ABSTRACT

TIPSUWAN, YODYIUM. Gain Scheduling for Networked Control System. (Under the direction of Dr. Mo-Yuen Chow.)

Performances of closed-loop control systems operated over a data network are typically degraded by network-induced delays. Furthermore, the closed-loop control systems can become unstable. The purpose of this research has been to develop a control methodology to handle network-induced delay effects using optimal gain scheduling on existing controllers. The proposed gain scheduling technique adapts controller gains externally by modifying a controller output to enable the controller for uses over a data network. Since existing controllers can still be utilized, the proposed methodology can reduce control system reinstallation and replacement costs. First, the effectiveness of the proposed gain scheduling technique on networked DC motor speed control using a PI (Proportional-Integral) controller is investigated. Also, the concept of network traffic condition measurement to select optimal controller gains is presented. Then, a middleware framework to measure network traffic conditions on an IP network based on delays and delay variations and to modify controller gains is described. Suggestion of using neural network in the gain scheduling scheme is also given. Finally, the gain scheduling technique with the middleware framework is then extended to mobile robot path-tracking control.

GAIN SCHEDULING FOR NETWORKED CONTROL SYSTEM

by

Yodyium Tipsuwan

A dissertation submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Department of Electrical and Computer Engineering

Raleigh

2003

APPROVED BY:

Dr. Douglas S. Reeves

Dr. Griff L. Bilbro

Dr. Hamid Krim

Dr. Mo-Yuen Chow
Chair of Advisory Committee

To my parents for their endless love, support, and encouragement

BIOGRAPHY

Yodyium Tipsuwan was born in Chiangmai, Thailand. He received the B.Eng. degree (with honors) in computer engineering in 1996 from Kasetsart University, Bangkok, Thailand, and the M.S. degree in electrical engineering in 1999 from North Carolina State University, Raleigh.

After his graduation in the B.Eng., he joined the Department of Computer Engineering, Kasetsart University as an instructor to mainly teach digital system designs and microprocessor-based control systems in 1996, and has been awarded the Royal Thai Scholarship to continue his study in M.S. and Ph.D. degrees since 1998. During his M.S. study, he focused on intelligent control and its implementation. He is currently a Ph.D. candidate at North Carolina State University. His main interests are distributed networked control systems, mobile robotics, and computational intelligence. His current research topic is networked control over IP networks.

Mr. Yodyium will join the Department of Computer Engineering, Faculty of Engineering, Kasetsart University, in August 2003 to teach students and conduct research there.

ACKNOWLEDGMENTS

This dissertation could not be completed without many helpful suggestions and motivation from several faculties, friends, and my parents. First, I would like to express my utmost gratitude to my advisor Dr. Mo-Yuen Chow. Dr. Chow has spent his tireless effort and precious time to supervise, encourage, and inspire me on my research since I have been arrived at North Carolina State University. He has continuously educated and guided me from a person who was new in the research area to a person who knows a little more with inspiration to continue challenging research works. I would also thank to Dr. Douglas S. Reeves for his discussion and suggestion on computer network issues, Dr. James J. Brickley for his substitution in my preliminary examination, Dr. Griff L. Bilbro and Dr. Hamid Krim for the valuable comments on this dissertation at the final stages of my study.

In addition, I would like to express my deepest appreciation to my parents, Mr. Prasit and Mrs. Chamaiporn Tipsuwan, for their endless love, support, understanding, and encouragement, and my closest friends, Ms. Saowanee Lertworasirikul, Mr. Somsak Pakpinyo, and Mr. Rangsarit Vanijjirattikhan, for their encouragement and support on various things during my study, and many other friends that I could not mention them all here.

Finally, I thank the Royal Thai Government for the financial support of my study.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

CHAPTER II

CHAPTER III

CHAPTER IV

CHAPTER V

CHAPTER I

INTRODUCTION

This chapter presents an overview of this dissertation including a brief introduction to networked control systems and applications, gain scheduling, neural networks, middleware, and gain scheduler middleware. A survey of the dissertation is given at the end of this chapter.

I. Networked Control Systems

A *networked control system* (*NCS*) [1] or a *networked-based control system* [2] is a closed-loop control system operated over a data communication network. To perform closed-loop networked control operations, a controller needs to send control data to a system plant, while also receiving feedback data from the plant through a communication network to update control data as shown in Fig. 1.

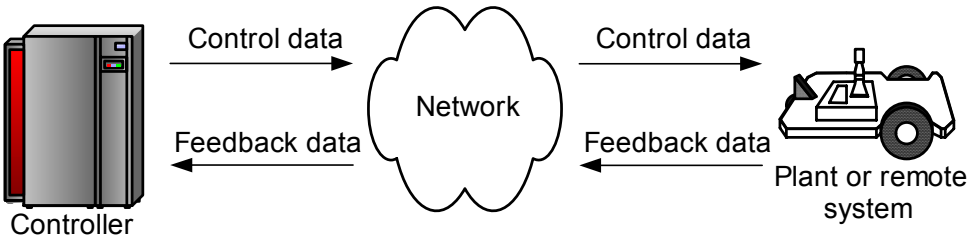


Fig. 1. Networked control system data transfer.

There are several control applications usually configured as networked control systems. In general, these applications can be categorized in two major groups as:

A. Complex control systems

A complex control system is a large-scale system containing several subsystems [3], which perform and collaborate together to achieve an overall system task. An example of a complex

control system is an altitude control system in an airplane [4] as shown in Fig. 2. In order to maintain the altitude during an autopilot mode, three subsystems, which are the flaps, the elevator, and the engine, must cooperate synchronously.

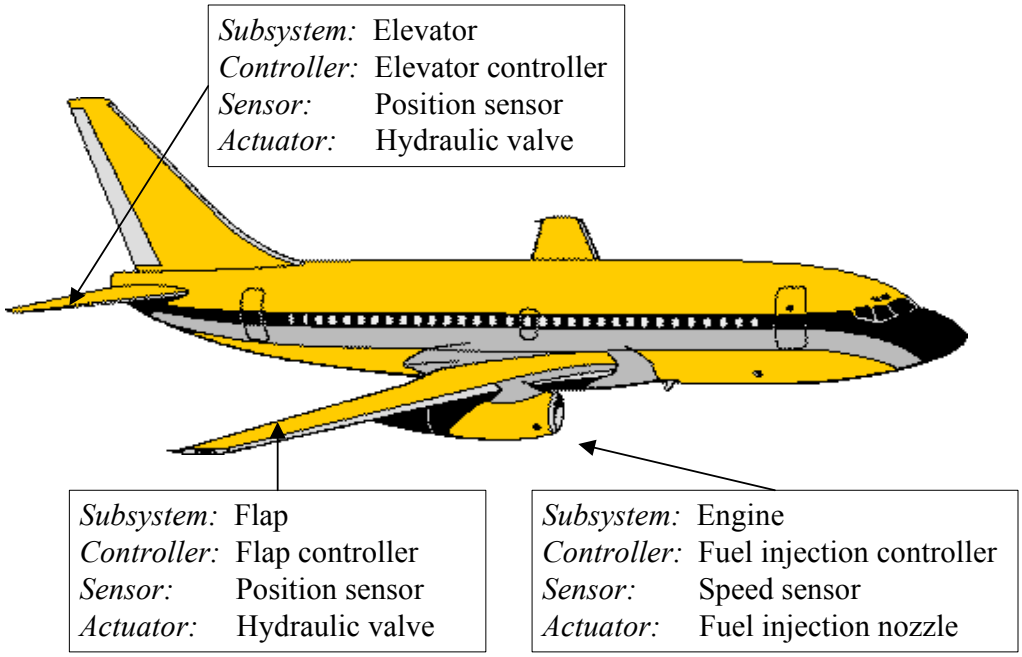


Fig. 2. Altitude control system of airplane.

Because an actual complex control system is typically large and sophisticated, connecting system components like sensors, actuators, and controllers together by direct electrical wiring usually results in complicated circuits. The system as a whole could be difficult to install and maintain. System connections can be even more cumbersome if system components are not closely located physically. Networked control system configuration can be applied on a complex control system to solve this problem by systematically organizing wiring connections into a shared data network. Several practical complex control applications have successfully utilized the networked control system concept for this purpose. These applications include automobiles [5, 6], chemical process [7], and manufacturing plants [8].

B. Remote control systems

Remote control systems have been used for two general reasons: convenience and safety. A remote control system saves human operator place-to-place traveling time and protects the operator from dangers in hazardous environments such as in the space and in a war zone. A widely known example is the Mars rover as shown in Fig. 3.

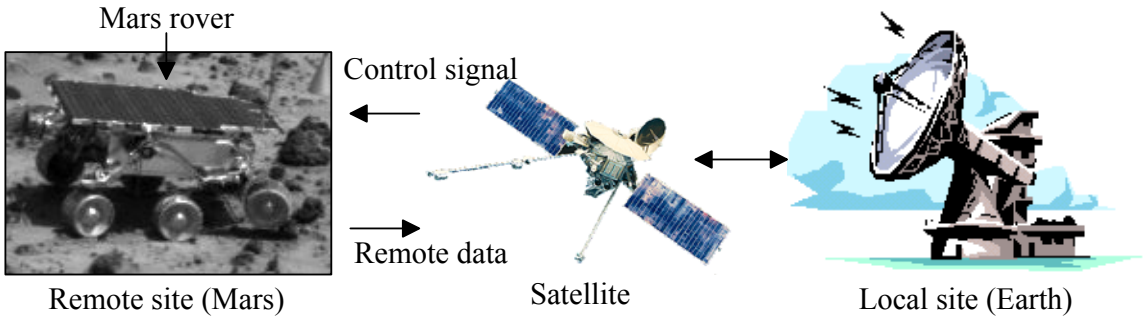


Fig. 3. Remote control system of Mars rover.

In the past, a remote control system usually requires a specific connection link or media, which may be limited to a point-to-point connection and have an expensive set-up cost. With the evolving of communication technologies, an emerging alternative to expand remote systems for more connections is to utilize wired or wireless data network resources by configuring the system as networked control systems. Thus, the network media can be shared among several remote control systems for expansions. In addition, the prices of network devices for networks such as Ethernet have become affordable, while the performances are rapidly improving. Therefore, the set-up cost for a remote control system can be much reduced than before. Furthermore, with the widely-used Internet, the remote system could even be controlled across continents without too much extra cost. Practical examples of networked remote control systems are robot teleoperation [9] and distant learning laboratory [10, 11].

Because a networked control system transfers control and feedback data through a network, the network can induce delays on the transfer signals and affect the overall feedback control

system performance. The performance could be degraded and the system can even become unstable. The network delay effect and impacts on system performance degradation and instability are based on the characteristics of the networked control system. For example, for a remote DC motor speed control, an oscillatory behavior may occur at the speed output if there are network delays in the control loop. Likewise, on a remote networked mobile robot path-tracking application, the robot may deviate from a desire position when it does not receive timely control data from a controller across a network. Fig. 4 shows a typical comparison of robot trajectories when the robot is controlled to track a path with and without network delays.

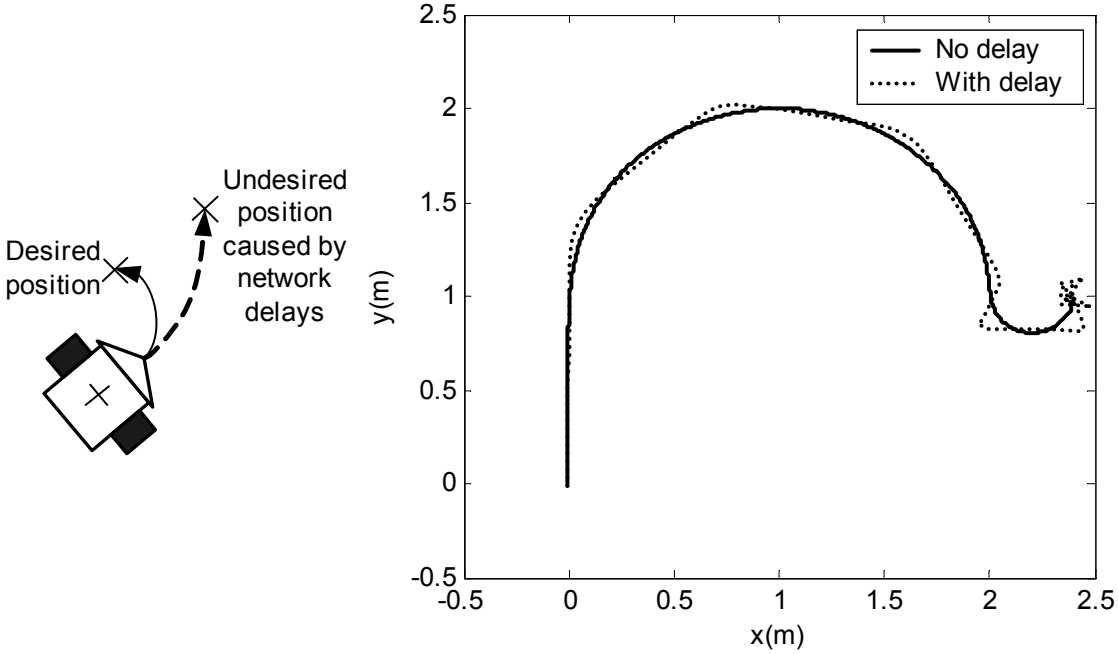


Fig. 4. Comparison of robot tracks with and without network delay in the feedback control loop.

Apparently, the robot with network delays cannot track the desired path closely. To maintain the networked control system performance as much as possible with the existence of network delays, a networked control methodology is required to compensate the network delay effect on the overall networked feedback control system. A survey on recent networked control

techniques and a novel networked control methodology using middleware gain adaptation will be presented in later chapters of this dissertation.

II. Gain Scheduling

Gain scheduling is a special kind of nonlinear feedback control techniques. The concept of gain scheduling originated from flight control applications [12-13], in which operating conditions always change. In gain scheduling, controller parameters are functions of operating conditions so called scheduling functions, which can usually be represented in a lookup table. These operating conditions are usually parameterized into variables called auxiliary or scheduling variables [13], that correlate with changes in the dynamics of the system. Choosing appropriate auxiliary variables depends on the physics and characteristics of a system. In this dissertation, data network traffic conditions will be characterized into network variables and will be used as auxiliary variables. Fig. 5 shows the block diagram of a feedback control system with gain scheduling.

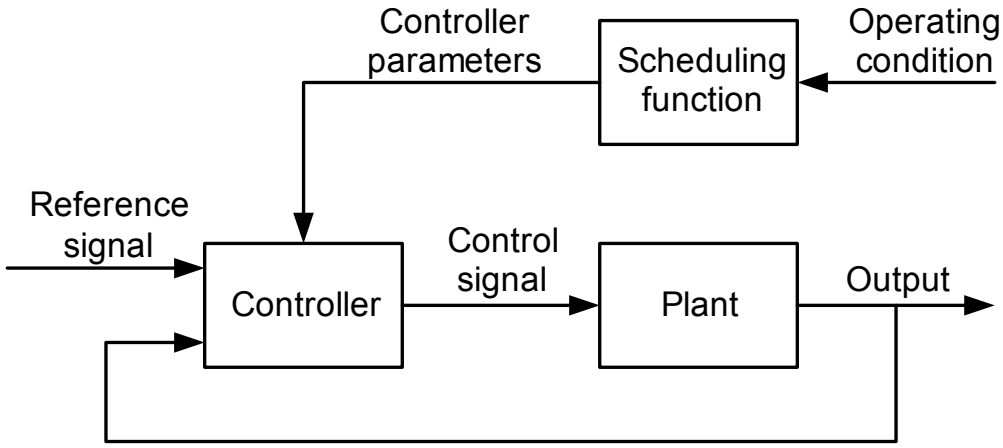


Fig. 5. Block diagram of feedback control system with gain scheduling.

When the operating condition of the control system changes, the controller parameters will be adjusted with respect to the measurements of auxiliary variables. Nevertheless, there is no feedback from the system output to the controller parameters. Gain scheduling, therefore, is

typically not considered as an adaptive control technique. Because the parameters are updated in an open-loop manner, parameter adaptation by gain scheduling could be viewed as a kind of feedforward compensation. The performance and stability of the system with gain scheduling are usually analyzed by extensive simulations. Analysis at the transition among operation conditions is normally required more attention. Until now, gain scheduling has been applied on various applications, and also in many cases along with other technologies such as robust control, optimal control, and fuzzy control. Example of these applications are flight control [14], arc welding [15], diesel engine [16], magnetic bearing [17], suspension system [18], and DC motor [19].

III. Neural Networks

An artificial neural network (ANN) is a mathematical structure designed to model a brain function of interests. In general, an ANN architecture is composed of processing elements called *neurons* or *nodes* [20]. Fig. 6 shows the fundamental structure of a neuron.

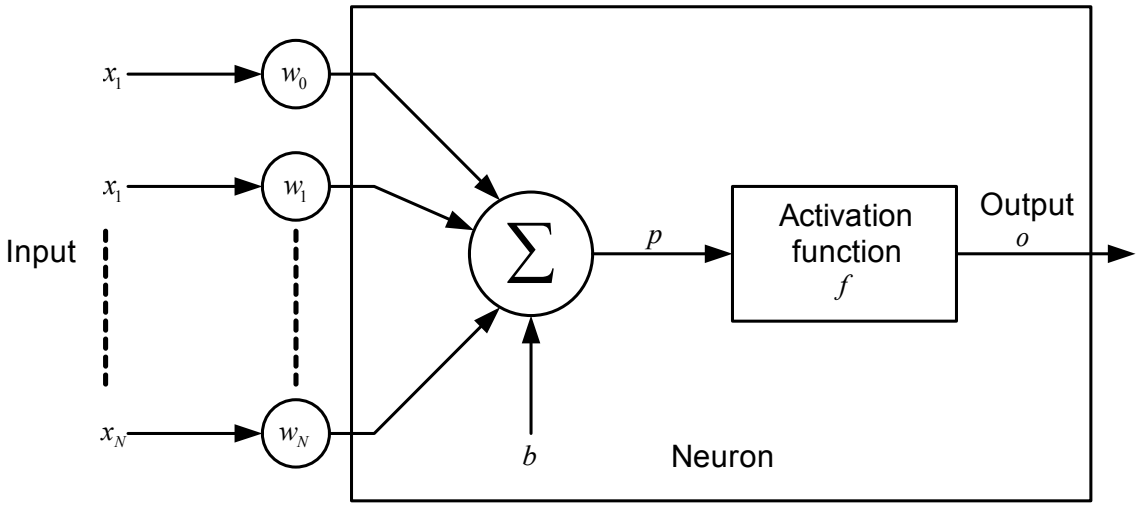


Fig. 6. Fundamental structure of a neuron.

The neuron receives the inputs $x_1, x_2, \ldots, x_N$, which are scaled by the weights $w_1, w_2, \ldots, w_N$, respectively. The summation of these inputs and the bias *b* is then applied as the input of an *activation function* or *a transfer function*. This operation of a neuron can be described by:

$$o = f(p), p = \sum_{i=1}^{n} (w_i x_i) + b \tag{1}$$

An activation function can be a linear or nonlinear function. However, popular choices of activation functions are usually nonlinear functions, which are monotonically non-decreasing and differentiable functions. Such activation functions include the sigmoid or logistic function, and the hyperbolic tangent function.

One of the most widely-used ANN structures is the multilayer feedforward network with one hidden layers illustrated in Fig. 7. Multilayer feedforward neural networks have been successfully applied to solve several nonlinear and complicated problems such as fault detection and diagnosis [21] and thermal modeling [22].

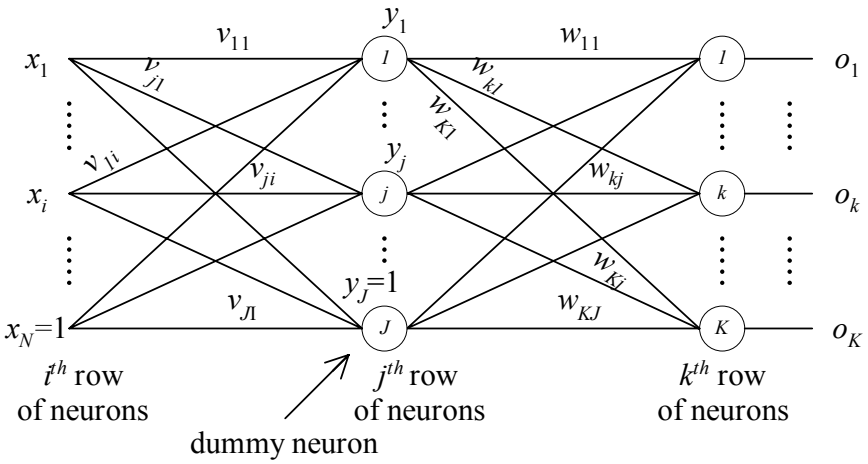


Fig. 7. Multilayer feedforward neural network.

A prominent feature of an ANN is its ability to approximate a highly nonlinear mapping by learning from a set of training data. This ability makes ANN very useful to approximate a scheduling function from a collection of simulation or experimental data. A training data set is composed of a set of input data and a set of the associated output data, which may be sampled from simulations or experiments. After being trained successfully by a training algorithm using

the training set, the ANN can represent the mapping between the inputs and outputs including the points that were not acquired in the training set. This learning process of the ANN is called supervised learning, which can be performed by a variety of training algorithms such as the highly popular error-backpropagation algorithm [23].

IV. Middleware

Middleware is an implementation to seamlessly manage connections among applications, hardware and software resources such as CPU processing time and data network resources. The interfaces of middleware to applications and resources are performed through abstract levels, while actual hardware and software implementations and operations are hidden inside the middleware. Applications developed from different platforms, languages, or procedures, can communicate together or access resources using the same abstraction. This framework provides modularity for application research and development, and ease to use and extend resources. The fundamental structure of middleware is illustrated in Fig. 8.

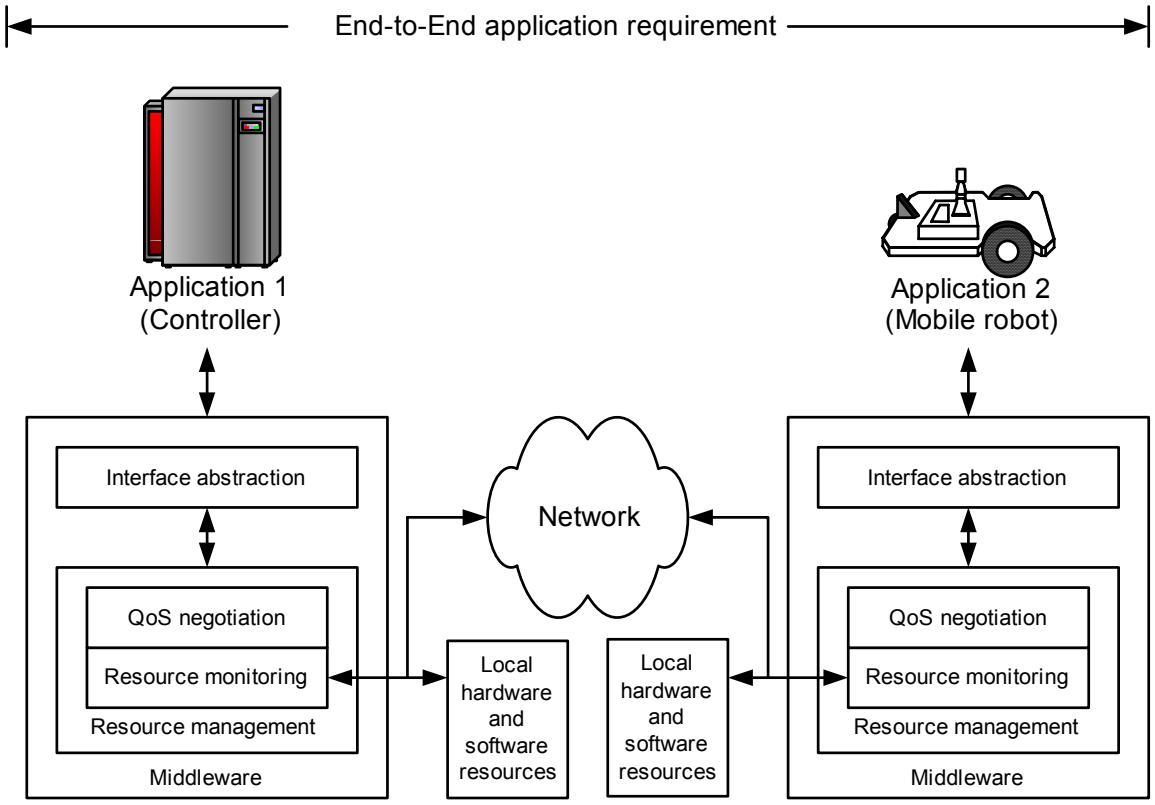


Fig. 8. A fundamental structure of middleware.

Middleware must have effective managing strategies for applications to access resources through abstraction. In general, an application can access resources by negotiating with middleware for its end-to-end application requirement. The requirement has to be mapped into specifications so called QoS (Quality-of-Service) specifications, which indicate the quantity of resource requirement such as required processing time, network bandwidth, delay bound, and jitter. For example, if we would like a robot to track a path over a data network to reach a destination in 15 s, the requirement can be mapped to the bandwidth and delay bound that the robot needs in order to complete the task. Middleware then check if there are adequate resources based on the resource monitoring measurements. If the requirement cannot be granted, the application may need to lower its performance and performs as best as possible. This action is called *graceful performance degradation* [24].

Middleware has been applied for control applications using various implementations such as RTPOOL [24] and CORBA (Common Object Request Broker Architecture) [25]. The main objective of middleware usages in these applications is to provide convenient interfaces and to guarantee QoS. Especially, middleware can offer appropriate network conditions, for example, bounds of delays and delay variations, for a networked control system. These applications include flight control [24], robot control [26-28], and factory automation [29, 30].

V. Gain Scheduler Middleware (GSM)

In this dissertation, a novel methodology to adapt controller gain parameters by a gain scheduling technique externally at the controller output through middleware is introduced. This methodology is developed based on the requirement to enable existing controllers for networked control purposes such that the controllers need not be redesigned or replaced by a complete new networked control system. Otherwise, the installations of new network control system are typically costly, inconvenient, and time consuming. The proposed methodology could benefit major industries in factory automation and industrial electronics because a major amount of existing controllers can be enabled or upgraded to be used over a data network. Therefore, these industries can save much investment and time for controller upgrades. The proposed methodology is defined as the gain scheduler middleware (GSM). The basic structure of a GSM is shown in Fig. 9.

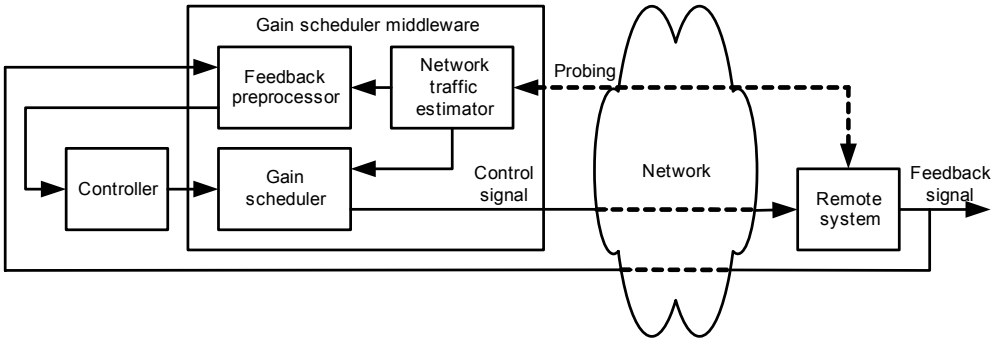


Fig. 9. Basic structure of gain scheduler middleware (GSM).

A GSM is composed of three main parts: a network traffic estimator, a feedback preprocessor, and a gain scheduler. The network estimator unit measures the current network traffic conditions by periodically sending probing packets to a remote system in order to characterize the network traffic conditions. The network traffic conditions and network variables can be thought of as operating conditions and auxiliary variables in gain scheduling. The feedback preprocessor unit preprocesses the feedback data such as filtering noise or prediction of remote system states, and forwards the preprocessed data to an existing controller. Note that the feedback preprocessor may not be necessary if the controller does not require preprocessing. The controller then computes the updated control data. The gain scheduler then captures the output of the controller, and modifies it by a gain scheduling algorithm with respect to the network variables in order to compensate network delay effects. Illustrations of GSM methodology applied on two applications: DC motor speed control system and mobile robot path tracking will be described in this dissertation.

VI. Overview of the Dissertation

This dissertation contains several manuscripts that have been submitted for publication to several journals. Chapter II of this dissertation is published in Control Engineering Practice, Special Issue on Control Methods for Telecommunication Networks [31]. Chapter III is published in IEEE Transactions on Industrial Electronics [32]. Chapter IV and V are the companion papers that have been submitted to an IEEE Transactions and are currently under review. Chapter VI is the paper to be submitted to another IEEE Transactions. There are also several other conference papers have been published along the research investigated in this dissertation [33-38].

Chapter II provides an introduction on networked control system technologies. General information about networked control systems including system configuration, data transfers, and network protocol characteristics for networked control systems are described. This chapter then provides a discussion on network delay effects on network control systems, and current

techniques to compensate network delays. Suggestion of using these techniques on different network protocol characteristics is also given.

Chapter III introduces the feasibility of using a gain scheduling technique to handle network delays based on network traffic conditions. The network considered in this chapter is assumed to be a QoS-enabled network, which can guarantee network traffic conditions for a networked control system. A PI (Proportional-Integral) DC motor speed control problem is used as an example to verify the feasibility and performance of the proposed gain scheduling technique. A method to adapt controller gain parameters by gain scheduling is described. Simulation and experimental results to show the effectiveness of the proposed approach are also provided.

Chapter IV and V are companion chapters. In these chapters, the concept of using gain scheduler middleware to adapt controller gains externally at the controller output with respect to the current IP network traffic conditions is introduced. Chapter IV provides the fundamental of network delay formulation in the frequency domain with the assumption that IP network delays are constant. Control system formulation for gain scheduling, sensitivity analysis, and optimal gain parameter finding are also described. Chapter V describes the generalization of the approach in chapter IV for actual IP network delays. The characteristics of actual IP network delays and characterization of IP network delays by a generalized exponential distribution are analyzed in this chapter. Optimal gain finding based on the generalized exponential distribution is then explained. The gain scheduler middleware concept for networked control is then described. Performance verification of the proposed approach on a PI DC motor speed control problem in simulations is also described in this chapter.

Chapter VI presents the use of gain scheduler middleware on a mobile robot application controlled with the existence of IP network delays. An example to illustrate the effectiveness of middleware in this chapter is a mobile robot path-tracking problem. Likewise, the gain scheduler middleware is used to adjust the output of the robot path-tracking controller with

respect to the current network traffic conditions. However, the structure of gain scheduler middleware is slightly different from chapter IV and V. The gain scheduler middleware in this chapter preprocesses the feedback data before forwarding preprocessed data to the robot path-tracking controller. Simulation and experimental results to verify the effectiveness of the gain scheduler middleware are described and presented at the end of this chapter.

References

[1] G. C. Walsh, H. Ye, and L. Bushnell, "Stability analysis of networked control systems," in American Control Conference, San Diego, CA, 1999, pp. 2876-2880.

[2] Y. H. Kim, H. S. Park, and W. H. Kwon, "Stability and a scheduling method for network-based control systems," in IEEE IECON 96, Taipei, Taiwan, 1996, pp. 934-939.

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

[4] B. Etkin and L. D. Reid, *Dynamics of Flight: Stability and Control*. New York: Wiley, 1996.

[5] Ü. Özgüner, H. Göktas, H. Chan, J. Winkelman, M. Liubakka, and R. Krotolica, "Automotive suspension control through a computer communication network," in IEEE Control Applications, Dayton, OH, 1992, pp. 895-900.

[6] N. Boustany, M. Folkerts, K. Rao, A. Ray, L. Troxel, and Z. Zhang, "A simulation based methodology for analyzing network-based intelligent vehicle control systems," in Intelligent Vehicles Symposium, 1992, pp. 138-143.

[7] B. P. Zeigler and J. Kim, "Extending the DEVS-Scheme knowledge-based simulation environment for real-time event-based control," *IEEE Transactions on Robotics and Automation*, vol. 9, no. 3, pp. 351-356, 1993.

[8] G. Kaplan, "Ethernet's winning ways," *IEEE Spectrum*, vol. 38, no. 1, pp. 113-115, 2001.

[9] K. Brady and T.-J. Tarn, "Internet-based teleoperation," in IEEE ICRA 2001, Seoul, South Korea, 2001, pp. 644-649.

[10] J. W. Overstreet and A. Tzes, "An Internet-based real-time control engineering laboratory," *IEEE Control Systems Magazine*, vol. 19, no. 5, pp. 19-34, 1999.

[11] G. V. Kondraske, R. A. Volz, D. H. Johnson, D. Tesar, J. C. Trinkle, and C. R. Price, "Network-based infrastructure for distributed remote operations and robotics research," *IEEE Transactions on Robotics and Automation*, vol. 9, no. 5, pp. 702-704, 1993.

[12] G. Stein, "Adaptive flight control-A pragmatic view," in *Applications of Adaptive Control*, K. S. Narendra and R. V. Monopoli, Eds. New York: Academic Press, 1980, pp. 291-312.

[13] K. J. Åström and B. Wittenmark, *Adaptive Control*. Reading, MA: Addison-Wesley, 1989.

[14] R. A. Nichols, R. T. Reichert, and W. J. Rugh, "Gain scheduling for H-infinity controllers: a flight control example," *IEEE Transactions on Control Systems Technology*, vol. 1, no. 2, pp. 69-79, 1993.

[15] J. B. Bjorgvinsson, G. E. Cook, and K. Andersen, "Microprocessor-based arc voltage control for gas tungsten arc welding using gain scheduling," *IEEE Transactions on Industry Applications*, vol. 29, no. 2, pp. 250-255, 1993.

[16] J. Jiang, "Optimal gain scheduling controller for a diesel engine," *IEEE Control Systems Magazine*, vol. 14, no. 4, pp. 42-48, 1994.

[17] F. Matsumura, T. Namerikawa, K. Hagiwara, and M. Fujita, "Application of gain scheduled H_∞ robust controllers to a magnetic bearing," *IEEE Transactions on Control Systems Technology*, vol. 4, no. 5, pp. 484-493, 1996.

[18] S.-H. Lee, S.-G. Kim, and J.-T. Lim, "Fuzzy-logic-based fast gain-scheduling control for nonlinear suspension systems," *IEEE Transactions on Industrial Electronics*, vol. 45, no. 6, pp. 953-955, 1998.

[19] M. R. Matausek, B. I. Jeftenic, D. M. Miljkovic, and M. Z. Bebic, "Gain scheduling control of DC motor drive with field weakening," *IEEE Transactions on Industrial Electronics*, vol. 43, no. 1, pp. 153-162, 1996.

[20] M.-Y. Chow and J. Teeter, *Methodologies of Using Artificial Neural Network and Fuzzy Logic Technologies for Motor Incipient Fault Detection*, 1 ed. Farrer Road, Singapore: World Scientific, 1997.

[21] B. Li, M.-Y. Chow, Y. Tipsuwan, and J. C. Hung, "Neural-network-based motor rolling bearing fault diagnosis," *IEEE Transactions on Industrial Electronics*, vol. 47, no. 5, pp. 1060-1069, 2000.

[22] M.-Y. Chow and Y. Tipsuwan, "Neural plug-in motor coil thermal modeling," in IEEE IECON 2000, Nagoya, Japan, 2000, pp. 1586-1591.

[23] D. E. Rumelhart and J. L. McClelland, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*. Cambridge, MA: MIT Press, 1986.

[24] T. F. Abdelzaher, E. M. Atkins, and K. G. Shin, "QoS negotiation in real-time systems and its application to automated flight control," *IEEE Transactions on Computers*, vol. 49, no. 11, pp. 1170-1183, 2000.

[25] D. G. Schmidt and F. Kuhns, "An overview of the Real-Time CORBA specification," *Computer*, vol. 33, no. 6, pp. 56-63, 2000.

[26] D. Brugali and M. E. Fayad, "Distributed computing in robotics and automation," *IEEE Transactions on Robotics and Automation*, vol. 18, no. 4, pp. 409-420, 2002.

[27] O. Kubitz, M. O. Berger, and R. Stenzel, "Client-server-based mobile robot control," *IEEE/ASME Transactions on Mechatronics*, vol. 3, no. 2, pp. 82-90, 1998.

[28] H. Utz, S. Sablatnog, S. Enderle, and G. Kraetzschmar, "Miro-middleware for mobile robot applications," *IEEE Transactions on Robotics and Automation*, vol. 18, no. 4, pp. 493-497, 2002.

[29] Z. Yang, G. Huang, R. L. Y. Guan, and R. Gay, "CORBA as object-oriented infrastructure for factory communication and control," in APCC/OECC 1999, Beijing, China, 1999, pp. 1083-1086.

[30] S. Song, J. Huang, P. Kappler, R. Freimark, and T. Kozlik, "Fault-tolerant Ethernet middleware for IP-based process control networks," in IEEE Local Computer Networks, Tampa, FL, 2000, pp. 116-125.

[31] Y. Tipsuwan and M.-Y. Chow, "Control methodologies in networked control Systems," to be published in *Control Engineering Practice*, vol. 10, no. 11, pp. 1099-1111, 2003.

[32] M.-Y. Chow and Y. Tipsuwan, "Gain adaptation of networked DC motor controllers based on QoS variations," to be published in *IEEE Transactions on Industrial Electronics*, vol. 50, no. 5, 2003.

[33] M.-Y. Chow and Y. Tipsuwan, "Network-based control adaptation for network QoS variation," in IEEE MILCOM 2001, Vienna, VA, 2001, pp. 257-261

[34] N. B. Almutairi, M.-Y. Chow, and Y. Tipsuwan, "Network-based controlled DC motor with fuzzy compensation," in IEEE IECON 2001, Denver, CO, 2001, pp. 1844-1849.

[35] Y. Tipsuwan and M.-Y. Chow, "Network-based controller adaptation based on QoS negotiation and deterioration," in IEEE IECON 2001, Denver, CO, 2001, pp. 1794-1799.

[36] Y. Tipsuwan and M.-Y. Chow, "Gain adaptation of networked mobile robot to compensate QoS deterioration," in IEEE IECON 2002, Sevilla, Spain, 2002, pp. 3146-3151.

[37] Y. Tipsuwan and M.-Y. Chow, "On the gain scheduling for networked PI controller over IP Network," in IEEE/ASME AIM, Kobe, Japan, 2003, pp. 640-645.

[38] Y. Tipsuwan and M.-Y. Chow, "Neural network middleware for model predictive path tracking of networked mobile robot over IP network," to be published in IEEE IECON 2003, Roanoke, VA, 2003.

C H A P T E R I I

CONTROL METHODOLOGIES IN NETWORKED CONTROL SYSTEMS

Yodyium Tipsuwan
ytipsuw@unity.ncsu.edu

Mo-Yuen Chow
chow@eos.ncsu.edu

Advanced Diagnosis And Control Lab
Department of Electrical and Computer Engineering
North Carolina State University, Raleigh NC 27606, USA
Tel: (919) 515-7360, Fax: (919) 515-5108

This chapter is published in Control Engineering Practice, Special Issue on Control Methods for Telecommunication Networks, vol. 10, no. 11, pp. 1099-1111.

CONTROL METHODOLOGIES IN NETWORKED CONTROL SYSTEMS

Abstract—The use of a data network in a control loop has gained increasing attentions in recent years due to its cost effective and flexible applications. One of the major challenges in this so called networked control system (NCS) is the network-induced delay effect in the control loop. Network delays degrade the NCS control performance and destabilize the system. A significant emphasis has been on developing control methodologies to handle the network delay effects in NCS. This survey paper presents recent NCS control methodologies. The overview on NCS structures and description of network delays including characteristics and effects are also covered.

Keywords—communication control applications, communication networks, communication protocols, distributed control, factory automation, delay analysis, delay compensation.

I. Introduction

The research and developments on shared data networks have a long history. Principle data networks such as Slotted ALOHA [1], and ARPANET [2], which were specially developed around 30-40 years ago, evolved to widely used modern network protocols like Ethernet and Internet for general usages, respectively. Data networking technologies provide several benefits on linking data points like computers. Networks enable remote data transfers and data exchanges among users, reduce the complexity in wiring connections and the costs of medias, and provide ease in maintenance.

Because of these attractive benefits, many industrial companies and institutes have shown interest in applying networks for remote industrial control purposes and factory automation. As a result of extensive research and development, several network protocols for industrial control

have been released. For example, CAN (Controller Area Network) was originally developed in 1983 by the German company Robert Bosch for use in car industries, and is also being used now in many other industrial control applications. Another example of industrial networks is Profibus developed by six German companies and five German institutes in 1987. Profibus is a broadcast bus protocol that operates as a multi-master/slave system. Many other industrial network protocols including Foundation Fieldbus and DeviceNet were also developed about the same time period. Most of these protocols are typically reliable and robust for real-time control purposes.

Meanwhile, the technologies on general computer networks especially Ethernet have also progressed very rapidly. With the decreasing price, increasing speed, widespread usages, numerous software and applications, and well-established infrastructure, these networks become major competitors to the industrial networks for control applications [3]. Furthermore, the popularity of the Internet has brought these networks into various organizations. Thus, the control applications can utilize these networks to connect to the Internet in order to perform remote control at much farther distances than in the past without investing on the whole infrastructure. Although the industrial networks have been enhanced for Internet connectivity, the cheaper price and widespread usages of the general networks are still attractive for use in control applications.

Regardless of the type of network used, the overall networked control system performance is always affected by network delays since the network is tied with the control system. Delays are widely known to degrade the performance of a control system. Network delays may not significantly affect an open-loop control system such as on-off relay systems in industrial plants. However, the open-loop control configuration may not be appropriate and adequate for time-sensitive high performance control applications such as telerobotics and telesurgery. These applications require feedback data sent across the network in order to correct the output error. Existing constant time-delay control methodologies may not be directly suitable for

controlling a system over the network since network delays are usually time-varying, especially in the Internet. Therefore, to handle network delays in a closed-loop control system over a network, an advanced methodology is required.

This survey paper provides recent control methodologies for a closed-loop control system over a data network. This closed-loop system configuration is known as a *network-based control system* [4] or *networked control system (NCS)* [5]. The two terms are somewhat interchangeable depending on different authors' preferences. The methodologies described in this paper have been applied and have shown promising results in many applications ranging from DC motors [6, 7] to automobiles [8, 9], aircrafts [10], mobile robots [11, 12], robotic manipulator [13], and distance learning [14, 15]. This paper provides the overview of NCS including system configuration, network delay characteristics, and the effects of networked delays in section II. The control methodologies for NCS will then be described in section III. The paper is concluded in section IV.

II. Overview of Networked Control System

A. Networked control system configuration

There are two general networked control system configurations listed as follows:

- *Direct structure*

The NCS in the direct structure is composed of a controller and a remote system containing a physical plant, sensors and actuators. The controller and the plant are physically located at different locations and are directly linked by a data network in order to perform remote closed-loop control as illustrated in Fig. 1.

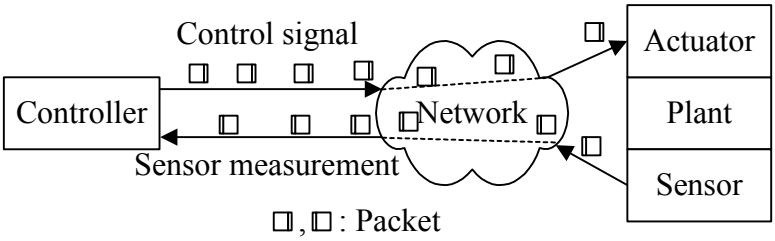


Fig. 1. NCS in the direct structure.

The control signal is encapsulated in a frame or a packet and sent to the plant via the network. The plant then returns the system output to the controller by putting the sensor measurement into a frame or a packet as well. In a practical implementation, multiple controllers can be implemented in a single hardware unit to manage multiple NCS loops in the direct structure. Some examples of NCS in the direct structure are a distance learning lab [15] and a DC motor speed control system [6].

- *Hierarchical structure*

The basic hierarchical structure consists of a main controller and a remote closed-loop system as depicted in Fig. 2.

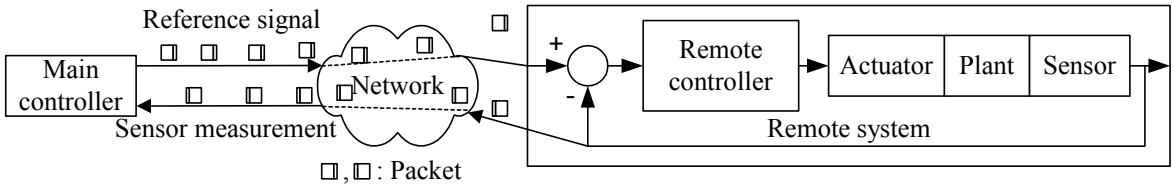


Fig. 2. NCS in the hierarchical structure.

Periodically, the main controller computes and sends the reference signal in a frame or a packet via a network to the remote system. The remote system then processes the reference signal to perform local closed- loop control and returns to the sensor measurement to the main controller for networked closed-loop control. The networked control loop usually has a longer sampling period than the local control loop since the remote controller supposes to satisfy the reference signal before processes the newly arrival reference signal. Similar to the direct

structure, the main controller can be implemented to handle multiple networked control loops for several remote systems. This structure is widely used in several applications including mobile robots [12], and teleoperation [13].

The use of either the direct structure or the hierarchical structure is based on application requirements and designer's preferences. For example, a robotic manipulator usually requires several motors at the joints of the robot to simultaneously and smoothly rotate together. It may be more convenient and more robust to use an existing robot controller and formulate the networked control problem in the hierarchical structure. On the other hand, a designer may require a networked DC motor speed control system [6] to have a faster control response over the network. The direct structure may be preferred in this case.

This survey paper mainly focuses on the fundamental and control methodologies for NCS in the direct structure. Nevertheless, control and analysis methodologies for the direct structure could also be applied for the hierarchical structure by treating the remote closed-loop system as a pure plant. In this case, the remote closed-loop system is represented by a state-space model or a transfer function similar to the plant.

B. Delays in-the-loop

Since an NCS operates over a network, data transfers between the controller and the remote system will induce network delays in addition to the controller processing delay. Fig. 3 shows network delays in the control loop, where **r** is the reference signal, **u** is the control signal, **y** is the output signal, k is the time index, and T is the sampling period. Most of networked control methodologies use the discrete-time formulation shown in Fig. 3. Fig. 4 shows the corresponding timing diagram of network delay propagations.

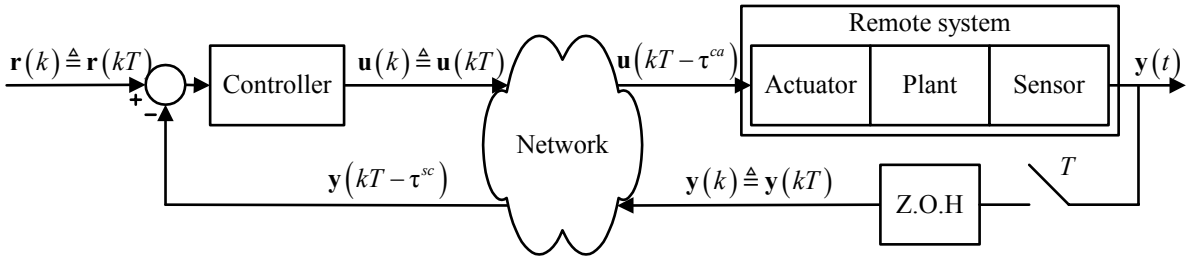


Fig. 3. General NCS configuration and network delays for NCS formulations.

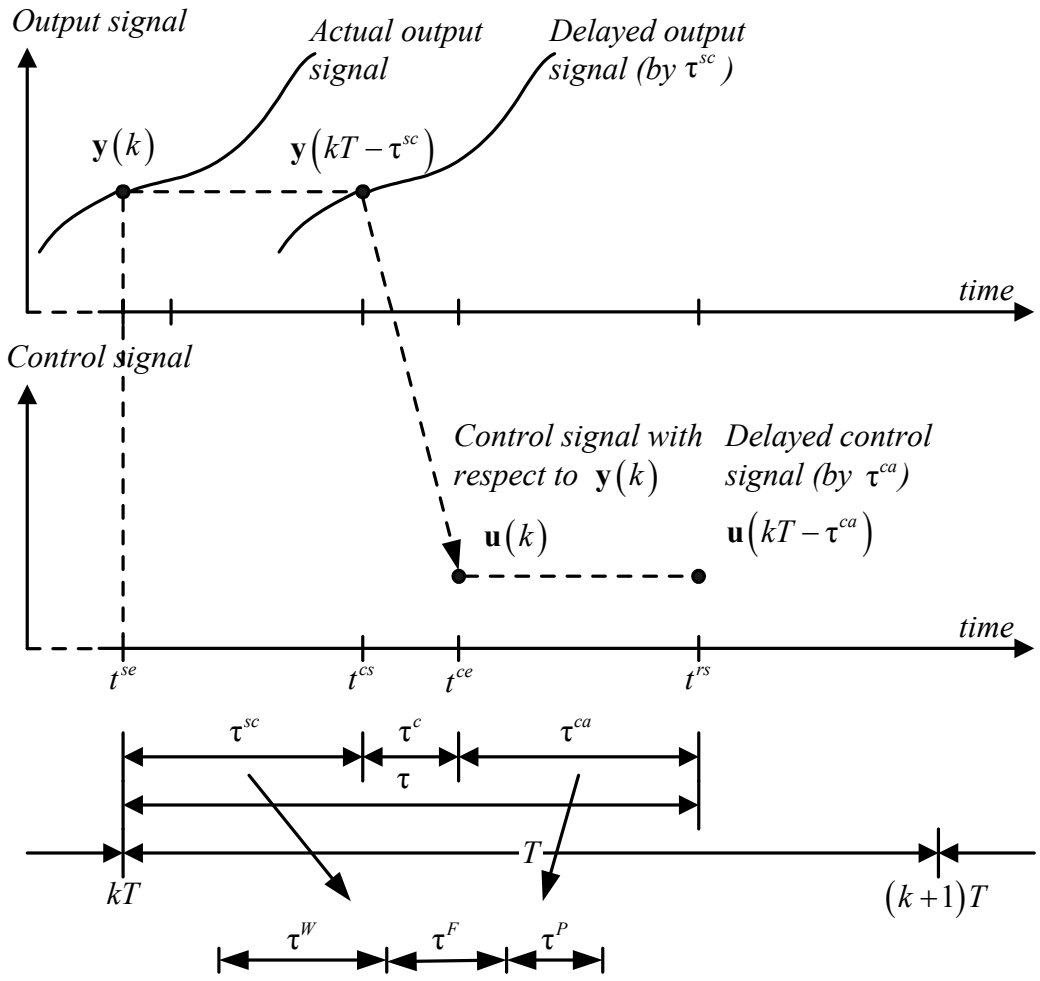


Fig. 4. Timing diagram of network delay propagations.

Network delays in an NCS can be categorized from the direction of data transfers as the sensor-to-controller delay τ^{sc} and the controller-to-actuator delay τ^{ca}. The delays are computed as:

$$\tau^{sc} = t^{cs} - t^{se}, \quad (1)$$

$$\tau^{ca} = t^{rs} - t^{ce}, \quad (2)$$

where t^{se} is the time instant that the remote system encapsulates the measurement to a frame or a packet to be sent, t^{cs} is the time instant that the controller starts processing the measurement in the delivered frame or packet, t^{ce} is the time instant that the main controller encapsulates the control signal to a packet to be sent, and t^{rs} is the time instant that the remote system starts processing the control signal. In fact, both network delays can be longer or shorter than the sampling time *T*. The controller processing delay τ^{c} and both network delays can be lumped together as the control delay τ for ease of analysis. This approach has been used in some networked control methodologies. Although the controller processing delay τ^{c} always exists, this delay is usually small compared to the network delays, and could be neglected. In addition, the sampling periods of the main controller and of the remote system may be different in some cases.

The delays τ^{sc} and τ^{ca} are composed of at least the following parts [16].

- *Waiting time delay* τ^{W}

The waiting time delay is the delay, of which a source (the main controller or the remote system) has to wait for queuing and network availability before actually sending a frame or a packet out.

- *Frame time delay* τ^{F}

The frame time delay is the delay during the moment that the source is placing a frame or a packet on the network.

- *Propagation delay* τ^P

The propagation delay is the delay for a frame or a packet traveling through a physical media. The propagation delay depends on the speed of signal transmission and the distance between the source and destination.

These three delay parts are fundamental delays that occur on a local area network. When the control or sensory data travel across networks, there can be additional delays such as the queuing delay at a switch or a router, and the propagation delay between network hops. The delays τ^{sc} and τ^{ca} also depend on other factors such as maximal bandwidths from protocol specifications, and frame or packet sizes.

Higher layer network protocols such as TCP may require retransmission if an error occurs in a packet, or a switch or a router drops the packet. This incident is a trade-off for an NCS. Even though some control or sensory signals are lost due to network transmissions, some NCS may operate acceptably. In this case, retransmission may be undesirable because the NCS may be severely affected by the extending delays as a result from retransmission.

C. Delay characteristics

The delay characteristics on NCS basically depend on the type of a network used, which are described as follows.

- *Cyclic service network*

In local area network protocols with cyclic service such as IEEE 802.4, SAE token bus, PROFIBUS, IEEE 802.5, SAE token ring, MIL-STD-1553B, and FIP, control and sensory signals are transmitted in a cyclic order with deterministic behaviors. Thus, the delays are periodic and can be simply modeled as a periodic function such that $\tau_k^{sc} = \tau_{k+1}^{sc}$ and $\tau_k^{ca} = \tau_{k+1}^{ca}$, where τ_k^{sc} and τ_k^{ca} are the sensor-to-controller delay and the controller-to-actuator delay at the sampling time period k [17]. The models work perfectly in the ideal case. In practice, NCS may experience small variations on periodic delays due to several reasons. For examples, the

discrepancies in clock generators on a controller and a remote system may result in delay variations.

- *Random access network*

Random access local area networks such as CAN and Ethernet involve with more uncertain delays. The significant parts of random network delays are the waiting time delays due to queuing and frame collision on the networks. When an NCS operates across networks, several more factors can increase the randomness on network delays such as the queuing time delays at a switch or a router, and the propagation time delays from different network paths. In addition, a cyclic service network connected to a random access network also results in random delays.

In the networking area, random network delays have been modeled by using various formulations based on probability and the characteristics of sources and destinations. The techniques range from simple approaches such as the Poisson process to more sophisticated approaches such as Markov chain [18], fluid flow model [19], ARMA model [20], etc. These techniques have been brought to NCS formulations in several studies, but may have to be modified or reformulated for specific networked control methodologies. For example, Markov chain is applied in [21, 22], and simple independent transfer-to-transfer probability distribution models are used in [22] as follows.

$$f_{\tau}\left(\tau_k^{ca}\right)=\delta\left(\tau_k^{ca}-a\right)\cdot\left(1-p_{ca}\right)+\delta\left(\tau_k^{ca}-b\right)\cdot p_{ca}, \tag{3}$$

$$f_{\tau}\left(\tau_k^{sc}\right)=\begin{cases}\delta\left(\tau_k^{sc}-a\right)\cdot\left(1-p_{sc}\right), & \tau_k^{sc}=a,\\ p_{sc}\big/\left(b-a^{+}\right), & \tau_k^{sc}\in\left(a,b\right],a<b,\\ 0, & \tau_k^{sc}\notin\left[a,b\right],\end{cases} \tag{4}$$

where $\delta(\cdot)$ is the Dirac delta function, a and b are constants, and $p_{sc},p_{ca}\in[0,1]$ are parameters of the network.

D. Effects of delays in-the-loop

- *Performance degradation*

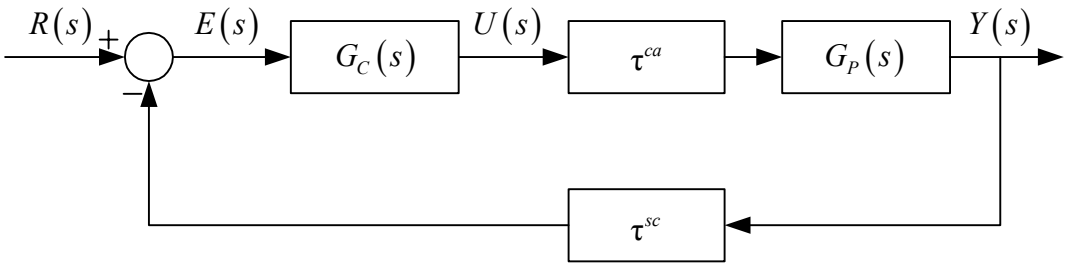
Delays in a control loop are widely known to degrade system performances of a control system, so are the network delays in an NCS. The closed-loop PI (Proportional-Integral) control system with delays in Fig. 5 (a) is used to briefly illustrate system performance degradations by delays in-the-loop, where $R(s)$, $U(s)$, $Y(s)$, and $E(s)=R(s)-Y(s)$ are the reference, control, output, and error signals in Laplace domain according to the reference, control, output, and error signals in time domain, respectively.

The transfer functions of the controller and the plant are described, respectively, as follows:

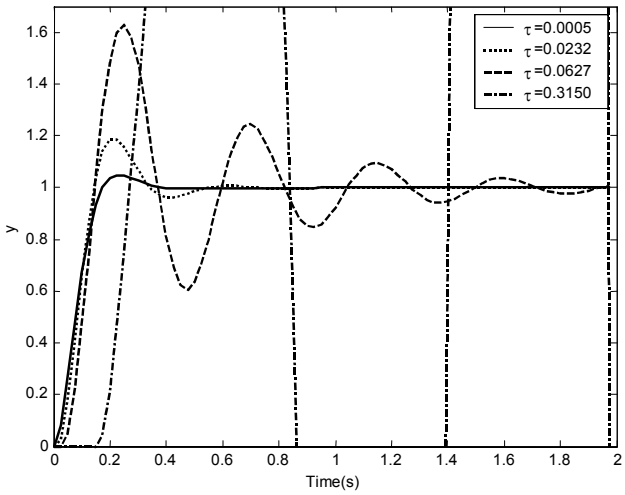
$$G_C(s)=\frac{\beta K_P\left(s+\frac{K_I}{K_P}\right)}{s}, K_P=0.1701, K_I=0.378, \tag{5}$$

$$G_P(s)=\frac{2029.826}{(s+26.29)(s+2.296)}, \tag{6}$$

where $G_C(s)$ is a PI controller, K_P is the proportional gain, K_I is the integral gain, $G_P(s)$ is the plant of a DC motor in [23], β is a parameter to adjust K_P and K_I. In this case, $\beta=1$. As shown in Fig. 5 (b), obvious system performance degradations are the higher overshoot and the longer settling time when the delays $\tau^{ca}=\tau^{sc}=\tau/2$ are longer. Other kinds of performance degradations can be evaluated based on different performance measures. Analyses on the effects of delays on system performance measures can be used for developing appropriate networked control methodologies [6, 24].



(a)



(b)

Fig. 5. System performance degradations caused by delays in-the-loop: (a) Closed-loop control system example. (b) Step response with respect to various τ, where $\tau^{ca} = \tau^{sc} = \tau/2$ are constant, and $\beta = 1$.

- *Destabilization*

Delays in-the-loop including network delays in an NCS can destabilize the system by reducing the system stability margin. Again, the system in Fig. 5 (a) is used to illustrate how the delays can reduce the stability region. Fig. 6 shows the branches of the root locus of the system in Fig. 5 (a) with respect to the parameter β. In this case, increasing β is equivalent to increasing K_P and K_I while maintaining the same ratio between both controller gains. Only

primary branches are shown because they are sufficient to approximate the stability region [25].

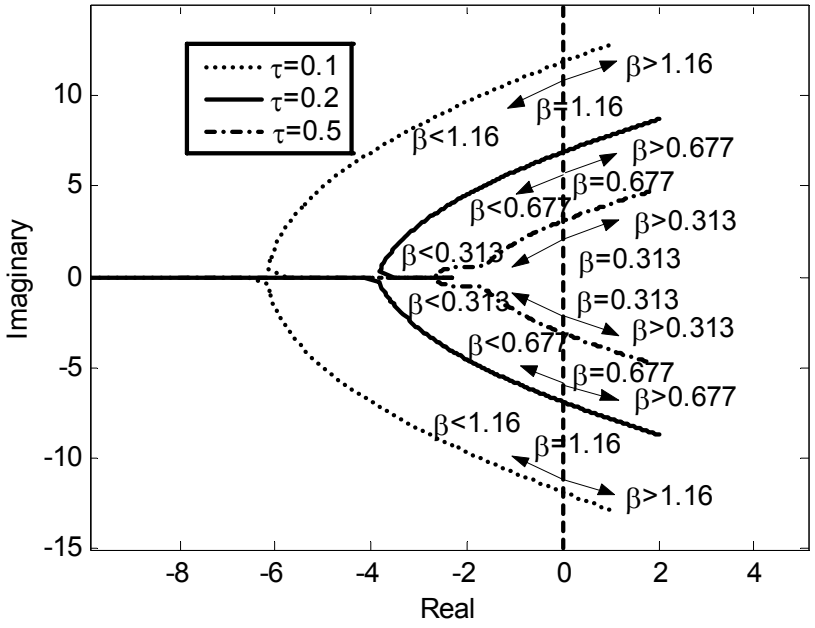


Fig. 6. Primary branches of the root locus of the system in Fig. 5 (a) with respect to β, where $\tau^{ca} = \tau^{sc} = \tau / 2$ are constant.

As shown in Fig. 6, when the delay τ is longer, the primary branches of the root locus bend toward the right of the imaginary axis, and β, at the point at which the branches cross the imaginary axis, is smaller. This result indicates the narrower stability region since the PI controller has the smaller range of feasible values to use for a stable closed-loop control.

There have been several studies to derive stability criteria for an NCS in order to guarantee that the NCS can remain stable in a certain condition. However, there is no generic stability analysis that can be applied on every NCS. Most of stability analysis techniques are subject to network configurations, network protocols, assumptions, and control techniques used.

Simple stability analysis for a discrete-time delayed system in [26] can be applied to a constant delay NCS. A periodic delay NCS requires more sophisticated analysis based on

various system formulations. For example, an NCS on a periodic delay network in [17] is stable if all eigenvalues of a specific formulation are contained in a unit circle. Another formulation in [27] uses a general frequency domain analysis for checking stability, but the stability criterion is limited to a single-dimensional system.

Stability analysis for an NCS with random network delays is more challenging, since more advanced algorithms are usually required. Varieties of techniques have been used for different NCS formulations. For example, in [22] and [21], stabilities of NCS were analyzed based on stochastic stability analysis, but with different formulations. Nonlinear control and perturbation theories were applied for NCS stability analysis in [5] using Bellman-Gronwall Lemma. A hybrid system technique is used to analyze the stability of an NCS in [28].

III. Recent Networked Control Methodologies

Due to network delay concerns, the methodologies to control an NCS have to maintain the stability of the system in addition to controlling and maintaining the system performance as much as possible. Various methodologies have been formulated based on several types of network behaviors and configurations in conjunction with different ways to treat the delay problems. Some assumptions may be required. For example:

- Network transmissions are error-free.
- Every frame or packet always has the same constant length.
- The difference between the sampling times of the controller and of the sensor, called time skew Δ_k, is constant.
- The computational delay τ^c is constant and is much smaller than the sampling period T.
- The network traffic cannot be overloaded.
- Every dimension of the output measurement or the control signal can be packed into one single frame or packet.

Some methodologies are denoted by some specific terminologies defined by the authors of this paper in order to unify and distinguish them.

A. Augmented deterministic discrete-time model methodology

Halevi and Ray [17] proposed a methodology named here as the augmented deterministic discrete-time model methodology to control a linear plant over a periodic delay network. The structure of the augmented discrete-time model is straightforward and easy. In addition, this methodology can be modified to support non-identical sampling periods of a sensor and a controller as mentioned in [29]. The linear plant used in this methodology has the following form:

$$\mathbf{x}(k+1) = \mathbf{\Phi x}(k) + \mathbf{\Gamma u}(k), \tag{7}$$

$$\mathbf{y}(k) = \mathbf{Cx}(k), \tag{8}$$

where $\mathbf{\Phi} = \exp(\mathbf{A}T)$, $\mathbf{\Gamma} = \int_0^T \exp(\mathbf{A}\zeta)d\zeta\mathbf{B}$, and $\{\mathbf{A,B,C}\}$ is the realization of the system. With requiring a set point to be zero, the dynamics of the linear controller used in this methodology can be described by:

$$\boldsymbol{\xi}(k+1) = \mathbf{F}\boldsymbol{\xi}(k) - \mathbf{Gz}(k), \tag{9}$$

$$\mathbf{u}(k) = \mathbf{H}\boldsymbol{\xi}(k) - \mathbf{Jz}(k), \tag{10}$$

where $\boldsymbol{\xi}$ is the controller state vector, $\mathbf{z}(k) = \mathbf{y}(k-i)$, $i = \{1,...,j\}$ is the past measurement at the instant when $\mathbf{u}(k)$ is processed by the controller, and **F**, **G**, **H**, and **J** are constant matrices describing the dynamics of the controller. The control **u** in (10) is the output of this controller.

The main idea to handle network delays in this methodology is to combine and rearrange (7)-(10) into an augmented state-space equation as follows:

$$\mathbf{X}(k+1) = \mathbf{\Omega}(k+1)\mathbf{X}(k), \tag{11}$$

where $\mathbf{X}(k) = \left[\mathbf{x}^T(k), \mathbf{y}^T(k-1), \ldots, \mathbf{y}^T(k-j), \boldsymbol{\xi}^T(k), \mathbf{u}^T(k-1), \mathbf{u}^T(k-l)\right]^T$ is the augmented state vector, and $\mathbf{\Omega}(k+1)$ is the augmented state transition matrix computed from $\mathbf{\Phi}$, $\mathbf{\Gamma}$, **C**, **F**, **G**, **H**, and **J**.

For periodic delays, there exists a positive integer M such that $\tau_{k+M}^{sc} = \tau_k^{sc}$. Using this property, the authors of [17] proved that the system in (11) is asymptotically stable if all eigenvalues of $\Xi_k^M = \prod_{j=1}^{M} \mathbf{\Omega}(k+M-j)$ are contained within the unit circle. Ray and Halevi also suggested an approach to improve the networked control methodology by appropriately selecting Δ_k [30].

B. Queuing methodology

Queuing mechanisms can be used to reshape random network delays on an NCS to deterministic delays such that the NCS becomes time-invariant. The methodologies to control an NCS that is based on queuing mechanisms are defined here as the queuing methodologies. These methodologies have been developed by utilizing some deterministic or probabilistic information of an NCS for the control algorithm formulation.

An early queuing methodology was developed by Luck and Ray [31, 32] denoted here as the deterministic predictor-based delay compensation methodology. This methodology uses an observer to estimate the plant states and a predictor to compute the predictive control based on past output measurements. The control and past output measurements are stored in a FIFO (First-In-First-Out) queue and a shift register defined as Q_1 and Q_2, where the sizes of Q_1 and Q_2 are μ and θ, respectively, as depicted in Fig. 7.

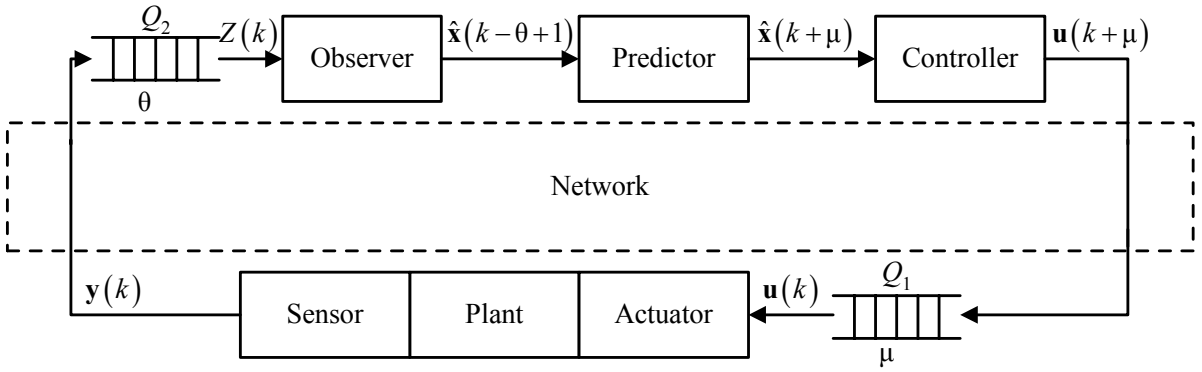


Fig. 7. Configuration of NCS in the deterministic predictor-based delay compensation methodology.

The steps for applying the delay compensation methodology are listed as follows.

- Using the set of past measurements $Z(k)=\{y(k-\phi), y(k-\phi-1),\ldots\}$ in Q_2, where ϕ is the number of packets in Q_2, the observer estimates the plant state $\hat{\mathbf{x}}(k-\theta+1)$.
- The predictor uses $\hat{\mathbf{x}}(k-\theta+1)$ to predict the future state $\hat{\mathbf{x}}(k+\mu)$.
- The controller computes the predictive control $\mathbf{u}(k+\mu)$ from $\hat{\mathbf{x}}(k+\mu)$, and then sends $\mathbf{u}(k+\mu)$ to be stored in Q_1.

Since the performances of the observer and the predictor highly depend on the model accuracy, the dynamic model of the plant has to be very precise.

Chan and Özgüner [33] developed another queuing methodology for controlling an NCS on random delay networks. This methodology, named here as the probabilistic predictor-based delay compensation methodology, utilizes probabilistic information along with the number of packets in a queue to improve state prediction. Nevertheless, this queuing methodology itself is not really a control algorithm, but is more likely a scheme to predict state variables. The configuration of the NCS in probabilistic predictor-based delay compensation methodology is illustrated in Fig. 8.

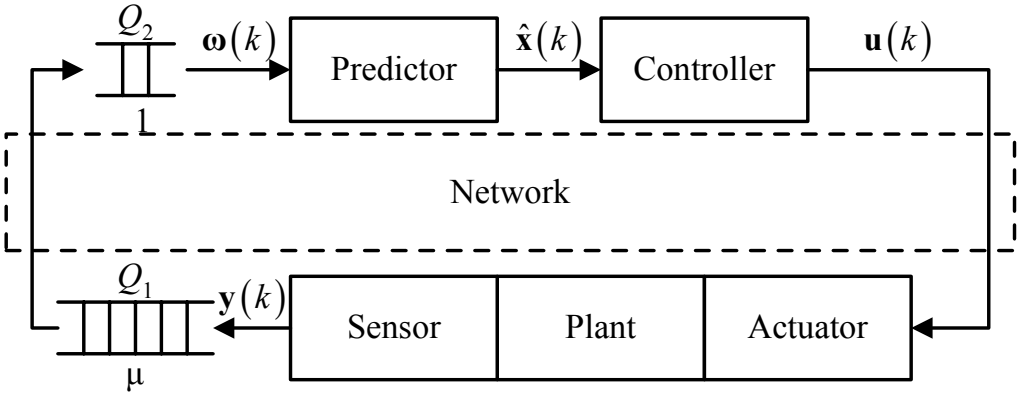


Fig. 8. Configuration of NCS in the probabilistic predictor-based delay compensation methodology.

As shown in Fig. 8, the queue Q_1 at the sensor has a capacity of μ, while the shift register Q_2 can store only one packet. The output $\mathbf{y}(k)$ is stored in Q_1 waiting to be sent to Q_2 when the network is available for a transmission. To describe the compensation methodology, let the number of packets stored in Q_1 and the output from Q_2 be defined as i and $\boldsymbol{\omega}(k)$, respectively. At the sampling time k, if the sensor cannot send $\mathbf{y}(k)$ before Q_2 is read, $\boldsymbol{\omega}(k)$ is set to the previous value $\boldsymbol{\omega}(k-1)$. Otherwise, $\boldsymbol{\omega}(k)$ can be identical to any value in $\{\mathbf{y}(k), \mathbf{y}(k-1), \ldots, \mathbf{y}(k-\mu)\}$. However, the possible choices of $\boldsymbol{\omega}(k)$ can be reduced to either $\mathbf{y}(k-i)$ or $\mathbf{y}(k-i+1)$, if $i = 1, \ldots, \mu$, defined as the delay index, is known. This condition requires that the value of i has to be attached to every packet of $\mathbf{y}(k)$. The predictor then estimates the current state $\hat{\mathbf{x}}(k)$ by:

$$\hat{\mathbf{x}}(k) = \mathbf{P}_0\left(\boldsymbol{\Phi}^{i-1}\omega(k) + \mathbf{W}_i\right) + \mathbf{P}_1\left(\boldsymbol{\Phi}^{i}\omega(k) + \mathbf{W}_{i+1}\right), \tag{12}$$

$$\mathbf{W}_i = \begin{cases} 0, & ,i = 1, \\ \left[\boldsymbol{\Gamma} \quad \boldsymbol{\Phi}\boldsymbol{\Gamma} \quad \ldots \quad \boldsymbol{\Phi}^{i-2}\boldsymbol{\Gamma}\right] \cdot \left[\mathbf{u}^T(k-1) \quad \mathbf{u}^T(k-2) \quad \ldots \quad \mathbf{u}^T(k-i+1)\right]^T & ,i \neq 1, \end{cases} \tag{13}$$

where $\mathbf{P}_0$ and $\mathbf{P}_1$ are weighting matrices. The weighting matrices are computed from the probabilities of the occurrences of $\mathbf{y}(k-i)$ and $\mathbf{y}(k-i+1)$. These equations require full state information (i.e., $\mathbf{y}(k) = \mathbf{x}(k)$). If the full state information is not available, an observer can also be applied with minor modification. With the predictive states, a control law from various control algorithms can be applied in this methodology.

C. Optimal stochastic control methodology

Nilsson [22] proposed the optimal stochastic control methodology to control an NCS on random delay networks. The optimal stochastic control methodology treats the effects of random network delays in an NCS as an LQG (Linear-Quadratic-Gaussian) problem. Other than the assumptions mentioned earlier, this methodology assumes that $\tau < T$.

The dynamics of a remote system plant in this methodology is described by:

$$\mathbf{x}(k+1) = \boldsymbol{\Phi}\mathbf{x}(k) + \boldsymbol{\Gamma}_0(\boldsymbol{\tau}_k)\mathbf{u}(k) + \boldsymbol{\Gamma}_1(\boldsymbol{\tau}_k)\mathbf{u}(k-1) + \mathbf{v}(k), \tag{14}$$

$$\mathbf{y}(k) = \mathbf{C}\mathbf{x}(k) + \mathbf{w}(k), \tag{15}$$

where $\boldsymbol{\tau}_k = \left[\tau_k^{sc}, \tau_k^{ca}\right]^T$ indicates network delays at the sampling time k, $\boldsymbol{\Phi} = \exp(\mathbf{A}T)$, $\boldsymbol{\Gamma}_0(\boldsymbol{\tau}_k) = \int_0^{T-\tau_k^{sc}-\tau_k^{ca}} \exp(\mathbf{A}\zeta)d\zeta\mathbf{B}$, and $\boldsymbol{\Gamma}_1(\boldsymbol{\tau}_k) = \int_{T-\tau_k^{sc}-\tau_k^{ca}}^{T} \exp(\mathbf{A}\zeta)d\zeta\mathbf{B}$. The stochastic processes $\mathbf{v}(k)$ and $\mathbf{w}(k)$ are uncorrelated Gaussian white noises with zero means. These equations are modified from the constant delay system in [26].

The goal of the optimal stochastic control methodology is to minimize the following cost function in the case that full state information is available.

$$J(k) = E\left[\mathbf{x}^T(N)\mathbf{Q}_N\mathbf{x}(N)\right] + E\sum_{k=0}^{N-1}\begin{bmatrix}\mathbf{x}(k)\\ \mathbf{u}(k)\end{bmatrix}^T \mathbf{Q}\begin{bmatrix}\mathbf{x}(k)\\ \mathbf{u}(k)\end{bmatrix}, \tag{16}$$

where $E[\cdot]$ is the expected value, and $\mathbf{Q}_N$ and $\mathbf{Q}$ are weighting matrices. The control law for the optimal state feedback is derived by using dynamic programming and is described as:

$$\mathbf{u}(k) = -\mathbf{L}(k, \boldsymbol{\tau}_k)\begin{bmatrix}\mathbf{x}(k)\\ \mathbf{u}(k-1)\end{bmatrix}, \tag{17}$$

where $\mathbf{L}$ is the optimal gain matrix after solving the formulated LQG problem. The network delay $\boldsymbol{\tau}_k$ is assumed to be independent. The past information of the delay is also required. If the full state information is not available, an optimal estimator such as the Kalman filter can be applied for (17). Nevertheless, this case requires the past information of output and input $\{\mathbf{y}(0),\ldots,\mathbf{y}(k),\mathbf{u}(0),\ \ldots,\mathbf{u}(k-1)\}$ in conjunction with the past information of the delay. Another control law to use with the delays modeled by a Markov Chain is also derived in the same study. Based on (16), the optimal stochastic control methodology has shown to give better performance than the deterministic predictor-based delay compensation methodology.

D. Perturbation methodology

Walsh, Beldiman, Ye, and Bushnell [5, 34] used non-linear and perturbation theory to formulate network delay effects in an NCS as the vanishing perturbation of a continuous-time system under the assumption that there is no observation noise. This methodology, denoted here as the perturbation methodology, can be applied on an NCS on periodic delay networks

and random delay networks at the sensor-to-controller transmission. However, these networks are restricted to be *priority-based networks*, which can assign different priorities to data transmissions. These priorities can be managed by priority scheduling algorithms proposed in [35]. In addition, this methodology requires a very small sampling time so that an NCS can be approximated as a continuous-time system. A control loop in the perturbation methodology consists of a nonlinear controller and a nonlinear plant, but the analysis and derivations used can be similarly applied to linear systems, as described in [5]. Fig. 9 shows the block diagram of the NCS in this methodology.

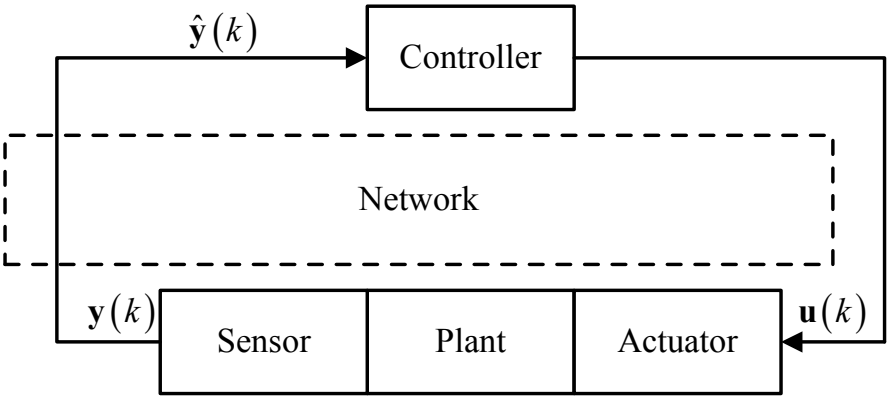


Fig.9. Configuration of NCS in the perturbation methodology.

The dynamics of the NCS in the perturbation methodology is represented by:

$$\dot{\mathbf{x}}(t) = \mathbf{f}\left(\mathbf{x}(t), \mathbf{e}(t), t\right), \tag{18}$$

where $\mathbf{x}(t) = \left[\mathbf{x}_p^T(t), \mathbf{x}_c^T(t)\right]^T$ is the augmented state vector containing the plant state vector $\mathbf{x}_p(t)$ and the controller state vector $\mathbf{x}_c(t)$. The error of the NCS is described by:

$$\mathbf{e}(t) = \mathbf{y}(t) - \hat{\mathbf{y}}(t), \tag{19}$$

where $\mathbf{y}(t)$ is the plant output, and $\hat{\mathbf{y}}(t) \triangleq \mathbf{y}\left(t - \tau^{sc}\right)$ is the most updated output which the controller receives. Also, $\mathbf{e}(t)$ is assumed to have certain dynamics as follows.

$$\dot{\mathbf{e}}(t) = \mathbf{g}\left(t, \mathbf{x}(t), \mathbf{e}(t)\right). \tag{20}$$

The dynamics equation in (20) is treated as the vanishing perturbation to derive a delay bound ρ such that the NCS remains stable if $\tau^{sc} < \rho$.

E. Sampling time scheduling methodology

Hong [27] developed the sampling time scheduling methodology to appropriately select a sampling period for an NCS such that network delays do not significantly affect the control system performance, and the NCS remains stable. This methodology is originally used for multiple NCS on a periodic delay network, in which all connections of every NCS on the network are known in advance. However, it was also modified to apply on random delay networks such as CAN in [36]. This methodology requires $\tau < T$, and is applicable to only a single-dimensional NCS.

To briefly describe the sampling time scheduling methodology, let the number of NCS on the network be *M*. The sampling times of all *M* NCS on the network are calculated from the sampling time of the most sensitive NCS based on the general frequency domain analysis on its worst-case delay bound. The most sensitive NCS, denoted as NCS_1, has the shortest delay bound defined as φ_1. The sampling time scheduling algorithm is formulated from the window concept illustrated in Fig. 10, where *L* and σ are the transmission periods of a pure data message and its overhead, respectively; T_1 is the sampling time of NCS_1, and *r* is the number of data messages that can be served by the network during the worst-case network traffic.

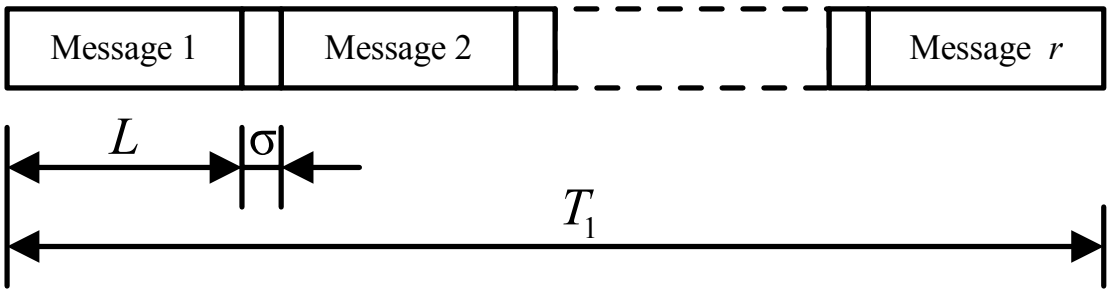


Fig. 10. Windows of data transmissions in the sampling period T_1 of the sampling time scheduling methodology.

The sampling time T_1 is computed from:

$$T_1 = \frac{\varphi_1 + L}{3}. \tag{21}$$

In order to find the sampling times of other NCS on the same network, these systems have to be indexed from the worst-case delay bounds of the systems in an ascending order as NCS_2, …, NCS_M. For example, the system NCS_2 has the worst-case delay bound longer than the worst-case delay bound of NCS_1, but is shorter than then the worst-case delay bound of NCS_3. The sampling times of NCS_2, …, NCS_M are determined from T_1 using different rules with respect to network conditions. In a generic case, all other sampling times are multiples of T_1 as expressed by:

$$T_i = k_i T_1,\ i = 2,3,\ldots,M, \tag{22}$$

$$k_i = \Lambda\left(\frac{\varphi_i - (T_1 - L)}{2T_1}\right), \tag{23}$$

where T_i is the sampling time of NCS_i, and $a = \Lambda(b)$ indicates that $a = 2^{\nu_i}, \nu_i \in \{0,1,2,\ldots\}$, which is the "closest" to, but does not exceed b.

In a special case, in which the number of NCS and other resources connected on the same network is less than r, the sampling times of NCS_2, …, NCS_M are determined by:

$$T_i = \frac{\varphi_i - (T_1 - L)}{2},\ i = 2,3,\ldots,M. \tag{24}$$

In addition, the optimality of the network utilization can be achieved by this methodology, which is an advantage among other methodologies. The condition for the optimality is:

$$2\sum_{i=1}^{M}\frac{T_1}{T_i} = r. \tag{25}$$

Kim, Kwon, and Park [4, 7] enhanced the concept of sampling time scheduling to develop another algorithm for the multi-dimensional NCS in. In this work, the delay bound of each system is obtained from different stability criteria. The dynamics of such a multi-dimensional NCS is briefly expressed as follows:

$$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{A}_1\mathbf{x}\left(t - \tau^{sc}\right) + \mathbf{A}_2\mathbf{x}\left(t - \tau^{sc} - \tau^{c}\right) + \mathbf{A}_3\mathbf{x}\left(t - \tau^{ca} - \tau^{c}\right), \tag{26}$$

where $\mathbf{x}(t)=\left[\mathbf{x}_p^T(t),\mathbf{x}_c^T(t)\right]^T$, $\mathbf{x}_p(t)$ is the plant state vector, $\mathbf{x}_c(t)$ is the controller state vector. The matrices $\mathbf{A}$, $\mathbf{A}_1$, $\mathbf{A}_2$, and $\mathbf{A}_3$ are calculated from the realizations of the plant and the controller. Two existing asymptotically stability criteria based on Lyapunov function can be used to find the delay bound in this generalized methodology.

F. Robust control methodology

Göktas [37] designed a networked controller in the frequency domain using robust control theory. This methodology is denoted here as the robust control methodology. A major advantage of this methodology is that it does not require a priori information about the probability distributions of network delays. In the robust control methodology, the network delays τ^{ca} and τ^{sc} are modeled as simultaneous multiplicative perturbation. Both delays τ^{sc} and τ^{ca} are also assumed to be bounded and able to be approximated by the fluid-flow model [19]. The network delay formulation is described as follows:

$$\begin{aligned}\tau^n &= \frac{1}{2}\left(\tau_{\max}+\tau_{\min}\right)+\frac{1}{2}\left(\tau_{\max}-\tau_{\min}\right)\delta,\ -1\le\delta\le 1,\\ &= \left(1-\alpha\right)\tau_{\max}+\alpha\tau_{\max}\delta,\ 0\le\alpha\le 1/2,\end{aligned} \tag{27}$$

where τ^n can be τ^{sc} and τ^{ca}, $\tau_{\max}$ is the upper bound of τ^n, $\tau_{\min}$ is the lower bound of τ^n, α and δ are real numbers to be determined based on an application. The first term in (27) represents a constant delay, whereas the second term represents the uncertain delay varying from the first term. The delay in (27) is converted for use in the frequency domain, and approximated by the first-order Padé approximation as:

$$e^{-\tau^n s}=e^{-s(1-\alpha)\tau_{\max}}e^{-s\alpha\tau_{\max}\delta}\approx\frac{1-s\tau^n/2}{1+s\tau^n/2}\approx\left(\frac{1-s\left(1-\alpha\right)\tau_{\max}/2}{1+s\left(1-\alpha\right)\tau_{\max}/2}\right)\left(\frac{1-s\alpha\tau_{\max}\delta/2}{1+s\alpha\tau_{\max}\delta/2}\right). \tag{28}$$

The uncertain delay part is then treated as the simultaneous multiplicative perturbation expressed as follows:

$$\left(\frac{1-s\alpha\tau_{\max}\delta/2}{1+s\alpha\tau_{\max}\delta/2}\right)=1+W_m(s)\Delta. \tag{29}$$

where Δ is the perturbation function, and $W_m(s) = \dfrac{\alpha\tau_{\max} s}{1+\alpha\tau_{\max} s/3.465}$ is a multiplicative uncertainty weight which covers the uncertain delay. The factor 3.465 is selected based on a designer's preference. This formulation is then put in H_∞ framework, and μ-synthesis is used to design a continuous time controller $G_C(s)$ for a plant $G_P(s)$. The control loop in the robust control methodology using this formulation is depicted in Fig. 11, where $R(s)$, $U(s)$, $Y(s)$, and $E(s) = R(s) - Y(s)$ are the reference, control, output, and error signals in the frequency domain, respectively.

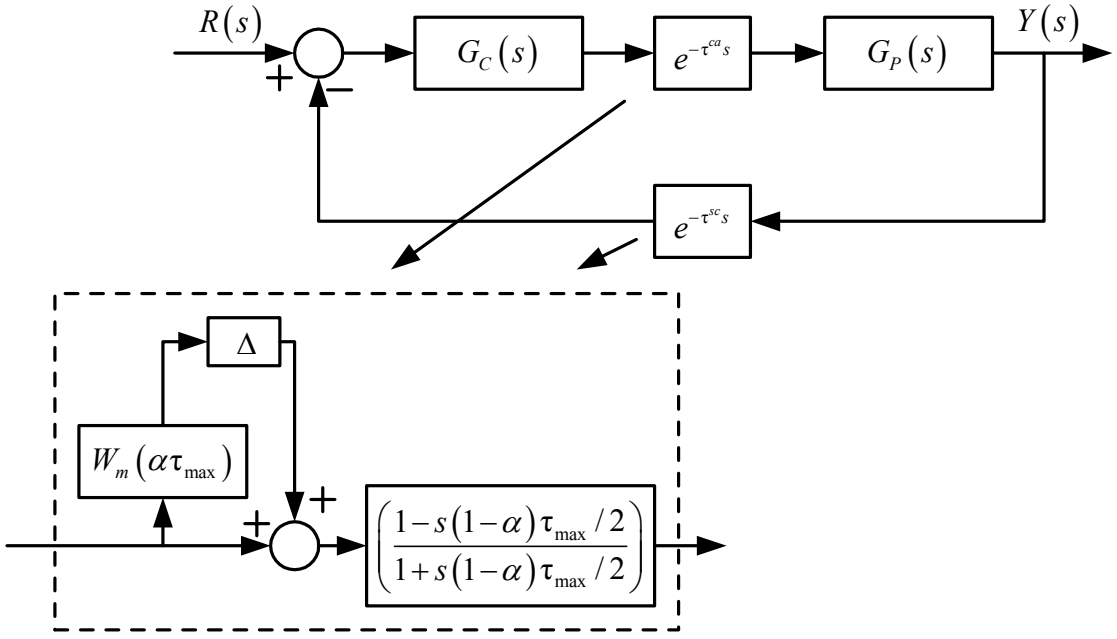


Fig. 11. Configuration of NCS in the robust control methodology.

The controller is discretized using the bi-linear transformation on an actual network. The author also suggested an approach to apply the robust control methodology with network Quality-of-Service on an ATM network in order to achieve the maximum tolerable error on a mobile robot application.

G. *Fuzzy logic modulation methodology*

Almutairi, Chow, and Tipsuwan [38] proposed the fuzzy logic modulation methodology for an NCS with a linear plant and a modulated PI controller to compensate the network delay effects based on fuzzy logic [39]. In this methodology, the PI controller gains are externally updated at the controller output with respect to the system output error caused by network delays. Thus, the PI controller needs not to be redesigned, modified, or interrupted for use on a network environment. A DC motor speed control problem is used to illustrate the proposed methodology. The system configuration of the fuzzy logic modulator methodology is depicted in Fig. 12, where $r(t)$, $e(t)$, $y(t)$, are the reference, error, and output of the system. The output of the PI controller is defined as $u_{PI}(t)$, and the modified PI controller output by the fuzzy logic modulation methodology is defined as $u_C(t)$.

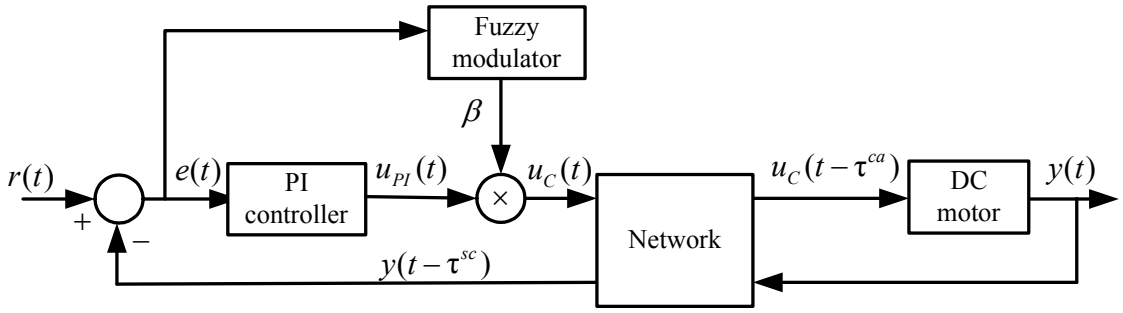


Fig. 12. Configuration of NCS in the fuzzy logic modulation methodology.

The fuzzy logic modulation methodology can be implemented in a unit called the fuzzy logic modulator, which modifies the control $u_{PI}(t)$ by:

$$u_C(t) = \beta u_{PI}(t) = \beta K_P e(t) + \beta K_I \int_{t_0}^{t} e(\xi) d\xi . \tag{30}$$

The multiplicative factor β is used to externally adjust the controller gains at the output without interrupting the original PI controller. The value of β is selected from two fuzzy rules based on the network delay effects as follows:

If $e(t)$ is SMALL, then $\beta = \beta_1$,

If $e(t)$ is LARGE, then $\beta = \beta_2$,

where $0 < \beta_1 < \beta_2 < 1$. The membership functions of $e(t)$ are depicted in Fig. 13, where μ_{SMALL} and μ_{LARGE} are the membership functions representing the degrees of memberships for the linguistic variable SMALL and LARGE, respectively; α_1 and α_2 are factors to adjust the shapes of the membership functions.

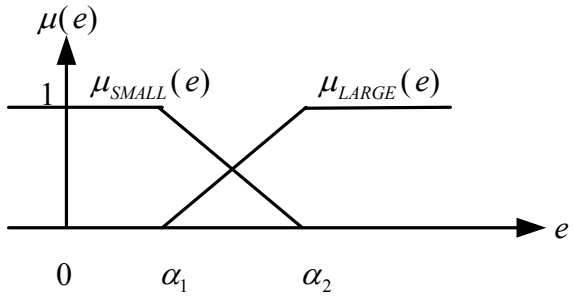


Fig. 13. Membership functions of $e(t)$.

The shapes of the membership functions and the values of β_1 and β_2 are fine tuned by on-line and off-line optimization using the steepest descent algorithm based on cost functions. The cost functions for the on-line optimization are:

$$J(k) = e^2(k), \tag{31}$$

$$J(k) = \sum_{i=k-m}^{k} e^2(i), \tag{32}$$

where the costs (31) and (32) indicate the instantaneous error, and the summing error evaluated from a moving window with the size of *m*. On the other hand, the off-line optimization uses the different cost function as follows:

$$\mathbf{J} = \lambda \mathbf{J}_1 + (1-\lambda)\mathbf{J}_2, \tag{33}$$

where

$$\mathbf{J}_1(\mathbf{p}) = \frac{\sum_{k=0}^{N} e(k)^2}{\left\| \mathbf{J}_1(\mathbf{p}) \right\|_{\infty}}, \tag{34}$$

$$\mathbf{J}_2(\mathbf{p}) = \frac{\sum_{i=1}^{M} \Delta e_b(i)^2}{\left\| \mathbf{J}_2(\mathbf{p}) \right\|_\infty}, \tag{35}$$

and $\{\Delta e_b(i)\} = \{\Delta e(k) \mid e(k)\Delta e(k) > 0\}$. The cost $\mathbf{J}_1$ places the penalty on the system response time and poor convergence; the cost $\mathbf{J}_2$ gives the extra penalty on the system overshoot, undershoot, and oscillatory behaviors, and λ is a weighting factor. The parameter vector **p** represents the membership function parameters and β_1 and β_2.

H. Event-based methodology

Tarn and Xi [13] introduced the event-based methodology for networked control of a robotic manipulator over the Internet. This methodology was originally developed for the hierarchical structure, but could be applied for the direct structure as well. The concept of the event-based methodology is quite different from all the previously mentioned methodologies. Instead of using time, this methodology uses a system motion as the reference of the system. The motion reference defined as *s* can be, for example, the distance traveled by an end-effector of a robotic manipulator. The motion reference *s* has to be a non-decreasing function of time in order to guarantee the system stability. The configuration of NCS in the event-based methodology is depicted in Fig. 14.

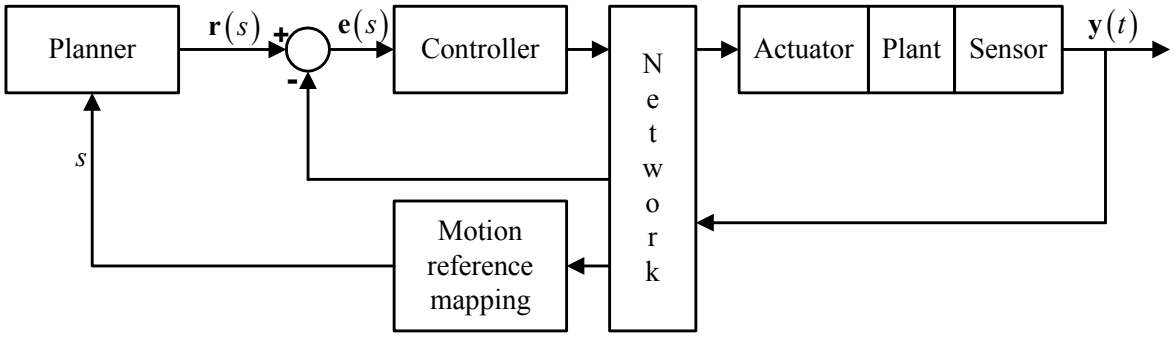


Fig. 14. Configuration of NCS in the event-based methodology.

The output measurement $y(t)$ sent across a network is used as an input for a motion reference mapping. The mapping converts $y(t)$ to the motion reference *s*, which is then used

as the input for the planner to compute the reference $\mathbf{r}(s)$. Thus, $\mathbf{r}(s)$ becomes a function of $y(t)$, and is updated in real-time to compensate all disturbances and unexpected events including network delays. Because the overall system is not based on time, network delays will not destabilize the system.

I. End-user control adaptation methodology

Tipsuwan and Chow [6] proposed the end-user control adaptation methodology. The main concept of end-user control adaptation is to adapt controller parameters (e.g., controller gains) with respect to the current network traffic condition or the current given network QoS (Quality-of-Service). In this methodology, the controller and the remote system are assumed to be able to measure network traffic conditions. The traffic condition measurement in this case could be measured through middleware [40]. The end-user control adaptation methodology is originally designed to cooperate with real-time QoS negotiation scheme [41], in which the controller can request and update network QoS requirements from the network. If the desired QoS requirements cannot be granted, the controller will adapt the parameters to aim for the best possible performance. The parameters are optimal with respect to the current traffic condition.

An application used to demonstrate the end-user control adaptation methodology is a DC motor speed control system controlled over a network link with random network delays. The DC motor speed is controlled by a PI controller, and the parameters to be adapted are the proportional gain K_P and the integral gain K_I. The system performance is measured by using the mean-squared error as follows:

$$J = \frac{1}{N}\sum_{k=1}^{N}\left|r(k) - y(k)\right|. \tag{36}$$

The network QoS measure used in this case is defined as $QoS^{(n)} = \begin{bmatrix} QoS_1 & QoS_2 \end{bmatrix}^T$, where *n* is the index to indicate a QoS condition, and:

- QoS_1: point-to-point network throughput.
- QoS_2: point-to-point maximal delay bound of the largest packet.

The optimal controller parameters with respect to controller gains under different $QoS^{(n)}$ are pre-computed by simulations and stored in a look-up table. The controller gains will be updated when the network traffic condition changes. An example of the cost surface of (36) with respect to PI controller gains is shown in Fig. 15.

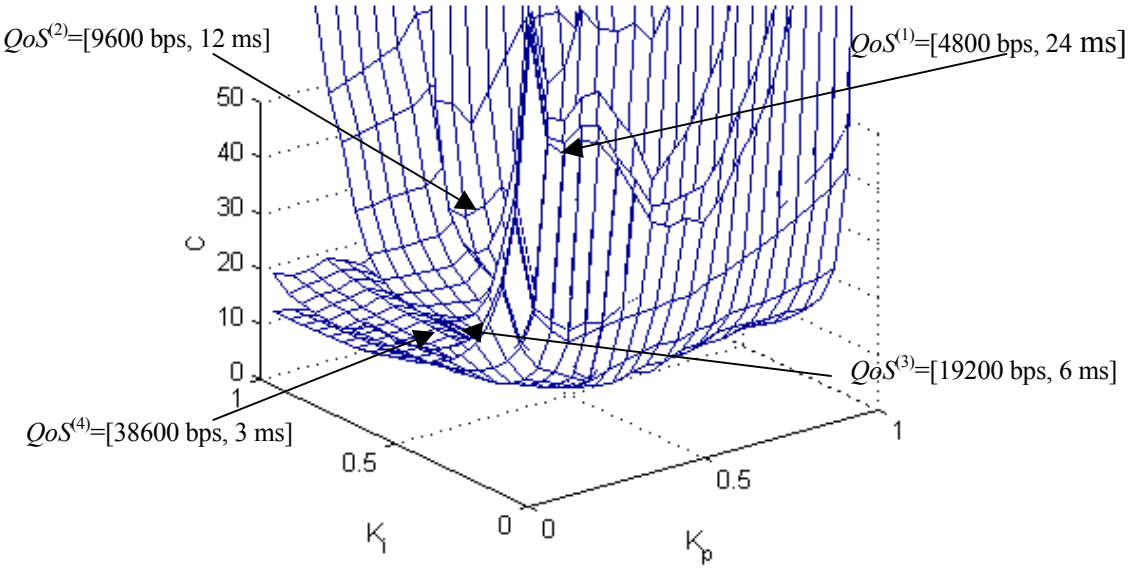


Fig.15. Cost surface with respect to controller gains under different QoS conditions.

The authors illustrated the performance of the end-user control adaptation methodology by letting the network condition changes from $QoS^{(1)} = [38400 \text{ bps}, 5 \text{ ms}]$ to $QoS^{(2)} = [19200 \text{ bps}, 8 \text{ ms}]$ when the reference changes from 200 rad/s to 300 rad/s at $t = 3.5$ sec. The step response of the actual DC motor speed control system is shown in Fig. 16.

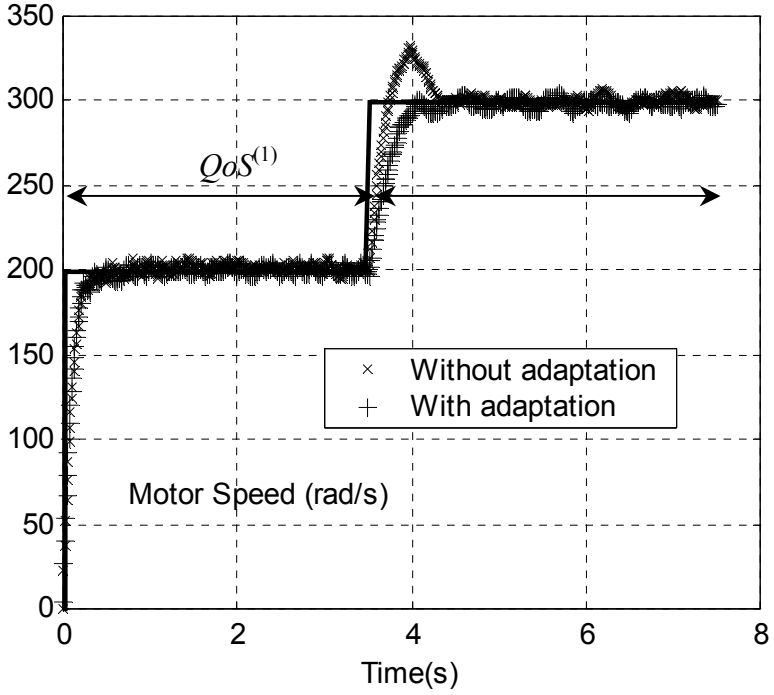


Fig.16. Step responses of an actual networked DC motor speed control system in the end-user control adaptation methodology; ×: without adaptation, +: with adaptation.

As shown in Fig. 16, the networked DC motor speed control system with control gain adaptation has superior performance than without gain adaptation as indicated by the lower overshoot. The end-user control gain adaptation methodology can also be applied for the hierarchical structure as shown in [12].

IV. Conclusion

This survey paper has introduced the fundamental and recent control methodologies for NCS. An NCS can be designed in the direct structure and/or the hierarchical structure depending on the application requirements and the designer's preferences. Regardless of the structure used, the system performance of NCS will degrade due to the existences of network delays in the control loop. In the worst case, the network delays can destabilize the NCS by reducing the system stability region. Random network delays in-the-loop are more difficult to

handle than constant or periodic delays because there is no existing criterion to generally guarantee the stability of an NCS. Stability criteria for NCS are usually subject to specific methodologies and network protocols. Therefore, to design an NCS with a certain networked control methodology, a designer has to clearly understand an application whether it is feasible, acceptable, and reliable enough to be controlled by the methodology under a selected network protocol. There are also additional factors of concern including the price for the network protocol, and the size and distance of the application. The control methodologies described in this paper cover a large variety of systems and protocols. For example:

- If a plant in NCS is linear, every methodology can be applied. However, if a plan is nonlinear, only the perturbation methodology, robust control methodology, and event-based control methodology can be used at this stage.
- The queuing methodology should not be used on a cyclic service network since the methodology will result in longer delays unnecessarily.
- If the network delays are unbounded, and the final time of the system is not critical, the event-based methodology is preferred because the system can remain stable.
- The end-user adaptation methodology is preferred when the network QoS can be provided or monitored.

Even though the methodologies described in this paper are mostly applied on wired local area networks, the networked control applications and research can be extended on progressing network technologies including wireless networks, ad hoc networks, and Internet technologies. Furthermore, certain issues in NCS can be investigated such as the effect of packet loss on NCS, QoS requirements of NCS, stability analysis on various wired and wireless protocols in order to advance and strengthen networked control applications. With the rapid spreading of network applications to every place including homes, offices, and manufacturing plants, the research and reward in NCS could be substantial in the near future.

Acknowledgement

The authors would like to thank the Royal Thai Government for partially supporting this study.

References

[1] W. Stallings, *Data & Computer Communication*, 6 ed. Upper Saddle River, NJ: Prentice Hall, 2000.

[2] D. Minoli and A. Schmidt, *Internet Architectures*: John Wiley & Sons, 1999.

[3] G. Kaplan, "Ethernet's winning ways," *IEEE Spectrum*, vol. 38, no. 1, pp. 113-115, 2001.

[4] Y. H. Kim, H. S. Park, and W. H. Kwon, "Stability and a scheduling method for network-based control systems," in IEEE IECON 96, Taipei, Taiwan, 1996, pp. 934-939.

[5] G. C. Walsh, H. Ye, and L. Bushnell, "Stability analysis of networked control systems," in American Control Conference, San Diego, CA, 1999, pp. 2876-2880.

[6] Y. Tipsuwan and M.-Y. Chow, "Network-based controller adaptation based on QoS negotiation and deterioration," in IEEE IECON 2001, Denver, CO, 2001, pp. 1794-1799.

[7] Y. H. Kim, H. S. Park, and W. H. Kwon, "A scheduling method for network-based control systems," in American Control Conference, Philadelphia, Pensilvania, USA, 1998, pp. 718-722.

[8] N. Boustany, M. Folkerts, K. Rao, A. Ray, L. Troxel, and Z. Zhang, "A simulation based methodology for analyzing network-based intelligent vehicle control systems," in Intelligent Vehicles Symposium, Detroit, MI, 1992, pp. 138-143.

[9] Ü. Özgüner, H. Göktas, H. Chan, J. Winkelman, M. Liubakka, and R. Krotolica, "Automotive suspension control through a computer communication network," in IEEE Control Applications, Dayton, OH, 1992, pp. 895-900.

[10] A. Ray, "Performance Evaluation of Medium Access Control Protocols for Distributed Digital Avionics," *Journal of Dynamic systems, Measurement, and Control*, vol. 109, pp. 370-377, 1987.

[11] M. Wargui, M. Tadjine, and A. Rachid, "Stability of real time control of an autonomous mobile robot," in IEEE International Workshop on Robot and Human Communication, Tsukuba, Japan, 1996, pp. 311-316.

[12] Y. Tipsuwan and M.-Y. Chow, "Gain adaptation of networked mobile robot to compensate QoS deterioration," in IEEE IECON 2002, Sevilla, Spain, 2002, pp. 3146-3151.

[13] T.-J. Tarn and N. Xi, "Planning and control of internet-based teleoperation," in Proceedings of SPIE: Telemanipulator and Telepresence Technologies V, Boston, MA, 1998, pp. 189-193.

[14] G. V. Kondraske, R. A. Volz, D. H. Johnson, D. Tesar, J. C. Trinkle, and C. R. Price, "Network-based infrastructure for distributed remote operations and robotics research," *IEEE Transactions on Robotics and Automation*, vol. 9, no. 5, pp. 702-704, 1993.

[15] J. W. Overstreet and A. Tzes, "An Internet-based real-time control engineering laboratory," *IEEE Control Systems Magazine*, vol. 19, no. 5, pp. 19-34, 1999.

[16] F.-L. Lian, J. R. Moyne, and D. M. Tilbury, "Performance evaluation of control networks: Ethernet, ControlNet, and DeviceNet," *IEEE Control Systems Magazine*, vol. 21, no. 1, pp. 66-83, 2001.

[17] Y. Halevi and A. Ray, "Integrated communication and control systems : Part I - Analysis," *Journal of Dynamic Systems, Measurement, and Control*, vol. 110, pp. 367-373, 1988.

[18] S. Shakkottai, A. Kumar, A. Karnik, and A. Anvekar, "TCP performance over end-to-end rate control and stochastic available capacity," *IEEE/ACM Transactions on Networking,* vol. 9, no. 4, pp. 377-391, 2001.

[19] J. Filipiak, *Modelling and Control of Dynamic Flows in Communication Networks*. Berlin, Germany: Springer-Verlag, 1988.

[20] Q. Li and D. L. Mills, "Jitter-based delay-boundary prediction of wide-area networks," *IEEE/ACM Transactions on Networking*, vol. 9, no. 5, pp. 578-590, 2001.

[21] R. Krtolica, Ü. Özgüner, H. Chan, H. Göktas, J. Winkelman, and M. Liubakka, "Stability of linear feedback systems with random communication delays," *International Journal of Control*, vol. 59, no. 4, pp. 925-953, 1994.

[22] J. Nilsson, “Real-time control systems with delays,” *Ph.D. dissertation*, Lund Institute of Technology, Lund, Sweden, 1998.

[23] Y. Tipsuwan and M.-Y. Chow, "Fuzzy logic microcontroller implementation for DC motor speed control," in IEEE IECON 99, San Jose, CA, 1999, pp. 1271-1276.

[24] J. K. Yook, D. M. Tilbury, and N. R. Soparkar, "A design methodology for distributed control systems to optimize performance in the presence of time delays," in American Control Conference, Chicago, IL, 2000, pp. 1959-1964.

[25] B. C. Kuo, *Automatic Control Systems*, 5 ed. Englewood Cliffs, NJ: Prentice-Hall, 1987.

[26] K. J. Åström and B. Wittenmark, *Computer-controlled systems: Theory and Design*, 2 ed. Englewood Cliffs, NJ: Prentice-Hall, 1990.

[27] S. H. Hong, "Scheduling algorithm of data sampling times in the integrated communication and control systems," *IEEE Transactions on Control Systems Technology*, vol. 3, no. 2, pp. 225-230, 1995.

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

[29] L.-W. Liou and A. Ray, "Integrated communication and control systems: Part III - Nonidentical sensor an controller sampling," *Journal of Dynamic Systems, Measurement, and Control*, vol. 112, pp. 357-364, 1990.

[30] A. Ray and Y. Halevi, "Intergrated communication and control systems: Part II - Design considerations," *Journal of Dynamic Systems, Measurement, and Control*, vol. 110, pp. 374-381, 1988.

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

[32] R. Luck and A. Ray, "Experimental Verification of a delay compensation algorithm for integrated communication and control systems," *International Journal of Control*, vol. 59, no. 6, pp. 1357-1372, 1994.

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

[34] G. C. Walsh, O. Beldiman, and L. Bushnell, "Asymptotic behavior of networked control systems," IEEE Control Applications, Kohala Coast-Island, Hawaii, 1999, pp. 1448-1453.

[35] G. C. Walsh, O. Beldiman, and L. Bushnell, "Error encoding algorithms for networked control systems," in IEEE Decision and Control, Phoenix, AZ, 1999, pp. 4933-4938.

[36] S. H. Hong and W.-H. Kim, "Bandwidth allocation scheme in CAN protocol," *IEE Proceeding-Control Theory and Applications*, vol. 147, no. 1, pp. 37-44, 2000.

[37] F. Göktas, "Distributed control of systems over communication networks," *Ph.D. dissertation*, University of Pennsylvania, Philadelphia, 2000.

[38] N. B. Almutairi, M.-Y. Chow, and Y. Tipsuwan, "Network-based controlled DC motor with fuzzy compensation," in IEEE IECON 2001, Denver, CO, 2001, pp. 1844-1849.

[39] L. A. Zadeh, "Outline of a new approach to the analysis complex systems and decision processes," *IEEE Transactions on Systems, Man, and Cybernatics*, vol. SMC-3, pp. 28-44, 1973.

[40] B. Li and K. Nahrstedt, "A control-based middleware framework for quality-of-service adaptations," *IEEE Journal on Selected Areas in Communications*, vol. 17, no. 9, pp. 1632-1650, 1999.

[41] T. F. Abdelzaher, E. M. Atkins, and K. G. Shin, "QoS negotiation in real-time systems and its application to automated flight control," *IEEE Transactions on Computers*, vol. 49, no. 11, pp. 1170-1183, 2000.

C H A P T E R I I I

GAIN ADAPTATION OF NETWORKED DC MOTOR CONTROLLERS BASED ON QOS VARIATIONS

Mo-Yuen Chow
chow@eos.ncsu.edu

Yodyium Tipsuwan
ytipsuw@unity.ncsu.edu

Advanced Diagnosis And Control Lab
Department of Electrical and Computer Engineering
North Carolina State University, Raleigh NC 27606, USA
Tel: (919) 515-7360, Fax: (919) 515-5108

This chapter is published in IEEE Transactions on Industrial Electronics, vol. 50, no. 5, Oct. 2003.

GAIN ADAPTATION OF NETWORKED DC MOTOR CONTROLLERS BASED ON QOS VARIATIONS

Abstract—Connecting a complex control system with various sensors, actuators, and controllers as a networked control system by a shared data network can effectively reduce complicated wiring connections. This system is also easy to install and maintain. The recent trend is to use networked control systems for time-sensitive applications, such as remote DC motor actuation control. The performance of a networked control system can be improved if the network can guarantee QoS (Quality-of-Service). Due to time-varying network traffic demands and disturbances, QoS requirements provided by a network may change. In this case, a network has to reallocate its resources and may not be able to provide QoS requirements to a networked control application as needed. Therefore, the application may have to gracefully degrade its performance and perform the task as best as possible with the provided network QoS. This paper proposes a novel approach for networked DC motor control systems using controller gain adaptation to compensate for the changes in QoS requirements. Numerical and experimental simulations, and prototyping, are presented to demonstrate the feasibility of the proposed adaptation scheme to handle network QoS variation in a control loop. The effective results show the promising future of the use of gain adaptation in networked control applications.

Keywords—communication networks, control systems, adaptive control, real time systems, distributed control, decentralized control, DC motors, stability.

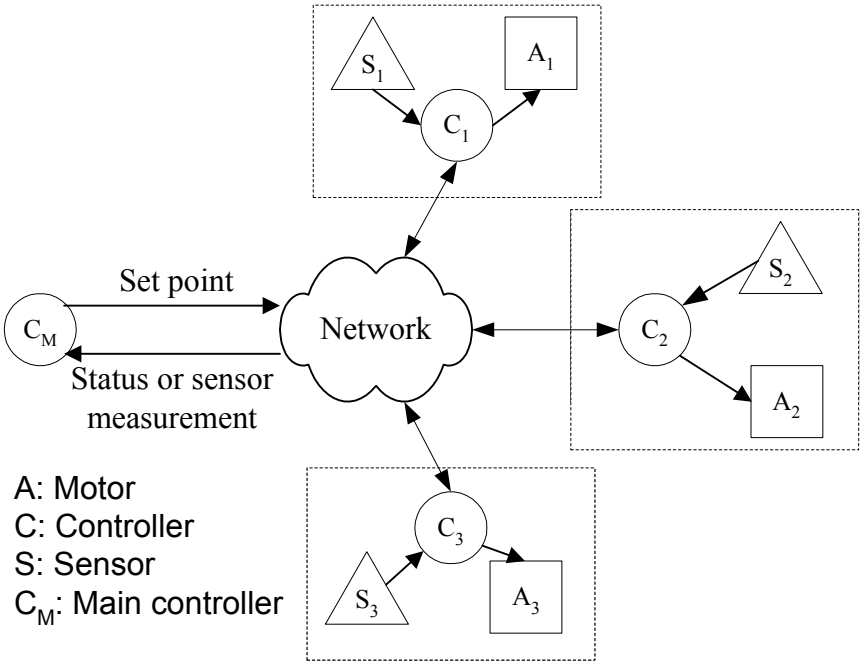
I. Introduction

DC motors have been widely utilized in many industrial applications and have a large interest in the industrial electronics community for control applications. Thus, a DC motor

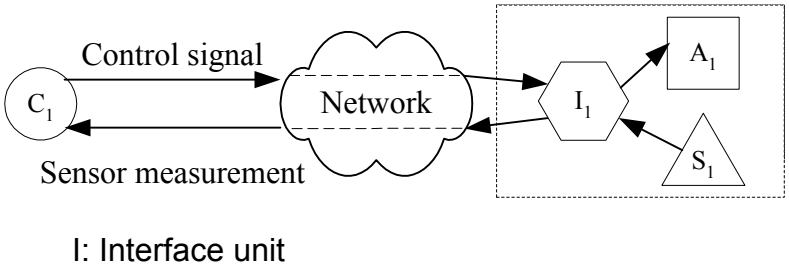
system will be used to illustrate the gain adaptation in a networked control system through out this paper. In an application composed of a small number of DC motors, such as a robotic manipulator, a conveyor belt, or a CNC milling machine, DC motors can be connected to controllers, drivers and sensors by directly wiring these devices together to perform closed-loop control without much complication. Nevertheless, direct wiring becomes more complicated and is not cost-effective for installation and maintenance of a large number of DC motors in large-scale or complex systems such as automobiles, aircrafts, and manufacturing plants. The wiring can be organized systematically by applying a shared data network instead of hardwired connections. Moreover, this network-based wiring provides much more modularity, remote-control capability, and ease in diagnosis for the control systems.

Conceptually, there are two approaches to utilize a data network [1, 2] for control of DC motors. In the first approach, each DC motor has its own controller. The controller and the DC motor are physically located close to each other. This controller receives a set point remotely from the main controller through the network, and then uses the given set point to perform closed-loop control locally as shown in Fig. 1 (a). A status or a sensor measurement of the DC motor is sent back to the main controller via the network. This approach provides modularity for each DC motor system. Each system is easy to be reconfigured. However, poor interaction between the main controller and each DC motor controller is a major drawback of this approach.

On the other hand, as shown in Fig. 1 (b), the second approach uses a network as a medium to directly transfer control signals and sensor measurements between a DC motor and a controller. Closed-loop control of the DC motor is communicated over the network. Each DC motor in this case is attached to a simple interface unit (middleware). This unit converts a data frame from the controller to an actual control signal and the sensor measurement to a data frame for sending back to the controller. In fact, this interface can also be thought of as a



(a)



(b)

Fig. 1. Control system configurations using a shared data network: (a) Hierarchical structure. (b) Direct structure.

simple remote controller. Systems formulated by this approach are so-called network-based or networked control system [3, 4], which can provide better interaction and higher flexibility for controlling DC motors.

Several standard industrial networks such as CAN, Profibus, etc.[5-7], have been widely used for networked control systems for years. Mostly, these networks have deterministic delays and enough bandwidth to perform networked closed-loop control using available control techniques without causing significant performance degradation or instability. However, investment on expensive network devices of these protocols is a problematic factor in today's competitive market environment.

Because of the affordability, simplicity, rapid development, and widespread usage, general-purpose networks such as Ethernet have been studied and suggested as alternatives for networked control applications [8]. Moreover, their connectivity to the Internet using Internet Protocol (IP) or Asynchronous Transfer Mode (ATM), can provide great benefits for remote access, control, and monitoring [9, 10] in industrial electronics [11, 12] and factory automation [13] applications. Applying networked control on these networks require more sophisticated control algorithms to handle random network delay problems. Various control techniques and stability criteria have been proposed to solve the delay problems. For examples, Luck and Ray suggested using buffers to reduce the variation of network delays [14]; Chan and Özgüner also used buffers [15], but they included some information about the amount of data in a queue to improve the results; Nilsson and Wittenmark formulated the effects of network delays as an LQG (Linear-Quadratic-Gaussian) problem and applied optimal control to handle the delays [16]; Walsh, et. al. applied non-linear control and perturbation theory to treat a network delay as a vanishing perturbation [17]; Hong introduced sampling time scheduling methods to obtain a large sampling time such that a networked control system remains stable [18]. Nevertheless, these algorithms require many assumptions, such as no communication error, which may not be feasible in real-world applications.

The networked control system performance does not only depend on the control algorithm used, but also on the network conditions. Several network conditions such as bandwidth, end-to-end delay, and packet loss rate are major impacts on networked control systems. The overall control performance can be improved if QoS (Quality-of-Services) requirements of these conditions can be provided. QoS can be viewed as bounds and limits of an end-to-end network application requirement on network conditions. The concepts of QoS have led to the development of many protocols such as ATM, RSVP, IP V6, and MPLS, which can provide even better platforms for networked control applications.

Due to changes in network user demands to or disturbances in network environments, such as the loss of a link, the availability of network resources may change unexpectedly. Therefore, the end-to-end control devices may have to renegotiate with the network resource counterparts for reallocation. If the QoS requirements cannot be provided as needed, the networked system may need to lower its performance and use the available QoS requirements to perform a control task as best as it can. In many cases, when anomalies happen, the first issue is to stabilize the closed-loop control system (with a network in the loop in this case), then aim for the best control performance under the given QoS conditions. Recent works on this issue have been extended to real-time QoS negotiation and graceful performance degradation [19], and the application of control theory on middleware QoS resource management and scheduling [20, 21].

This paper proposes a novel approach for networked DC motor control by using controller gain adaptation to compensate for the changes in QoS requirements through a real-time QoS negotiation process. In this paper, the gains of a networked PI controller are the main focus. These gains are adapted with respect to the given QoS conditions. A numerical simulation is set up to investigate the network delay issues and the performance of the adaptation scheme under fully control environments. We then develop a hardware prototype to verify the adaptation scheme experimentally. The successful results obtained from both numerical and

experimental prototyping show the promising future of gain adaptation in networked control technologies for industrial applications and factory automation.

II. Problem Formulation

A networked control system can be divided into three parts: (1) the remote systems and remote controllers, (2) the central controller, and (3) the data network. A general block diagram of the networked control system under investigation is shown in Fig. 2. Each component is described in the following sections.

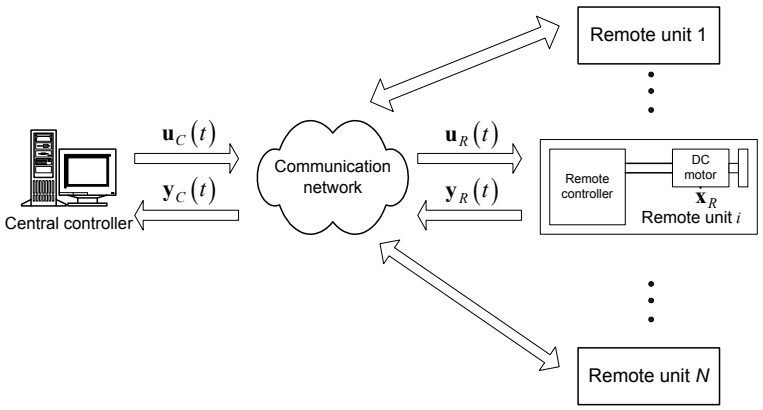


Fig. 2. An overall real-time networked control system.

A. Remote system and remote controller

Each distributed remote controller can be assumed to have enough computing power to do relatively simple pre-programmed control – such as converting the control signal received from the central controller via the network into a PWM signal to drive a motor (the remote process). Each remote controller can send local measurements, such as motor current, speed, temperature, and local environment information, back to the central controller via the network. Each remote process has its own system dynamics that can be described by the state-space description [11] shown in Eq. (1), where the state vector $\mathbf{x}_R = \left[x_{R1}, \cdots, x_{Rn}\right]^{\mathrm{T}} \in X^n$, the state space; $\mathbf{p}_R = \left[p_{R1}, \cdots, p_{Rq}\right]^{\mathrm{T}} \in \Re^q$ are the system parameters; the input vector

$\mathbf{u}_R = \left[u_{R1}, \cdots, u_{Rr}\right]^{\mathrm{T}} \in U^r$, the input space; $t \in \Re^+$ is the time parameter; and $\mathbf{f}_R \in \Re^n$ is the state transfer function of the remote plant:

$$\dot{\mathbf{x}}_R = \mathbf{f}_R\left(\mathbf{x}_R, \mathbf{p}_R, \mathbf{u}_R, t\right). \tag{1}$$

Depending on the design of the networked control system, the remote controller, C_R, performs a certain task, such as regulating the performance of the plant P_R, as described by Eq. (2):

$$\mathbf{u}_R = \mathbf{g}_R\left(\boldsymbol{\alpha}_R, \cdot\right), \tag{2}$$

where $\boldsymbol{\alpha}_R = \left[\alpha_{R1}, \cdots, \alpha_{Ra}\right]^{\mathrm{T}}$ is the adjustable controller parameter vector and $(\cdot)$ represents other appropriate information. The combination of the remote controller and the remote plant can be viewed as a remote system, S_R, which has its own system dynamics that can be described by a set of differential equations:

$$\dot{\mathbf{x}}_R = \mathbf{f}_R\left(\mathbf{x}_R, \mathbf{p}_R, \mathbf{g}_R\left(\boldsymbol{\alpha}_R, \cdot\right), t\right). \tag{3}$$

We denote the quality of service provided by the network as a function of time, as $QoS\left(t\right)$, which can be varied.

B. Central controller

The central controller can be a highly sophisticated controller that requires lots of computing power and memory, and is therefore not suitable to be installed at the remote site. The central controller is powerful and can provide advanced real-time control laws to all remote units, including fault diagnosis and accommodation control, network traffic condition monitoring and adaptation to the QoS provided by the network. The central controller will provide the control signal $\mathbf{u}_C\left(t\right)$ to each of the remote systems. The control signal is assumed to be the result of optimizing a cost function *C*, described as:

$$\min_{\mathbf{u}_C(t)} C\left(\mathbf{x}_R, \cdot\right). \tag{4}$$

Let $z^{-\tau}$ be a time delay operator, and let $QoS(t)$ be the current QoS provided by the network. We define:

$$\mathbf{u}_R(t) = \mathbf{u}_C\left(z^{-\tau_R}, QoS(t)\right), \tag{5}$$

$$\mathbf{y}_C(t) = \mathbf{y}_R\left(z^{-\tau_C}, QoS(t)\right), \tag{6}$$

where τ_R is the time delay in transmitting a signal from the central site to the remote site, and τ_C is the time delay in transmitting a signal from the remote site to the central site. The time delays $z^{-\tau}$ and $QoS(t)$ are functions of network variables such as the type and number of signals to be transmitted, the network traffic congestion condition, the network throughput, the network protocol used, the network management/policy used, and the controller processing time.

III. Case Study

In order to focus our discussion on how the QoS conditions can affect the closed-loop control performance, and how the central control can adapt its control parameters to compensate for the QoS variation and deficiency to provide the *best* control performance with a given QoS requirement, we select a simple central controller and a simple remote system (end-user to end-user) connected via a shared network with different QoS levels as a case study. With the simple networked control system, the effects of the QoS conditions are more obvious for illustration than a complicated system.

A. Remote system description

This section briefly describes the dynamics of our remote process: a dc motor driving a load. The load can be a robot arm or an unmanned electric vehicle, for instance. The loop equation for the electrical circuit is [11,12]:

$$u(t) = e_a = L\frac{di_a}{dt} + Ri_a + e_b\,. \tag{7}$$

The mechanical torque balance based on Newton's law is:

$$J\frac{d\omega}{dt}+B\omega+T_l=T_e=Ki_a\,, \tag{8}$$

where $u=e_a$ is the armature winding input voltage; $e_b=K_b\omega$ is the back-EMF voltage; L is the armature winding inductance; i_a is the armature winding current; R is the armature winding resistance; J is the system moment of inertia; B is the system damping coefficient; K and K_b are the torque constant and the back-EMF constant, respectively; T_l is the load torque; and ω is the rotor angular speed.

By letting $x_1=i_a$ and $x_2=\omega$, the electro-mechanical dynamics of the dc motor can be described by the following state-space description:

$$\dot{x}_1=-\frac{R}{L}x_1-\frac{K_b}{L}x_2+\frac{1}{L}u\,, \tag{9}$$

$$\dot{x}_2=\frac{K}{J}x_1-\frac{B}{J}x_2-\frac{1}{J}T_l\,. \tag{10}$$

The parameters of the motor used in this paper are shown in Table 1.

Table 1. DC motor parameters.

J	Inertia	42.6 e-6 Kg-m^2
L	Inductance	170 e-3 H
R	Resistance	4.67 Ω
B	Damping coefficient	47.3 e-6 N-m-sec/rad
K	Torque Constant	14.7 e-3 N-m/A
K_B	Back-EMF Constant	14.7 e-3 V-sec/rad

To keep the illustration simple, we assume that the remote controller is used to simply convert the control voltage data sent from the central controller into a PWM signal to drive the dc motor. The remote control value u_R can be mathematically expressed as:

$$u_R(t)=u_C(t-\tau_R)\,, \tag{11}$$

where τ_R is the time delay to transmit the control signal u_C from the central controller to the remote controller. The remote controller also sends the monitored signals $y_R(t)$ of the remote system back to the central controller, $y_C(t)$, and these two signals are related as:

$$y_C(t) = y_R(t - \tau_C), \tag{12}$$

where τ_C is the time delay to transmit the measured signal from the remote controller to the central controller.

In fact, there are also processing delays, denoted as τ_{PC} and τ_{PR}, at the central and remote controllers, respectively. However, both τ_{PC} and τ_{PR} could be approximated as small constants, or even neglected because these delays are usually small compared to τ_C and τ_R.

B. Central control system description

The central controller will monitor the QoS conditions of each remote system link and provide appropriate control signals to each remote system. In this paper, the central controller uses a PI control algorithm to compute the control to the remote system for step tracking, based on the monitored system signals sent from the remote system via the network link. The PI controller used has the form [11,12]:

$$u(t) = K_p e(t) + K_I \int_0^t e(s) ds, \tag{13}$$

where K_P is the proportional gain; K_I is the integral gain; $r(t)$ is the reference signal for the system to track; $y(t)$ is the system output; and $e(t) = r(t) - y(t)$ is the error function. In our case, $y = \omega$ is the motor speed, and $u(t)$ is the input voltage to the motor system.

C. Network link QoS conditions

There are different ways to define Quality-of-Service for end-to-end (from the central control to a specific remote system) user conditions. Two of the most popular QoS measures, which are used in this paper, are defined as follows:

- QoS_1 denotes the point-to-point (from the central controller to the remote controller) network throughput; it is used to indicate how fast the signal can be sampled and sent as a packet through the network.

- QoS_2 denotes the point-to-point maximal delay bound of the largest packet; it is used to indicate how long of a packet is expected to be delivered from the central controller to the remote controller.

One factor of interest in this paper is the sampling time *h*. In this paper, we set the periodic sampling time *h* with respect to the following criterion:

$$h > \tau_C + \tau_{PC} + \tau_R + \tau_{PR} . \tag{14}$$

This criterion is used to guarantee that packet transfers between the central and remote controllers can be processed in a sampling period. QoS_1 and QoS_2 are significant factors to determine *h* with knowledge of packet sizes used in order to satisfy the criterion. With given QoS_1 and QoS_2, τ_C and τ_R can be estimated by computing delays such as transmission delay and propagation delay. If this criterion cannot be satisfied, the central controller will not have the measured signals to process. This situation is not discussed in this paper.

IV. Simulation Setups

Both numerical and experimental simulations are set up to illustrate the effects of QoS variations on networked control, as explained in the following sections.

A. Numerical simulation

In the numerical simulation scenario, the networked DC motor control system is simulated using MATLAB/ SIMULINK under fully controlled environments. The motor equations in Eq. (9) and Eq. (10) are used as the main model, and it is controlled by the PI controller in Eq. (13) with the insertions of network delays according to Eq. (11) and Eq. (12). These delays can be varied according to different effects of interests. The numerical system setup is illustrated in Fig. 3.

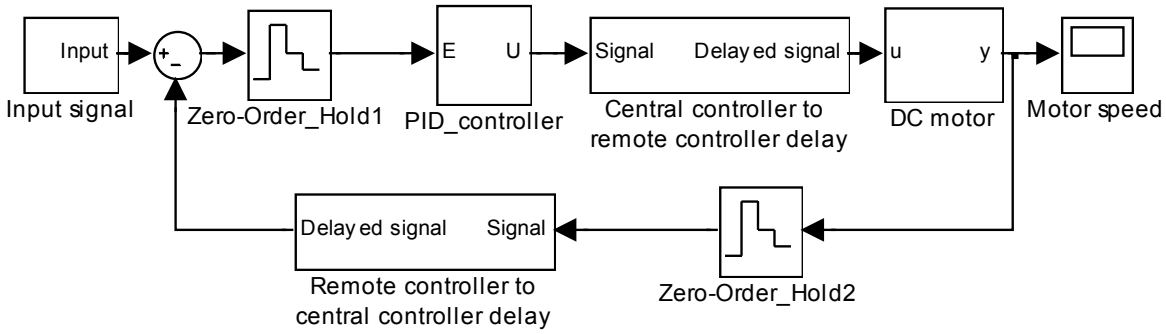


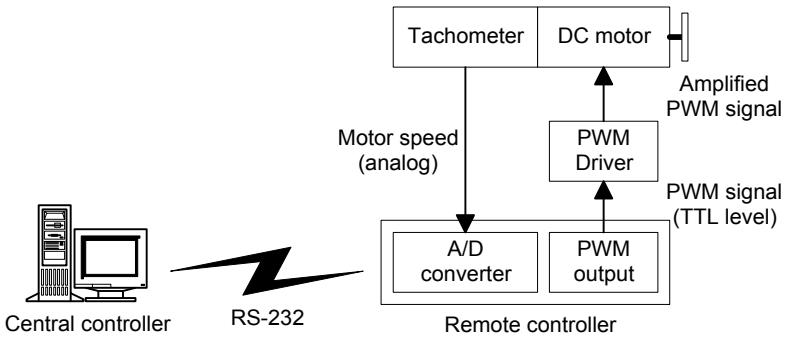
Fig. 3. Block diagram of the networked DC motor control system in the numerical simulation.

B. Experimental simulation

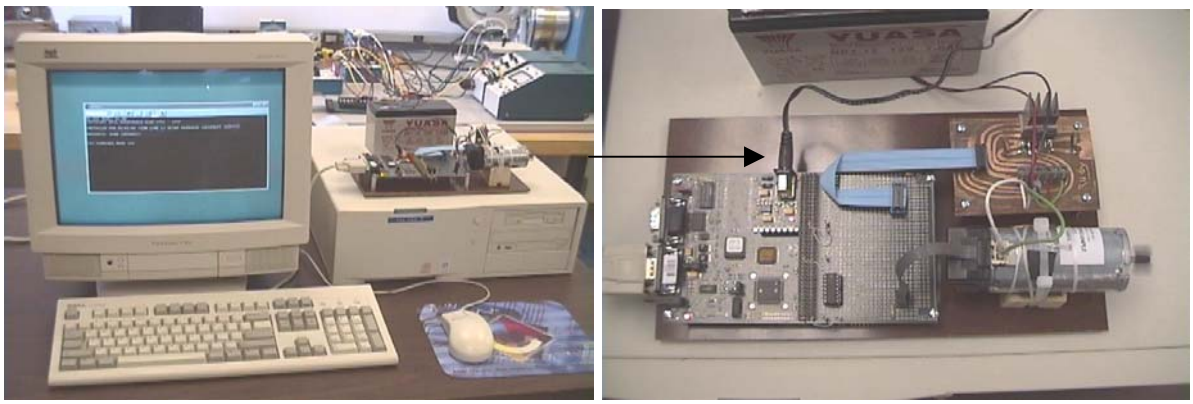
A simple peer-to-peer networked DC motor control system is set up to experimentally demonstrate the effects of interests. The block diagram of the networked system is shown Fig. 4 (a), and the actual system setup is depicted in Fig. 4 (b).

The central controller is implemented on a PC running RT-Linux, while the remote controller is implemented on a Siemens C-515C microcontroller board. Mainly, the remote controller is used for two tasks. The first task is to convert the instantaneous motor speed measurement to a packet and send it to the central controller. This packet is then used at the central controller to compute the input voltage by the PI control algorithm. The second task is to convert the input voltage in a packet form from the central controller to a duty cycle of a PWM signal, and then send it to the motor.

The network between the central and remote controllers in the setup is simulated by an RS-232 serial link. The baudrate of the serial link is assigned as QoS_1. Three packet formats used in this case are shown in Fig. 5.



(a)



(b)

Fig. 4. (a) Block diagram of a peer-to-peer networked DC motor control system. (b) Actual networked DC motor control system setup.

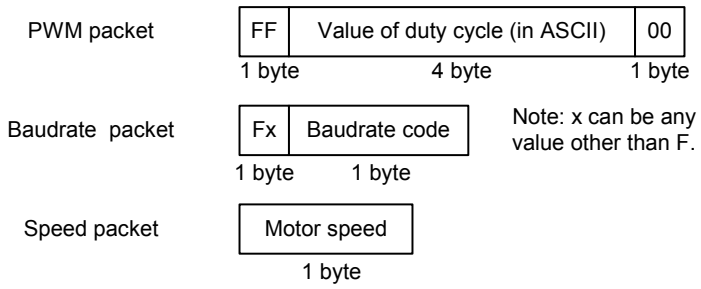


Fig. 5. Packet formats.

A PWM packet is used for sending the control voltage signal as the duty cycle of the PWM signal from the central controller to the remote controller. After the remote controller processes the PWM packet, it returns the motor speed using the speed packet format. The A/D converter used for sampling and quantization of the motor speed in the experimental setup has the resolution of 8 bits. Therefore, one byte is enough to represent the motor speed in the experimental simulation. On the other hand, the central controller sends the baudrate packet to the remote controller when the central and remote controllers are negotiating for a new baudrate used. Therefore, the maximal packet size with respect to our custom protocol is set to be 6 bytes, and QoS_2 in this case is determined by the maximal delay bound of the 6-byte packet transmission.

C. Simulation results and discussions

- *System response with different PI gains*

In the networked control system under consideration, both the control gains and the network QoS can significantly affect the closed-loop system performance. Fig. 6 shows typical closed-loop responses of a networked control system from the numerical simulation, subject to tracking a 200 rad/s reference signal with respect to different PI gains, given a sampling period of 2 ms and without time delays.

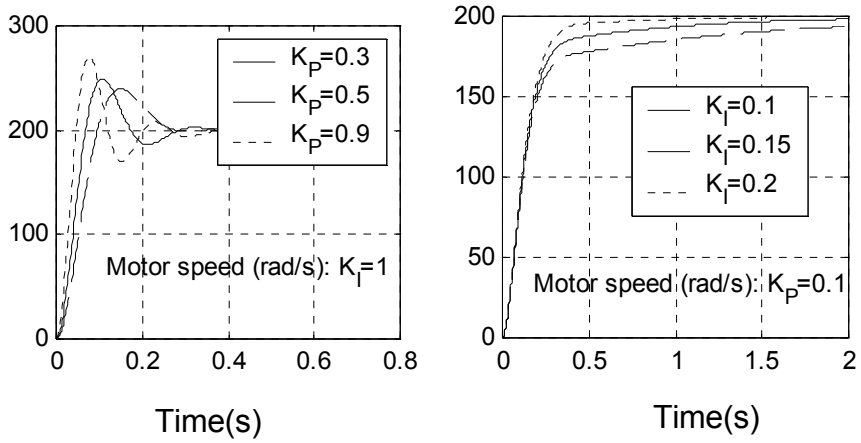


Fig. 6. Networked motor control performance with a sampling time of 2 ms and different PI gain values and without time delays.

- *System response with different QoS*

The closed-loop system will respond differently given a controller gain and different network QoS conditions. To demonstrate this case, we assume:

$$QoS_2 = Maximal\ packet\ size(bits)/QoS_1 . \tag{15}$$

Also, the sum of τ_{PC} and τ_{PR} is estimated to be 0.1 ms based on the equipment settings. Applying these estimations in conjunction with Eq. (14), the sampling time is estimated by:

$$h > 2QoS_2 + 0.1 \text{ ms}. \tag{16}$$

With the largest packet size of 6 bytes, possible valid sampling times for $QoS_1 = \{4800,$ 9600, 19200 and 38400$\}$ bps can be set to $\{24,$ 12, 6, and 3$\}$ ms, respectively. These settings are applied in the numerical simulation using a 6-byte packet size for the input voltage transmission and a single-byte size for the measured signal transmission. The network delays are set to be random, ranging from 90% to 100% of the maximal delays with respect to the packet size used in different settings. This delay variation is generated using a uniform distribution. Fig. 7 clearly indicates that sampling times and network delays, which are determined from different QoS, can significantly influence the closed-loop control system performance by increasing the maximum overshoots and the settling times in the step responses, sometimes to the point of destabilizing the system.

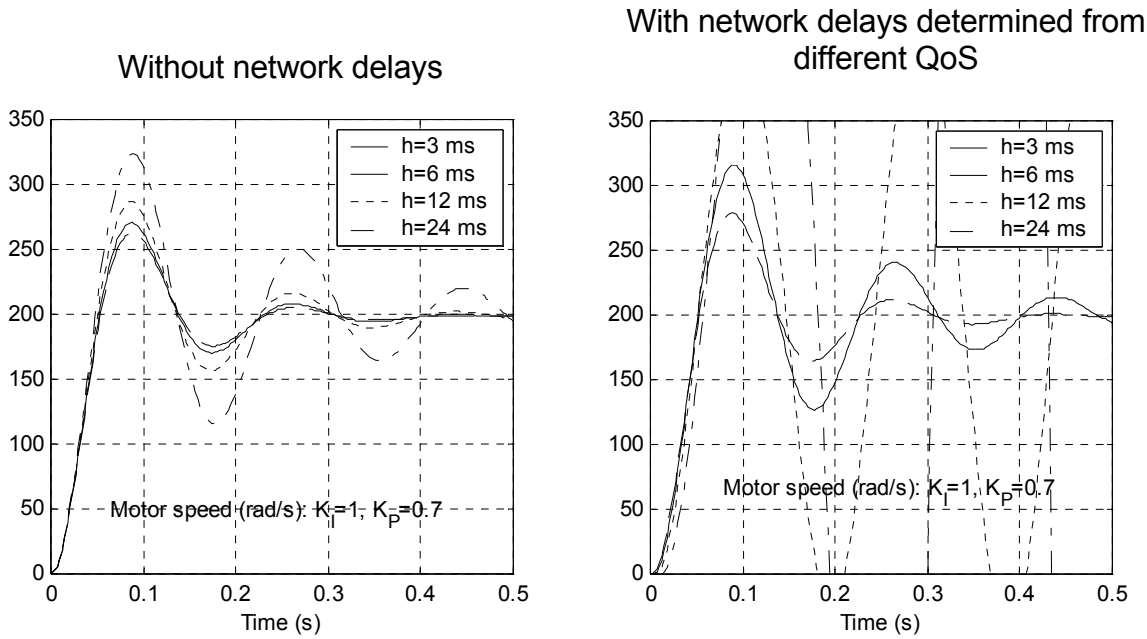


Fig. 7. Networked motor control performance given PI gain values under different QoS conditions.

D. Performance measure and gain adaptation

In many cases, an end-user cannot control the network QoS, while the end-user can monitor the network QoS via appropriate middleware. Thus, the end-user can then adapt its controller gains to provide the best closed-loop control performance possible while the network QoS is varying or deteriorating. In this paper, we use a cost function approach to proactively adapt the PI gains in response to the change in network QoS. Different performance measures can be used to construct the cost function. In this paper, we define the cost function as:

$$C = \frac{1}{N}\sum_{k=1}^{N}\left|r(k) - y(k)\right|, \tag{17}$$

where N is the appropriate time index such that the tracking has arrived at the steady state, $r(k)$ and $y(k)$ are the measurements on r and y at time k. The cost function C considers how different the transient and steady-state system response is from the reference signal. Fig. 8

shows a typical cost for different PI gains under different QoS conditions, from the numerical simulation.

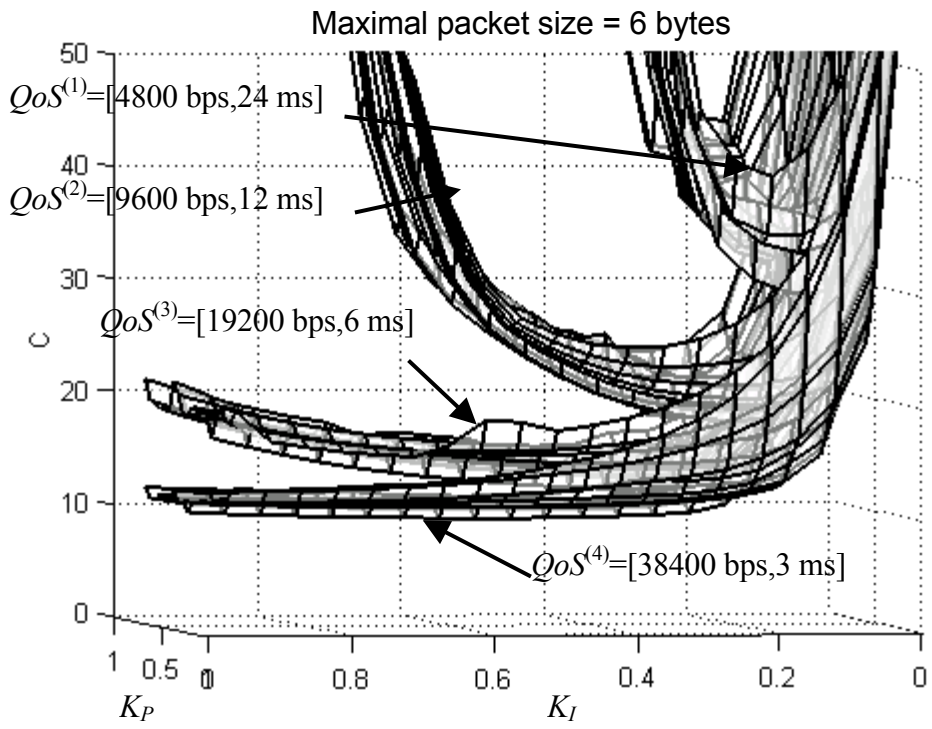


Fig. 8. Cost with respect to different PI gains and QoS.

Each arrow points to a cost surface with respect to each QoS setting. The shading in Fig.8 indicates the overlapped parts of the surfaces. The lighter shading of the surface lines implies more parts of surfaces overlap the lines. The figure clearly indicates that given a QoS and the maximal packet size, there are certain values of K_P and K_I that can provide the optimal cost, hence the optimal system performance. Different types of optimization techniques can be used to find the optimal values of K_P and K_I. For instance, we can construct the cost function C with respect to the influential factors K_P, K_I, QoS_1 and QoS_2. Note that if there is a closed-form relationship among network QoS conditions, controller gains, and the cost function, the

problem can be formulated as an optimal control problem such as the LQG problem. However, such the relationship is usually difficult to find and may not be able to obtain analytically.

Analytical searching techniques, such as steepest descent, may take a long time to find the optimal solution and may not be suitable for real-time applications. Other searching techniques, such as Newton Raphson, may provide significant faster earching results, yet it can be sensitive to the cost function model error. In this paper, we use a look-up table technique to perform the searching and to show the feasibility of the proposed proactive gain adaptation algorithm with respect to network QoS. The cost values are stored in a table form with respect to QoS_1, QoS_2, K_P, and K_I. Using the monitored network QoS, we search for the best K_P and K_I from the stored table that can provide the optimal cost. Linear interpolation is used when the exact entry cannot be found from the table with the given inputs. For other high dimensional problems, the size of the lookup table may be very large due to the curse of dimensionality. Computational intelligence approaches such as fuzzy logic and neural networks can be applied to construct the mapping from the controller gains to the network QoS conditions by using the lookup table as a priori knowledge.

In order to show the effects of network QoS variations on the networked control system, different network QoS and control gains with and without adaptation were simulated and analyzed in both numerical and experimental simulations.

- *Numerical Simulation*

Fig. 9 shows a typical networked control response from the numerical simulation when the network QoS deteriorates from $QoS^{(1)} = \left[QoS_1^{(1)}, \quad QoS_2^{(1)} \right] = \left[38400 \text{ bps}, 3 \text{ ms} \right]$ to $QoS^{(2)} = \left[19200 \text{ bps}, \ 6 \text{ ms} \right]$, and to $QoS^{(3)} = \left[9600 \text{ bps}, 12 \text{ ms} \right]$. Step references are issued for the controller to track when the network changes QoS so that we can obviously see how the network QoS variation affects the networked control response with and without gain adaptation. The solid line represents the reference signal for the networked controller to track; the dotted line represents the closed-loop system response *without* gain adaptation; and the

dashed line represents the closed-loop system response *with* gain adaptation to compensate for the QoS deterioration. The numerical simulation results clearly show the advantages and improved performance of using gain adaptation to maintain the system performance while the network QoS is varying or deteriorating.

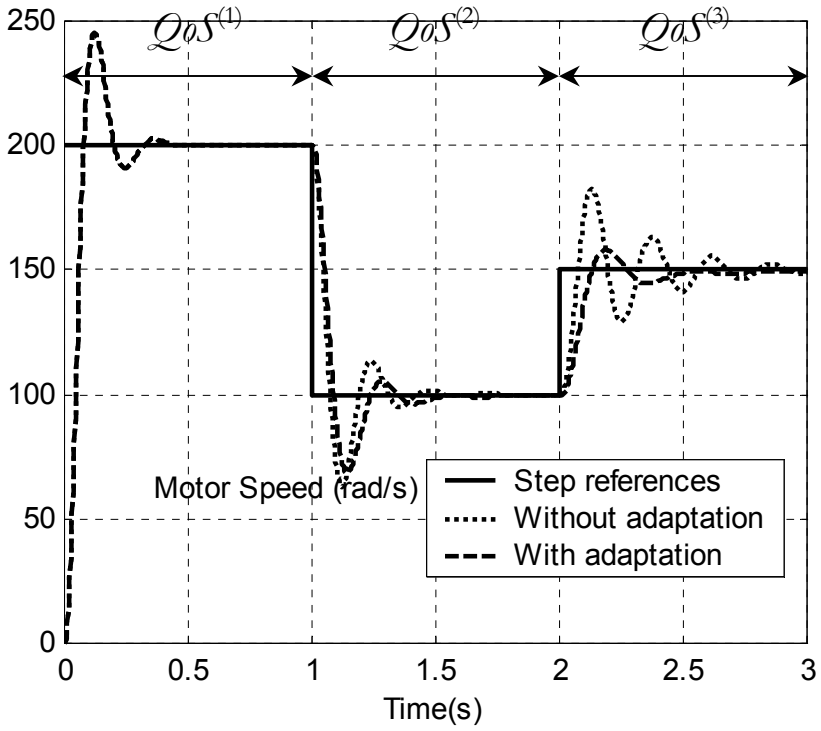


Fig. 9. A typical networked control system performance from the numerical setup with and without gain adaptation when network QoS deteriorates.

- *Experimental Simulation*

A networked control response from the experimental simulation is depicted in Fig. 10. Due to the processing delays at the central and remote controllers, the network QoS in this case is modified to deteriorate from $QoS^{(1)} = [38400 \text{ bps}, 5 \text{ ms}]$ to $QoS^{(2)} = [19200 \text{ bps}, 8 \text{ ms}]$ instead, with respect to the new step references. The solid line represents the reference signal for the networked controller to track; the cross-sign line represents the closed-loop system

response *without* gain adaptation; and the plus-sign line represents the closed-loop system response *with* gain adaptation to compensate for the QoS deterioration. The experimental results explicitly support the advantages and performance of using gain adaptation.

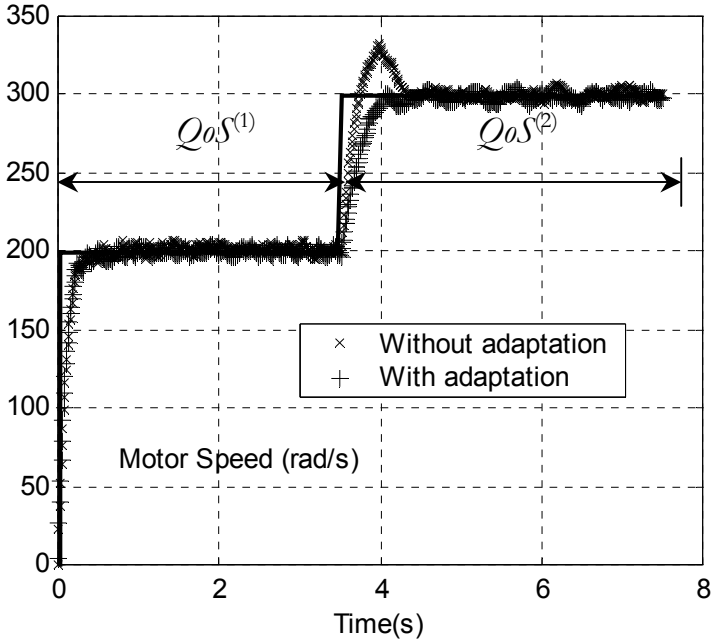


Fig. 10. A typical networked control system performance from the experimental setup with and without gain adaptation when network QoS deteriorates.

V. Conclusion

Networks and their applications are promising for wide deployment of real-time high performance networked control in industrial applications. However, one of the major concerns is the QoS that can be provided by the network and how it affects the performance of the networked control system. This paper has described, formulated, and shown promising results on the use of a proactive gain adaptation in a networked DC motor control system by the end-user in response to network QoS variations and deteriorations. Both the numerical and experimental results have shown that this is a promising and feasible approach in performing network QoS negotiation and graceful performance degradation and in maintaining networked

control system availability. The analysis on more advanced issues such as packet losses can improve and strengthen the gain adaptation approach for more practical uses in the future.

Acknowledgement

The authors would like to thank the Royal Thai Government for partially supporting this study.

References

[1] J. W. Overstreet and A. Tzes, "An Internet-based real-time control engineering laboratory," *IEEE Control Systems Magazine*, vol. 19, no. 5, pp. 19-34, 1999.

[2] G. V. Kondraske, R. A. Volz, D. H. Johnson, D. Tesar, J. C. Trinkle, and C. R. Price, "Network-based infrastructure for distributed remote operations and robotics research," *IEEE Transactions on Robotics and Automation*, vol. 9, no. 5, pp. 702-704, 1993.

[3] G. C. Walsh, O. Beldiman, and L. Bushnell, "Error encoding algorithms for networked control systems," in IEEE Decision and Control, Phoenix, AZ, 1999, pp. 4933-4938.

[4] Y. H. Kim, H. S. Park, and W. H. Kwon, "Stability and a scheduling method for network-based control systems," in IEEE IECON 96, Taipei, Taiwan, 1996, pp. 934-939.

[5] J. M. Lee, S. Lee, M. H. Lee, and K. S. Yoon, "Integrated wiring system for construction equipment," *IEEE/ASME Transactions on Mechatronics*, vol. 4, no. 2, pp. 187-195, 1999.

[6] P. Sink, "A comprehensive guide to industrial networks," *Sensors*, vol. 18, no. 6, pp. 28-43, 2001.

[7] W. T. Strayer and A. C. Weaver, "Performance measurement of data transfer services in MAP," *IEEE Network*, vol. 2, no. 3, pp. 75-81, 1988.

[8] G. Kaplan, "Ethernet's winning ways," *IEEE Spectrum*, vol. 38, no. 1, pp. 113-115, 2001.

[9] A. C. Weaver, J. Luo, and X. Zhang, "Monitoring and control using the Internet and Java," in IEEE IECON 99, San Jose, CA, 1999, pp. 1152-1158.

[10] A. Malinowski, T. Konetski, B. Davis, and D. Schertz, "Web-controlled robotic manipulator using Java and client-server architecture," IEEE IECON 99, San Jose, CA, 1999, pp. 827-830.

[11] Y. Tipsuwan and M.-Y. Chow, "Fuzzy logic microcontroller implementation for DC motor speed control," in IEEE IECON 99, San Jose, CA, 1999, pp. 1271-1276.

[12] J. T. Teeter, M.-Y. Chow, and J. J. Brickley, Jr., "A novel fuzzy friction compensation approach to improve the performance of a DC motor control system," *IEEE Transactions on Industrial Electronics*, vol. 43, no. 1, pp. 113-120, 1996.

[13] R. Zurawski, "Verifying correctness of interfaces of design models of manufacturing systems using functional abstractions," *IEEE Transactions on Industrial Electronics*, vol. 44, no. 3, pp. 307-320, 1997.

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

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

[16] J. Nilsson, B. Bernhardsson, and B. Wittenmark, "Stochastic analysis and control of real-time systems with random time delays," *Automatica*, vol. 34, no. 1, pp. 57-64, 1998.

[17] G. C. Walsh, O. Beldiman, and L. Bushnell, "Asymptotic behavior of networked control systems," in IEEE Control Applications, Kohala Coast, HI, 1999, pp. 1448-1453.

[18] S. H. Hong, "Scheduling algorithm of data sampling times in the integrated communication and control systems," *IEEE Transactions on Control Systems Technology*, vol. 3, no. 2, pp. 225-230, 1995.

[19] T. F. Abdelzaher, E. M. Atkins, and K. G. Shin, "QoS negotiation in real-time systems and its application to automated flight control," *IEEE Transactions on Computers*, vol. 49, no. 11, pp. 1170-1183, 2000.

[20] B. Li and K. Nahrstedt, "A control-based middleware framework for quality-of-service adaptations," *IEEE Journal on Selected Areas in Communications*, vol. 17, no. 9, pp. 1632-1650, 1999.

[21] D. C. Schmidt, V. Kachroo, Y. Krishnamurthy, and F. Kuhns, "Developing next-generation distributed applications with QoS enabled DPE middleware," *IEEE Communications Magazine*, vol. 38, no. 10, pp. 112-123, 2000.

C H A P T E R I V

METHODOLOGY OF USING NETWORKED PI CONTROLLER GAIN SCHEDULING OVER IP NETWORK:

PART I – FOUNDATION

Yodyium Tipsuwan
ytipsuw@unity.ncsu.edu

Mo-Yuen Chow
chow@eos.ncsu.edu

Advanced Diagnosis And Control Lab
Department of Electrical and Computer Engineering
North Carolina State University, Raleigh NC 27606, USA
Tel: (919) 515-7360, Fax: (919) 515-5108

This chapter and chapter V were submitted for publication to an IEEE Transactions, as parts I and II.

METHODOLOGY OF USING NETWORKED PI CONTROLLER GAIN SCHEDULING OVER IP NETWORK: PART I – FOUNDATION

Abstract—The potential use of networks for real-time high performance control and automation is enormous and appealing. Replacing a widely used PI controller by a new networked controller for networked control capability can be costly and time-consuming. This paper proposes a methodology based on gain scheduling to enhance the PI controller so it can be used over IP networks with a general computer protocol like Ethernet. Part I of this paper presents the foundation of the proposed approach. Problem formulation by using a rational function to approximate constant network delays is described, and a DC motor speed control problem is used to illustrate the proposed methodology. Simulation results also indicate that the delay approximation can provide good enough accuracy with about 1% error compared to the actual delay. This paper also describes PI controller parameterization by a single parameter for gain scheduling at the controller output. Performance of the proposed approach is measured from a defined cost function. The cost function indicates performance degradation from a nominal condition when there is no network in the control loop. A searching algorithm is presented to find the optimal parameter to schedule the control gains based on the root locus analysis with the constant delay approximation. The sensitivity of the parameter used with respect to the network delay is also analyzed and illustrated. In addition, Part I of this paper shows that the optimal parameter decreases and becomes more sensitive to the delay with more performance degradation when the delay is longer. Part II of this paper will extend the foundation in Part I for practical implementation on actual IP network delays.

Keywords—Internet, networks, adaptive control, control systems, DC motors, distributed control, real time system.

I. Introduction

Computer networks and their applications have undergone major changes in the last two decades. Recently, a new and promising use of networks is in the area of high performance distributed control and automation, which is developing into an attractive market with wide applications in factory automation, teleoperation, and industrial electronics. The potential use of networks for real-time high performance control and automation is enormous and appealing. Before networks can be widely and efficiently used for real-time high performance control and automation, certain issues need to be addressed, including time delays, delay variations, and bandwidth constraints. Time delays are well known to degrade system performance and reduce the stability region. The system performance can be further aggravated with the existence of delay variations. Specialized networks such as Profibus [1] and CAN (Controller Area Network) [2] have been developed for networked control applications, in which both the network delays and bandwidth constraints are relatively easy to predict, and the delay variations are low.

A recent and advancing trend in the networked control area is to substitute specialized industrial networks with a general computer network such as Ethernet and wireless Ethernet in order to control a system remotely over the Internet or IP (Internet Protocol) network [3, 4]. A general protocol like Ethernet has several advantages due to its affordability, widespread usage, and well-developed infrastructure for Internet connection. Nevertheless, once a networked control system is connected through the Internet, the network delays induced by IP are no longer constant and can vary depending on traffic conditions. Several methodologies have been developed to handle the time-varying network delay effects, and some promising results have been reported. These control methodologies are based on different techniques such as state augmentation [5], optimal stochastic control [6], nonlinear control and perturbation theory [7],

robust control [8], buffering and prediction [9, 10], Smith predictor [11], event-based control [12], sampling time scheduling [13, 14], and fuzzy logic [15, 16]. Many of these techniques require a completely redesigned controller to handle network delays. Practical controllers that exist in many industrial applications such as a PI controller have to be replaced with a new controller. This replacement is usually time-consuming, which implies high installation cost.

An alternative choice to maintain the system performance under random IP network delays is to provide an appropriate IP network traffic condition for supporting a networked control system by applying QoS (Quality-of-Service) concepts to impose network delays, delay variations, and bandwidth constraints [17, 18]. Several QoS concepts can be used. For example, applying an IP QoS protocol [19] such as RSVP to reserve network resources [20], providing differentiate service (Diff-Serv) to prioritize packets [21], or using middleware to handle QoS negotiation and resource reservation [22, 23]. Nevertheless, these methods cannot totally guarantee IP network QoS measures likes packet loss, delay bound, or delay variation bound, since IP network does not design for perfect real-time performance. Anomalies such as changes in user demands and a loss of a link may affect the IP network traffic. Therefore, the controller may still need to adapt itself to handle these anomalies in addition to relying on the network protocol used.

The main goal of this paper is to propose a methodology that enhances the widely used PI (Proportional-Integral) controller so it can be used over IP network with a general computer protocol like Ethernet. The proposed approach is based on PI controller gain scheduling. The optimal PI controller gains are scheduled in real-time with respect to the monitored IP network traffic condition in order to maintain the best possible system performance. Therefore, changes in IP network traffic conditions are always captured by the proposed approach. Furthermore, the controller gains are not directly adapted or scheduled, but are instead adjusted by modifying the output of the PI controller. Without requiring redesign and reinstallation of a new networked control system, the proposed approach could save much cost and time for practical

uses. This paper is organized as follows. Section II provides the system description including system configuration, mathematical system formulation, and case study. Section III describes PI controller parameterization, performance measures, and a searching algorithm to find the optimal controller gains. Part I concludes in section IV. The generalization of this paper for actual IP network delays is continued in Part II.

II. System Description

A. System configuration

In this paper, we consider a distributed networked control system configuration over an IP network. An overall setup of the distributed networked control system shown in Fig. 1 is composed of:

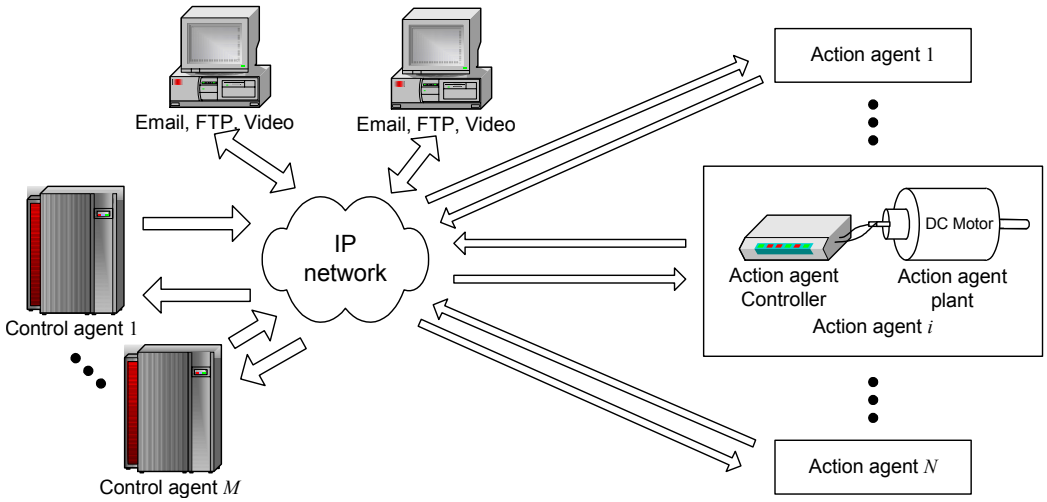


Fig. 1. An overall distributed networked control system over IP network.

- *IP network*

The IP network under consideration links all networked control devices including the control agent and action agents together by sharing the same communication media. The network can be also shared with other end-to-end applications ranging from email and FTP to

video streams. In this paper, we assume that the networked control devices use UDP (User Datagram Protocol) as the layer-4 protocol on IP network to avoid additional delays from retransmissions.

- *Control agent*

In general, the control agent is a high performance computing unit to manage operations of action agents, and also make some critical "group" decisions for some of the action agents. There can be many different control agents for different control purposes in a distributed control system. Without loss of generality, this paper uses one of the control agents as an illustration to demonstrate the proposed control technique. The control agent can handle high level overall networked system control with advanced features such as fault detection and gain scheduling on the low level control to aim for the best possible performance when anomalies happen. The control agent uses a PI controller to control each action agent in the low level. Periodically, the control agent converts the sensory signals in a packet sent across IP network from each action agent to numerical feedback data for a PI control response. The control signal from the PI controller is then sent back as a packet to each action agent via the IP network as well. We also assumed that the control agent is capable of monitoring an IP network condition such as measuring delay and delay variation. Several actual IP traffic delays and statistics will be shown in later sections.

- *Action agent*

Each action agent contains an action agent controller and an action agent plant. The action agent controller is a simple hardware unit to periodically convert the control signal in a packet from the control agent to an actual signal to drive the action agent plant. The action agent plant used as an example in this paper is a DC motor, which has many applications in industry such as for process control and robot control. The sensory output of the action agent plant such as the motor speed is also monitored and sent back to the control agent.

B. Mathematical formulation

In order to analyze how to schedule the PI controller gains on the control agent with respect to an IP network traffic condition, let us formulate the problem mathematically in a continuous-time approach by first *assuming* IP network delays are constant. A typical single networked control system is formulated as shown in Fig. 2, where $R(s)$, $U(s)$, $Y(s)$, and $E(s) = R(s) - Y(s)$ are the reference, control, output, and error signals in Laplace domain according to the reference, control, output, and error signals in time domain defined as $r(t)$, $u(t)$, $y(t)$, and $e(t)$, respectively. The other action agents can be controlled using the similar formulation.

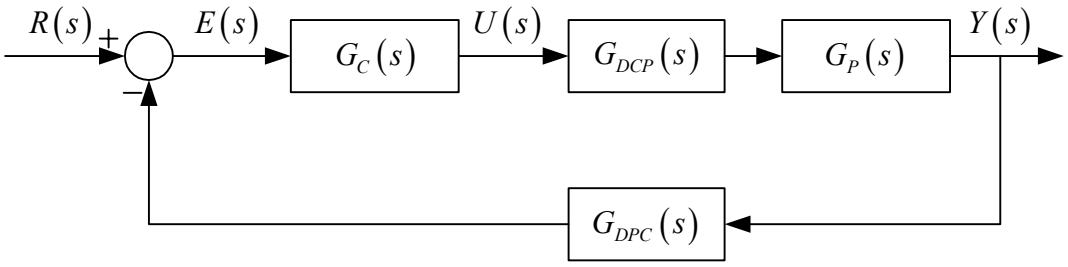


Fig. 2. A point-to-point networked control system formulation.

The action agent plant dynamics is expressed as a transfer function $G_P(s)$, where the PI controller $G_C(s)$ is described by:

$$G_C(s) = \frac{K_P\left(s + \frac{K_I}{K_P}\right)}{s} = \frac{K_P(s + z_C)}{s}, \tag{1}$$

where K_P and K_I are the proportional gain and integral gain, respectively, and $z_C = K_I / K_P$ is a constant. The IP network delays for sending the control $U(s)$ to the action agent plant $G_P(s)$, and for sending the system output $Y(s)$ to the PI controller $G_C(s)$, are represented by $G_{DCP}(s)$ and $G_{DPC}(s)$, respectively, of which the analytical forms are:

$$G_{DCP}(s) = e^{-\tau_{DCP}s}, \tag{2}$$

$$G_{DPC}(s) = e^{-\tau_{DPC}s}, \tag{3}$$

where τ_{DCP} and τ_{DPC} are the delay from the controller to the plant, and the delay from the plant to the controller in time domain, respectively. The closed-loop transfer function including the network delays becomes:

$$\frac{Y(s)}{R(s)} = \frac{G_C(s)G_{DCP}(s)G_P(s)}{1+G_C(s)G_{DCP}(s)G_P(s)G_{DPC}(s)}. \tag{4}$$

Transfers functions in the exponential form like (2) and (3) have infinite dimensions because the exponential form has infinite order as expressed in a series. In order to analyze the closed-loop control system with network delay effects, a typical approach is to use a rational function with the numerator degree zero to approximate the delays as follows [24]:

$$e^{-\tau s} \cong \frac{1}{\left(1+\frac{\tau s}{n}\right)^n}, \tag{5}$$

where τ can be τ_{DCP} or τ_{DPC}. Using (5), the system dynamics can be only partially realized. For example, only the primary branches of the root locus can be plotted, whereas there are infinite branches of the root locus for the system with (2) and (3). Therefore, the characteristic of the approximated model using (5) will not be exactly the same as the actual one. However, (5) is adequate for many practical applications, because the primary branches usually contain the dominant eigenvalues of the system [24]. There are also other popular delay approximations such as the following power series:

$$e^{-\tau s} \cong 1-\tau s+\frac{\tau^2 s^2}{2!}-\frac{\tau^3 s^3}{3!}+\ldots+\frac{\tau^n s^n}{n!}, \tag{6}$$

and Padé approximation:

$$e^{-\tau s} \cong \frac{N_r(\tau s)}{D_r(\tau s)}, \tag{7}$$

where $N_r = \sum_{n=0}^{r} \frac{(2r-n)!}{n!(r-n)!}(-s\tau)^n$, and $D_r = \sum_{n=0}^{r} \frac{(2r-n)!}{n!(r-n)!}(s\tau)^n$. To distinguish (5) from other delay approximations, we name (5) as *denominator approximation* for future reference. The delay approximations in (6) and (7) are more difficult to analyze because the system poles are

more difficult to find computationally. Padé approximation is even more complicated since it gives additional zeros, which further complicates the overall closed-loop system analysis. In addition, Padé approximation may even give a negative value of $e^{-\tau s}$ if τ is significantly large and the degree of approximation is not adequate. More important, (5) is more suitable for real-time application analysis due to its computational simplicity. Thus, we will use (5) to approximate time delay effects on the networked control system. A later section will show that (5) gives good enough accuracy for the practical problem under our investigation.

C. Case study

A DC motor speed control problem is used as an action agent for a case study to demonstrate the proposed approach throughout this paper. The DC motor parameters obtained from [25] are shown in Table 1.

Table 1. DC motor parameters.

J	Inertia	42.6 e-6 Kg-m^2
L	Inductance	170 e-3 H
R	Terminal Resistance	4.67 Ω
B	Damping coefficient	47.3 e-3 N-m-sec/rad
K	Torque Constant	14.7 e-3 N-m/A
K_B	Back-EMF Constant	14.7 e-3 V-sec/rad

The corresponding action agent plant dynamics can be derived and described by the following transfer function.

$$G_P(s) = \frac{2029.826}{(s+26.29)(s+2.296)}. \tag{8}$$

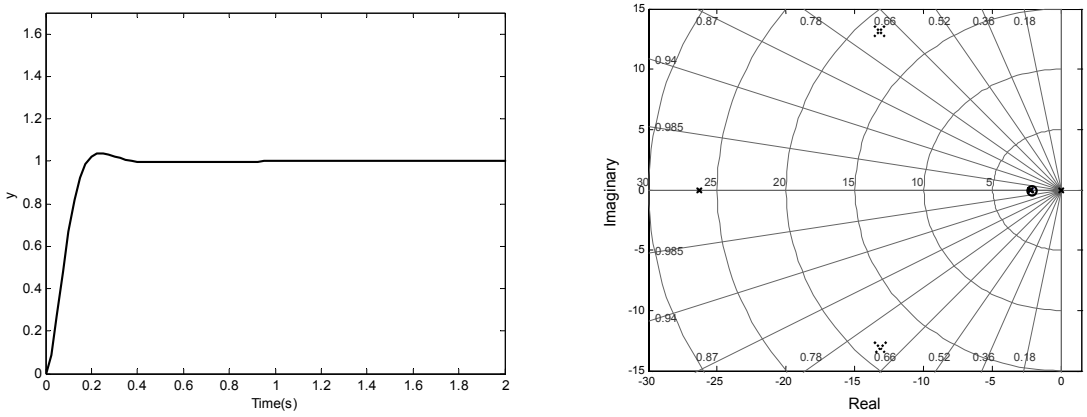
Let us assume that the practically utilized DC motor PI speed controller is designed and tuned without concerning for the network delays for the following step function as the reference signal:

$$r(t) = \begin{cases} 0, & t < 0, \\ c, & t \geq 0, \end{cases} \tag{9}$$

where c is the steady-state value. The overall closed-loop system performance is required to have the relative damping ratio of 0.707 and to satisfy the following specifications with the step response.

1) Percentage overshoot (*P.O.*): $P.O. \leq 5\%$.
2) Settling time (t_s): $t_s \leq 0.309$ sec.
3) Rise time (t_r): $t_r \leq 0.117$ sec.

Using the root locus design approach without considering network delays, a feasible choice of $\left(K_P, K_I\right) = \left(K_P^0, K_I^0\right) \triangleq \left(0.1701, 0.378\right)$, and these gains satisfy all the design specifications as listed. We will use $\left(K_P^0, K_I^0\right)$ as a baseline reference to compare with the proposed gain scheduling algorithm when network delay effects are considered. The DC motor step response with $\left(K_P^0, K_I^0\right)$ is illustrated in Fig. 3 (a), where the open-loop poles and zeros, and closed-loop poles are depicted Fig. 3 (b).



(a) (b)

Fig. 3. DC motor characteristics with respect to $\left(K_P^0, K_I^0\right)$: (a) Step response. (b) Open-loop poles and zeros (on real axis), and closed-loop poles (dotted cross signs).

III. Parameterization for Gain Scheduling: Constant Network Delay

A. PI controller parameterization

With the existence of network delays in the control loop, the initial $\left(K_P^0, K_I^0\right)$ may no longer satisfy the design specifications. The system performance will also degrade, and the system may become unstable. To remain the best possible system performance with network delays, the controller gains need to be adapted with respect to the current network condition. In this section, we introduce the β gain, where $\beta \geq 0$, as a multiplicative factor to externally adapt $\left(K_P^0, K_I^0\right)$ without completely redesigning the existing controller. The idea of β gain adaptation is adopted from [26]. The β gain has to be greater than zero to avoid positive feedback. The β gain is placed in front of the initial PI controller as depicted in Fig. 4.

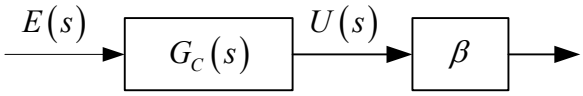


Fig. 4. Adaptation of PI controller gains at the controller output by β.

Analytically, β adjusts both K_P^0 and K_I^0, while keeping the ratio between both gains at z_C as follows:

$$\beta G_C(s) = \beta \frac{K_P^0 (s + z_C)}{s}. \tag{10}$$

This parameterization enables PI gain scheduling to be tractable for real-time on-line analysis with existing theories such as root locus so that the control agent could quickly analyze the system to perform additional advanced control schemes such as fault detection and diagnosis. Adjusting K_P and K_I separately with no concern about the ratio z_C could maintain equivalent or better system performance than adjusting β. However, separately adjusting K_P^0 and K_I^0 requires a more complicated approach like root contour, which is quite tedious and time-consuming. Thus, this approach may not be suitable for real-time on-line analysis. In addition, without β parameterization, $\left(K_P, K_I\right)$ has to be chosen for adjustment from a two

dimensional feasible region. By searching for β instead, the feasible region is only single dimensional, which is easier to search for the optimal value.

B. Optimizing β gain

In order to evaluate the best possible system performance with respect to β under different IP network conditions, we use a cost function approach to find the optimal β. The cost function to be minimized is defined as follows.

$$J = w_1 J_1 + w_2 J_2 + w_3 J_3 , \tag{11}$$

$$J_1 = \begin{cases} \left(MSE - MSE_0\right)^2 , & MSE > MSE_0 , \\ 0 , & MSE \le MSE_0 , \end{cases} \tag{12}$$

$$J_2 = \begin{cases} \left(P.O. - P.O._0\right)^2 , & P.O. > P.O._0 , \\ 0 , & P.O. \le P.O._0 , \end{cases} \tag{13}$$

$$J_3 = \begin{cases} \left(t_r - t_{r0}\right)^2 , & t_r > t_{r0} , \\ 0 , & t_r \le t_{r0} , \end{cases} \tag{14}$$

where

$$MSE = \frac{1}{N}\sum_{k=0}^{N} e^2\left(k\right) \tag{15}$$

is the mean-squared error, MSE_0 is the nominal mean-squared error, $P.O._0$ is the nominal percentage overshoot, and t_{r0} is the nominal rise time. The error $e\left(k\right) = y\left(k\right) - r\left(k\right)$ is computed by sampling $y\left(t\right)$ at $t = kT$, where T is the sampling period, and k is the time index. The costs J_1, J_2, and J_3 are mainly used to provide the penalty when the system performance degrades from the nominal system performance. In this case, the nominal performance can be adopted from the design specifications mentioned earlier such that $P.O._0 =$ 5%, $t_{r0} = 0.117$, whereas MSE_0 has to be determined from a simulation or an experiment. In this paper, we use $MSE_0 = 0.00595$. Therefore, when $\beta = 1$ without network delays in the system, $J = 0$. The cost J_1 gives the penalty on poorer response time and convergence, while the cost J_3 provides

an extra penalty on the slower response. The cost J_2 gives the particular penalty on the higher value of the percentage overshoot. These costs will increase if the networked control system performs worst than the nominal condition. If the system performs equally to the nominal condition or better, the costs are zero.

The weights w_1, w_2, and w_3 are used to specify the relative significance of J_1, J_2, and J_3, respectively, on the overall system performance. These weights are determined from different application requirements. For example, a precision machine like a wirecut EDM (Electrical Discharge Machine) requires more concern on *P.O.* when cutting a piece of metal over a pre-defined path. Thus, w_2 should be higher among the others. On the other hand, a magnetic stirrer requires more accuracy on the steady-state value so that a chemical solution can be mixed appropriately by giving the significance to w_1. In addition, the costs J_1, J_2, and J_3 represent different physical meanings, and may be in different ranges. Therefore, weights to adjust these costs to have the same significances at a nominal condition could be considered as initial weights first. Then, the weights can be later adjusted to support application requirements.

The initial weights could be determined by normalizing J_1, J_2, and J_3 from simulation data at the nominal condition to the same interval. To find the weights by normalization, we can run simulations at the nominal condition for different values of β in a certain range (e.g., $\beta \in [0,2]$), and record J_1, J_2, and J_3 for each simulation with respect to β. Then, J_1, J_2, and J_3 from all simulations can be normalized into [0,1] by using the weight:

$$w_i = 1 \Big/ \left(\max_{\beta} J_i - \min_{\beta} J_i \right), i = 1,2,3, \tag{16}$$

where $\max_{\beta} J_i$ and $\min_{\beta} J_i$ are the maximal and minimal cost J_i obtained from the simulations with different β in the range. Because $\beta = 1$ always satisfies the performance requirements at the nominal condition, $\min_{\beta} J_1 = \min_{\beta} J_2 = \min_{\beta} J_3 = 0$.

Typical behaviors of J_1, J_2, and J_3 with respect to β at the nominal condition are shown in Fig. 5.

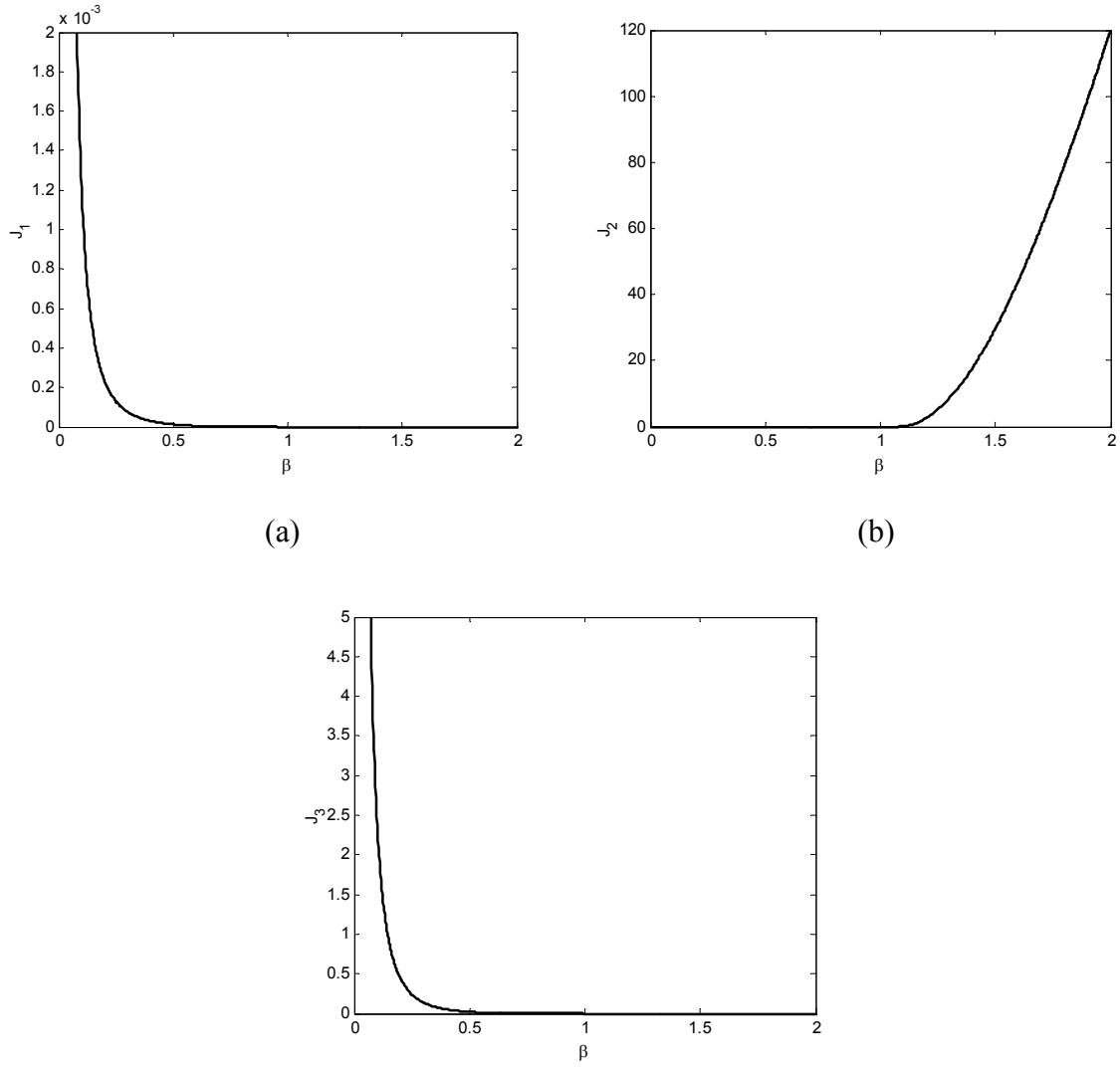


Fig. 5. Typical behaviors of (a) J_1, (b) J_2, and (c) J_3 with respect to β.

As shown in Fig. 5, J_1 and J_3 are large when β is low. In this case, the system feedback response is slower than in the nominal condition in order to reduce $e(t)$. These costs decrease when β increases, and becomes zero at $\beta = 1$. On the other hand, J_2 is large when β is large due to the higher *P.O.* than in the nominal condition. The cost J_2 is smaller when β

decreases, and becomes zero at $\beta = 1$. Thus, with any positive w_1, w_2, and w_3, $\beta = 1$ is the optimal β if there is no network delay.

With the existence of network delays, $\beta = 1$ may no longer be optimal. Thus, the optimal gain has to be obtained by evaluating J with concern for current network delays. Unfortunately, J usually does not have a closed-form relationship with β. Unlike the LQG problem, this optimization problem may not be solved analytically. Therefore, a feasible approach to search for the optimal β is to rely on a simulation according to the feasible set of β. We define $\mathcal{F}$ as the feasible set containing all β that do not cause system instability. The feasible set $\mathcal{F}$ can be estimated by the root locus analysis and the denominator approximation in (5). The characteristic equation of the DC motor speed control system with network delays can then be approximated as:

$$\begin{aligned} 1 + \beta G_C(s) G_{DCP}(s) G_P(s) G_{DPC}(s) &= e^{-\tau s} \frac{2029.826 \beta K_P^0 (s + z_C)}{s(s + 26.29)(s + 2.296)}, \\ &\cong \frac{2029.826 \beta K_P^0 n^n (s + z_C)}{\tau^n s(s + 26.29)(s + 2.296)\left(s + \frac{n}{\tau}\right)^n}, \end{aligned} \tag{17}$$

where τ is defined as $\tau = \tau_{CP} + \tau_{PC}$. For example, setting $n = 4$, the root locus of (17) with respect to $\tau = 0.1, 0.2,$ and 0.5 are shown in Fig. 6. Only the primary branches of the root locus are shown in Fig. 6 to determine the stability of the closed-loop system. As discussed in [24], the primary branches are sufficient to be used for stability region approximation.

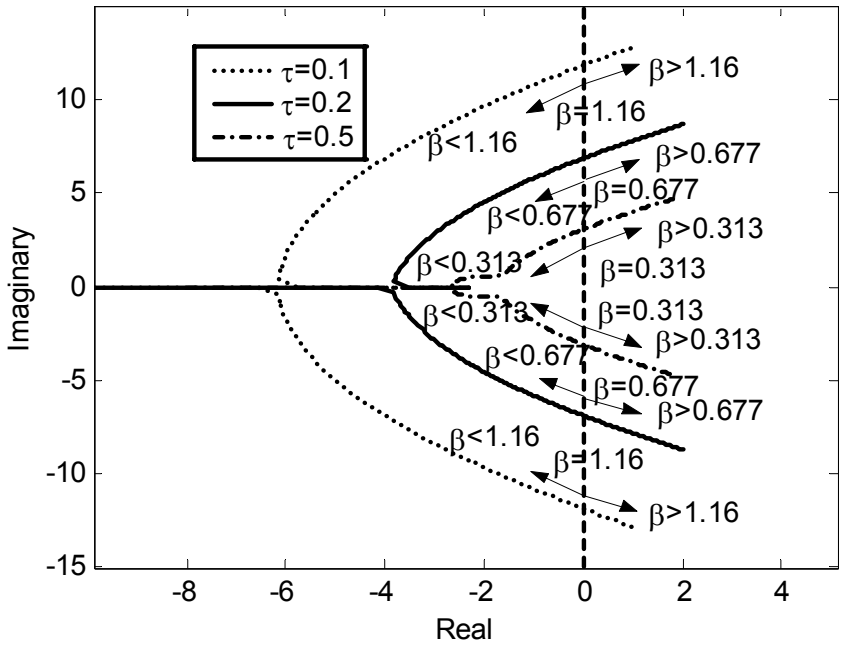
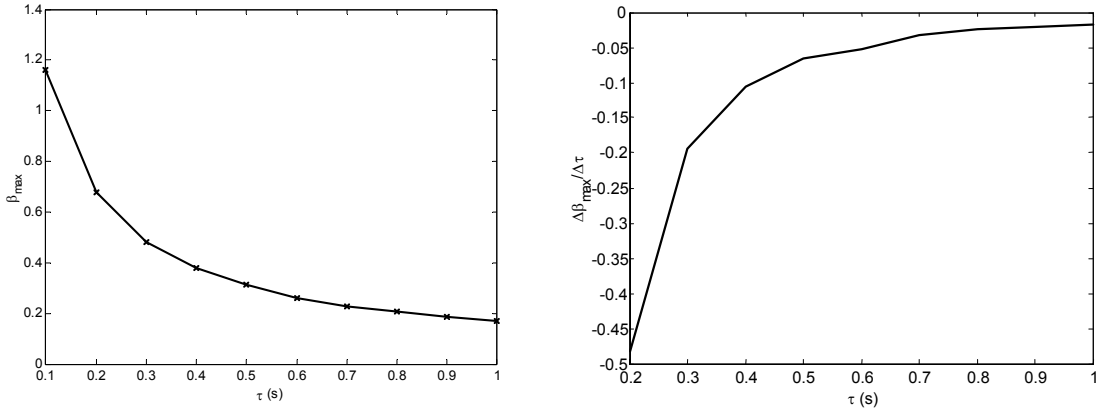


Fig. 6. Primary branches of the root locus of the networked DC motor PI speed control system using denominator approximation to approximate network delays.

In a stable system, β can range from 0 to the value, of which the root locus crosses the imaginary axis. Thus, $\mathcal{F}$ can be defined as $\mathcal{F} = \left\{ \beta \middle| \beta \in \left[0, \beta_{\max}(\tau) \right) \right\}$, where $\beta_{\max}(\tau)$ is the β gain at the point that the root locus crosses the imaginary axis with respect to τ. In fact, $\beta_{\max}(\tau)$ does not need to be very precise since the optimal β is unlikely to be close to $\beta_{\max}(\tau)$. If β is close to $\beta_{\max}(\tau)$, J_1 and J_2 can be very large because the closed-loop system is nearly unstable. Table 2 shows $\beta_{\max}(\tau)$ of the networked DC motor PI speed control system in addition to Fig. 6. Also, $\beta_{\max}(\tau)$ and the change in $\beta_{\max}(\tau)$ with respect to τ defined as $\Delta\beta_{\max}(\tau)/\Delta\tau$ are illustrated in Fig. 7. These $\beta_{\max}(\tau)$ values were obtained by observing the root locus branches of (17) with respect to β that cross the imaginary axis. The root locus calculation is performed in MATLAB 6.1.

Table 2. $\beta_{\max}(\tau)$ of the networked DC motor PI speed control system with respect to τ.

τ (sec)	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
$\beta_{\max}(\tau)$	1.16	0.677	0.483	0.378	0.313	0.261	0.229	0.205	0.185	0.168



(a) (b)

Fig. 7. (a) $\beta_{\max}(\tau)$ and (b) $\Delta\beta_{\max}(\tau)/\Delta\tau$ of the networked DC motor PI speed control system with respect to τ.

As indicated in Table 2 and Fig. 7, $\beta_{\max}(\tau)$ becomes smaller and less sensitive to τ when τ increases. This implies that a longer delay τ gives a smaller feasible set $\mathcal{F}$. Simple search for the optimal β for a specific τ can be easily accomplished according to the algorithm with flowchart depicted in Fig. 8, where $\Delta\beta$ is the bin size, $J_{\min}$ is the minimal cost, β^* is the optimal β, *M* is the number of bins, and *i* is the searching index. The gain searching is divided into two stages to easily investigate the influence of w_1, w_2, and w_3, on the cost *J*. The first stage of the algorithm outlined in Fig. 8 (a) is to compute J_1, J_2, and J_3. The second stage of the algorithm, as outlined in Fig. 8 (b), is to compute *J* and find β. However, both stages can be combined to save computation time when w_1, w_2, and w_3 are exactly chosen.

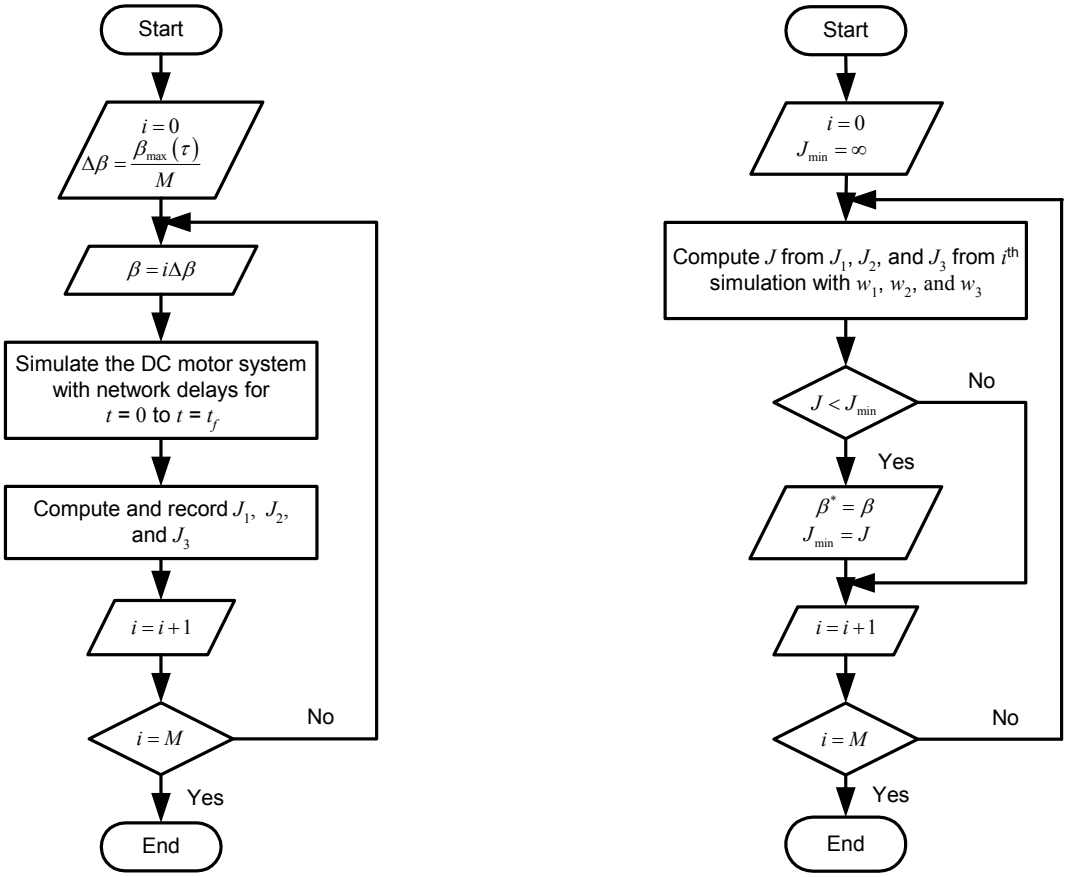


Fig. 8. Searching algorithm for the optimal β.

As shown in Fig. 8 (a), at the beginning of gain searching, $\Delta\beta$ is computed as the searching step. The searching index i is set to zero to begin the first iteration. Then, β is calculated, and used in a DC motor simulation from $t = 0$ to $t = t_f$, where t_f is the final time, with a constant delay τ. After a simulation is completed, J_1, J_2, J_3 for the iteration i are evaluated and recorded. The index i and β are increased by one and $\Delta\beta$, respectively for the next iteration. The simulation is repeated until $i = M$.

To determine β^*, the searching index i is again set to zero, and $J_{\min}$ is set to infinity to be replaced later as shown in Fig. 8 (b). The cost J of the i^{th} simulation is computed from w_1, w_2,

and w_3, and is compared with $J_{\min}$. If the new cost J is less than $J_{\min}$ and constraints are not violated, $J_{\min}$ and β^* are then replaced by J and β of the i^{th} simulation, respectively. The constraints in this case can be bounds of J_1, J_2, and J_3. The index i is then increased by one. The comparison continues until $i = M$.

For example, we optimize J using (2) and (3) as delays for $U(s)$ and $Y(s)$, respectively, with $w_1 = 1.64902,\ w_2 = 0.00833,\ w_3 = 0.01395,\ M = 100,\ t_f = 10$ sec, $\tau = 0.1, 0.2, 0.6$ sec. The weights are obtained by (16). The cost J from this optimization is shown in Fig. 9.

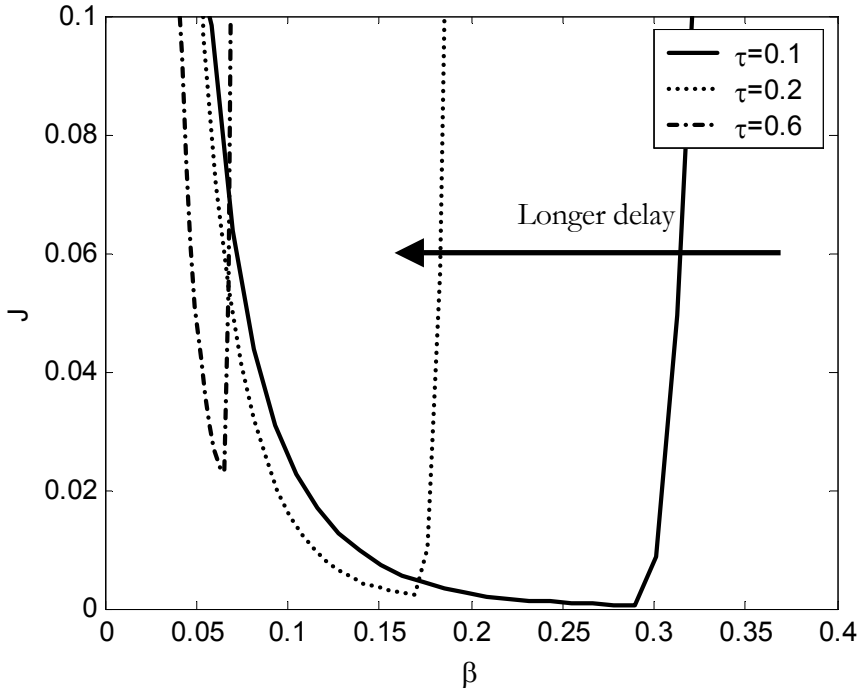


Fig. 9. Cost J from optimization using (2) and (3) to delay $U(s)$ and $Y(s)$, respectively, with $w_1 = 1.64902,\ w_2 = 0.00833,\ w_3 = 0.01395, M = 100,\ t_f = 10$ sec, $\tau = 0.1, 0.2, 0.6$ sec.

As shown in Fig. 9, when the delay τ is longer, the optimal β shifts to the left, and J becomes more *sensitive* to β. Table 3 shows the optimal β from the optimization without constraint, and with constraints $J_1 \le 0.02$, $J_2 \le 0.01$, $J_3 \le 0.15$ for comparison.

Table 3. Optimal β from optimization using (2) and (3) to approximate delays with $w_1 = 1.64902,\ w_2 = 0.00833,\ w_3 = 0.01395,\ M = 100,\ t_f = 10$ sec : woc-(without constraint), wc-(with constraint $J_1 \le 0.02$, $J_2 \le 0.01$, $J_3 \le 0.15$), N/A-not available.

τ (sec)	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Optimal β wc	0.293	0.171	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
Optimal β woc	0.293	0.171	0.121	0.094	0.077	0.064	0.056	0.049	0.044	0.040

Noticeably, with some values of τ such as $\tau = 0.3$, there can be no β that can satisfy the constraints. The performance degradation caused by such a long delay is too much to handle with adjusting $\left(K_P^0, K_I^0\right)$ by β in order not to violate the constraints.

To verify whether the denominator approximation can provide enough accuracy for the delay approximation, we rerun the same simulations without constraint except the delays are approximated by denominator approximation. If the denominator approximation is accurate, J from this case will be similar to the previous simulation such that the optimization will give the similar optimal β. Table 4 shows the percentage error between the optimal β from the previous simulation defined as β_{act} and the simulation using denominator approximation defined as β_{app}. Both β_{act} and β_{app} are truncated to fit into Table 4.

Table 4. Percentage error between the optimal β obtained by using (2) and (3) defined as β_{act} and the optimal β obtained by using denominator approximation defined as β_{app} from simulations.

τ (sec)	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
β_{act}	0.293	0.171	0.121	0.094	0.077	0.064	0.056	0.049	0.044	0.040
β_{app}	0.291	0.170	0.120	0.093	0.076	0.064	0.055	0.049	0.043	0.039
$\left\lvert\frac{\beta_{act} - \beta_{app}}{\beta_{act}}\right\rvert \times 100\%$	0.791	0.791	0.797	0.807	1.225	0.810	1.230	1.250	1.261	1.266

As shown in Table 4Table , the largest percentage error is not over 1.3%. These small percentage errors can be accurate enough for many practical requirements. However, if more accuracy is required, the degree *n* in (5) can be increased to fulfill the requirement.

In Part II of this paper, we will extend the concepts, methods, and results, from Part I to actual IP network delays for networked PI controller gain scheduling. Part II will also describe the proposed gain scheduling algorithm. The performance of the gain scheduling approach will be verified by simulations in Part II.

IV. Conclusion

Part I of the two companion papers presented the foundation for the proposed gain scheduling scheme for a networked PI control system over IP networks. The networked PI control system is initially formulated with constant network delays, which are approximated by a rational function denoted as denominator approximation. The accuracy of the delay approximation can be improved by increasing the order of the rational function. In a case study using a DC motor speed control problem, the fourth-order approximation is reasonably accurate.

Scheduling the PI controller gains with respect to a network delay by using a multiplicative factor β to adjust K_P and K_I allows the control agent to perform tractable real-time on-line analysis. Moreover, the K_P and K_I gains can be updated externally at the controller output without interrupting the original PI controller. By using the root locus analysis on the system transfer function including the transfer function of the network delay approximation model to estimate stability regions, the bounds of feasible regions to search for the optimal β for gain scheduling can be obtained. An iterative searching method can be used to find the optimal β as suggested. When the network delay is longer, the stability region and the bound of the feasible region of β are smaller as well as the optimal β. On the other hand, with a longer delay, the bound becomes less sensitive to the delay, whereas the optimal β is more sensitive. In addition, when a constraint is enforced into gain searching, there may be no solution due to

too much performance degradation by the network delay. The foundation described in Part I will be extended to networked control with random IP network delays and described in Part II of the two companion papers along with gain scheduling algorithm and simulation results.

Acknowledgement

The authors would like to thank the Royal Thai Government for partially supporting this study, and Dr. Douglas S. Reeves for his helpful comments related to this paper.

References

[1] E. Tovar and F. Vasques, "Real-time fieldbus communications using Profibus networks," *IEEE Transactions on Industrial Electronics*, vol. 46, no. 6, pp. 1241-1251, 1999.

[2] M. Farsi, K. Ratcliff, and M. Barbosa, "An overview of controller area network," *Computing & Control Engineering Journal*, vol. 10, no. 3, pp. 113-120, 1999.

[3] G. Kaplan, "Ethernet's winning ways," *IEEE Spectrum*, vol. 38, no. 1, pp. 113-115, 2001.

[4] F.-L. Lian, J. R. Moyne, and D. M. Tilbury, "Performance evaluation of control networks: Ethernet, ControlNet, and DeviceNet," *IEEE Control Systems Magazine*, vol. 21, no. 1, pp. 66-83, 2001.

[5] Y. Halevi and A. Ray, "Integrated communication and control systems: Part I-Analysis," *Journal of Dynamic Systems, Measurement, and Control*, vol. 110, pp. 367-373, 1988.

[6] J. Nilsson, "Real-time control systems with delays," *Ph.D. dissertation*, Lund Institute of Technology, Lund, Sweden, 1998.

[7] G. C. Walsh, H. Ye, and L. G. Bushnell, "Stability analysis of networked control systems," *IEEE Transactions on Control Systems Technology*, vol. 10, no. 3, pp. 438-446, 2002.

[8] F. Göktas, "Distributed control of systems over communication networks," *Ph.D. dissertation,* University of Pennsylvania, Philadelphia, PA, 2000.

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

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

[11] M. L. Sichitiu, "Control of data networks: models, stability and controllers," *Ph.D. dissertation*, University of Notre Dame, Notre Dame, IN, 2001.

[12] T.-J. Tarn and N. Xi, "Planning and control of internet-based teleoperation," in Proceedings of SPIE: Telemanipulator and telepresence technologies V, Boston, MA, 1998, pp. 189-193.

[13] S. H. Hong, "Scheduling algorithm of data sampling times in the integrated communication and control systems," *IEEE Transactions on Control Systems Technology*, vol. 3, no. 2, pp. 225-230, 1995.

[14] Y. H. Kim, H. S. Park, and W. H. Kwon, "Stability and a scheduling method for network-based control systems," in IEEE IECON 96, Taipei, Taiwan, 1996, pp. 934-939.

[15] S. Lee, S. H. Lee, and K. C. Lee, "Remote fuzzy logic control for networked control system," in IEEE IECON 2001, Denver, CO, 2001, pp. 1822-1827.

[16] N. B. Almutairi, M.-Y. Chow, and Y. Tipsuwan, "Network-based controlled DC motor with fuzzy compensation," in IEEE IECON 2001, Denver, CO, 2001, pp. 1844-1849.

[17] Y. Tipsuwan and M.-Y. Chow, "Network-based controller adaptation based on QoS negotiation and deterioration," in IEEE IECON 2001, Denver, CO, 2001, pp. 1794-1799.

[18] S. Soucek, T. Sauter, and T. Rauscher, "A scheme to determine QoS requirements for control network data over IP," in IEEE IECON 2001, Denver, CO, 2001, pp. 153-158.

[19] C. Metz, "IP QOS: Traveling in first class on the Internet," *IEEE Internet Computing*, vol. 3, no. 2, pp. 84-88, 1999.

[20] T. P. Barzilai, D. D. Kandlur, A. Mehra, and D. Saha, "Design and implementation of an RSVP-based quality of service architecture for an integrated services Internet," *IEEE Journal on Selected Areas in Communications*, vol. 16, no.3, pp. 397-413, 1998.

[21] J. Shin, J. W. Kim, and C.-C. J. Kuo, "Quality-of-service mapping mechanism for packet video in differentiated services network," *IEEE Transactions on Multimedia*, vol. 3, no. 2,pp. 219-231, 2001.

[22] T. F. Abdelzaher, E. M. Atkins, and K. G. Shin, "QoS negotiation in real-time systems and its application to automated flight control," *IEEE Transactions on Computers*, vol. 49, no. 11, pp. 1170-1183, 2000.

[23] B. Li and K. Nahrstedt, “A control-based middleware framework for quality-of-service adaptations,” *IEEE Journal on Selected Areas in Communications*, vol. 17, no. 9, pp. 1632-1650, 1999.

[24] B. C. Kuo, *Automatic Control Systems*, 5 ed. Englewood Cliffs, NJ: Prentice-Hall, 1987.

[25] Y. Tipsuwan and M.-Y. Chow, “Fuzzy logic microcontroller implementation for DC motor speed control,” in IEEE IECON 99, San Jose, CA, 1999, pp. 1271-1276.

[26] N. B. Almutairi and M.-Y. Chow, “A modified PI control action with a robust adaptive fuzzy controller applied to DC motor,” in IJCNN'01, Washington, DC, 2001, pp. 503-508.

C H A P T E R V

METHODOLOGY OF USING NETWORKED PI CONTROLLER GAIN SCHEDULING OVER IP NETWORK:

PART II – NETWORKED CONTROL ON ACTUAL IP NETWORK

Yodyium Tipsuwan
ytipsuw@unity.ncsu.edu

Mo-Yuen Chow
chow@eos.ncsu.edu

Advanced Diagnosis And Control Lab
Department of Electrical and Computer Engineering
North Carolina State University, Raleigh NC 27606, USA
Tel: (919) 515-7360, Fax: (919) 515-5108

This chapter and chapter IV were submitted for publication to an IEEE Transactions, as parts I and II.

METHODOLOGY OF USING NETWORKED PI CONTROLLER GAIN SCHEDULING OVER IP NETWORK: PART II – NETWORKED CONTROL ON ACTUAL IP NETWORK

Abstract—This paper is the second of two companion papers. The foundation for the proposed gain scheduling approach to enhance a PI controller for use with constant network delays was given in Part I. Part II extends the concepts and methods in Part I to networked control on actual IP networks. Actual IP network RTT (Round-Trip Time) delays are analyzed by statistical measures and histograms. Parameterization of actual IP network delays by extending the constant network delay concept in Part I to a generalized exponential distribution model is described. The searching algorithm to find the optimal parameter for gain scheduling is also extended to support the distribution model. This paper also covers the gain scheduling algorithm with respect to the monitored current IP traffic condition by estimating the probabilistic parameters of the distribution model in real-time. Simulation results show that the high values of the parameters of the distribution model can degrade the networked system performance and reduce the optimal parameter for gain scheduling. With reasonably long IP network delays, the proposed gain scheduling scheme can substantially maintain the system performance and stabilize the system based on a real-time IP traffic condition satisfactorily as illustrated by simulations using artificial data and actual IP network measurements. The PI controller with gain scheduling provides the overall networked control system performance significantly better than the system using the nominal gains without the delay concern.

Keywords—Internet, networks, adaptive control, control systems, DC motors, distributed control, real time system.

I. Introduction

Part I [1] of this paper introduces the foundation of a methodology that enhances the widely used PI (Proportional-Integral) controller so it can be used over IP networks. The proposed approach is based on PI controller gain scheduling, where the PI controller gains are updated at the controller output. The approach could save much cost and time for practically existing PI controllers because the controllers can still be used without requiring redesign and reinstallation of a new networked control system. In order to maintain the best possible system performance, the optimal PI controller gains are scheduled in real-time with respect to the monitored IP network traffic condition. The foundation for the proposed approach including parameterization of a networked PI control system for gain scheduling and stability region approximation with respect to constant network delays will be extended for actual IP network delays here in Part II. An algorithm to search for the optimal parameter to schedule the PI controller gains in Part I is also continually utilized. Characterization and parameterization of actual IP network delays based on the foundation in Part I are described in Section II. Section III presents the real-time gain scheduling algorithm. Simulation results to verify the performance of the proposed gain scheduling approach are shown in Section IV. The paper is concluded in Section V.

II. Parameterization for Gain Scheduling: Actual IP Network Delay

A. IP network delay characteristics

Actual IP network delays are not constant, but stochastic in nature. In addition, because packets are sent in discrete-time according to the applications and protocols used, the network delays are not necessarily continuous. The time-varying characteristic in IP network delays is caused by several factors such as physical media (e.g., wire or wireless), network configuration, routing protocols, traffic conditions, and network usages. To illustrate actual IP network delays, RTT (roundtrip time) delays measured for 24 hours (00:00-24:00) from an

Ethernet network in ADAC (Advanced Diagnosis And Control) Lab at North Carolina State University (NCSU) to the destinations listed in Table 1 are shown in Fig. 1. Statistical measures of the RTT delays are shown in Table 2. The corresponding histograms of the RTT delays to approximate probability densities are shown in Fig. 2. Note that the scales of delays on the x-axis in Fig 2 (a), (b), (c), and (d) are different, which are [4e-4, 9e-9], [0.022, 0.034], [0.062, 0.0645], and [0,1] sec, respectively.

Table 1. Destinations to measure RTT delays from ADAC Lab at NCSU, their location, and distance: www,lib.ncsu.edu, www.visitnc.com, www.utexas.edu, and www.ku.ac.th.

Destination host	**Description**	**Location**	**Distance from ADAC Lab (miles)**
www.lib.ncsu.edu	NCSU library	NCSU, Raleigh, NC, USA	0.15
www.visitnc.com	Department of Commerce, NC	Releigh, NC, USA	1.5
www.utexas.edu	University of Texas at Austin	Austin, TX, USA	1169
www.ku.ac.th	Kasetsart University	Bangkok, Thailand	9015

Table 2. Statistical measures (minimum, median, mean, and maximum) of RTT delays measured from ADAC Lab at NCSU to www.lib.ncsu.edu, www.visitnc.com, www.utexas.edu, and www.ku.ac.th.

Destination host	$\tau_{\min}$ **(sec)**	τ_{median} **(sec)**	τ_{mean} **(sec)**	$\tau_{\max}$ **(sec)**
www.lib.ncsu.edu	0.000435	0.000471	0.000580	0.0862
www.visitnc.com	0.0166	0.0232	0.0326	0.7562
www.utexas.edu	0.0622	0.0627	0.0629	0.1187
www.ku.ac.th	0.0045	0.3150	0.3730	227.7095

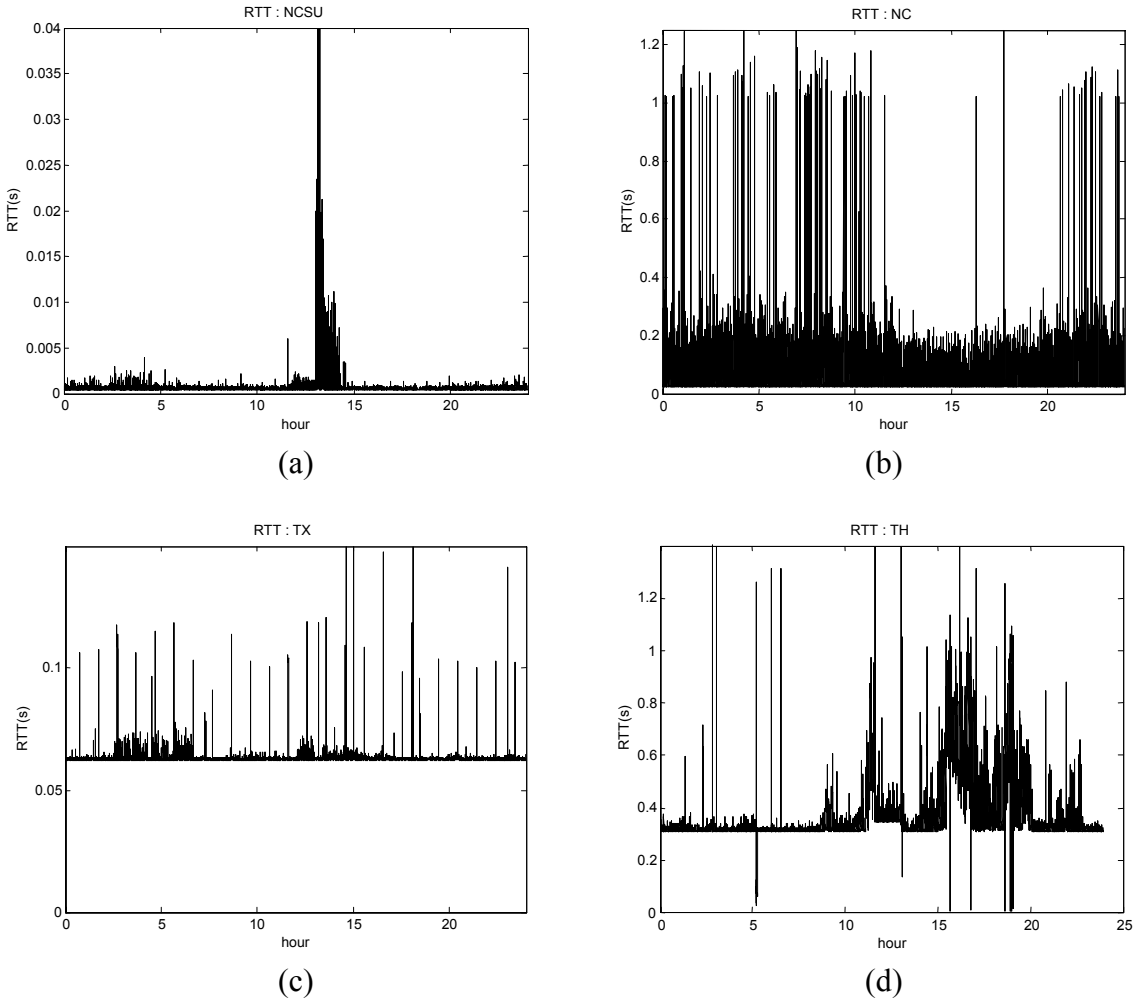


Fig.1. RTT delays measured from ADAC Lab at NCSU to: (a) www.lib.ncsu.edu. (b) www.visitnc.com. (c) www.utexas.edu. (d) www.ku.ac.th.

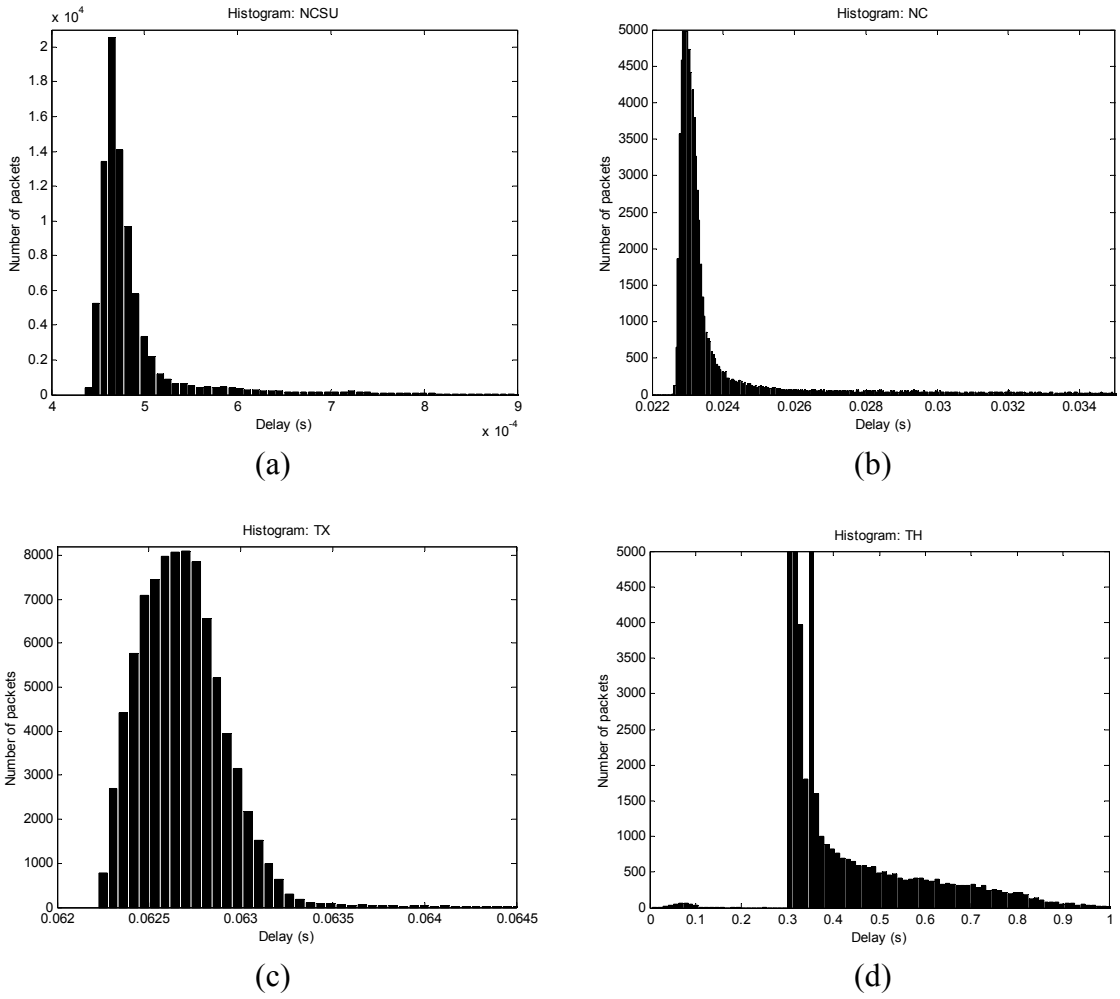


Fig. 2. Histograms of RTT delays measured from ADAC Lab at NCSU to: (a) www.lib.ncsu.edu. (b) www.visitnc.com. (c) www.utexas.edu. (d) www.ku.ac.th.

As shown in Fig. 1 (a), (b), and (c), the RTT delay in each case can be considered to vary above a constant delay. The constant delay is the minimal RTT delay in Table 1. The minimal delay is mainly caused by the propagation delay and the necessary transmission delay in a minimal-hop transmission. RTT delay that is greater than the minimal delay may be induced by various causes including queuing delays and congestion conditions in routers. The RTT delay characteristic in Fig. 1 (d) is quite different from the other cases. The RTT delay in this case could not be considered to vary above a constant delay. Changes of medias used to link IP

networks between the USA and Thailand at different time frames such as different fiber optic lines of several country gateways may be a reason for this situation.

The physical distance between ADAC Lab and a destination is a factor for RTT delays, and the distance in general directly relates to the propagation delay among the sites. As shown in Table 2, the further destinations have longer minimal, median, and mean RTT delays. However, the distance does not always imply a longer delay. For example, www.visitnc.com has a longer maximal RTT delay than www.utexas.edu that is much further actually. Packets from ADAC Lab to www.visitnc.com may take more hops than packets to www.utexas.edu.

Furthermore, the RTT delay may depend on other factors like local activities on the destination site and demands to access the site. For example, www.lib.ncsu.edu may be routinely backing up data around midnight so that the RTT delay increases during this time period. Likewise, not many people would access NC information on www.visitnc.com overnight. Thus, the RTT delay at this time period is lower than the RTT delay measured during the daytime. In this paper, we assume that the control agent only performs the control tasks including monitoring IP traffic conditions, computing control signals, and exchanging control packets and sensory output packets with action agents as mentioned in Part I. Likewise, the activities of action agents are limited to data conversion and packet exchanges with the control agent. The control agent and action agents are not allowed to perform other unnecessary local activities to limit RTT delays.

B. Parameterization of IP network delay characteristics for gain scheduling

Fig. 2 shows that the histograms skew to the left. These shapes of the histograms indicate the higher probability to have RTT delay that is shorter than the median and mean. RTT delay can be much longer than the median and mean, but with much lower probability. In this paper, we will focus more on well-regulated traffic IP networks with a small number of hops that have the probability density of RTT delay similar to Fig. 2 (a) and (b). To investigate how this stochastic behavior can affect the optimality of β, RTT delay is modeled by a random

probability distribution. The random distribution model should be simple in order to estimate in real-time for gain scheduling, while providing reasonable accuracy in representing different IP network conditions. Based on these reasons, we propose to use the generalized exponential distribution to describe IP network delays as follows:

$$P[\tau] = \begin{cases} \frac{1}{\phi} e^{-(\tau-\eta)/\phi}, \tau \geq \eta, \\ 0, \qquad \tau < \eta, \end{cases} \tag{1}$$

where the expected value of the RTT delay $E[\tau] = \phi + \eta$, and variance $\sigma^2 = \phi^2$. If η is known, ϕ can be easily approximated from η, and an experimental value of $E[\tau]$ or the mean μ. Some researchers have also used the exponential distribution to approximate IP network delays for control over IP networks [2]. An important concern is how to select η in real-time. Both η and ϕ parameters have to be updated frequently with respect to the current IP traffic condition. This concern relates to how to generalize the approach based on a constant τ in the previous section. RTT delay can be treated as the delay τ except that RTT delay is stochastic and happens as discrete events. In this paper, we treat the IP network stochastic behavior as a parameter variation of the system transfer function. Therefore, η should be an appropriate value to serve as a base for the parameter variation described as $\tau = \eta + \Delta\tau$, where $\Delta\tau$ is the delay parameter variation. Also, based on (1), $P[\tau = \eta]$ should be the peak of the probability density function. A feasible choice of η is the median of RTT delay. The median can be easily computed in real-time and is representative for a majority of RTT delay. For example, the medians in Table 2 are very representative because they locate closely to the peaks of the histograms in Fig. 2, which are the majority of RTT delays. We ignore the case $\Delta\tau < 0$ (i.e. $\tau < \eta$) since this variation is relatively small. In addition, $P[\tau = \eta]$ could be used as the worst-case RTT delay distribution for $\tau < \eta$. Based on (1) and the RTT delay statistics in Table 2, we also assume that the ideal RTT delays have $\mu > \eta$.

C. Optimizing β with actual IP network delay concern

The controller used in the real IP network environment has to be a discrete-time PI controller to support actual IP packet transmissions. The PI controller in Part I can be discretized by several approximation methods such as forward Euler, backward Euler, or Tustin approximations. In part II of this paper, we use a discrete-time PI controller approximated by forward Euler approximation as an example. With the forward Euler approximation, a stable continuous-time system could be possibly mapped in an unstable discrete-time system [3]. The optimal β obtained by the procedure in Part I with a continuous-time PI controller may cause instability when the corresponding discrete-time PI controller is used. Therefore, the optimal β for a discrete-time PI controller should be obtained by using the discrete-time controller in the searching algorithm directly, where $\beta_{\max}(\tau)$ obtained from the root locus in the continuous-time domain can still be used. The sampling time of the controller is defined as $T = 1$ msec so that the discrete-time PI controller behaves closely to the continuous-time controller at the nominal condition without violating the performance specifications mentioned in Part I.

If RTT delay is constant, ϕ will be zero, and the optimal β for $\tau = \eta$ can be immediately applied. With the stochastic behavior of actual IP networks, ϕ could affect the optimal setting of the PI controller as the delay transfer functions in the control loop are changed. Thus, the optimal β under different actual IP network traffic conditions has to be evaluated with respect to the updated η and ϕ. To find the optimal β with respect to the updated η and ϕ, the same procedure in Part I can be used in simulations except that the delay τ has to vary by the generalized exponential distribution in (1) with given η and ϕ. Fig. 3 shows the cost J with respect to $\phi = 0, 0.002, 0.004, 0.006$, while holding $\eta = 0.01$. The surface of J with respect to $\eta = 0.01, 0.02, 0.03, \ldots, 0.1,$ and $\phi = 0, 0.001, 0.002, ..., 0.009,$ is also shown in Fig. 4. The cost J uses $w_1 = 1.64902,\ w_2 = 0.00833,\ w_3 = 0.01395,\ M = 100,\ t_f = 10$ sec as used in Part I without constraint. Some of the optimal β found from J are shown in Table 3.

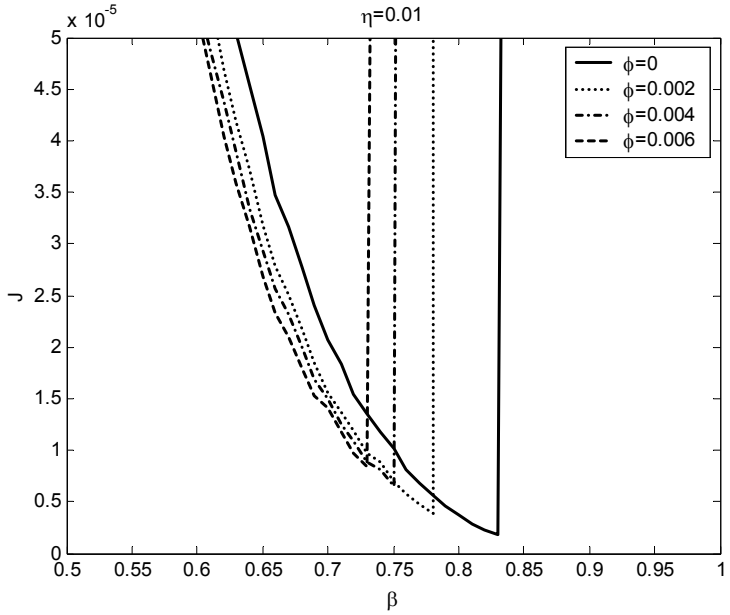


Fig. 3. Typical effect of various ϕ on the optimal β selecting while holding η constant at $\eta = 0.01$.

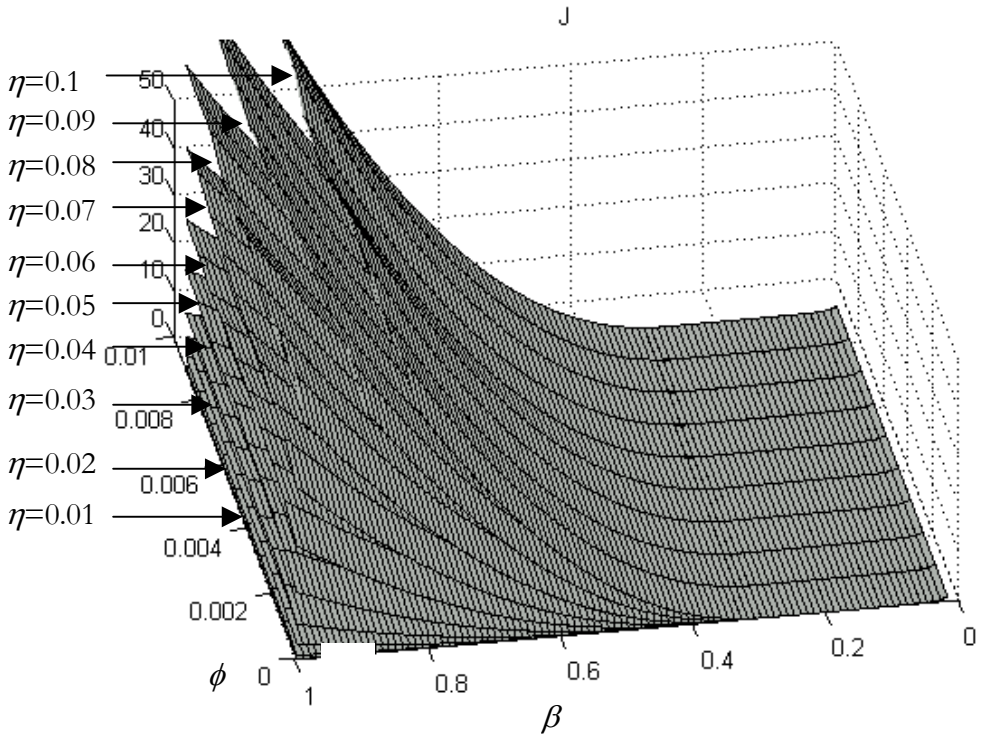


Fig. 4. Typical cost surfaces of J with respect to various η and ϕ without constraint.

Table 3. Optimal β with respect to $\eta = 0.01, 0.02, 0.03, 0.04,$ and $\phi = 0, 0.001, 0.002, ..., 0.009$ obtained with $w_1 = 1.64902$, $w_2 = 0.00833$, $w_3 = 0.01395$, $M = 100$, $t_f = 10$ sec without constraint.

ϕ \ η	**0**	**0.001**	**0.002**	**0.003**	**0.004**	**0.005**	**0.006**	**0.007**	**0.008**	**0.009**
0.01	0.83	0.68	0.58	0.51	0.45	0.40	0.37	0.34	0.31	0.29
0.02	0.79	0.65	0.56	0.49	0.44	0.39	0.36	0.33	0.30	0.28
0.03	0.78	0.65	0.55	0.48	0.43	0.39	0.36	0.33	0.30	0.28
0.04	0.76	0.64	0.55	0.48	0.43	0.39	0.35	0.32	0.30	0.28

As shown in Fig. 3 and Table 3, the optimal β shifts to the left and decreases in value when there is more delay variation indicated by a higher value of ϕ. A higher η also lowers the optimal β as shown in Table 3 and Fig. 4.

III. Gain Scheduling Algorithm

Since the traffic condition on actual IP networks does not always remain the same, the networked PI controller has to adapt itself to handle this time-varying traffic condition. A possible solution to adapt the networked PI controller is to adjust $\left(K_P^0, K_I^0\right)$ by scheduling β with respect to the current traffic condition characterized by η and ϕ. We assume that the gain scheduling approach is implemented as a hardware or software part of the control agent called the β scheduler middleware, which is physically (in hardware) or virtually (in software) attached in front of the original PI controller, respectively. The β scheduler middleware must have enough memory to store the optimal β with respect to η and β as a lookup table. In addition, the β scheduler middleware must have IP network interface for RTT monitoring and packet exchanges between the PI controller and an action agent plant without interrupting the original PI controller. An output queue is also required in the β scheduler middleware to store outgoing packets when the IP network is not ready for a network transmission. The gain scheduling procedure is depicted in Fig. 5 with steps 1, 2, 3, 4, and 5, to be described as follows.

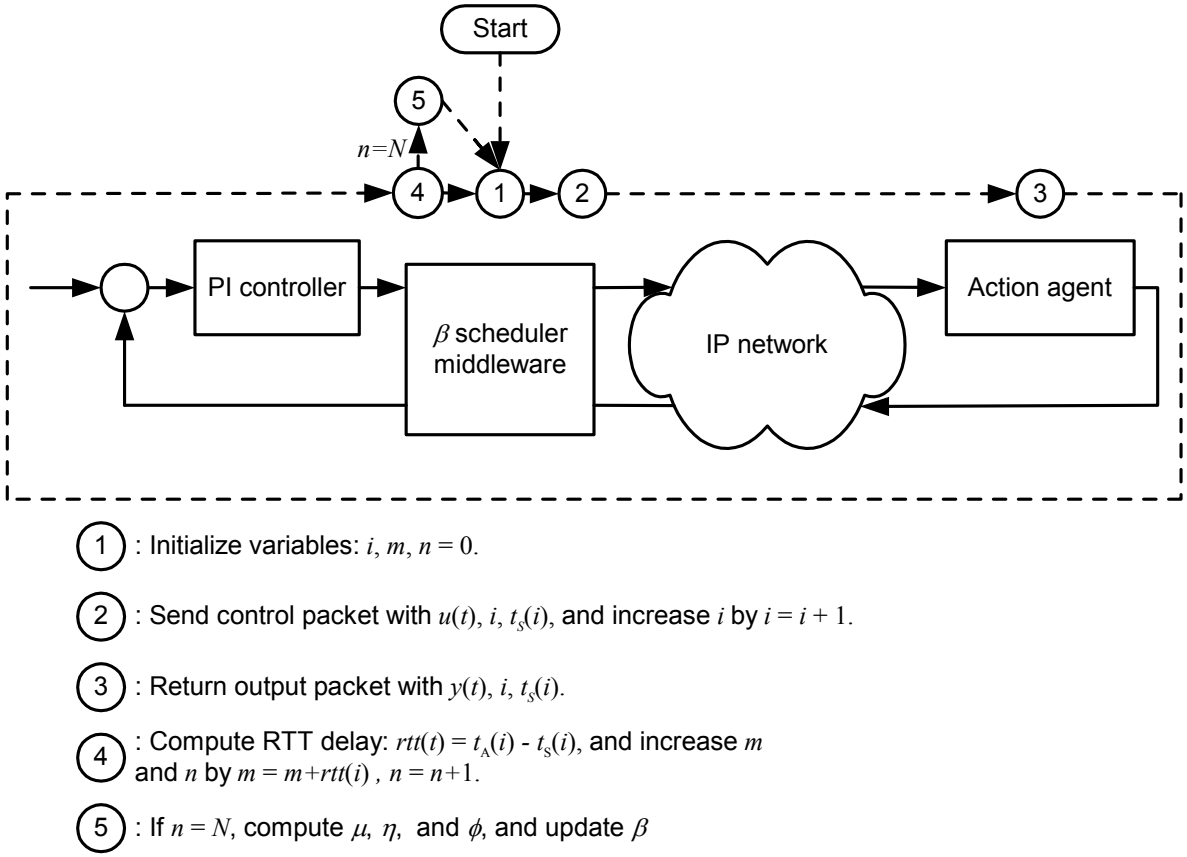


Fig. 5. β gain scheduler middleware operation.

1) The β gain scheduler middleware initializes the packet index defined as $i = 0,\ldots,N$, to $i = 0$, the summation of RTT delay defined as m to $m = 0$, and the number of successful packet roundtrips defined as n to $n = 0$ to be used in later steps.

2) To send $u(t)$ out to an action agent according to the original controller operation, the β scheduler middleware captures and puts $u(t)$ in a UDP packet at every sampling time T with i, and the current time defined as the sending time $t_S(i)$. The control $u(t)$ in the packet i is defined as $u(t,i)$ for future reference. The packet i should be sent out immediately. However, the IP network may not be always available for a transmission. Thus, the packet i may have to be stored in the output queue to wait for sending at the instant that the IP network is ready. Once the packet i is pushed in the queue, or sent out immediately without being

stored, the β scheduler middleware will increase the packet index by $i = i + 1$. A possible control packet format used is illustrated in Fig. 6 (a).

(a) (b)

Fig. 6. Possible UDP packet formats for β scheduler middleware:
(a) Control packet of $u(t,i)$. (b) Output packet of $y(t,i)$.

3) The action agent will return the output $y(t)$, i, and $t_S(i)$, as a packet shown in Fig. 6 (b) back to the β scheduler middleware once it receives and processes $u(t,i)$ periodically using the sampling period T. This corresponding feedback is defined as $y(t,i)$. Likewise, we assume the action agent has the same queuing mechanism to handle outgoing packets. The feedback packet will arrive at the β scheduler middleware with a random IP network delay.

4) When the β scheduler middleware receives a packet containing $y(t,i)$ from the action agent at time t during a sampling period, the β scheduler middleware will compute:

$$rtt(i) = t_A(i) - t_S(i), \tag{2}$$

$$m = m + rtt(i), \tag{3}$$

$$n = n + 1, \tag{4}$$

where $rtt(i)$ is the RTT delay of the packet roundtrip i, and $t_A(i)$ is the arrival time of the corresponding feedback packet i. The summation m is used to later compute the mean μ. The β scheduler middleware will store $rtt(i)$ in memory along with other RTT denoted as $rtt(j)$, $\forall j \in \mathbb{N} < i$ that are previously computed. The RTT delay $rtt(i)$ is placed in the memory, at which $rtt(a) < rtt(i) \le rtt(b)$, $\forall a, \forall b \in \mathbb{N} \le i$ for sorting RTT delays in the memory to later compute η. For future reference, the RTT delay stored at position l in the memory is defined as $RTT[l]$.

Packets transmitted between the β scheduler middleware and an action agent may be lost because of reasons such as IP network congestion and a router's packet dropping policy.

Therefore, there would be some unsuccessful packet roundtrips. In this case, the PI controller and the action agent will opt to use the most updated data to compute $u(t,i)$ and $y(t,i)$, respectively. In this paper, we focus on the effect of IP network delay and variation, and assume that the number of unsuccessful packet roundtrips is small such that it does not significantly affect the control performance.

5) Once $n = N$, the β scheduler middleware will calculate:

$$\mu = m/N, \tag{5}$$

$$\eta = \begin{cases} \left(RTT\left[\left(N/2\right)\right] + RTT\left[\left(N/2\right)+1\right]\right)/2, & N \text{ is even,} \\ RTT\left[\left\lceil N/2 \right\rceil\right] & , N \text{ is odd,} \end{cases} \tag{6}$$

$$\phi = \begin{cases} \mu - \eta, & \mu > \eta, \\ 0 \quad , & \mu \le \eta, \end{cases} \tag{7}$$

where N is the number of packets used to approximate the characteristic of RTT delay. When $\mu \le \eta$, $\phi = 0$ and η becomes the representative worst-case delay to avoid a negative ϕ, which violates the shape of (1). The β scheduler middleware then updates β by picking the optimal β from the table with respect to η and ϕ. Steps 1-5 will be repeated for the next iteration.

Before the proposed gain scheduling is practically applied, the β scheduler middleware should first probe the network traffic condition to pick up an appropriate initial β from the look-up table. The probing can be performed by informing the action agent for probing, and following steps 1-5 for a single iteration without actually controlling the action agent.

IV. Case Study and Simulation Results

The performance of the proposed β scheduler middleware approach on the networked DC motor PI speed control system described in section II of Part I of this paper [1] is verified by simulations implemented on Matlab/Simulink 6.1. The following environment is used to illustrate the effectiveness of the proposed β scheduler middleware control scheme.

- The steady-state reference value: $c = 1$.

- The final simulation time: $t_f = 10$ sec.
- The size of the optimal β lookup table $\eta \times \phi$ is 14×26, where $\eta = 0.01, 0.02, 0.03, \ldots, 0.09, 0.1, 0.2, 0.3, \ldots, 0.5$, and $\phi = 0, 0.001, 0.002, \ldots, 0.019, 0.02, 0.03, 0.04, \ldots, 0.07$. The optimal β from η and ϕ that is not in the table is obtained by the linear-interpolation technique. The optimal β is computed by following the procedure in Part I using the number of bins $M = 100$.
- The sampling time of the PI controller, the β scheduler middleware and the action agent controller: $T = 1$ msec.
- The number of packets to evaluate the characteristic of RTT delay: $N = 100$.
- The DC motor parameters are picked up from [4] as same as in Table 1 of Part I.
- The nominal PI controller gains: $\left(K_P^0, K_I^0\right) \triangleq \left(0.1701, 0.378\right)$.

Multiple RTT delay data sets are prepared for the simulations by pre-computing with certain values of η and ϕ and also obtaining from actual measurements. In this paper, we assume that the delay from the β scheduler middleware to the DC motor (τ_{CP}) and the delay from the DC motor to the β scheduler middleware (τ_{PC}) have similar characteristics. Therefore, we prepare these delays for the simulation by dividing all data points in a RTT delay data set by two, and selecting some values from this set to apply as τ_{CP}, and τ_{PC}. Each value of τ_{CP} chosen from data points in the RTT data set is different from the data points used for τ_{PC}.

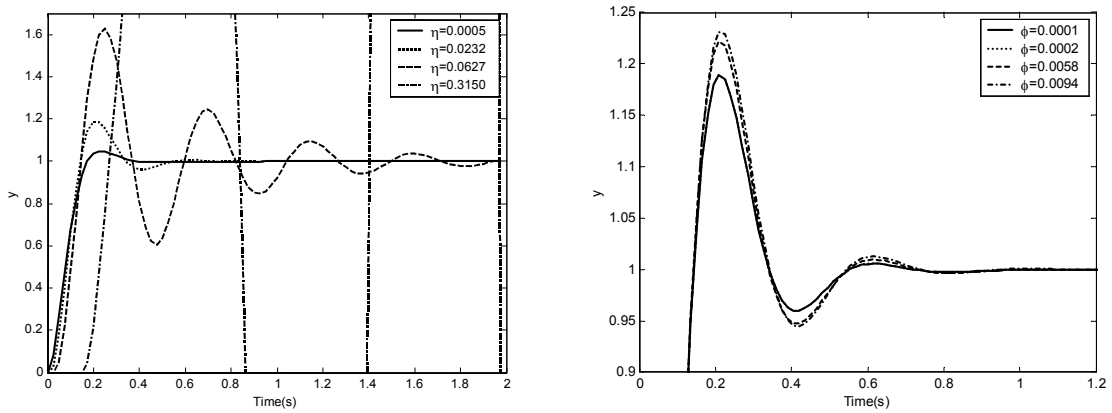
Both η and ϕ have effects on the step response of the networked DC motor PI speed control system as shown in Table 4, Table 5, and Fig. 7. The η and ϕ parameters shown in the tables and figure are adopted from τ_{median} and $\tau_{\text{mean}} - \tau_{\text{median}}$ in Table , respectively. The RTT delay data sets used are pre-computed from these parameters by using Matlab 6.1.

Table 4. Cost J from the networked DC motor PI speed control system simulation with various η while holding constant at $\phi = 0$.

η	0.0005	0.0232	0.0627	0.3150
J	0	1.6171	28.0574	4.6974e+033

Table 5. Cost *J* from the networked DC motor PI speed control system simulation with various ϕ while holding η constant at $\eta = 0.0232$.

ϕ	0.0001	0.0002	0.0058	0.0094
J	1.6171	1.6219	2.4693	2.7568



(a) (b)

Fig. 7. Effects of η and ϕ on the step response of the networked DC motor PI speed control system: (a) η is varied while holding ϕ constant at $\phi = 0$. (b) ϕ is varied while holding η constant at $\eta = 0.0232$.

As demonstrated in Table 4, Table 5, and Fig. 7, higher values of η and ϕ can degrade the system step response performance, and result in higher *J*. The obvious performance degradation in Fig. 7 is the increasing *P.O.* However, when ϕ is very small, a small increment in ϕ (i.e., 0.0001 to 0.0002) may not obviously result in system performance degradation. As shown in Fig. 7 (b), the step response for $\phi = 0.001$ is almost identical to the step response for $\phi = 0.002$.

To investigate the performance of the β scheduler middleware, three scenarios are tested and compared in the simulations:

1. The DC motor is controlled over IP networks by the PI controller with the nominal gains $\left(K_P^0, K_I^0\right)$.

2. The DC motor is controlled over IP networks by the PI controller with a fixed β. The fixed β is obtained by pre-estimating η and ϕ from an RTT delay data set.

3. The DC motor is controlled over IP networks by the PI controller with β gain scheduler middleware using η and ϕ from real-time measurements as described in the previous section.

First, the three scenarios are simulated by using pre-computed RTT delay data sets. The RTT data sets are generated for 4 sets to simulate 4 cases: 1) $\eta = 0.0005$, $\phi = 0.0001$, 2) $\eta = 0.0232$, $\phi = 0.0094$, 3) $\eta = 0.0627$, $\phi = 0.0002$, 4) $\eta = 0.3150$, $\phi = 0.0058$. The parameters in cases 1 to 4 are computed from τ_{median} and $\tau_{\text{mean}} - \tau_{\text{median}}$ of www.lib.ncsu.edu, www.visitnc.com, www.utexas.edu, and www.ku.ac.th, in Table 2, respectively, in order to simulate the networked control system in similar environments to the actual IP networks. The costs *J* from the simulations on the three scenarios are shown in Table 6, whereas the step responses from all 4 cases are illustrated in Fig. 8.

Table 6. Cost *J* from the networked DC motor PI speed control simulation using pre-computed RTT delays generated from (a) $\eta = 0.0005$, $\phi = 0.0001$. (b) $\eta = 0.0232$, $\phi = 0.0094$. (c) $\eta = 0.0627$, $\phi = 0.0002$, and (d) $\eta = 0.3150$, $\phi = 0.0058$.

Control scheme	**Parameters for RTT delay generation**			
	$\eta = 0.0005$, $\phi = 0.0001$	$\eta = 0.0232$, $\phi = 0.0094$	$\eta = 0.0627$, $\phi = 0.0002$	$\eta = 0.3150$, $\phi = 0.0058$
Nominal gains	0	2.7568	28.0577	6.5781e+033
Fixed β	0	4.5827e-005	2.0504e-004	0.0063
β scheduler	0	5.0967e-005	2.0195e-004	0.0101

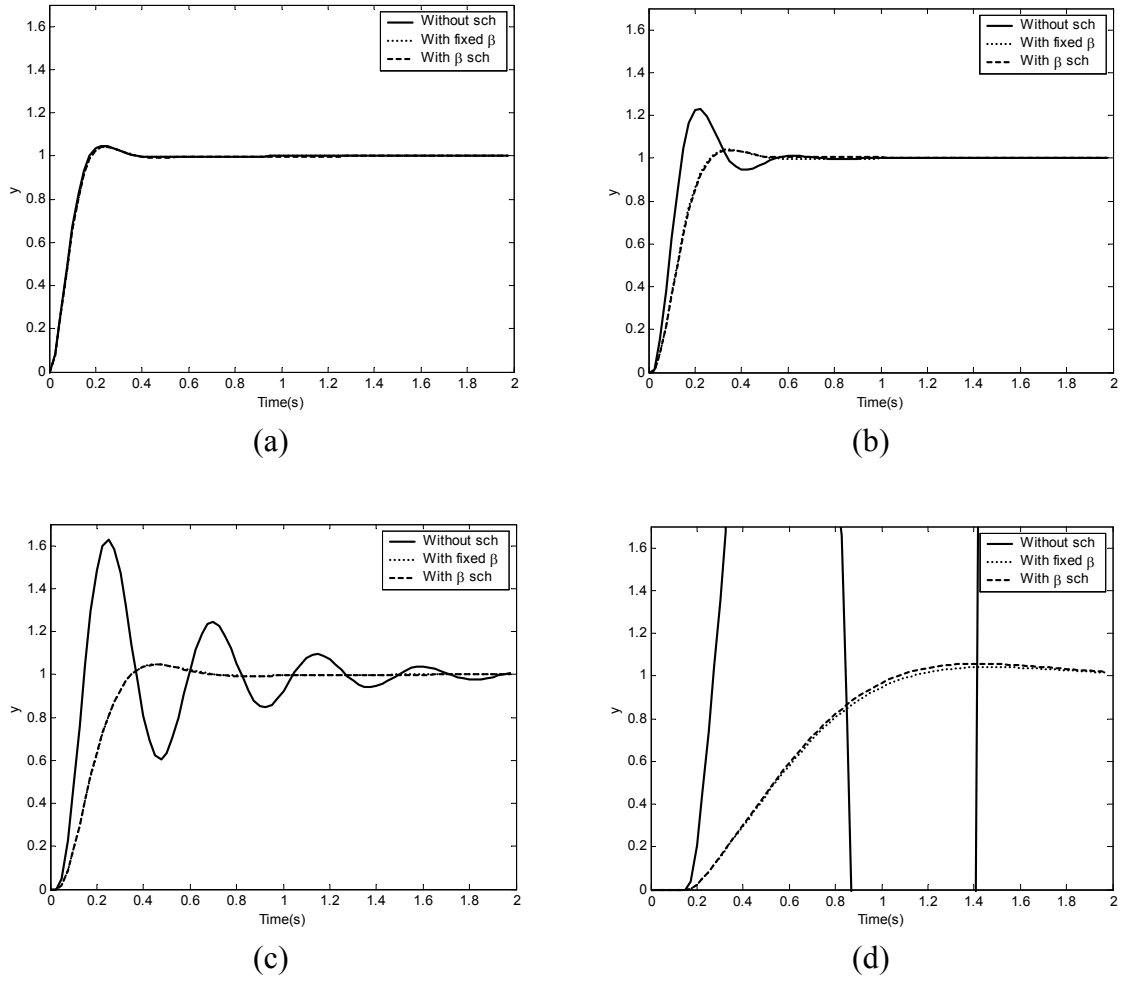


Fig. 8. Step responses from the networked DC motor PI speed control simulation using pre-computed RTT delays generated from (a) $\eta = 0.0005$, $\phi = 0.0001$. (b) $\eta = 0.0232$, $\phi = 0.0094$. (c) $\eta = 0.0627$, $\phi = 0.0002$, and (d) $\eta = 0.3150$, $\phi = 0.0058$.

As shown in Table 6 and Fig. 8 (a), the networked PI controller with the nominal $\left(K_P^0, K_I^0\right)$ can still meet the design specifications listed in section II when RTT delay and its variation is very low (e.g., $\eta = 0.0005$, $\phi = 0.0001$). In this case, $J = 0$ because the nominal $\left(K_P^0, K_I^0\right)$ meets the design specifications at the nominal condition. There is still a gap for the networked control system to degrade by IP network delays before violating the performance requirements. However, when RTT delay is longer and with more variation (e.g. $\eta = 0.0232$, $\phi = 0.0094$ and $\eta = 0.0627$, $\phi = 0.0002$), the system performance obviously degrades as shown in Table 6,

and Fig. 8 (b) and (c). Moreover, the system can become unstable when RTT delay is very long with very high variation (e.g. $\eta = 0.3150$, $\phi = 0.0058$) as shown in Table 6 and Fig. 8 (d).

On the other hand, the networked PI control system using fixed β and β scheduler middleware could maintain the system performance well. As shown in Table 6 and Fig. 8 (a), both control schemes can satisfy the performance requirements by resulting in $J = 0$ because RTT delay and its variation is relatively low. With longer RTT delay and more delay variation, the requirements cannot be satisfied as shown in Table 6, and Fig. 8 (b) and (c). Nevertheless, both schemes can still maintain the system performance in a satisfactory level. The performance is much better than the system with the nominal $\left(K_P^0, K_I^0\right)$ since the PI controller gains in this case are adapted to be more suitable for the network traffic conditions. Nevertheless, as shown in Table 6 and Fig. 8 (d), both schemes cannot perfectly maintain the system performance to meet the specifications, but still can reasonably stabilize the networked control system.

To investigate the effectiveness of the β scheduler middleware on actual RTT delays, the same simulations are repeated except that the RTT delay data sets in cases 1 to 4 are replaced by actual RTT delays measured from ADAC Lab at NCSU to the destinations in Table 1. The costs *J* from the simulations with actual RTT delays on the three scenarios are shown in Table 7, whereas the step responses from the simulations are illustrated in Fig. 9.

Table 7. Cost *J* from network DC motor PI speed control simulation using RTT delays from ADAC Lab at NCSU to: (a) www.lib.ncsu.edu. (b) www.visitnc.com. (c) www.utexas.edu. (d) www.ku.ac.th.

Control scheme	**Destination host**			
	www.lib.ncsu.edu	**www.visitnc.com**	**www.utexas.edu**	**www.ku.ac.th**
Nominal gains	0	1.7634	28.0574	7.1413e+033
Fixed β	0	4.9965e-005	2.0517e-004	0.0062
β gain scheduling	0	3.6111e-005	2.0147e-004	0.0208

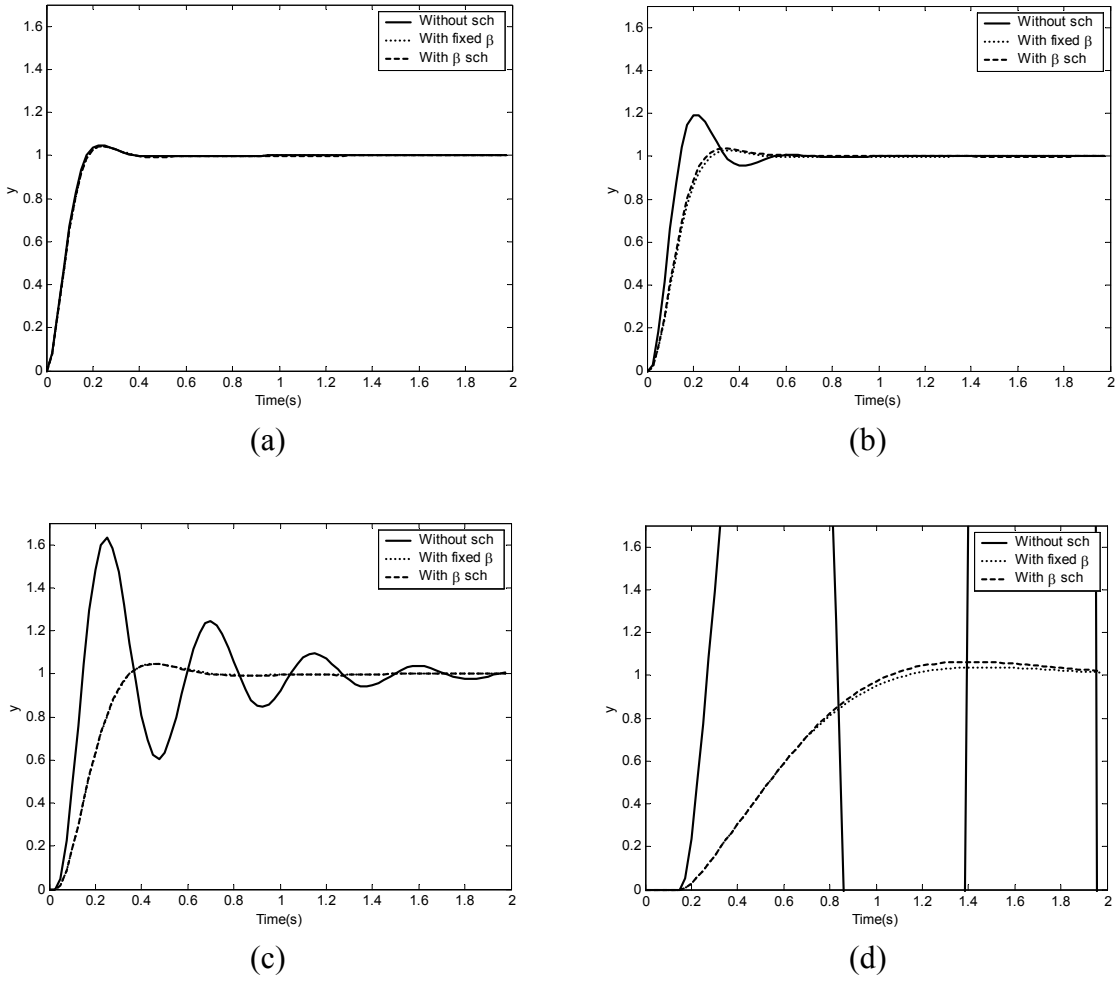


Fig. 9. Step responses from the networked DC motor PI speed control simulation using RTT delays from ADAC Lab at NCSU to: (a) www.lib.ncsu.edu. (b) www.visitnc.com. (c) www.utexas.edu. (d) www.ku.ac.th.

As shown in Table 7 and Fig. 9, the networked DC motor PI speed control system simulations with measured RTT delays give similar step response behaviors and costs J to the simulations with pre-computed RTT delays. Therefore, the same discussion on the control scheme performance can be drawn. There are small differences in the costs J in Table 6 and Table 7, which implies imperfect matching between RTT delays generated exactly by η and ϕ and RTT delays on actual IP networks. However, the similar behaviors of the step responses

can still indicate that the generalized exponential distribution is a good enough approximation for IP network delays to find the optimal β for a current traffic condition.

In addition, the PI controllers using the pre-computed optimal β and β gain scheduler middleware with respect to real-time RTT delay characteristics have similar performance. The only exception in these examples is that when RTT delays are very long with high variation (e.g., RTT delays from ADAC Lab to www.ku.ac.th). In actual IP networks, the probability density of RTT delay may vary, and sometimes the variation can be large. Therefore, the fixed optimal β gain approach may no longer be suitable in some situations such as IP network congestion. Using the β gain scheduling middleware to adjust the PI controller gains provides an acceptable and flexible control scheme in real IP network environment.

V. Conclusion

This paper has extended the concepts and methods described in Part I of the companion paper to enhance a PI controller for use over actual IP network delays based on RTT (Round-Trip Time) delay measurements and the generalized exponential distribution model. The generalized exponential distribution model is chosen for parameterization of IP networks RTT delays because this model can provide reasonably good accuracy and be quickly estimated in real-time to support the gain scheduling algorithm. Simulation results from the networked DC motor PI speed control problem using the generalized exponential distribution model as RTT delays and using actual RTT delays are similar, which indicates the good model accuracy.

The concept of finding the optimal β to schedule the PI controller externally at the controller output is extended to the distribution model by considering the median of RTT delays η as the constant delay in Part I and letting the delay variation be parameterized by ϕ. The same iterative gain searching algorithm can be applied, but the PI controller in this case has to be converted to a discrete-time controller. As indicated in simulations, higher η and ϕ parameters imply the smaller optimal β similar to the case with a longer constant delay in Part I.

The proposed gain scheduling approach iteratively estimates the parameters η and ϕ of the distribution model from a number of RTT delay measurements in real-time. The real-time gain scheduling has an advantage over a PI controller with fixed gains because the suitable PI gains can always be updated when the IP network traffic changes. When the IP network delay is low with a small variation, the gain scheduling approach does not result in the superior system performance than the controller with the nominal gains since the nominal gains can still satisfy the design specification. Under reasonably long random IP network delays, the gain scheduling can adapt the controller gain suitably for the current traffic condition and maintain the system performance in a satisfactory level much better than does the controller with the nominal gains.

The proposed approach promisingly permits PI controllers that are already available in industries to be used over IP networks. The research can be extended to include several concerns such as the optimal number of packet roundtrips to approximate the parameters η and ϕ, packet loss effects, and the experimental performance in order to improve and strengthen the gain scheduling approach. In addition, the system performance under different IP network QoS protocols as mentioned in Section I of Part I of this paper should be studied since the gain scheduling approach has the potential to gain advantages from an improved real-time IP traffic condition by IP QoS.

Acknowledgement

The authors would like to thank the Royal Thai Government for partially supporting this study, and Dr. Douglas S. Reeves for his helpful comments related to this paper.

References

[1] Y. Tipsuwan, and M.-Y. Chow, "Methodology of using networked PI controller gain scheduling over IP network: Part I – Foundation,".

[2] J. W. Park and J. M. Lee, “Transmission modeling and simulation for Internet-based control,” in IEEE IECON 2001, Denver, CO, 2001, pp. 165-169.

[3] K. J. Åström and B. Wittenmark, *Computer-Controlled Systems: Theory and Design*, 2 ed. Englewood Cliffs, NJ: Prentice-Hall, 1990.

[4] Y. Tipsuwan and M.-Y. Chow, "Fuzzy logic microcontroller implementation for DC motor speed control," in IEEE IECON 99, San Jose, CA, 1999, pp. 1271-1276.

APPENDIX FOR CHAPTER IV AND V

NEURAL-NETWORK-BASED GAIN SCHEDULING FOR NETWORKED PI CONTROLLER OVER IP NETWORK

I. Relationship Approximation by Neural Network

Although the optimal β may not have a closed form relationship with respect to η and ϕ, computational techniques such as artificial neural networks can be used to approximate this relationship with adequate accuracy. Artificial neural networks are composed of simple mathematical elements called neurons, which operate in parallel. Neural networks can be trained so that they can provide target output values with a specific set of input values. Both input and output sets for training are called together as a training set. Different types of neural networks can be used to approximate the relationship among the optimal β, η, and ϕ. A widely used 3-layer feedforward neural network is used for the approximation in this paper. The numbers of hidden neurons is defined as *h*. We have used $h = 3$, 5, 7, and 10 in the section to evaluate its performance with respect to different hidden neurons used.

To avoid a neural network to over fit a training set, and improve generalization to new situations, the optimal β, η and ϕ data collected from simulations can be divided into three sets. The first set is the training set. The second set is the validation set, which is used to check the performance of the neural network during training. While a network is trained, the error from the validation set as well as from the training set will usually decrease. However, the error from the validation set will begin to increase when the neural network over fits the training set. The third set is used as the testing set for a trained network. The size of the optimal β with respect to $\eta \times \phi$ to form the training, validation, and testing sets is $14 \times 26 = 364$ patterns,

where $\eta = 0.01,0.02,0.03,\ldots,0.09,0.1,0.2,0.3,\ldots,0.5$ and $\phi = 0,0.001,\ldots,0.019,0.02,0.03,$ $0.04,\ldots,0.07$. The data is separated as the training set for 243 patterns, as the validation set for 121 patterns, and as the testing set for 121 patterns. The network is trained by using Levenberg-Marquardt algorithm [1] with the mean-squared error tolerance of 4e-5. The training errors and validation errors during network training using different numbers of hidden neurons are shown in Fig. 1.

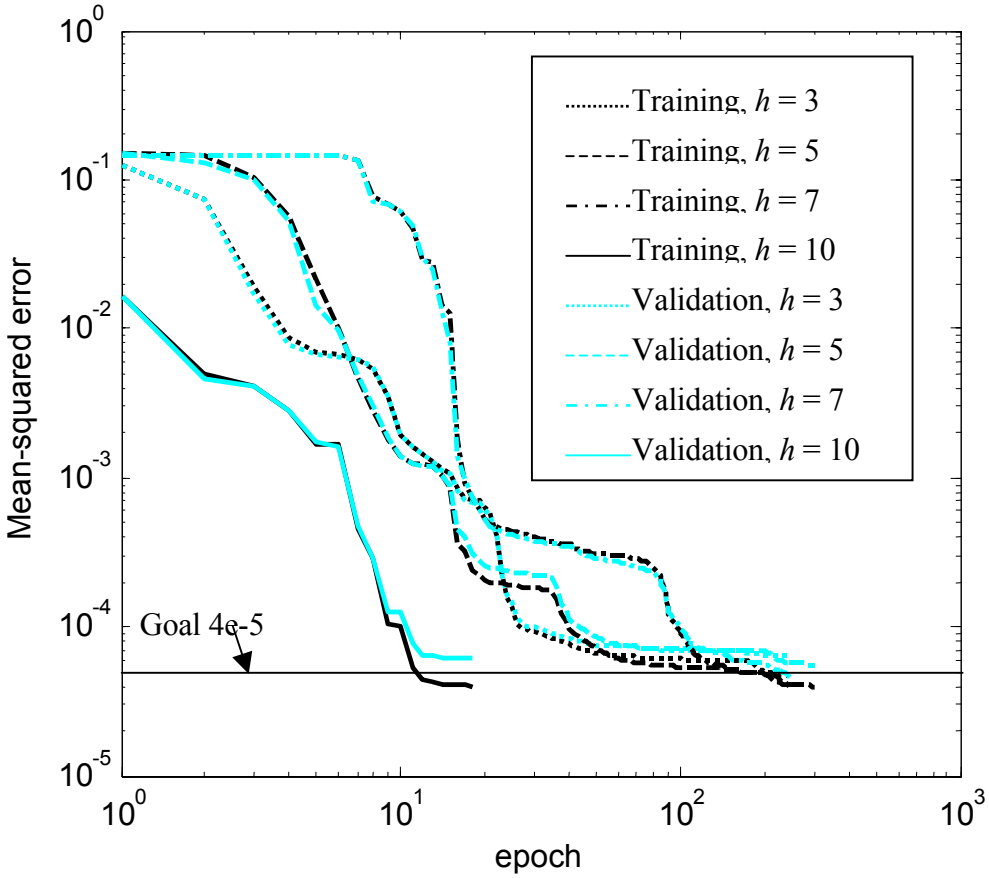


Fig. 1. Mean-squared errors from network trainings.

All network training can reach the training goal except the network with $h = 3$. This results indicate that three hidden neurons may not be enough to approximate the relationship among β, η, and ϕ in this case. The surfaces of the optimal β with respect to η and ϕ from the original 364 patterns used as a lookup table, and from the neural network with $h = 3,5,7,$ and

10 using the same set of η and ϕ are shown in Fig. 2. The mean-squared error from applying the testing set on these neural networks are shown in Table 1.

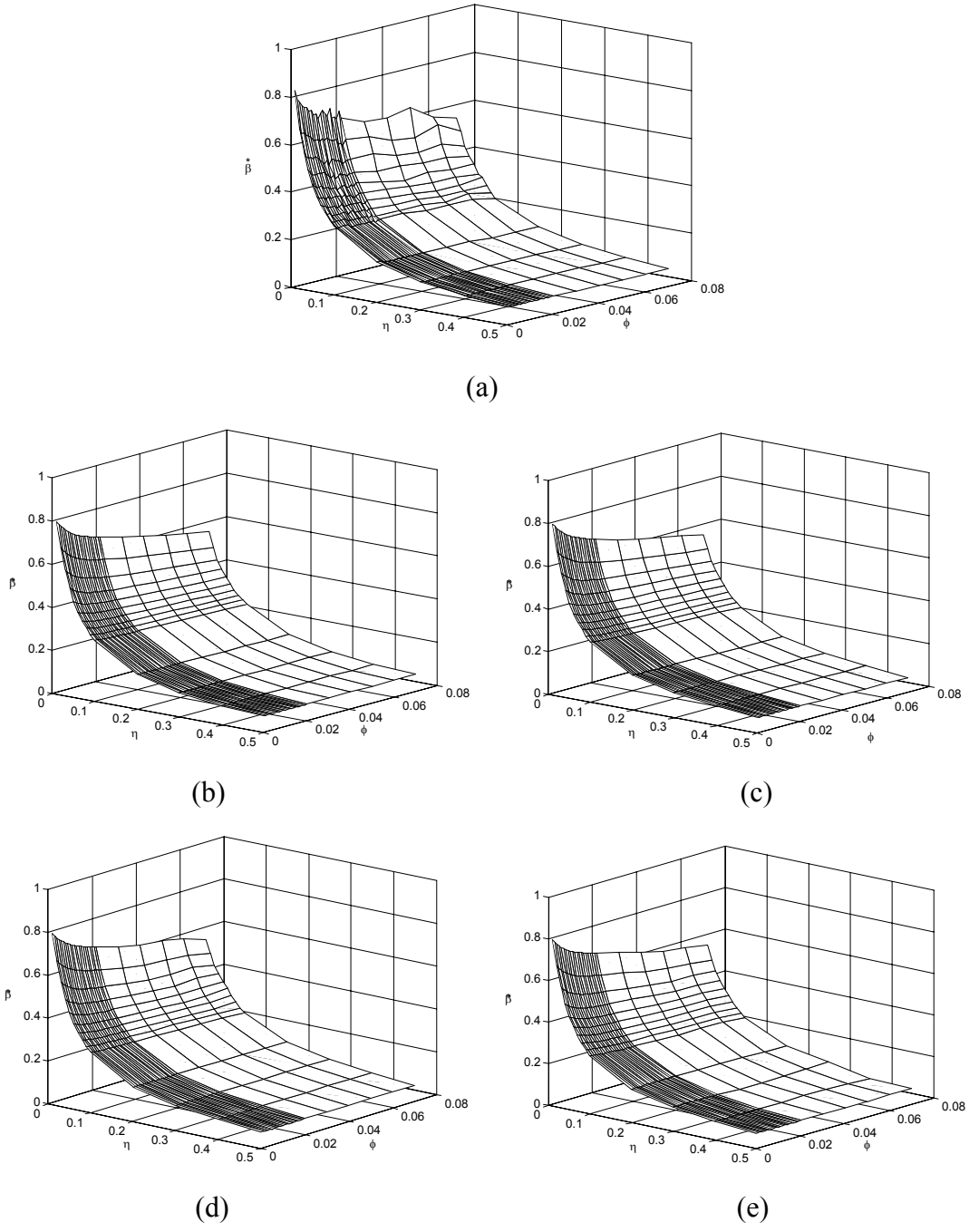


Fig. 2. Surfaces of the optimal β with respect to η and ϕ from (a) the lookup table and from neural networks with the number of neurons: (b) 3, (c) 5, (d) 7, and (e) 10.

Table 1. Mean-squared error from applying the testing set on neural networks with $h = 3$, 5, 7, and 10.

Number of neurons	3	5	7	10
MSE $\times$ **e-5**	5.438	5.058	4.201	6.106

The similar shape of the surfaces in Fig. 1 (b)-(e) to the lookup table surface in Fig. 1 (a) imply that the neural networks could provide reasonably good approximation for the lookup table. Nevertheless, as shown in Table 1, the mean-squared errors from the neural networks with $h = 3,5$ and 10 are higher than the training error tolerance. Thus, we will use the neural network with $h = 7$ to represent the relationship among the optimal β, η, and ϕ. Another consideration is the computational performance of the neural network. In Matlab, the computation time for finding the optimal β using any of four neural networks takes about 0.02 s, whereas the lookup table approach uses about 0.01 s. The lookup table approach has shorter computation time in this case due to the relatively small table size. However, the lookup table approach may not be effective when the size of the table is large. The neural network approach should be more effective since the order of computational iteration is $O(1)$, which implies the computation is finished in one iteration by its parallel structure. The lookup table approach usually uses more number of iterations to search for the optimal β. For example, a linear search algorithm for the optimal may have the order of $O(n^2)$. In addition, a neural network usually requires less memory than a lookup table to store parameters and variables especially when the size of the table is large. For example, the neural network with *h*=7 needs to store only $7 \times 3 = 24$ parameter values as the weights of the network, whereas the lookup table needs to store 364 parameter values.

II. Simulation result

The performance of the proposed β scheduler middleware approach using a neural network on the networked DC motor PI speed control system is verified by simulations

implemented on Matlab/Simulink 6.1. The following environment is used to illustrate the effectiveness of the proposed approach:

- The steady-state reference value: $c = 1$.
- The final simulation time: $t_f = 10$ sec.
- The sampling time of the PI controller, the β scheduler middleware, and the plant: $T = 1$ msec.
- The number of packets to evaluate the characteristic of RTT delay: $N = 100$.

To investigate the effectiveness of the β scheduler middleware on actual RTT delays, the four scenarios are simulated by the RTT delay data sets measured from ADAC Lab at NCSU to the destinations in chapter V:

1. The DC motor is controlled over IP networks by the PI controller with the nominal gains $\left(K_P^0, K_I^0\right)$.

2. The DC motor is controlled over IP networks by the PI controller with a fixed β. The fixed β is obtained by pre-estimating η and ϕ from an RTT delay data set.

3. The DC motor is controlled over IP networks by the PI controller with β scheduler middleware using the lookup table, and η and ϕ from real-time measurements.

4. The DC motor is controlled over IP networks by the PI controller with β scheduler middleware using the network with $h = 7$, and η and ϕ from real-time measurements.

In this paper, we assume that the delay from the β scheduler middleware to the DC motor (τ_{CP}) and the delay from the DC motor to the β scheduler middleware (τ_{PC}) have similar characteristics. Therefore, we prepare these delays for the simulations by dividing all data points in an RTT delay data set by two, and selecting some values from this set to apply as τ_{CP}, and τ_{PC}. Each value of τ_{CP} chosen from data points in the RTT data set is different from the data points used for τ_{PC}. The costs *J* from the simulations are shown in Table 2, whereas the step responses from the simulations are illustrated in Fig. 3.

Table 2. Costs J from network DC motor PI speed control simulations using RTT delays from ADAC lab at NCSU to: (a) www.lib.ncsu.edu. (b) www.visitnc.com. (c) www.utexas.edu. (d) www.ku.ac.th.

Control scheme / **Destination host**	Nominal gains	Fixed β	β gain scheduling	Neural network with $h = 7$
www.lib.ncsu.edu	0	0	0	0
www.visitnc.com	1.7634	4.997e-5	3.611e-5	3.611e-5
www.utexas.edu	28.1036	2.052e-4	2.015e-4	2.027e-4
www.ku.ac.th	7.14e+33	0.0062	0.0070	0.0061

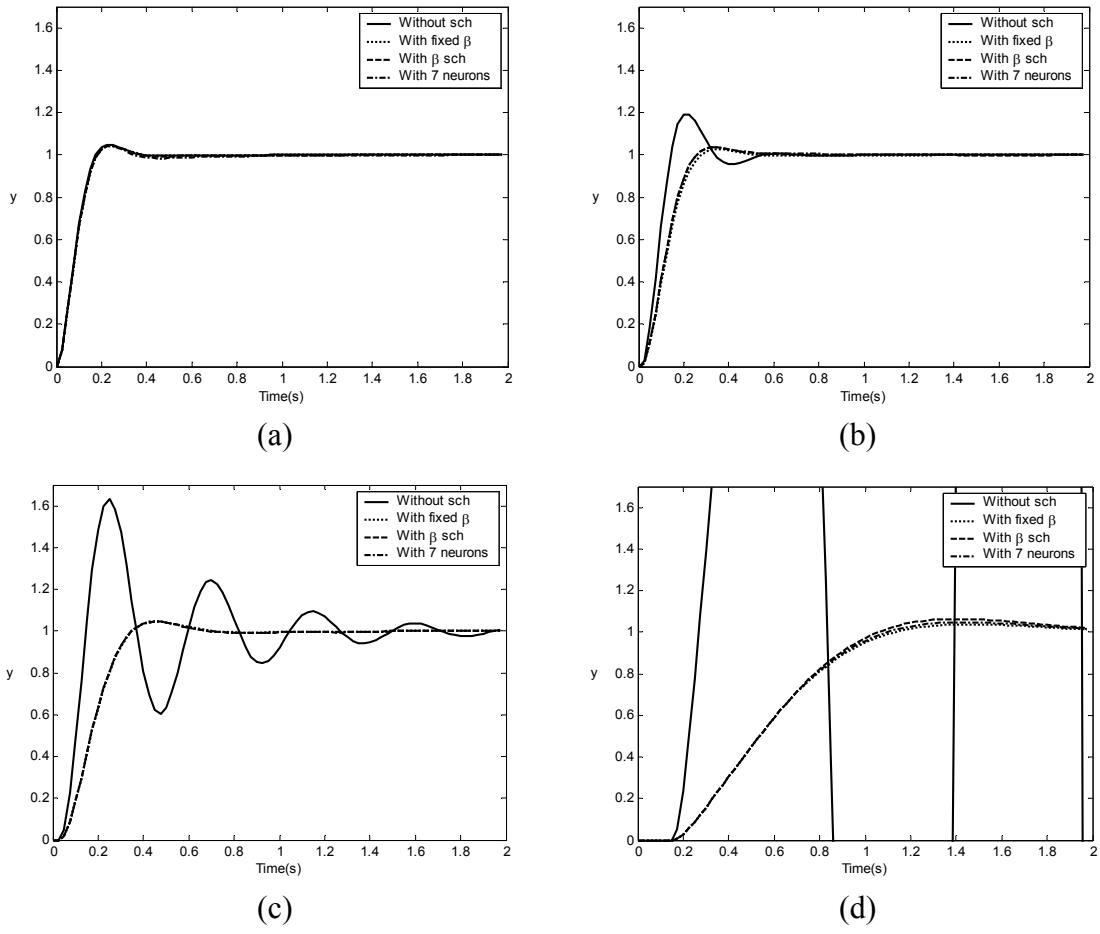


Fig. 3. Step responses from network DC motor PI speed control simulations using RTT delays from ADAC lab at NCSU to: (a) www.lib.ncsu.edu. (b) www.visitnc.com. (c) www.utexas.edu. (d) www.ku.ac.th.

The performance of the PI controller using the nominal gains is significantly lower when RTT delay is longer and its variation is larger as shown in Table 2 and Fig. 3. The networked PI control system using fixed β and β scheduler middleware with the lookup table and the neural network could maintain the system performance much better. As shown in Table 2 and Fig. 3 (a), all control schemes can satisfy the performance requirements by resulting in $J = 0$ because RTT delay and its variation are relatively low. With longer RTT delay and more delay variation, the requirements cannot be satisfied as shown in Table 2, and Fig. 3 (b) and (c). However, the performances from the fixed β and both β scheduler middleware schemes are satisfactorily maintained and is much better than the system with the nominal $\left(K_P^0, K_I^0\right)$. The PI controller gains in these case are adapted to be more suitable for the network traffic conditions. However, as shown in Table 2 and Fig. 3 (d), both schemes cannot perfectly maintain the performance to meet the specifications with very long RTT delay and high variation, but can still stabilize the system.

Noticeably, the PI controllers using the pre-computed optimal β and both β scheduler middleware schemes have similar performance. However, in actual IP networks, the probability density of RTT delay may vary, and sometimes the variation can be large. Therefore, this approach may no longer be suitable in some situations such as IP network congestion. Real-time adaptation by either β scheduler middleware schemes should be able to handle these situations better since the network traffic is always monitored. Nevertheless, the β scheduler middleware scheme with the lookup table may operate slowly if the table size is large. The neural-network-based approach could be a good alternative.

References

[1] M. T. Hagan and M. B. Menhaj, "Training feedforward networks with the Marquardt algorithm," *IEEE Transactions on Neural Networks*, vol. 5, no. 6, pp. 989-993, 1994.

C H A P T E R V I

GAIN SCHEDULER MIDDLEWARE FOR MOBILE ROBOT PATH-TRACKING OVER IP NETWORK

Yodyium Tipsuwan
ytipsuw@unity.ncsu.edu

Mo-Yuen Chow
chow@eos.ncsu.edu

Advanced Diagnosis And Control Lab
Department of Electrical and Computer Engineering
North Carolina State University, Raleigh NC 27606, USA
Tel: (919) 515-7360, Fax: (919) 515-5108

This chapter was submitted to an IEEE Transactions.

GAIN SCHEDULER MIDDLEWARE FOR MOBILE ROBOT PATH-TRACKING OVER IP NETWORK

Abstract—For robotic control applications to be used over a data network, conventionally, the robot controller needs to be redesigned or replaced by a new controller system and algorithms in order to compensate network delay effects. These processes are usually costly, inconvenient, and time-consuming. In this paper, a novel methodology to enable an existing robot controller for networked control by *middleware* is introduced. The proposed methodology uses middleware to modify the output of an existing robot controller based on a gain scheduling algorithm with respect to the current network traffic conditions. Since the existing robot controller can still be utilized, this approach could save much time and investment cost. A mobile robot path-tracking problem over an IP network is used as a case study to illustrate the effectiveness of the middleware approach. Simulation and experimental results on a mobile robot path-tracking platform show that the middleware approach can significantly maintain the robot path-tracking performance with the existence of IP network delays. The research results indicate the promising future of using middleware approach for IP network teleoperations.

Keywords—Internet, networks, adaptive control, control systems, DC motors, distributed control, real time system, mobile robots, telerobotics.

I. Introduction

Due to the rapid advancement in data and communication network technologies, especially the Internet, real-time networked control applications including teleoperation and remote mobile robots have gained increasing attentions in industries such as factory automation and industrial electronics. By organizing wiring connections among control system devices via network resources, networked control systems can be conveniently and systematically

maintained. Furthermore, this configuration also enables remote control operations. Nevertheless, the networked control system performance could be degraded and even become unstable by network-induced delays. The performance can be aggravated if the network delays are time-varying and random (e.g., IP network delays). Several techniques have been developed to handle network delay effects, and some promising results have been reported. These control methodologies are based on different techniques such as buffering [1, 2], nonlinear and perturbation theory [3], optimal stochastic control [4], optimal gain scheduling [5], and sampling time scheduling [6]. Some control techniques are developed for a specific kind of applications such as robots. These techniques include robust gain scheduling [7], wave variables [8, 9], and event-based control [10]. However, applying and implementing these control techniques on existing systems that are extensively being used in industrial plants could be costly, inconvenient, and time-consuming. The main reason is that all existing controllers may have to be redesigned, replaced, or, reinstalled in order to be used over data networks.

Recently, there have been several efforts to apply middleware to assist networked control systems [11, 12]. Middleware is an implementation to seamlessly link applications and/or function calls together. In several implementations, middleware could also handle network resource allocation and reservation between two applications over a data network [13-16]. A networked control system can utilize middleware to achieve certain network conditions such as guaranteed bandwidth, delay bound, or loss rate, by negotiating with the network counterpart for resource reservation. However, the network requirements may not always be granted due to several reasons such as network traffic congestion, inadequate resources, or loss of a link.

This paper proposed a novel methodology to utilize middleware to enable an existing non-network-based mobile robot controller so it can be used for networked control. The proposed methodology applies middleware to modify the controller output with respect to the current network traffic condition. Controller output modification is performed based on gain scheduling. Since the controller does not need to be replaced, reinstalled, or redesigned, the

proposed approach can be cost-effective, and conveniently applied on existing systems [17]. To illustrate the effectiveness of the proposed middleware approach for complex industrial applications, a mobile robot path-tracking problem over an IP network is used as a case study. The structure of middleware is discussed in section II. The system description of the case study is described in section III. Use of middleware for the case study is explained in section IV. The effectiveness of the proposed approach is verified in section V. Finally, the paper concludes in section VI.

II. System Description

The middleware in this paper is defined as the *Gain Scheduler Middleware (GSM)*. We assume that the GSM handles all network connections between the controller and the remote system to be controlled over a network. These include typical network operations such as sending and receiving packets, and other general middleware operations such as negotiation and resource reservation. The structure of the GSM is shown in Fig. 1.

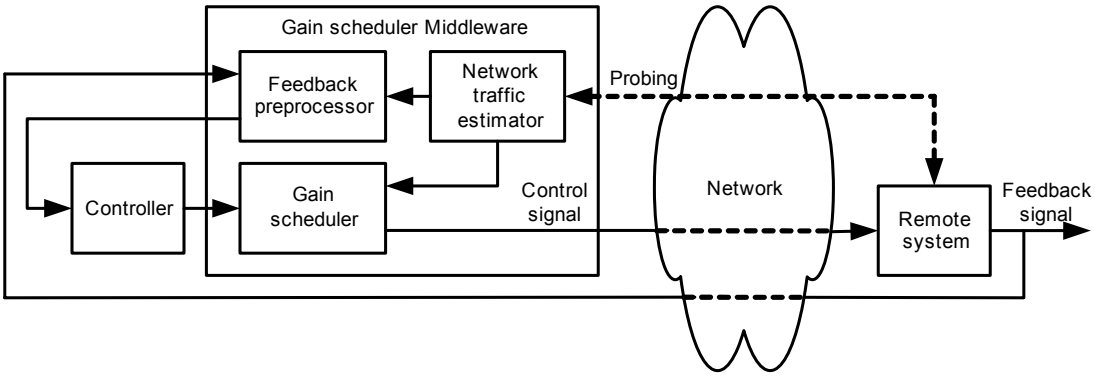


Fig. 1. Structure of gain scheduler middleware (GSM).

The basic components of the GSM shown in Fig. 1 are:

1. Network traffic estimator

The function of the network traffic estimator unit is to estimate the current network traffic conditions, which can be characterized into a set of network variables. For example, the

network variables in this case can be the statistics of the IP traffic data such as mean delay and loss rate. These network variables are then utilized by feedback preprocessor and gain scheduler, depending on the control algorithm used. After initialization at the beginning of the teleoperation process, network traffic estimator will periodically monitor the network conditions by sending a probing packet to the remote system with sampling time T_P. The estimator then characterizes the network conditions with the updated network variables based on the monitored probing packet roundtrip measurement.

2. Feedback preprocessor

The feedback preprocessor unit is used to preprocess the feedback data such as motor speed and current from the remote system before forwarding the signal to the controller. Preprocessing in this case can be, for example, filtering noises in the feedback data, or prediction of remote system states. Necessity of these operations depends on the gain-scheduling algorithm used in the gain scheduler. Thus, this part may or may not be required. In some cases of the GSM for a PI (Proportional-Integral) DC motor speed controller in [17], feedback preprocessor was not applied.

3. Gain scheduler

By using a gain-scheduling algorithm, the gain scheduler unit modifies the controller output with respect to the current network conditions (characterized by network variables). The algorithm to modify the controller output depends on the overall system configuration of the controller and the remote system.

The overall GSM operations for networked control can be summarized as follows:

1) Feedback preprocessor waits for feedback data from the remote system. Once the feedback data arrives, the preprocessor processes the data using the current values of network variables and passes the preprocessed data to the controller.

2) The controller computes the control signals and sends them to the gain scheduler.

3) The gain scheduler modifies the controller output based on the current values of network variables and sends the updated control signals to the remote system.

III. A Case Study: Mobile Robot Path-tracking

A mobile robot path-tracking problem is used to illustrate the GSM concept and its effectiveness. The robot model and path-tracking algorithm are described as follows:

A. The mobile robot model

The robot used to illustrate the proposed approach is a differential drive mobile robot with two driving wheels and two caster wheels [18] as shown in Fig. 2.

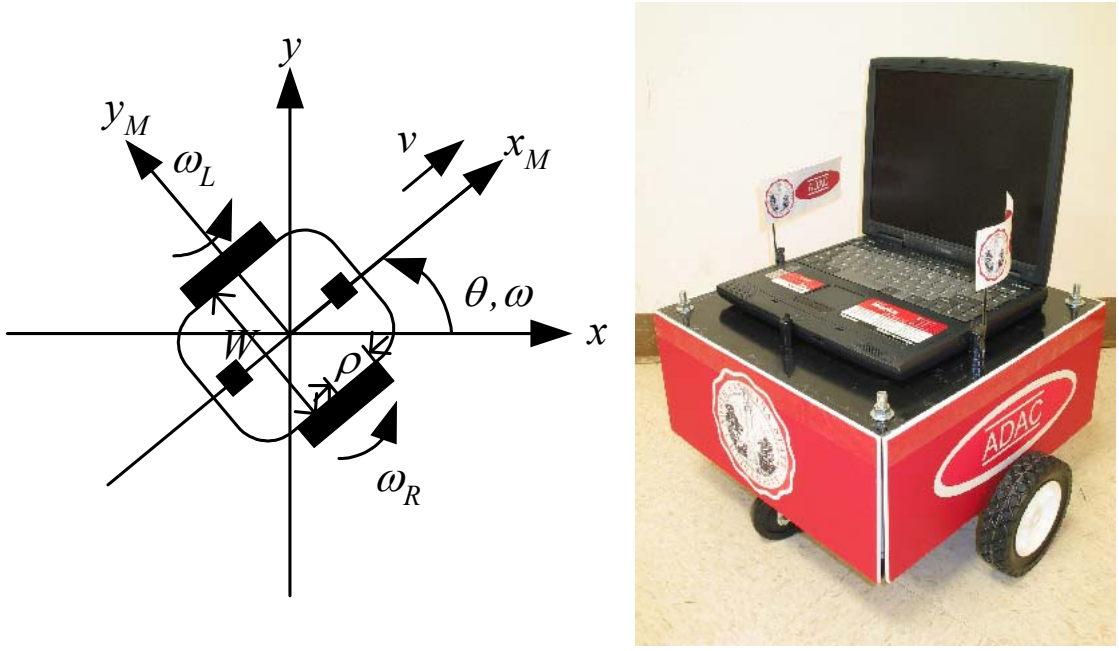


Fig. 2. Differential drive mobile robot: (a) Robot schematic drawing. (b) Actual mobile robot platform.

The mobile robot model can be described by the following equations:

$$\begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} \frac{\rho}{2} & \frac{\rho}{2} \\ \frac{\rho}{W} & -\frac{\rho}{W} \end{bmatrix} \begin{bmatrix} \omega_R \\ \omega_L \end{bmatrix}, \tag{1}$$

$$\begin{bmatrix} \omega_R \\ \omega_L \end{bmatrix} = \begin{bmatrix} \omega_{R,r} + \varepsilon_R \\ \omega_{L,r} + \varepsilon_L \end{bmatrix}, \tag{2}$$

$$\dot{x} = v\cos\theta, \tag{3}$$

$$\dot{y} = v\sin\theta, \tag{4}$$

$$\dot{\theta} = \omega, \tag{5}$$

where (x, y) is the position in the inertial coordinate, (x_M, y_M) is the position in the robot coordinate, θ is the azimuth angle of the robot, v is the linear velocity of the robot, W is the distance between the two wheels, ρ is the radius of the wheels, ω is the angular velocity of the robot, ω_L and ω_R are the angular velocities of the left and right wheels, $\omega_{L,r}$ and $\omega_{R,r}$ are the reference angular velocities for wheel speed controllers at the left and right wheels, and ε_L and ε_R are the differences between the reference velocities and the actual velocities of the left and right wheels, respectively. The speed of each wheel is controlled by a PI controller:

$$u(t) = K_P e(t) + K_I \int_0^t e(\xi) d\xi, \tag{6}$$

where K_P is the proportional gain, K_I is the integral gain, $u(t)$ is the input voltage to a DC motor, and $e(t)$ is the error, which can be either ε_L or ε_R.

B. Path-tracking algorithm

A generalization of the quadratic curve approach proposed in [18] is used as the path-tracking algorithm in our illustration. The main concept of this path-tracking algorithm is to move the robot along a quadratic curve to a reference point on a desired path. A point on the path is described in the inertial coordinate as $(x_p(s), y_p(s))$, where s is the distance traveled on the path. By assuming that the orientation of the mobile robot moves close to the desired value in the motion along the reference path, this algorithm controls only the position of the robot regardless to its orientation. This algorithm is suitable for real-time usage because of its simple computation with minimal amount of information compared to other approaches. The algorithm is outlined as:

1) Based on the current robot position $\mathbf{x}(i)=\begin{bmatrix} x(i) & y(i) & \theta(i) \end{bmatrix}^{\mathrm{T}}$, where $i \in \mathbb{N}^{+}$ is the iteration number, optimize:

$$\min_{s} \sqrt{\left(x_p(s)-x(i)\right)^2+\left(y_p(s)-y(i)\right)^2}\,, \tag{7}$$

to find $s=s_0$ for the *i*-th iteration that gives the closest distance between the robot and the path. Depending on the forms of $x_p(s)$ and $y_p(s)$, this optimization could be performed in real-time by using a closed-form solution, or a lookup table and a numerical technique such as linear interpolation. The iteration number *i* can be thought of as the sampling time index of the path-tracking controller if $t_{i+1}-t_i$ is constant.

2) Compute the reference point for the robot to track. Without loss of generality, in this paper, the path is constructed by a combination of lines and curves with different curvatures. Each line or curve has a constant curvature. Fig. 3 shows an example of a robot path, which is the combination of:

- Segment 1: Straight line:

$$x_p(s)=0\,,\ y_p(s)=s\,,\ \text{if } s_{e,0}\le s\le s_{e,1}, s_{e,0}=0, s_{e,1}=1\,, \tag{8}$$

- Segment 2: Arc with a radius of 1:

$$x_p(s)=1-\cos(s-1)\,,\ y_p(s)=1+\sin(s-1)\,,\ \text{if } s_{e,1}<s\le s_{e,2}, s_{e,2}=1+\pi\,, \tag{9}$$

- Segment 3: Arc with a radius of 0.2:

$$x_p(s)=2.2-0.2\cos 5(s-1-\pi)\,,\ y_p(s)=1-0.2\sin 5(s-1-\pi)\,,\ \text{if } s_{e,2}<s\le s_{e,3},$$
$$s_{e,3}=1+1.2\pi\,, \tag{10}$$

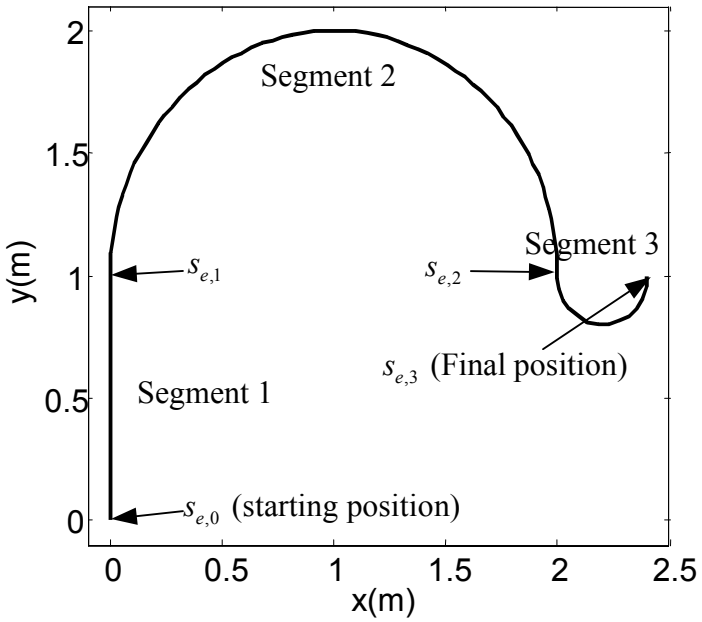


Fig. 3. Example of robot path.

where j is the index of the j-th segment of the path, $\kappa_j = \dfrac{d\theta_p(s)}{ds}$ is the curvature of the j-th segment, $s_{e,j}$ is the endpoint of the j-th segment. The reference position $\mathbf{x}_r(i) = \left[x_r(i) \quad y_r(i) \quad \theta_r(i) \right]^{\mathrm{T}}$ is computed from $x_r(i) = x_p\left(s(i)\right)$, $y_r(i) = y_p\left(s(i)\right)$, $\theta_r(i) = \theta_p\left(s(i)\right)$, where $s(i)$ is the reference distance traveled and is determined by the procedures shown in Fig. 4.

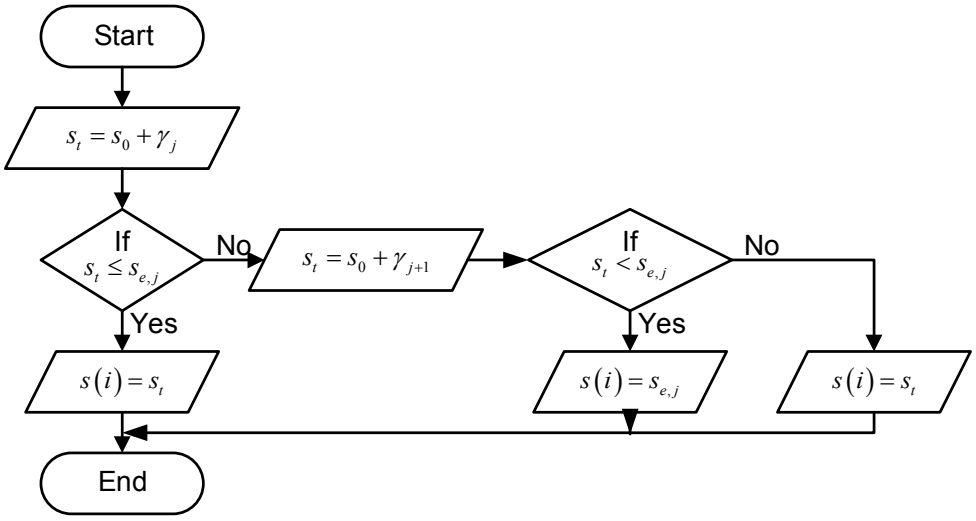


Fig. 4. Procedures for determining the reference distance traveled $s(i)$.

The value of $s(i)$ is initially determined from s_0 as:

$$s(i) = s_t = s_0 + \gamma_j, \gamma_j = \frac{s_{\max}}{1 + \beta\kappa_j}, \tag{11}$$

where s_t is a temporary variable, γ_j is the projecting distance, $s_{\max} \le s_{e,j} - s_{e,j+1}, \forall j$, is the maximal projecting distance, and $\beta \in \mathbb{R}^+$ is a positive constant. The projecting distance indicates how far the reference distance traveled should be projected ahead from $\left(x_p(s_0), y_p(s_0)\right)$. The values of the constants $s_{\max}$ and β depend on the robot path, the robot configuration, and the designer's preference, whereas the curvature κ_j depends only on the *j*-th segment of the path to track. The reference point will be closer to $\left(x_p(s_0), y_p(s_0)\right)$ if κ_j is high.

However, if $s(i) = s_t > s_{e,j}$, $s(i)$ will not be on the *j*-th segment. In this case, the controller needs to evaluate if the robot should track the path based on segment *j* or segment $j+1$. For evaluation, s_t is recomputed by:

$$s(i) = s_t = s_0 + \gamma_{j+1}. \tag{12}$$

When $s(i) = s_t < s_{e,j}$, $s(i)$ may be less than $s(i-1)$, which causes the robot to move backtrack if $\left(x_p(s(i)), y_p(s(i))\right)$ is used as the reference position. Therefore, the better choice of $s(i)$ in this case should be $s_{e,j}$ in order to guarantee that the robot will not move backtrack.

3) Compute the error $\mathbf{x}_r(i)-\mathbf{x}(i)$, and transform the error from the inertial coordinate to the robot coordinate as:

$$\mathbf{e}(i)=\begin{bmatrix} e_x & e_y & e_\theta \end{bmatrix}^{\mathrm{T}}=\begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}\left(\mathbf{x}_r(i)-\mathbf{x}(i)\right). \tag{13}$$

4) Find a quadratic curve that links between $\mathbf{x}(i)$ and $\mathbf{x}_r(i)$ from:

$$y_M = A(i)x_M^2, \text{ where } A(i)=\operatorname{sgn}(e_x)\frac{e_y}{e_x^2}. \tag{14}$$

The robot will move forward if $\mathbf{x}_r(i)$ is in front of the robot ($e_x>0$). On the other hand, the robot will move backward if $\mathbf{x}_r(i)$ is behind of the robot ($e_x<0$).

5) Compute the reference linear and angular velocities of the robot along the quadratic curve. The original equations of the velocities are:

$$v_r(i)=\operatorname{sgn}(e_x)\sqrt{\dot{x}_M^2\left(1+4A^2(i)x_M^2\right)}, \tag{15}$$

$$\omega_r(i)=\frac{2A(i)\dot{x}_M^3}{v_r^2(i)}. \tag{16}$$

Let x_M at $t_i \le t < t_{i+1}$ be given by:

$$x_M = K(i)(t-t_i), \tag{17}$$

where

$$K(i)=\operatorname{sgn}(e_x)\frac{\alpha}{1+|A(i)|}, \tag{18}$$

and α is a positive constant used as a speed factor. The robot will move fast if α is set to a high value, and vice versa. If $t-t_i$ is small (e.g., in the order of ms), $v_r(i)$ can be approximated during $t_i \le t < t_{i+1}$ by:

$$v_r^2(i)=K^2(i)\left(1+4A^2(i)K^2(i)(t-t_i)^2\right)\simeq K^2(i). \tag{19}$$

Thus, (15) and (16) can be approximated by:

$$\hat{v}_r(i)\simeq K(i), \tag{20}$$

$$\hat{\omega}_r(i)\simeq 2A(i)K(i). \tag{21}$$

The reference speeds of both wheels are then calculated by:

$$\omega_{R,r}(i) = \frac{\hat{v}_r(i)}{\rho} + \frac{W\hat{\omega}_r(i)}{2\rho}, \tag{22}$$

$$\omega_{L,r}(i) = \frac{\hat{v}_r(i)}{\rho} - \frac{W\hat{\omega}_r(i)}{2\rho}. \tag{23}$$

6) Repeat steps 1)-5) and set $i = i+1$.

IV. Use of GSM for Mobile Robot Path-Tracking Control over a Network

In order to control a mobile robot to track a predefined path over a network, the path-tracking controller computes and sends the reference speed $\omega_{L,r}(i)$ and $\omega_{R,r}(i)$ in a packet across the network at every iteration *i* to the robot as shown in Fig 5.

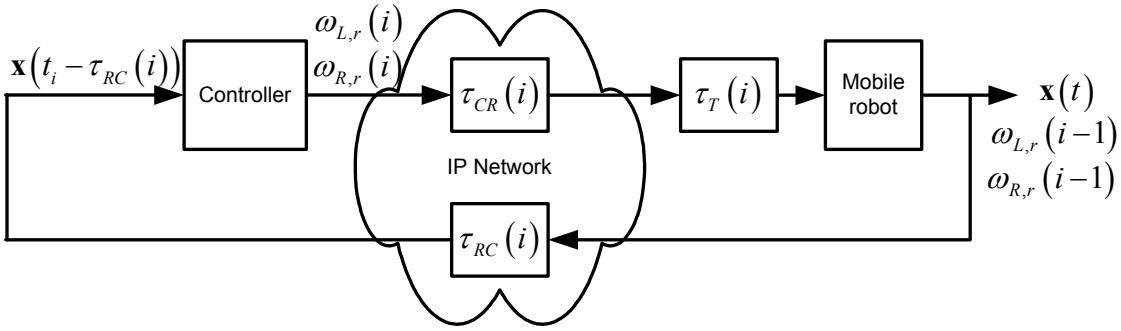


Fig. 5. Data flow of networked mobile robot.

The path-tracking computation at iteration *i* starts when the controller receives the feedback data in a packet from the mobile robot at time $t = t_i$. Compared with the network delays, the computation time at the controller is usually and relatively insignificant. Thus, the computation could be assumed to finish at $t = t_i$ as well. The basic arrival feedback data in this case are the reference speeds $\omega_{L,r}(i-1)$ and $\omega_{R,r}(i-1)$, and the robot position $\mathbf{x}(t_i - \tau_{RC}(i))$, where $\tau_{RC}(i)$ is the network delay from the robot to the controller at *i*. The controller then sends $\omega_{L,r}(i)$ and $\omega_{R,r}(i)$ to the robot once the computation is finished. Likewise, $\omega_{L,r}(i)$ and $\omega_{R,r}(i)$ are also delayed by the network. The network delay to send these reference speeds to the mobile robot is defined as $\tau_{CR}(i)$. The robot then periodically monitors and updates the

reference speeds by the newly arrival data of $\omega_{L,r}(i)$ and $\omega_{R,r}(i)$ at every sampling time period *T*. The waiting time to update the reference speeds is defined as $\tau_T(i)$.

A. Network delay effect on path-tracking algorithm

To modify the controller output with respect to network conditions characterized by the GSM, the effects of network delays on the mobile robot have to be analyzed. There are two concerns in the use of the original path-tracking algorithm due to $\tau_{CR}(i)$ and $\tau_{RC}(i)$:

1. Due to $\tau_{RC}(i)$, the controller does not have the current robot position $\mathbf{x}(t_i)$, but $\mathbf{x}(t_i - \tau_{RC}(i))$.

2. The reference speeds $\omega_{L,r}(i)$ and $\omega_{R,r}(i)$ are computed at $t = t_i$, but will be applied at $t = t_i + \tau_{CR}(i) + \tau_T(i)$.

If the controller directly uses $\mathbf{x}(t_i - \tau_{RC}(i))$ as $\mathbf{x}(i)$ to compute $\omega_{L,r}(i)$ and $\omega_{R,r}(i)$, and if $\mathbf{x}(t_i - \tau_{RC}(i))$ and $\mathbf{x}(t_i)$ are very different, then the result may be far away from what it actually should be. In addition, even if the controller uses $\mathbf{x}(t_i)$ to compute $\omega_{L,r}(i)$ and $\omega_{R,r}(i)$, the robot might have already moved to another position at $t = t_i + \tau_{CR}(i) + \tau_T(i)$ when $\omega_{L,r}(i)$ and $\omega_{R,r}(i)$ are applied. Thus, the robot response can be undesirable. The delay of $\omega_{L,r}(i)$ and $\omega_{R,r}(i)$ is crucial if the robot moves far away from a desired position. In addition, with long network delays, $t - t_i$ may be large, and the approximation in (19) may be no longer valid. Thus, the robot may not follow the desired quadratic trajectory.

B. Feedback preprocessor

In this paper, we use feedback preprocessor to predict the future position of the mobile robot at $t = t_i + \tau_{CR}(i) + \tau_T(i)$. The predicted position is then forwarded to the path-tracking controller. A future position at $t = t_i + \tau_{CR}(i) + \tau_T(i)$, defined as $\mathbf{x}(t_i + \tau_{CR}(i) + \tau_T(i))$, is predicted from $\mathbf{x}(t_i - \tau_{RC}(i))$. This predicted position, defined as $\hat{\mathbf{x}}(t_i + \tau_{CR}(i) + \tau_T(i))$, is then used instead of $\mathbf{x}(i)$ for the path-tracking controller. Thus, $\omega_{L,r}(i)$ and $\omega_{R,r}(i)$ should be

properly applied when the packet containing the reference speeds reaches the robot $t = t_i + \tau_{CR}(i) + \tau_T(i)$.

Before the mobile robot receives the reference speeds $\omega_{L,r}(i)$ and $\omega_{R,r}(i)$, both left and right wheels have been controlled by using $\omega_{L,r}(i-1)$ and $\omega_{R,r}(i-1)$ as the reference speeds. Thus, the robot can be assumed to move with constant linear and angular velocities if the wheel speed controllers of both wheels work perfectly such that $\varepsilon_L \to 0, \varepsilon_R \to 0$ quickly, and the weight of the robot is light. From this assumption, (3)-(5), (20), and (21), we can approximate the robot movement during $\left[t_i - \tau_{RC}(i),\ t_i + \tau_{CR}(i) + \tau_T(i)\right]$ using:

$$\begin{aligned}\boldsymbol{\Delta}_\tau \mathbf{x}(i) &= \mathbf{x}\left(t_i + \tau_{CR}(i) + \tau_T(i)\right) - \mathbf{x}\left(t_i - \tau_{RC}(i)\right), \\ &= \left[\Delta_\tau x(i) \quad \Delta_\tau y(i) \quad \Delta_\tau \theta(i)\right]^{\mathrm{T}}, \end{aligned} \tag{24}$$

where

$$\Delta_\tau \theta(i) \simeq \hat{\omega}_r(i-1)\tau(i), \tau(i) = \tau_{CR}(i) + \tau_T(i) + \tau_{RC}(i), \tag{25}$$

1) If $\hat{\omega}_r(i-1) \neq 0$:

$$\Delta_\tau x(i) \simeq \frac{\hat{v}_r(i-1)}{\hat{\omega}_r(i-1)}\left[\sin\theta\left(t_i + \tau_{CR}(i) + \tau_T(i)\right) - \sin\theta\left(t_i - \tau_{RC}(i)\right)\right], \tag{26}$$

$$\Delta_\tau y(i) \simeq \frac{\hat{v}_r(i-1)}{\hat{\omega}_r(i-1)}\left[\cos\theta\left(t_i - \tau_{RC}(i)\right) - \cos\theta\left(t_i + \tau_{CR}(i) + \tau_T(i)\right)\right], \tag{27}$$

2) If $\hat{\omega}_r(i-1) = 0$:

$$\Delta_\tau x(i) \simeq \hat{v}_r(i-1)\tau(i)\cos\theta\left(t_i - \tau_{RC}(i)\right), \tag{28}$$

$$\Delta_\tau y(i) \simeq \hat{v}_r(i-1)\tau(i)\sin\theta\left(t_i - \tau_{RC}(i)\right). \tag{29}$$

The delay variable $\tau(i)$ is estimated by the network traffic estimator. The predicted position $\hat{\mathbf{x}}\left(t_i + \tau_{CR}(i) + \tau_T(i)\right)$ is then computed from:

$$\hat{\mathbf{x}}\left(t_i + \tau_{CR}(i) + \tau_T(i)\right) = \mathbf{x}\left(t_i - \tau_{RC}(i)\right) + \Delta_\tau \hat{\mathbf{x}}(i). \tag{30}$$

where $\Delta_\tau \hat{\mathbf{x}}(i)$ is the approximation of $\boldsymbol{\Delta}_\tau \mathbf{x}(i)$ computed from (24)-(29).

C. Gain scheduler

To avoid the robot deviating far from a desired position, gain scheduler is used to first evaluate the predictive movement of the robot. If the robot tends to move too fast and could be

farther from the desired position because of network delays, gain schedule will modify $\omega_{L,r}(i)$ and $\omega_{R,r}(i)$ to compensate network delays $\tau(i)$ before sending the reference speed signals out. To evaluate the robot movement with respect to $\omega_{L,r}(i)$ and $\omega_{R,r}(i)$ ahead of time, we could utilize (24)-(30) to define the following cost function:

$$\begin{aligned} \text{Min } \hat{J}_1(i+1) &= \left\| \boldsymbol{\Delta}_\tau \hat{\mathbf{x}}(i+1) \right\|_2, \\ &= \left\| \hat{\mathbf{x}}\left(t_{i+1} + \tau_{CR}(i+1) + \tau_T(i+1)\right) - \hat{\mathbf{x}}\left(t_{i+1} - \tau_{RC}(i+1)\right) \right\|_2, \end{aligned} \tag{31}$$

$$\text{Min } \hat{J}_2(i+1) = -\left|K(i)\right|, \tag{32}$$

where $\|\bullet\|_2$ is the Euclidean norm, $\hat{\mathbf{x}}\left(t_{i+1} - \tau_{RC}(i+1)\right) \approx \hat{\mathbf{x}}\left(t_i + \tau_{CR}(i) + \tau_T(i)\right)$, and $\hat{\mathbf{x}}\left(t_{i+1} + \tau_{CR}(i+1) + \tau_T(i+1)\right)$ is the predicted position, which can be determined by using $\hat{\mathbf{x}}\left(t_{i+1} - \tau_{RC}(i+1)\right)$ and a predicted delay $\hat{\tau}(i+1) \approx \tau_{RC}(i+1) + \tau_{CR}(i+1) + \tau_T(i+1)$. Likewise, assume that $\hat{\tau}(i+1)$ is estimated by the network traffic estimator. The cost function $\hat{J}_1(i+1)$ represents the amount of robot movement with respect to the predicted network delay after the robot receives the reference speed signals $\omega_{L,r}(i)$ and $\omega_{R,r}(i)$. A large value of $\hat{J}_1(i+1)$ implies that the robot could significantly be affected by network delays. On the other hand, $\hat{J}_2(i+1)$ is linearly proportional to the speed of the robot since both $\omega_{L,r}(i)$ and $\omega_{R,r}(i)$ are a linear function of $K(i)$. Minimizing $\hat{J}_2(i+1)$ is equivalent to maximizing $\left|K(i)\right|$. Depending on the actual robot performance requirement (e.g., maximal efficiency control, minimal time control), other cost functions could be also used.

From (25)-(29), $\hat{J}_1(i+1)$ could be also expressed as:

1) If $\hat{v}_r(i) = 0$ and $\hat{\omega}_r(i) = 0$:

$$\hat{J}_1(i+1) = 0, \tag{33}$$

2) If $\hat{v}_r(i) = 0$ and $\hat{\omega}_r(i) \neq 0$:

$$\hat{J}_1(i+1) = \left|\hat{\omega}_r(i)\hat{\tau}(i+1)\right|, \tag{34}$$

3) If $\hat{v}_r(i) \neq 0$ and $\hat{\omega}_r(i) \neq 0$:

$$\hat{J}_1(i+1) = \sqrt{\frac{1-\cos\hat{\omega}_r(i)\hat{\tau}(i+1)}{2A^2(i)} + \left(\hat{\omega}_r(i)\hat{\tau}(i+1)\right)^2}, \tag{35}$$

4) If $\hat{v}_r(i) \neq 0$ and $\hat{\omega}_r(i) = 0$:

$$\hat{J}_1(i+1) = \left|\hat{v}_r(i)\hat{\tau}(i+1)\right|. \tag{36}$$

In a vigorous approach, to find the optimal $\omega_{L,r}(i)$ and $\omega_{R,r}(i)$, a weighted cost function based on $\hat{J}_1(i+1)$ and $\hat{J}_2(i+1)$ can be formed and an optimal control strategy can be applied to minimize $\hat{J}_1(i+1)$ and $\hat{J}_2(i+1)$. However, this approach may not be suitable for the path tracking algorithm used in real-time because the algorithm is highly nonlinear with uncertain delays and disturbances. A heuristic approach can provide a feasible solution by maximizing $\left|K(i)\right|$ while maintaining $\hat{J}_1(i+1) \leq \varepsilon$, where ε is defined as the tracking performance degradation tolerance. In this case, $\omega_{L,r}(i)$ and $\omega_{R,r}(i)$ are modified based on their original values so that the robot will move as fast as possible by minimizing $\hat{J}_2(i+1)$ while $\hat{J}_1(i+1)$ is maintained at an acceptable small value. This approach does not minimize $\hat{J}_1(i+1)$ as in the vigorous approach, but can provide feasible $\omega_{L,r}(i)$ and $\omega_{R,r}(i)$, which are optimal under the condition $\hat{J}_1(i+1) \leq \varepsilon$. In practice, gain scheduler will optimally modify $\omega_{L,r}(i)$ and $\omega_{R,r}(i)$ when $\hat{J}_1(i+1) > \varepsilon$ so that the robot will move as fast as possible based on $\hat{\tau}(i+1)$.

Because $A(i)$ is fixed by the path-tracking algorithm as the requirement of the robot trajectory in (21), the speed modification is equivalent to adjusting the gain $K(i)$ in (20) and (21). The optimal values of $\omega_{L,r}(i)$ and $\omega_{R,r}(i)$ in (34) and (36) can be determined by solving $\hat{\omega}_r(i)$ and $\hat{v}_r(i)$, respectively, whereas (35) requires a numerical method to solve (35) for $\hat{\omega}_r(i)$ to find the optimal $\omega_{L,r}(i)$ and $\omega_{R,r}(i)$. Since $\hat{\omega}_r(i) \simeq 2A(i)K(i)$, (35) could be viewed as a function of $A(i)$, $K(i)$ and $\hat{\tau}(i+1)$. Because $A(i)$ and $\hat{\tau}(i+1)$ are given, $\hat{J}_1(i+1)$ will be determined by $K(i)$. The optimal values of $K(i)$ with respect to $A(i)$ and $\hat{\tau}(i+1)$ subject to $\hat{J}_1(i+1) \leq \varepsilon$ can be found by computing $\hat{J}_1(i+1)$ from various combinations of $A(i)$, $K(i)$ and $\hat{\tau}(i+1)$ in actual ranges of operating conditions. Then, by fixing $A(i)$ and $\hat{\tau}(i+1)$, we can search for the optimal $K(i)$ with an iterative approach that gives $\hat{J}_1(i+1) \leq \varepsilon$. These optimal $K(i)$ values are then stored in a lookup table and will be

utilized by gain scheduler to compute the optimal $\omega_{L,r}(i)$ and $\omega_{R,r}(i)$. For example, Fig 6 shows the cost surfaces of $\hat{J}_1(i+1)$ with respect to $A(i)$, $K(i)$, and $\hat{\tau}(i+1)$ with $\varepsilon = 0.2$.

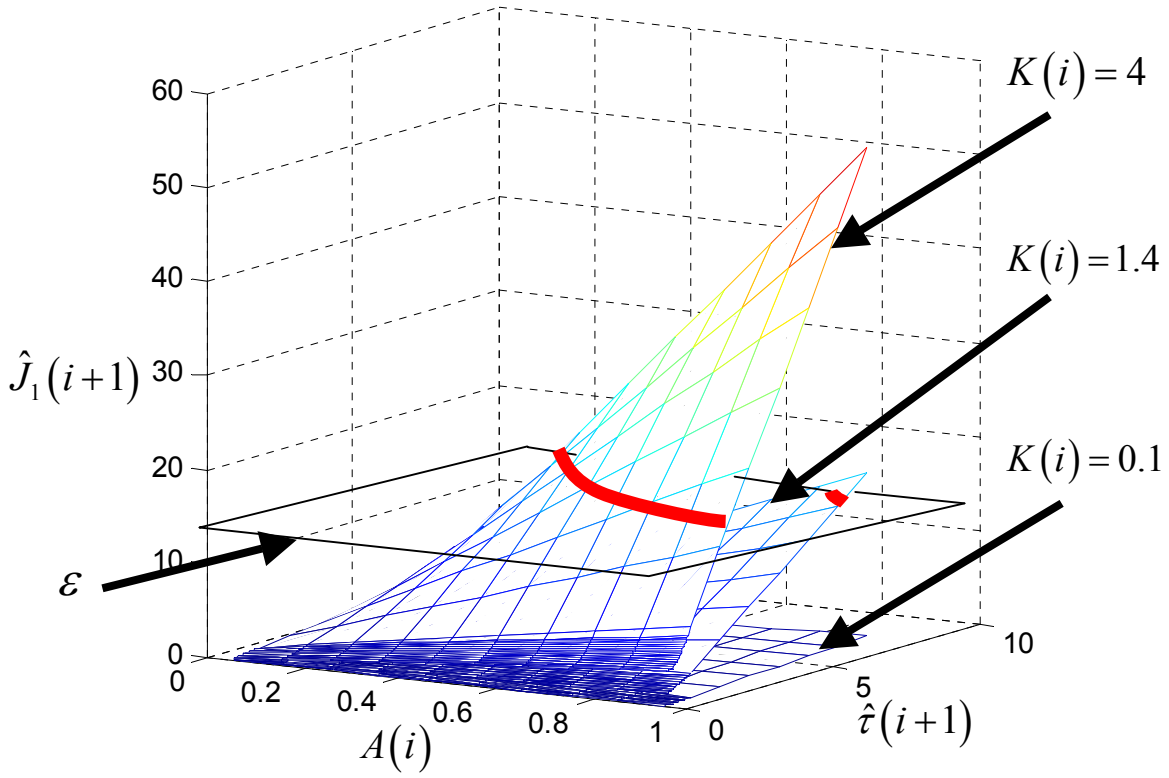


Fig. 6. Cost surfaces of $\hat{J}_1(i+1)$ with respect to $A(i)$, $K(i)$, and $\hat{\tau}(i+1)$ with $\varepsilon = 0.2$.

As shown in Fig. 6, ε can be thought of as a plane cutting through multiple surfaces of cost $\hat{J}_1(i+1)$ with different values of $K(i)$. The optimal $K(i)$ with respect to $A(i)$ and $\hat{\tau}(i+1)$ chosen to modify $\omega_{L,r}(i)$ and $\omega_{R,r}(i)$ in this case is the largest $K(i)$ that has to be under or at the ε plane.

Fig. 7 shows the surface of the optimal $K(i)$ with respect to $A(i)$ and $\hat{\tau}(i+1)$ with $\varepsilon = 0.2$.

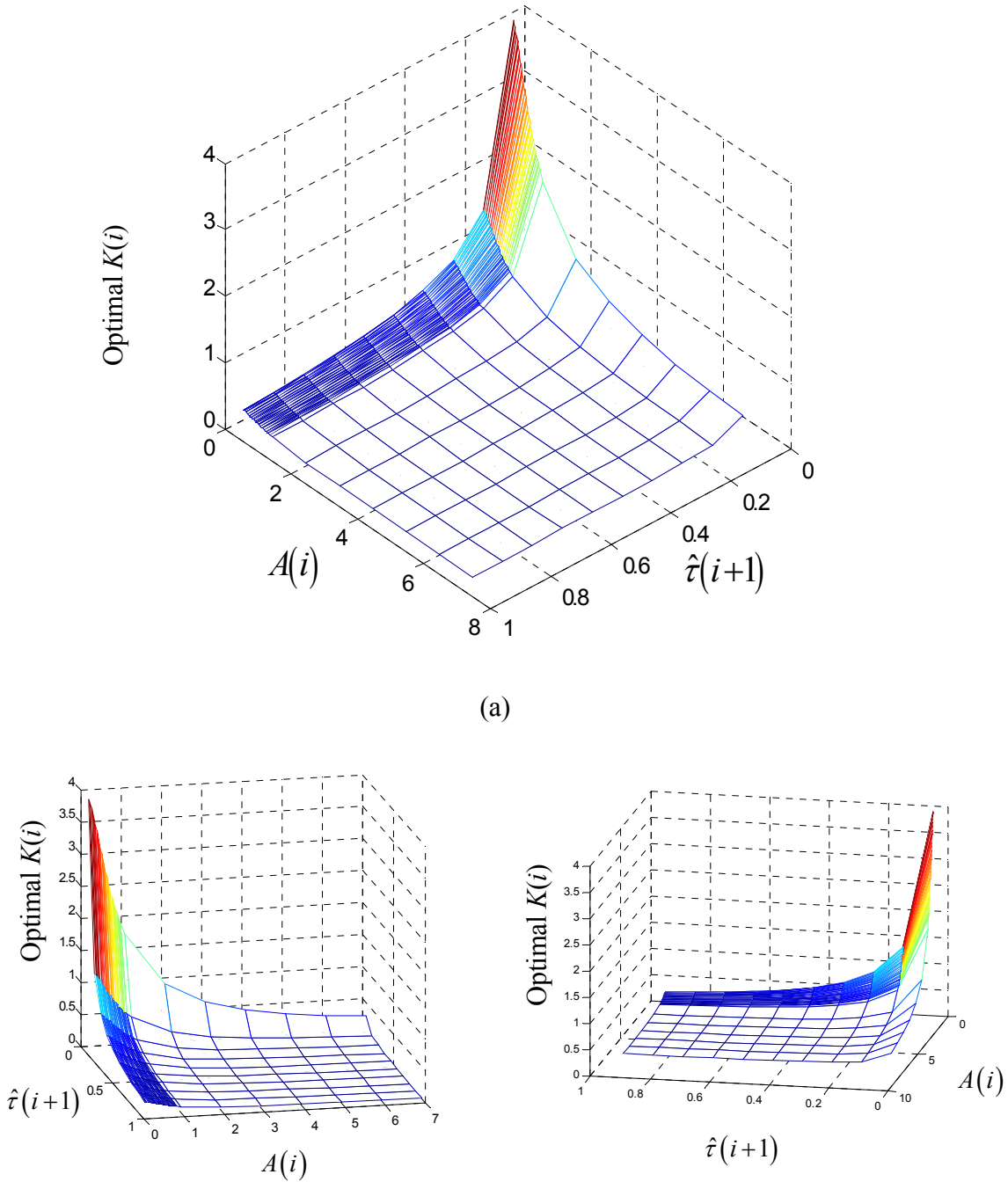


Fig. 7. Optimal $K(i)$ surface with respect to $A(i)$ and $\hat{\tau}(i+1)$ with $\varepsilon = 0.2$: (a) Front view. (b) Side view of. $A(i)$. (c) Side view of $\hat{\tau}(i+1)$.

As indicated in Fig. 7 (a), (b), and (c), if $A(i)$ and $\hat{\tau}(i+1)$ are low, the optimal $K(i)$ is large. This implies that the robot can move very fast if the curvature of the quadratic curve is small and the network delay is short. According to Fig. 7 (b), a larger $A(i)$ enforces the optimal $K(i)$ to be small because the GSM has to reduce the robot speed in order to follow the quadratic guideline with the higher curvature closely. Likewise, as shown in Fig. 7 (c), with a longer delay $\hat{\tau}(i+1)$, the GSM has to apply a small optimal $K(i)$ to reduce the robot speed so that the robot will not deviate far from the guideline and will still satisfy $\hat{J}_1(i+1) \le \varepsilon$.

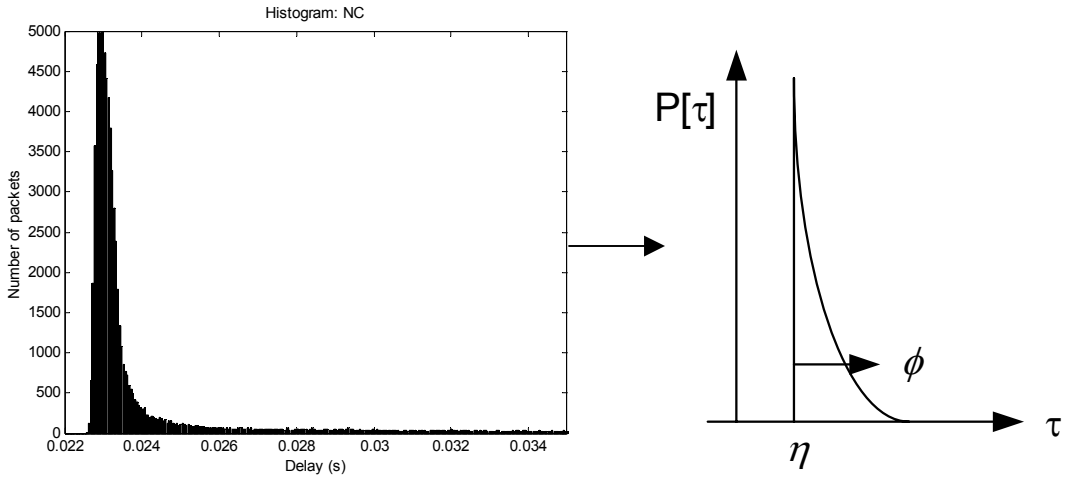
D. Network traffic estimator

Network variables representing network traffic conditions are typically subjective to the algorithm used in the feedback preprocessor and in the gain scheduler. Characterization of network conditions to network variables for use in both parts also depends on typical behaviors and characteristics of the network used. In this paper, we illustrate the GSM concept using delays from an actual IP network. As mentioned in earlier sections, the required network variable is the delay τ, which is estimated from the roundtrip time (RTT) delay measurements between the controller and the mobile robot on an IP network.

Several papers have proposed to approximate the RTT delay on IP networks by a generalized exponential distribution [17, 19]:

$$P[\tau] = \begin{cases} \frac{1}{\phi} e^{-(\tau-\eta)/\phi}, \tau \ge \eta, \\ 0, \qquad\quad \tau < \eta, \end{cases} \tag{37}$$

where the expected value of the RTT delay $E[\tau] = \phi + \eta$, and variance $\sigma^2 = \phi^2$. If η is known, ϕ can be easily approximated from η, and an experimental value of $E[\tau]$ or the mean μ by $\phi = E[\tau] - \eta$. A typical shape of (36) is depicted in Fig. 8.



(a) (b)

Fig. 8. (a) Typical histogram of RTT delays measured between ADAC (Advanced Diagnosis And Control) lab at North Carolina State University and North Carolina Department of Commerce, NC. (b) Typical probability density function of the generalized exponential distribution.

An important concern is what should be a good representative value of RTT delays to be used as the network variable τ. The feedback processor requires a delay value that is closed to the actually delay as much as possible. If RTT delays on an actual IP network are assumed to have the distribution similarly to the generalized exponential distribution, the median of RTT delays defined as $\mathrm{Med}(\tau)$ can be a good representative value [12]. In this case, a majority of RTT delays should not be much different from $\mathrm{Med}(\tau)$, and could be used in the feedback preprocessor. On the other hand, the gain scheduler requires the value of the delay τ so the networked mobile robot does not violate $\hat{J}(i+1) \le \varepsilon$. We can relax this value by using a slightly larger τ for some purposes such as reducing the effects from robot modeling errors or delay prediction errors in case that an actual RTT delay is larger than $\mathrm{Med}(\tau)$. By assuming the network traffic distribution is the generalized exponential distribution, we proposed to use the mean of RTT delays μ in this case, which is ideally larger than $\mathrm{Med}(\tau)$. However, in actual real-time traffic measurements with a limited number of probing packets, we may have

$\text{Med}(\tau) > \mu$. To handle this case, we propose to use the larger value between $\text{Med}(\tau)$ and μ::

$$\hat{\tau} = \max\{\text{Med}(\tau), \mu\}, \tag{38}$$

Both $\text{Med}(\tau)$ and μ in a specific time interval can be computed by sending probing packets as mention in section II.

V. Simulation and Experimental Results

A. Testing environment and parameters used for the simulation and experiment of the mobile robot path-tracking control over a network

1. IP network delay

To verify the effectiveness of the GSM concept, the RTT network delays of UDP (User Datagram Protocol) packets between ADAC (Advanced Diagnosis And Control) lab at North Carolina State University and Kasetsart University, Thailand, are measured for 24 hours (00:00-24:00). The reason to use UDP for networked control is to avoid additional delays from retransmission. The use of UDP is a common practice for real-time networked control applications. The measured RTT delay data is illustrated in Fig. 9 (a), whereas the histogram of the RTT delays is depicted in Fig. 9 (b). This data are used in the simulation and experimental setups of the networked mobile robot path-tracking control with the assumption that there is no packet loss. Each value of these RTT delays is divided by 2 and is utilized as τ_{RC} and τ_{CR}.

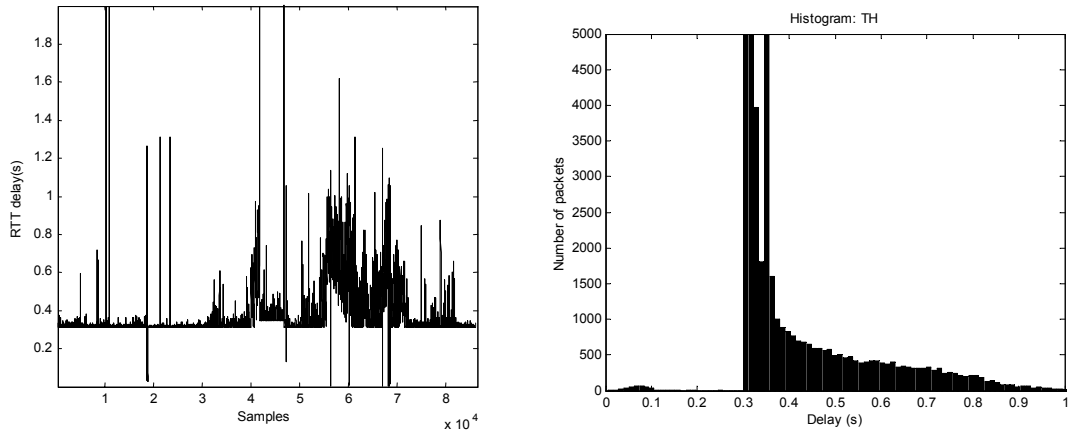


Fig. 9. (a) RTT delays between ADAC (Advanced Diagnosis And Control) lab at North Carolina State University and Kasetsart University, Thailand, measured for 24 hours (00:00-24:00). (b) Histogram of the corresponding RTT delays.

2. Path for the robot to track

The path of the robot used for simulation and actual experimental verification is the same path described in section III.

3. Controller and robot parameters

The controller and robot parameters used for the proposed GSM verification are listed in Table 1.

Table 1. Controller parameters

Parameter	Description	Value
$s_{\max}$	Maximal projecting distance	0.5
α	Speed factor	0.25
β	Projecting factor	0.5
ε	Cost tolerance	0.2
W	Distance between two wheels	0.4826 m
ρ	Radius of the wheels	0.073 m
T_P	Sampling time of probing packet transmission	0.01 s
T_C	Sampling time of path-tracking controller	0.1 s
T	Sampling time for robot to update reference speeds	0.002 ms

B. Simulation results

The networked mobile robot path-tracking simulation program is setup in a Matlab/Simulink environment to investigate the effectiveness of the GSM and is shown in Fig. 10.

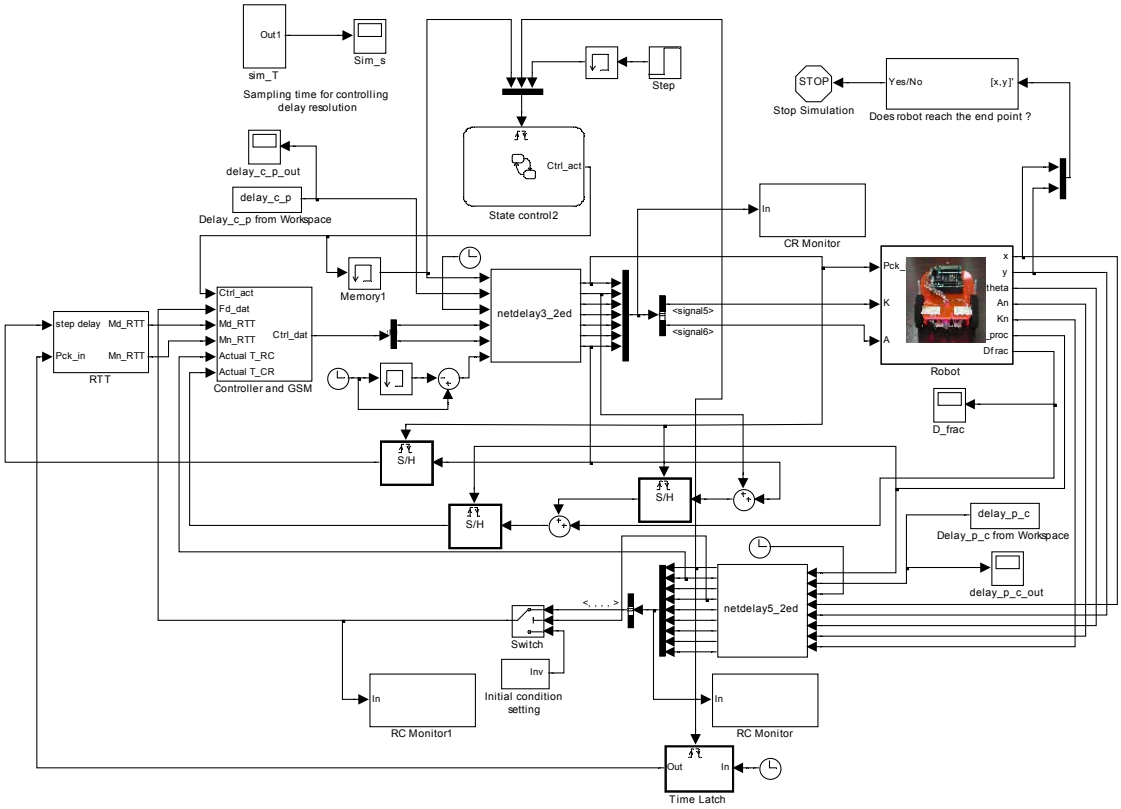


Fig. 10. Networked robot simulation setup in Matlab/Simulink.

The robot path-tracking program has five main blocks: *Controller and GSM*, *Robot*, *Netdelay3_2ed*, *Netdelay5_2ed,* and *RTT*. The *Controller and GSM* block computes and adjusts the reference speeds with respect to the mean and the median of RTT delays. The *Robot* block simulates the robot dynamics. The *Netdelay3_2ed* and *Netdelay5_2ed* blocks delay the outputs from the *Controller and GSM* block and the *robot* block, respectively, by using the delay data in section V.A preloaded from a text file. The *RTT* block computes the mean and median of RTT delays using the delay data from the *Netdelay3_2ed* and *Netdelay5_2ed*.

Three scenarios are compared:

1. The robot is controlled without IP network delay.

2. The robot is controlled with the existence of IP network delays from ADAC to Kasetsart University. GSM is not applied.

3. The robot is controlled with the existence of IP network delays from ADAC to Kasetsart University. GSM is applied.

The network traffic estimator is set to compute the mean and median of RTT delays for every 10 probing packet roundtrips. The initial position of the robot is arbitrarily set to $(-0.01,-0.01)$. The robot will stop if:

$$\sqrt{\left(x_p\left(s_{e,3}\right)-x(i)\right)^2+\left(y_p\left(s_{e,3}\right)-y(i)\right)^2}\le 0.05\,. \tag{39}$$

Fig. 11 shows the results from simulations.

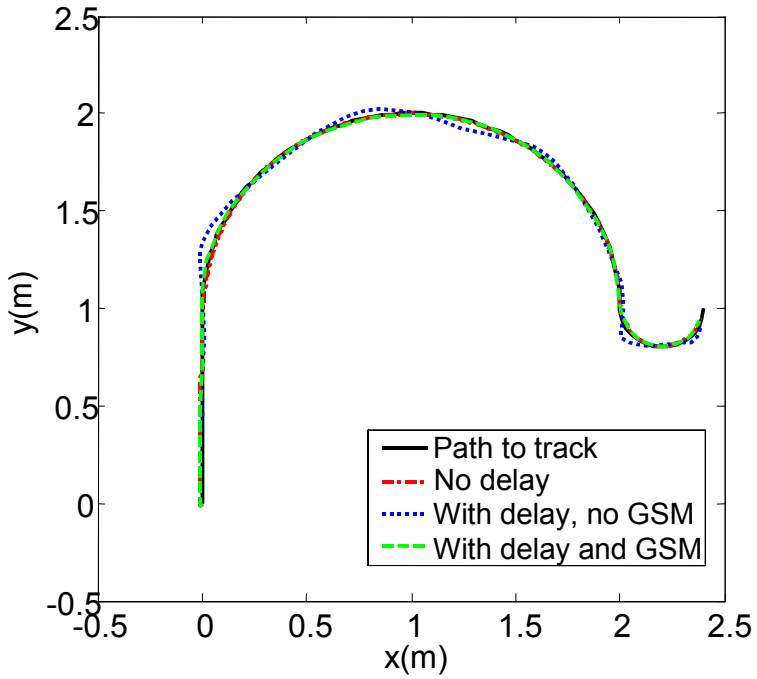


Fig. 11. Comparison of robot tracks from simulations: (a) Dashed-dotted: The robot is controlled without IP network delay; (b) Dotted line: The robot is controlled with the existence of IP network delays from ADAC to Kasetsart University. GSM is not applied; (c) Dashed line: The robot is controlled with the existence of IP network delays from ADAC to Kasetsart University. GSM is applied.

As shown in Fig. 11, the original path-tracking controller performs superbly when there is no IP network delay. The robot track is basically overlaps with the path to be tracked. With the existence of RTT network delays, the robot without GSM fails to track the path closely because the position feedback and the reference speeds are delayed by IP network delays. On the contrary, the robot with GSM can track the path much better. The predicted position applied for path-tracking computation and the gain scheduling scheme have compensated the network delay effects on the networked mobile robot system.

C. Experimental results

An experimental mobile robot platform is built to verify the effectiveness of the proposed GSM using the same delay scenario as in the simulations. The block diagram of the robot setup is illustrated in Fig. 12. The actual hardware setup is shown in Fig. 13.

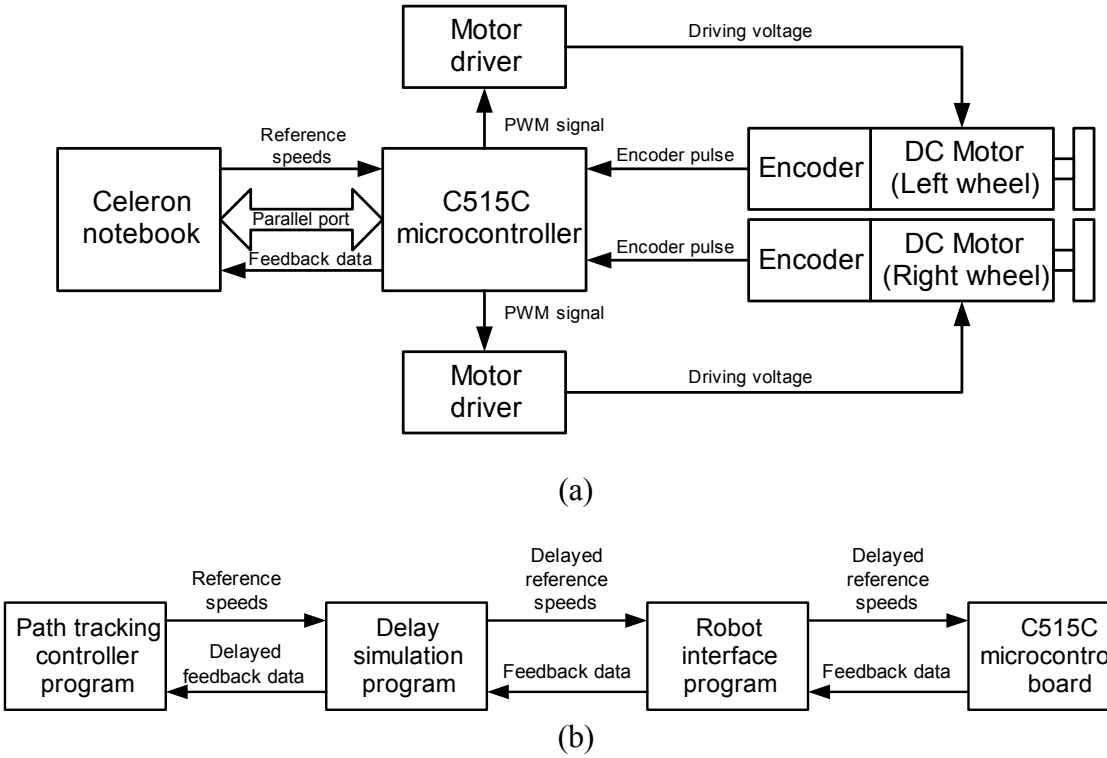


Fig. 12. Networked control robot experimental setup. (a) Hardware schematic diagram, (b) Software schematic diagram.

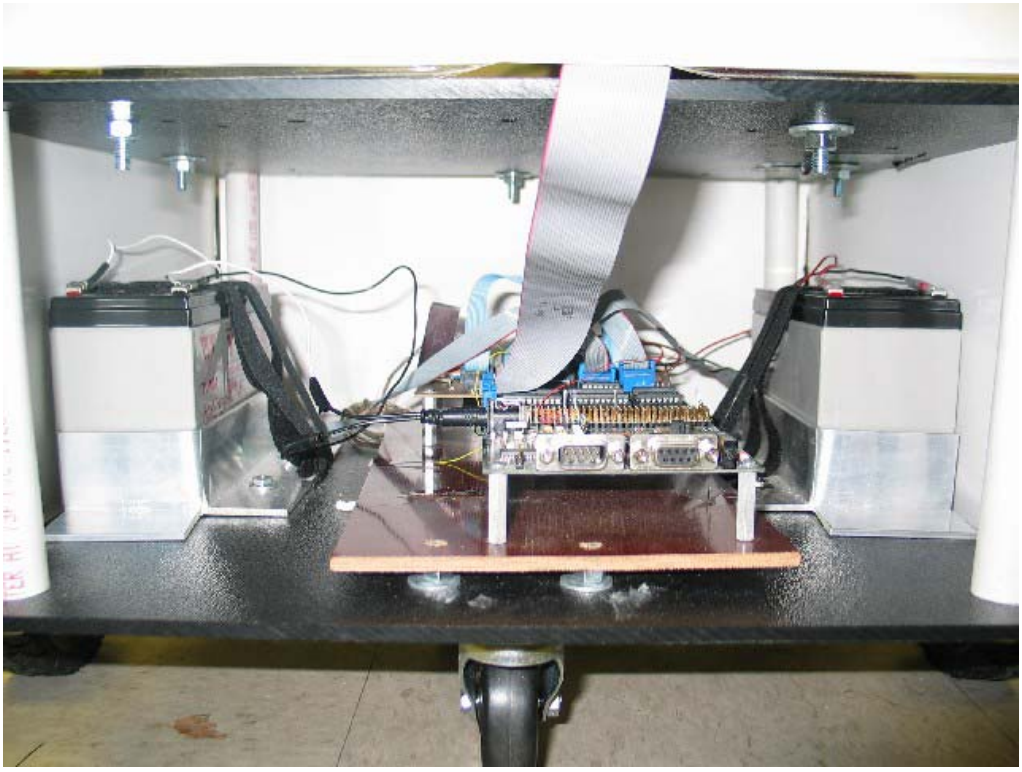


Fig. 13. Actual hardware setup.

The mobile robot platform is composed of a Celeron 1.5 GHz notebook computer, a C515C Siemens/Infineon microcontroller board, two LMD18200 motor driver chips, and two DC motors with two 500 pulses/rev optical encoders. The DC motors are Maxon A-max 236668 donated by Maxon Motor, USA. Each motor has a gearhead ratio 1:111. The notebook computer performs path-tracking computation and simulates IP network delay effects by using the actual IP network delay data collection, whereas the microcontroller board controls the speeds of the two DC motors by PI control algorithm.

The microcontroller board reads the motor rotations from the encoders and converts them to the feedback data. The feedback data is then sent to the notebook computer via a parallel port. The notebook computer uses this feedback data to compute the reference speeds of both DC motors, and sends them back to the microcontroller board via the parallel port as well. The

microcontroller board then applies these references to perform PI control computation, and send control signals in the forms of TTL-level PWM signals out to the motor drivers. The motor drivers will amplify the TTL-level signals to 0-12 V signals for driving the DC motors.

To focus specially on the effects of network delays, we create an experimental simulation scenario of IP network delays by delaying data transfers between the notebook computer and the microcontroller board using real-time software and a hardware timer. The reasons of using the collected IP delay data rather than using the real IP network are:

1) The experimental results can be compared with the simulation results using the same delay data.
2) The experiment is ensured to be repeatable for various future investigations.

This scenario is implemented by using three real-time programs running on RTLinux 3.2, a real-time operating system:

1. Delay simulation program

The delay simulation program delays data transfers between the path-tracking controller program and the robot interface program by using two linked-list structures. The delay applied in this program is the actual measured IP network delays in section V. A. The delay can range from 0 s to the maximal delay measurement with the resolution about 0.1 ms.

2. Robot interface program

The robot interface program handles parallel port communication between the notebook computer and the microcontroller board. The interface program receives the reference speeds from the path-tracking controller program via the delay simulation program. Likewise, it forwards the feedback data from the microcontroller board through the delay simulation program to the path-tracking controller program.

3. Path-tracking controller program

This program contains two parts: path-tracking implementation and the GSM. The path-tracking implementation performs the basic path-tracking algorithm computation as discussed

in Section II. The GSM is responsible for data communication with the robot interface program through the delay simulation program, for position prediction, and for the gain scheduling.

Fig. 14 shows the experimental results of the IP-based robot path-tracking. In addition, Fig. 15 shows the distance from the robot to the path. This distance indicates how close the robot is to the path when the robot is tracking the path.

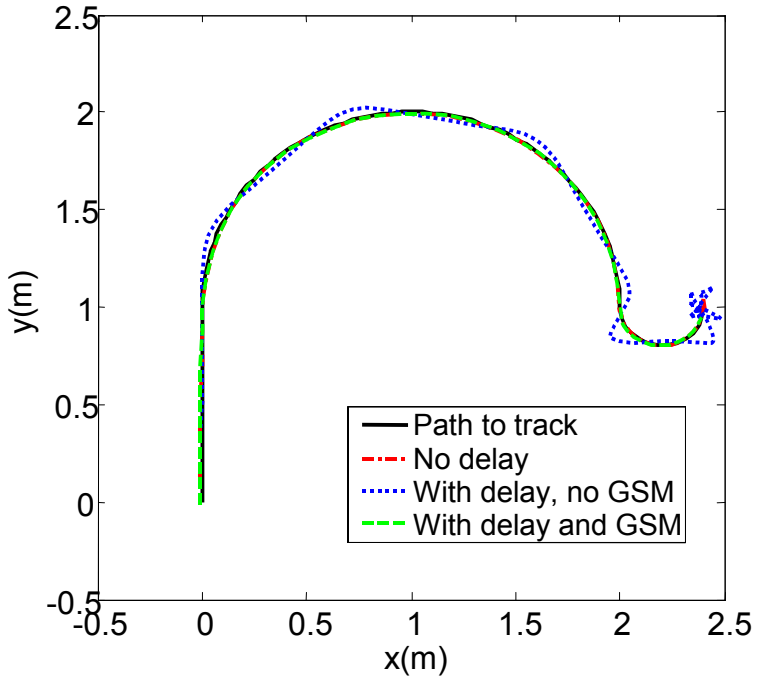


Fig. 14 Comparison of robot tracks from experiments: (a) Dashed-dotted line: The robot is controlled without IP network delay; (b) Dotted line: The robot is controlled with the existence of IP network delays from ADAC to Kasetsart University. The GSM is not applied; (c) Dashed line: The robot is controlled with the existence of IP network delays from ADAC to Kasetsart University. The GSM is applied.

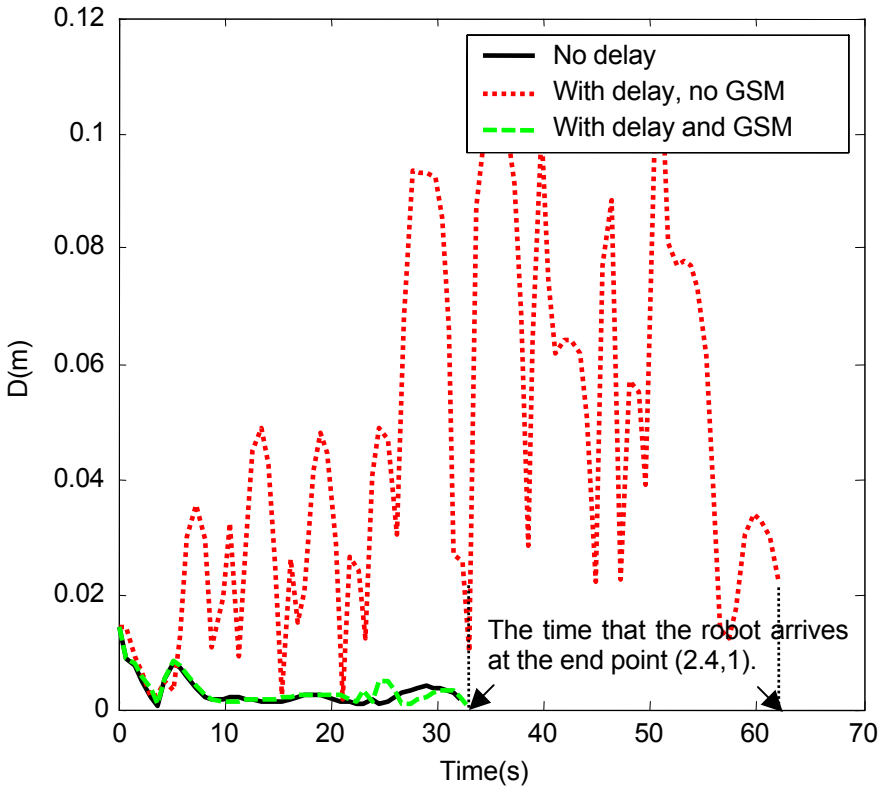


Fig. 15 Closest distances from the robot to the path obtained from experiments: (a) Solid line: The robot is controlled without IP network delay; (b) Dotted line: The robot is controlled with the existence of IP network delays from ADAC to Kasetsart University. The GSM is not applied; (c) Dashed line: The robot is controlled with the existence of IP network delays from ADAC to Kasetsart University. The GSM is applied.

As shown in Fig. 14, the experimental results are similar to the results obtained from the simulations. Without IP network delay, the original path-tracking controller performs superbly as well. When there are IP network delays, the robot without the GSM cannot track the path closely because the position feedback and the reference speeds are delayed by IP network delays. The GSM can improve the path-tracking performance by using predicted position and gain scheduling to compensate the delay effects so the robot can track the path more effectively as shown in Fig. 14. Fig. 15 also shows that the robot without the GSM deviates from the path relatively more than the other two cases, and spends a longer time to reach the final destination.

On the other hand, the robot with the GSM can track the path much closer to the robot without delay.

VI. Conclusion

This paper has proposed the concept of gain scheduling by using middleware to enable an existing robot controller for networked control over IP network. The middleware called gain scheduler middleware (GSM), adjusts the controller output with respect to the current network traffic conditions without interrupting the internal design or structure of the existing controller. Therefore, The GSM allows convenient installation and cost-effective upgrade for an existing controller without redesign or replacement by a whole new networked control system. The effectiveness of the GSM is verified by using a mobile robot path-tracking problem. The GSM approach has shown significant improvement on the mobile robot path-tracking performance with the existence of IP network delays. Because the GSM can be implemented separately from the controller, modification of the GSM by using other types of prediction algorithms or applying a different cost function for different applications could be achieved easily. In addition, since the GSM adapts the controller output based on the current network traffic conditions, the GSM concept can also take advantages from IP network QoS (Quality-of-Service). Furthermore, the GSM concept may be able to be applied on other robot control algorithms, but reformulation on the problem may be required. Future studies such as the effects of packet loss on networked control robots and the performance of GSM on a QoS-enabled network can be used to improve the effectiveness of GSM for more practical uses.

Acknowledgment

The authors would like to acknowledge the Royal Thai Government for partially supporting this study and Maxon Precision Motor, Inc. for the motor donation.

References

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

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

[3] G. C. Walsh, H. Ye, and L. G. Bushnell, "Stability analysis of networked control systems," *IEEE Transactions on Control Systems Technology*, vol. 10, no. 3, pp. 438-446, 2002.

[4] J. Nilsson, B. Bernhardsson, and B. Wittenmark, "Stochastic analysis and control of real-time systems with random time delays," *Automatica*, vol. 34, no. 1, pp. 57-64, 1998.

[5] Y. Tipsuwan and M.-Y. Chow, "Network-based controller adaptation based on QoS negotiation and deterioration," in IEEE IECON 2001, Denver, CO, 2001, pp. 1794-1799.

[6] S. H. Hong, "Scheduling algorithm of data sampling times in the integrated communication and control systems," *IEEE Transactions on Control Systems Technology*, vol. 3, no. 2, pp. 225-230, 1995.

[7] A. Sano, H. Fujimoto, and M. Tanaka, "Gain-scheduled compensation for time delay of bilateral teleoperation systems," in IEEE ICRA, Leuven, Belgium, 1998, pp. 1916-1923.

[8] S. Munir and W. J. Book, "Internet based teleoperation using wave variables with prediction," *IEEE/ASME Transactions on Mechatronics*, vol. 7, no. 2, pp. 124-133, 2002.

[9] C. Benedetti, M. Franchini, and P. Fiorini, "Stable tracking in variable time-delay teleoperation," in IEEE IROS, Maui, HI, 2001, pp. 2252-2257.

[10] K. Brady and T.-J. Tarn, "Internet-based teleoperation," in IEEE ICRA, Seoul, South Korea, 2001, pp. 644-649.

[11] H. Utz, S. Sablatnog, S. Enderle, and G. Kraetzschmar, "Miro-middleware for mobile robot applications," *IEEE Transactions on Robotics and Automation*, vol. 18, no. 4, pp. 493-497, 2002.

[12] D. Brugali and M. E. Fayad, "Distributed computing in robotics and automation," *IEEE Transactions on Robotics and Automation*, vol. 18, no. 4, pp. 409-420, 2002.

[13] D. C. Schmidt, "Middleware techniques and optimizations for real-time, embedded systems," in International Symposium on System Synthesis, San Jose, CA, 1999, pp. 12-16.

[14] B. Li and K. Nahrstedt, "A control-based middleware framework for quality-of-service adaptations," *IEEE Journal on Selected Areas in Communications*, vol. 17, no. 9, pp. 1632-1650, 1999.

[15] S. Song, J. Huang, P. Kappler, R. Freimark, and T. Kozlik, "Fault-tolerant Ethernet middleware for IP-based process control networks," in IEEE Local Computer Networks, Tampa, FL, 2000, pp. 116-125.

[16] T. F. Abdelzaher, E. M. Atkins, and K. G. Shin, "QoS negotiation in real-time systems and its application to automated flight control," *IEEE Transactions on Computers*, vol. 49, no. 11, pp. 1170-1183, 2000.

[17] Y. Tipsuwan and M.-Y. Chow, "On the gain scheduling for networked PI controller over IP Network," in IEEE/ASME International Conference on Advanced Intelligent Mechatronics, Kobe, Japan, 2003.

[18] K. Yoshizawa, H. Hashimoto, M. Wada, and S. M. Mori, "Path tracking control of mobile robots using a quadratic curve," in IEEE Intelligent Vehicles Symposium, Tokyo, Japan, 1996, pp. 58-63.

[19] J. W. Park and J. M. Lee, "Transmission modeling and simulation for Internet-based control," in IEEE IECON 2001, Denver, CO, 2001, pp. 165-169.