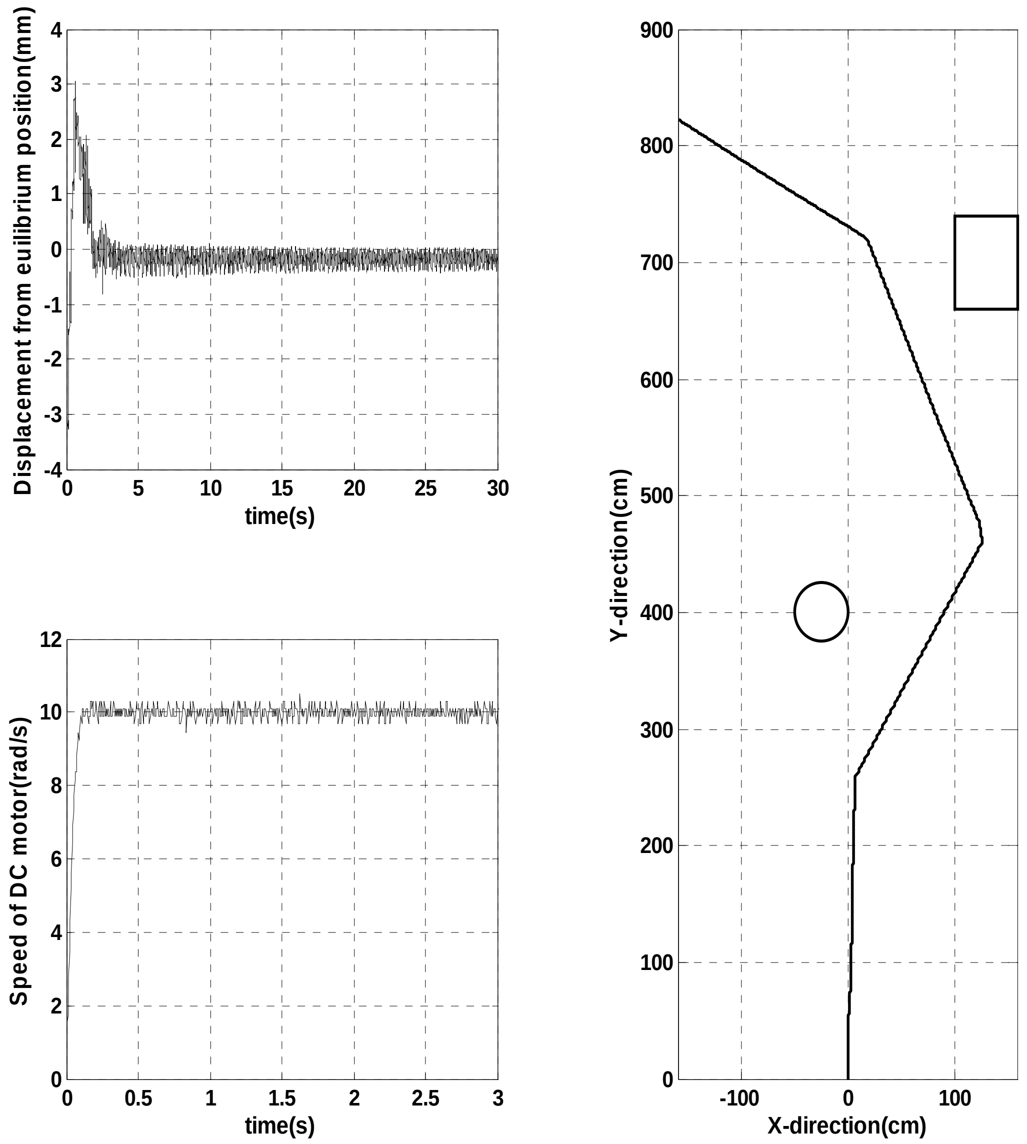


Fig. 5-10. Performance of suggested combination for Case 1

Case 2 also shows the stable system. Fig. 5-11 shows the performances of Clients 1, 2, and 3 as the stable combination.

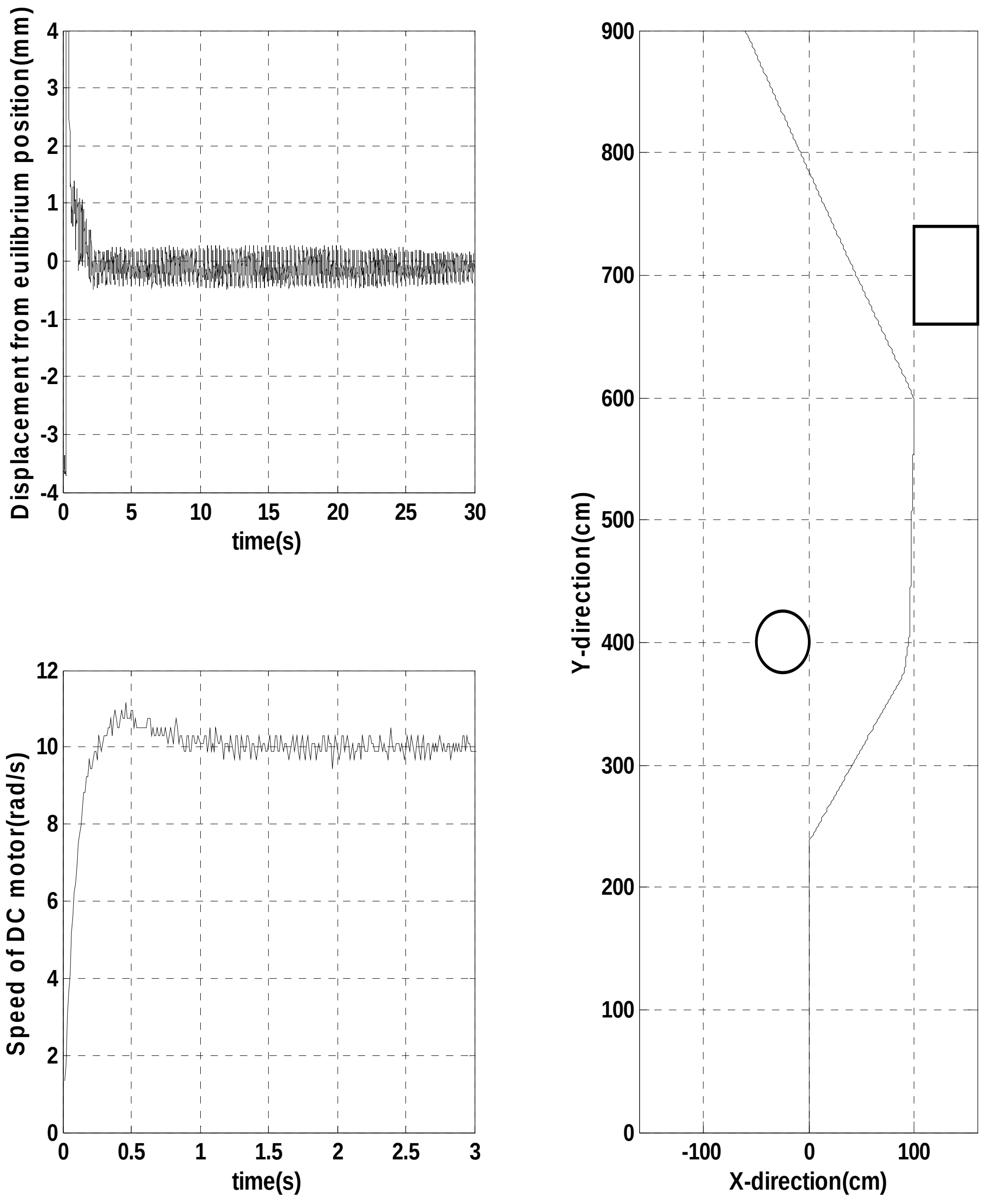


Fig. 5-11. Performance of suggested combination for Case 2

Since the total bandwidth utilization of this case is 99.9%, the stability of Client 1 is affected in Case 3 as shown in Fig. 5-12. Although Client 2 has a overshoot in its transient response, it is regarded as the stable system. Client 3 also arrived at the destination without collision.

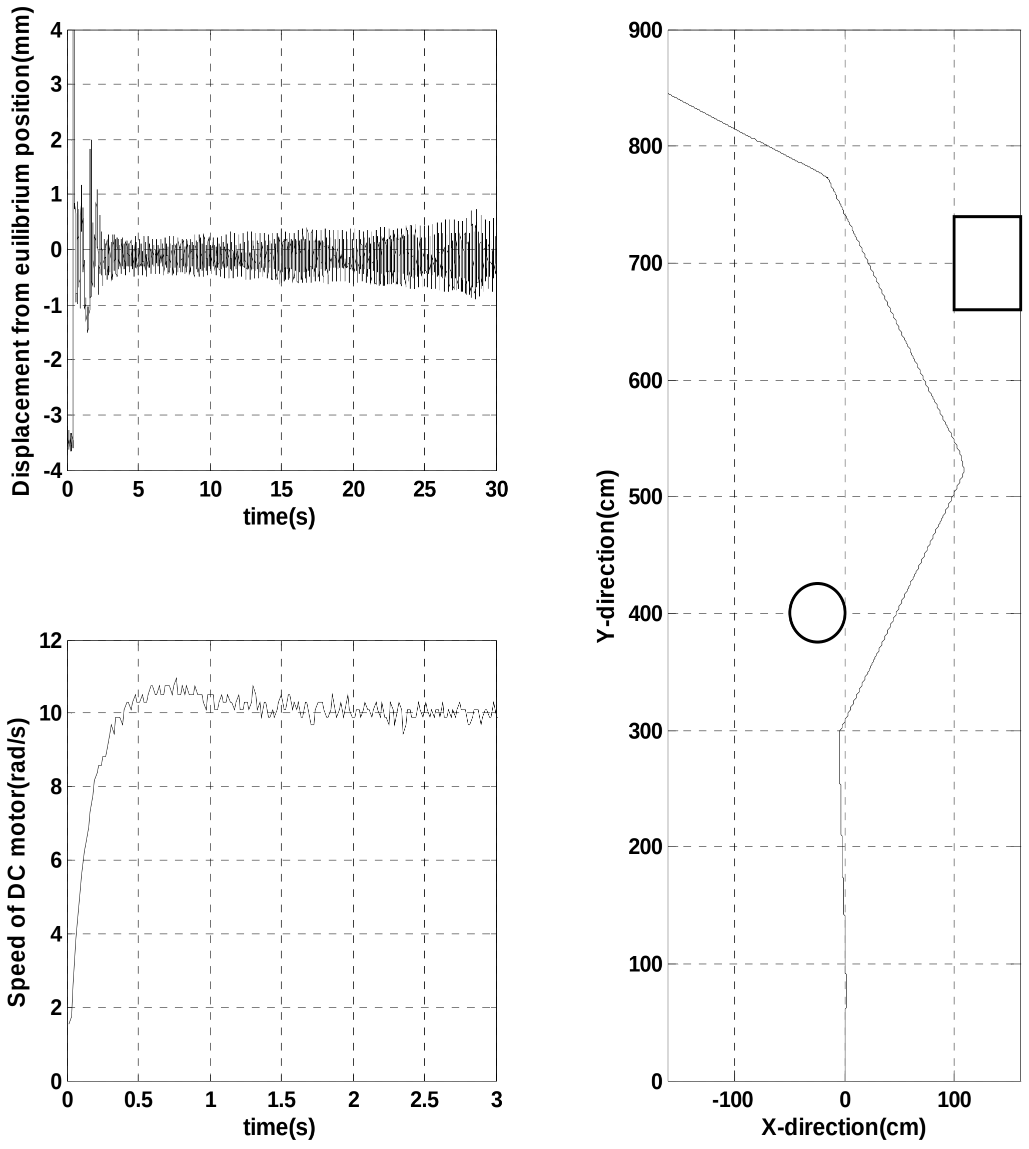


Fig. 5-12. Performance of suggested combination for Case 3

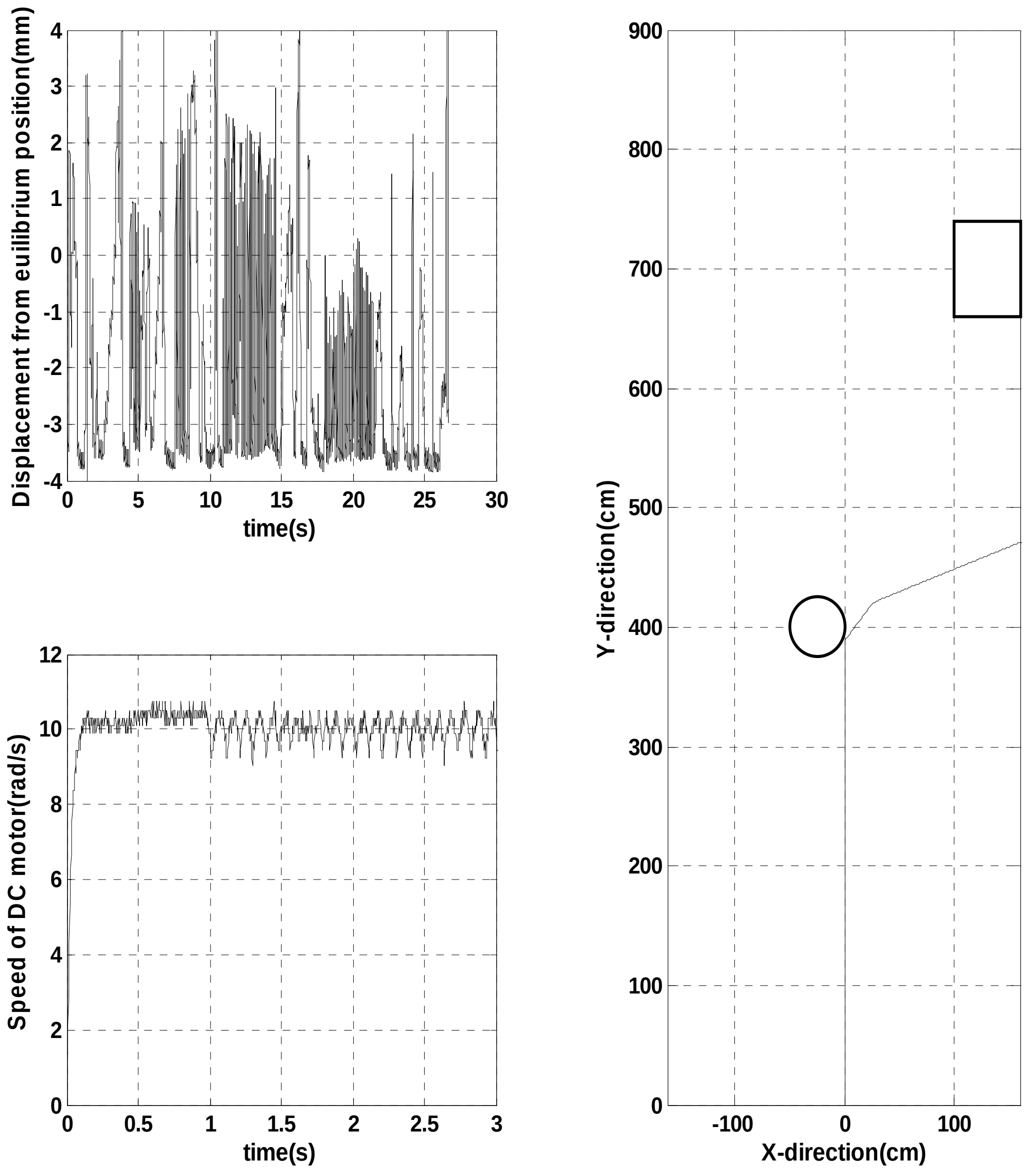


Fig. 5-13. Performance of suggested combination for Case 4

In Case 4, the total bandwidth utilization is 136.1%. The performance of Case 4 is shown in Fig. 5-13. Client 1 did not maintain its stability and the ball fell down from its equilibrium position. The performance of Client 2 is observed as the stable system. However, it is found that the output fluctuation is increased after 1 second. Client 3 did not arrive at the destination. The wheelchair robot hit the obstacle and stopped in front of a wall. Cases 1, 2, and 3 are experimented as the stable system whereas Case 4 is found as unstable. For the system stability, the conditions of combination for the sampling period should be considered.

CHAPTER VI

CONCLUSIONS AND SUGGESTED FUTURE WORK

6.1 Conclusions

The objective of this research was to demonstrate experimentally the feasibility of a real-time networked closed-loop control system with multiple clients. A steel-ball magnetic-levitation system, a DC motor speed-control system, and an autonomous wheelchair robot referred to as Clients 1, 2, and 3, respectively were used as NCS test beds to validate the proposed strategy. The multiple-client NCS was demonstrated successfully in this research.

For real-time operation, computing environments for the clients were discussed. Clients 1 and 2 need fast response from the server for system stability. Linux with RTAI and Comedi was used as the OS for them. Client 3's software was written in Microsoft Visual Basic 6.0 on Windows XP, modifying the previous code written by Hsieh [22]. Due to its low bandwidth requirement, this operating environment was good enough for Client 3. In order to convert the data type between Client 1 and Linux server, Gateway is developed. Using Samba, Client 3 is connected to Gateway on the same LAN with a fast data-transmission speed. The round-trip time between Client 3 and Gateway is just 11.2 ms with Samba whereas 30.8 ms is taken without using Samba.

UDP was used as the communication network protocol in this research due to its better real-time performance instead of TCP. Although TCP is a reliable protocol, it has large overheads for various services and may waste bandwidth and time. While UDP has no guarantee for transmitting data, it has smaller overheads and less time delay than TCP. A prediction algorithm [17] was needed to compensate for any delayed or lost data packet. For the server to identify the clients, the sensor data packets include a client identification number. After calculating the control data, the server sends back the data to the identified client.

The feedback control loop is limited by the bandwidth of the communication network. Therefore the system stability is affected by the sampling period of the system. The reduction of the sampling period improves the control loop's performance. However, a shorter sampling period requires more network bandwidth to transmit more sensor data or control data, which increases the network traffic load.

The transmission time for each connection is measured by the PING test, and the processing time for each client was measured by the `timeGetTime()` function for the Window-based OS and the `rt_get_cpu_time_ns()` function for Linux. With the transmission time and the processing time for each client, the operation time for each control loop was calculated. Using the operation time for each control loop, the sampling periods for the system stable were determined.

The sampling period of Client 1 was set to be 3 ms, and Client 1 used 43.5% of the total bandwidth. The bandwidth utilization of Client 2 with guaranteed stability was found to be from 9.1% to 45.3%. Thus, the range of the bandwidth utilization of Client 3 was between 11.8% and 46.8%. As long as the bandwidth utilizations of all clients were within these ranges, the multiple-client NCS could maintain its stability. In order to verify these conditions, three successful operation cases were suggested and tested.

6.2 Suggested Future Works

In the current system, Gateway converts and transfers the control data and the sensor data between Linux server and Client 3 because of the difference of their socket structures. For a sensor, it is waste of time and resource with an intermediary PC. It may also introduce time delay and data loss between Linux server and Client 3. With Client 3’s programs written in C on Linux OS, the communication efficiency could be enhanced.

Adding more servers to calculate the control data collaboratively could be a solution to bandwidth limitation. When one of the servers is busy with calculating the control data for a client, other servers could communicate with other clients. It could be seen as increased bandwidth for system performance.

In this research, the combinations of the sampling period for each client system are suggested statically. However, this static method may not be efficient when network conditions are changed. For this reason, a dynamic bandwidth-allocation method according to the control performance of each client could be developed.

REFERENCES

[1] W. Zhang, M. S. Branicky, and S. Phillips, “Stability of networked control systems,” *IEEE Control Systems Magazine*, vol. 21, no. 1, pp. 84–99, February 2001.

[2] T. B. Sheridan, “Teleoperation, telerobotics and telepresence: A progress report,” *Control Engineering Practice*, vol. 3, no. 2, pp.205–214, February 1995.

[3] G. Hirzinger, B. Brunner, J. Dietrich, and J. Heindl, “ROTEX – The First Remotely Controlled Robot in Space,” in *Proc. of the IEEE International Conference on Robotics and Automation*, vol. 3, pp.2604–2611, May 1994.

[4] Q. Lin and C. Kuo, “Virtual tele-operation of underwater robots,” in *Proc. of the 1997 IEEE International Conference on Robotics and Automation*, pp.1022–1027, April 1997.

[5] M. Evans, “Brokk - the world’s leading supplier of remote controlled demolition machines,” January 2004. [Online]. Available: http://www.brokk.com, Accessed on February 11, 2009.

[6] M. B. Cohn, M. Lam, and R. S. Fearing, “Tactile feedback for teleoperation,” in *Proc. SPIE*, vol. 1833, PP.240–254, January 2004.

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

[8] X. Wang, M. Moallem, and R. V. Patel, “An Internet-based distributed multiple-telerobot system,” *IEEE Transations on Systems, Man, and Cyvernetics - Part A : Systems and Humans*, vol. 33, no. 5, pp. 627–633, September 2003.

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

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

[11] A. Puri, “*Theory of Hybrid Systems and Discrete Systems*,” Ph.D. Dissertation, Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, CA, 1995.

[12] C. E. Garcia, R. Carelli, J. F. Postigo, and C. Soria, “Supervisory control of a telerobotic system: A hybrid control approach,” *Control Engineering Practice*, vol. 11, no. 7, pp. 805–817, July 2003.

[13] W. Wanga, H.-X. Lib, and J. Zhang,” A hybrid approach for supervisory control of furnace temperature,” *Control Engineering Practice*, vol. 11, no. 11, pp. 1325–1334, November 2003.

[14] K. Ji, W.-J. Kim, and A. Srivastava, “Internet-based real-time control architectures with time-delay/packet-loss compensation,” *Asian Journal of Control*, vol. 9, no. 1, pp.47–51, March 2007.

[15] J. Eker and A. Cervin, "Distributed wireless control using Bluetooth," in *Proc. of IFAC Conference on New Technologies for Computer Control,* Hong Kong, P. R. China, pp. 699–705 November 2001.

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

[17] A. Ambike, *Close-Loop Real-Time Control on Distributed Networks*, M.S. Thesis, Texas A&M University, College Station, TX, August 2004.

[18] A. Cervin and J. Eker, "Feedback scheduling of control tasks," in *Proc.of the 39th IEEE Conference on Decision and Control*. Sydney, Australia, pp. 4871–4876, 2000.

[19] H. S. Park, Y. H. Kim, D. -S Kim, and W. H. Kwon, "A Scheduling Method for Network-Based Control Systems," *IEEE Transactions on Control System Technology*, vol. 10, no. 3, pp. 318–330, May 2002.

[20] K. Ji and W. -J. Kim, "Optimal bandwidth allocation and QoS-adaptive control co-design for networked control systems," *International Journal of Control, Automation, and Systems*, vol. 6, no. 4, pp. 596–606, August 2008

[21] 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.

[22] P. Hsieh, *Autonomous Robotic Wheelchair with Collision-Avoidance Navigation,* M.S. Thesis, Texas A&M University, College Station, TX, August 2008.

[23] A. Srivastava, *Distributed Real-Time Control via the Internet*, M.S. Thesis, Texas A&M University, College Station, TX, May 2003.

[24] G. F. Franklin, J. D. Powell, and M. L. Workman, *Digital Control of Dynamic Systems*, 4th Ed., Prentice Hall, December 2002.

[25] Maxon Precision Motors, "*A-max 26 series*," maxon DC motor datasheet, July 2004.

[26] *DC Motor Closed-Loop Speed Control*, MEEN 667 Laboratory Manual, Department of Mechanical Engineering, Texas A&M University, TX, 2008.

[27] *Superlogics PCMDIO Users Manual*, January 2002. [Online] Available : http://www.SuperLogics.com. Accessed on October 20, 2008.

[28] *DIAPM RTAI - real-time application*, January 1999. [Online]. Available: http://www.rtai.org. Accessed on October 10, 2008.

[29] *Linux control and measurement device interface*, August 1999. [Online]. Available: http://www.comedi.org. Accessed on October 13, 2008.

[30] *Superlogics PCMDRIVE® Data Acquisition Software User's Manual*, January 2002. [Online]. Available: http://www.SuperLogics.com. Accessed on October 20, 2008.

[31] *Samba - Opening Windows to a Wider World*, May 1992. [Online]. Available: http://www.samba.org. Accessed on December 10, 2008.

[32] 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.

[33] K. Ji, *Real-Time Control Over Networks*, Ph.D. Dissertation, Department of Mechanical Engineering, Texas A&M University, College Station, TX, May 2006.

[34] *TAMUlink – WPA for Windows XP*, September 2008. [Online]. Available: https://hdc.tamu.edu/reference/documentation/?section_id=729. Accessed on November 10, 2008.

[35] A. Rogers, *Precision Mechatronics Lab Robot Development*, M.S. Thesis, Texas A&M University, College Station, TX, December 2007.

APPENDIX

GATEWAY.C

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

```
#include <inttypes.h>

#include "defines.h"

#define KEEP_STATIC_INLINE

//#include <rtai_lxrt_user.h>

#include <rtai_lxrt.h>


RTIME time_stamp;

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

double y_0, y_1, y_2, y_3, y_4, y_5, y_6;

double y_7=3;

double u_1, u_2;


//rtai declarations

unsigned long mtsk_name;

RT_TASK *mtsk;

struct sched_param mysched;

RTIME current_time_stamp;

void terminate_normally(int signo)

{

     fflush(stdin);

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

     {

          printf("Terminating the program normally\n");
          //make the process soft real time process

          rt_make_soft_real_time();

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

          rt_task_yield();

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

          stop_rt_timer();

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

          rt_task_delete(mtsk);

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

     exit(0);
}

main(int argc, char *argv[])
{

   int sockid, addrlen;

   int sd, clilen;

   struct sockaddr_in gate_addr, server_addr;

   struct sockaddr_in my_addr, client_addr;

   int nw, nr;

   int cw, cr;

   int cnt=0;

   int send_buffer_size, recv_buffer_size;

   unsigned short server_port = 0;

   unsigned short second_port = 0;
   struct send_data *send_buffer = NULL;

   struct recv_data *recv_buffer = NULL;

   double recv_msg[9] =
{0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0}; //client to
gateway

   char fwd[10] = "front";

   char back[10] = "back";

   char stop[10] = "stop";

   char left[10] = "left";

   char right[10] = "right";

   char *server_ip= "165.91.95.40";

   FILE *fp = NULL;

   fp = fopen("result.txt","w");


   if (fp==NULL)

   {

     printf("could not open file\n");

     exit(0);

   }

     RTIME start_time = 0;

     RTIME end_time = 0;

     RTIME actual_period = 0;

     RTIME difference = 0;
     struct sigaction sa;
     sa.sa_handler = terminate_normally;
     sa.sa_flags = 0;
     sigemptyset(&sa.sa_mask);
     if(sigaction(SIGINT, &sa, NULL))
     {
          perror("sigaction");
     }
     if(sigaction(SIGTERM, &sa, NULL))
     {
          perror("sigaction");
     }
   fprintf(stderr, "creating socket\n");
   if ( (sockid = socket(AF_INET, SOCK_DGRAM, 0)) < 0)
   {
     perror("1socket() failed ");
     fprintf(stderr, "%s: 1socket error: %d\n", argv[0],
errno);
     exit(2);
   }
   if ( (sd = socket(AF_INET, SOCK_DGRAM, 0)) < 0)
   {
     perror("2socket() failed ");
     fprintf(stderr, "%s: 2socket error: %d\n", argv[0],
errno);
     exit(2);
   }
   fprintf(stderr, "binding sockets\n");
   server_port = 4444;
   second_port = 3333;
   addrlen = sizeof(server_addr);
   clilen = sizeof(my_addr);
   memset((void *) &server_addr, (char) 0, addrlen);
   server_addr.sin_family = AF_INET;
   server_addr.sin_addr.s_addr = inet_addr(server_ip);
   server_addr.sin_port = htons(server_port);
   memset((void *) &my_addr, (char) 0, clilen);
   my_addr.sin_family = AF_INET;
   my_addr.sin_addr.s_addr = htonl(INADDR_ANY);
   my_addr.sin_port = htons(second_port);
   if ( (bind(sd, (struct sockaddr *) &my_addr,
      sizeof(my_addr)) < 0) )
   {
     perror("2bind() failed ");
     fprintf(stderr, "bind() errno = %d\n", errno);
     exit(4);
   }
   recv_buffer_size = sizeof(struct recv_data);
   if(( recv_buffer = (struct recv_data *)calloc(1,
sizeof(struct recv_data))) ==NULL)
   {
        fprintf(stderr, "cannot allocate memory for
buffer!\n");
     exit(4);
   }
   send_buffer_size = sizeof(struct send_data);
   if(( send_buffer = (struct send_data *)calloc(1,
sizeof(struct send_data))) ==NULL)
   {
        fprintf(stderr, "cannot allocate memory for
buffer!\n");
     exit(4);
   }

   fprintf(stderr, "%s: starting blocking message read\n",
argv[0]);
     mysched.sched_priority = 99;
     if( sched_setscheduler( 0, SCHED_FIFO, &mysched ) == -
1 ) {
     puts(" ERROR IN SETTING THE SCHEDULER UP");
     perror( "errno" );
     exit( 0 );
     }

     mlockall(MCL_CURRENT | MCL_FUTURE);
     mtsk_name = nam2num("MTSK");
     if (!(mtsk = rt_task_init(mtsk_name, 0, 0, 0))) {
          printf("CANNOT INIT MASTER TASK\n");
          exit(1);
     }
     start_time = rt_get_cpu_time_ns();
     printf("main: start_time = %lld\n", start_time);
     printf("MASTER TASK STARTS THE ONESHOT TIMER\n");
     //rt_set_oneshot_mode();
     actual_period = start_rt_timer(nano2count(25000));
     printf("actual_period = %lld\n", actual_period);
     printf("MASTER TASK MAKES ITSELF PERIODIC \n");
     rt_task_make_periodic(mtsk, rt_get_time()+
nano2count(3000000),_ nano2count(3000000));
   while( 1 )
   {
     cr = recvfrom(sd, recv_msg, 10, 0, (struct sockaddr *)
&client_addr, &clilen);
     if( cr <= -1 )
     {
          fprintf(stderr, "2recvfrom() errno = %d\n",
errno);
          exit(10);
     }
     start_time = rt_get_cpu_time_ns();
     y_0 = recv_msg[0];
     y_1 =  y_2 = y_3 = y_4 = y_5 = y_6 = 0;
     y_7 = 3;
     u_1 = u_2 = 0;
     send_buffer->y_0 = y_0;
     send_buffer->y_1 = y_1;
     send_buffer->y_2 = y_2;
     send_buffer->y_3 = y_3;
     send_buffer->y_4 = y_4;
     send_buffer->y_5 = y_5;
     send_buffer->y_6 = y_6;
     send_buffer->y_7 = y_7;
     send_buffer->u_1 = u_1;
     send_buffer->u_2 = u_2;
     send_buffer->time_stamp = current_time_stamp;
     nw=sendto(sockid, (const void *)send_buffer,
send_buffer_size, 0,(struct sockaddr *) &server_addr,
addrlen);
     if( nw <= -1 )
    {
       perror("1sendto failed ");
       fprintf(stderr, "sendto() errno = %d \n", errno);
       exit(12);
     }
     nr = recvfrom(sockid, (void *)recv_buffer,
recv_buffer_size, 0, (struct sockaddr *) &server_addr,
&addrlen);
     if( nr <= -1 )
     {
          fprintf(stderr, "1recvfrom() errno = %d\n",
errno);
          exit(10);
     }
     u0 = recv_buffer->u0;
     u1e = recv_buffer->u1e;
     u2e = recv_buffer->u2e;
     u3e = recv_buffer->u3e;
     u4e = recv_buffer->u4e;
     u5e = recv_buffer->u5e;
     u6e = recv_buffer->u6e;
     u7e = recv_buffer->u7e;
     u8e = recv_buffer->u8e;
     if(u0 == 10.0){
          cw = sendto(sd, fwd, 5, 0, (struct sockaddr *)
&client_addr, clilen);
     }
     else if(u0 == 7.0){
          cw = sendto(sd, right, 5, 0, (struct sockaddr *)
&client_addr, clilen);
     }
     else if(u0 == 2.0){
          cw = sendto(sd, left, 5, 0, (struct sockaddr *)
&client_addr, clilen);
     }
     else if(u0 == 0.0){
          cw = sendto(sd, back, 5, 0, (struct sockaddr *)
&client_addr, clilen);
     }
     else if(u0 == 5.0){
          cw = sendto(sd, stop, 5, 0, (struct sockaddr *)
&client_addr, clilen);
     }
     end_time = rt_get_cpu_time_ns();
     send_buffer->time_stamp = recv_buffer->time_stamp;
     printf("end_time - start_time = %lld\n", (end_time -
start_time));
     cnt = cnt +1;
   } //end while
fclose(fp);
     //make the process soft real time process
     //rt_make_soft_real_time();
     printf("MASTER TASK YIELDS ITSELF\n");
     rt_task_yield();
     printf("MASTER TASK STOPS THE PERIODIC TIMER\n");
     stop_rt_timer();
     printf("MASTER TASK DELETES ITSELF\n");
     rt_task_delete(mtsk);
     close(sockid);
     close(sd);
     free(send_buffer);
     free(recv_buffer);
     free(recv_msg);
```

```
    free(recv_msg);
}
```

VITA

The author, Minhyung Lee received his Bachelor of Engineering degree in weapons engineering from Korea Military Academy, Seoul, South Korea in March 2004. Since Fall 2007 he has been enrolled in the Master of Science degree program in the Mechanical Engineering Department, Texas A&M University, College Station.

Mr. Minhyung Lee may be reached at Department of Mechanical Engineering, 3123 TAMU, College Station, TX 77843-3123. His email is c14565@gmail.com.