

ABSTRACT

RICHARDS, TYLER V. Performance Characterization of IP Network-based Control

Methodologies for DC Motor Applications. (Under the direction of Dr. Mo-Yuen Chow.)

Using a communication network, such as an IP network, in the control loop is increasingly becoming the norm. This process of network-based control (NBC) has potential profound impact in areas such as: teleoperation, healthcare, military applications, and manufacturing. However, limitations arise as the communication network introduces delay that often degrades or destabilizes the control system. Four methods have been investigated that alleviate the IP network delays to provide stable real-time control. A performance measure is defined for a case study on a DC motor with a networked proportional-integral (PI) speed controller with various network delays and noise levels. Matlab simulation results show that NBC combined with these techniques can successfully maintain system stability, allowing control of real-time applications.

Performance Characterization of IP Network-based Control

Methodologies for DC Motor Applications

by

Tyler Richards

A thesis submitted to the Graduate Faculty of

North Carolina State University

In partial fulfillment of the

Requirements for the degree of

Masters of Science

In

Computer Engineering

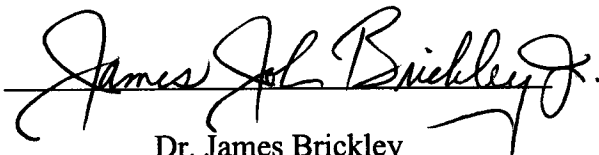
Raleigh, NC

2005


Approved by:



Dr. Mo-Yuen Chow
Chair of Advisory Committee



Dr. James Brickley



Dr. Wenye Wang

BIOGRAPHY

Tyler V. Richards was born February 4th, 1981 in Lexington, KY. He received his Bachelor of Science in Computer Engineering and a minor in Economics from North Carolina State University in May 2003. He then joined the Advanced Diagnosis, Automation and Control (ADAC) Laboratory, where he has pursued his Master of Science degree to be awarded in August 2005, in the areas of Mechatronics and Network-based Control. After graduation, Tyler will join the Mechatronics division of the Large Power and Systems Division of Caterpillar, Inc.

ACKNOWLEDGMENTS

I would like to thank my advisor, Dr. Mo-Yuen Chow, for providing direction and inspiration for my research. I would also like to thank Dr. James Brickley and Dr. Wenye Wang for agreeing to be part of my thesis committee, and for their continued support and knowledge they have given me along the way. Special thanks go out to Dr. Fen Wu for all the guidance and knowledge imparted to me with the Robust control. I would also like to thank all the members of the ADAC Lab. Without their vast knowledge and friendship as my support, this thesis would not have been possible. I would like to give special thanks to my parents, Varner and Lynne Richards, for all the love and support they have given me through the years.

TABLE OF CONTENTS

LIST OF FIGURES.	vii
LIST OF TABLES.	ix
CHAPTER 1. Introduction.	1
CHAPTER 2. Brief description of four NBC methods.	5
2-1. Gain scheduling middleware (GSM).	5
2-2. Optimal stochastic methodology.	6
2-3. Queuing methodology.	9
2-4. Robust control methodology.	13
CHAPTER 3. Case study: DC motor.	17
3-1. System description.	17
3-2. Performance characterization.	19
3-3. Network delay.	23
CHAPTER 4. Simulation results.	25
4-1. Gain scheduling middleware (GSM).	25
4.1.1 DC motor application.	25
4.1.2 Results.	28
4-2. Optimal stochastic method.	30
4.2.1 DC motor application.	30
4.2.2 Results.	31
4-3. Queuing methodology.	33
4.3.1 DC motor application.	33
4.3.2 Results.	33
4-4. Robust control methodology.	36
4.4.1 DC motor application.	36
4.4.2 Results.	37
CHAPTER 5. Noise.	40
5-1. Gain scheduling middleware.	42
5-2. Optimal stochastic method.	44

5-3.	Queuing methodology.	45
5-4.	Robust control methodology.	47
CHAPTER 6.	Conclusion.	49
6-1.	Benefits table.	49
6-2.	Future work.	51
REFERENCES.		52
APPENDIX.		54
A1.	GSM.	55
A1.1.	Simulink diagram.	55
A1.2.	Simulink files.	55
A1.2.1.	find_beta_costs.m	55
A1.2.2.	find_weights.m	59
A1.2.3.	find_weights_part2.m	62
A1.2.4.	test_findcost.m	62
A2.	Optimal stochastic method.	64
A2.1.	Simulink diagram	64
A2.2.	Simulink files	64
A2.2.1.	run_sim_once.m	64
A3.	Queuing methodology.	68
A3.1.	Simulink diagram.	68
A3.2.	Simulink files.	68
A3.2.1.	setup_test_sub10_init.m	68
A3.2.2.	get_results1.m	69
A3.2.3.	calculations.m	71
A4.	Robust control methodology.	72
A4.1.	Simulink diagram.	72
A4.2.	Simulink files.	72
A4.2.1.	setup_d1.m	72
A4.2.2.	d1_after.m	74

A4.2.3. find_cost.m	86
A4.3. dkitgui.	87
A4.3.1. Setup.	87
A4.3.2. Sample results.	88

LIST OF FIGURES

Figure 1.	Structure of a network-based control system.	2
Figure 2.	Structure of GSM.	6
Figure 3.	Configuration of networked control system utilizing queuing methodology.	9
Figure 4.	Delay compensation algorithm with predictor blocks.	12
Figure 5.	Timing diagram for $\tau^{sc} = \tau^{ca} = \delta$.	13
Figure 6.	Controller-Plant Modeling with Network Delays.	14
Figure 7.	Robust analysis for a general interconnection system.	15
Figure 8.	Nominal PI controller step response with $(K_I^*, K_P^*) = (25.572, 0.995)$.	19
Figure 9.	MSE cost vs. β .	21
Figure 10.	P.O. cost vs. β .	21
Figure 11.	t_r cost vs. β .	22
Figure 12.	Histogram for $\eta = 0.0001$ sec RTT delay.	24
Figure 13.	Cost vs. β for $\eta = [0.0001, 0.001, 0.005]$ network delays.	27
Figure 14.	Cost vs. β for $\eta = [0.01, 0.05, 0.1]$ network delays.	27
Figure 15.	Nominal PI controller vs. GSM step responses for a) $\eta = 0.0001$ b) $\eta = 0.001$ c) $\eta = 0.005$ d) $\eta = 0.01$ e) $\eta = 0.05$ f) $\eta = 0.1$ seconds.	29
Figure 16.	Nominal PI controller vs. Optimal stochastic method step responses for a) $\eta = 0.0001$ b) $\eta = 0.001$ c) $\eta = 0.005$ d) $\eta = 0.01$ e) $\eta = 0.05$ f) $\eta = 0.1$ seconds.	32
Figure 17.	Nominal PI controller vs. Queuing methodology step responses for a) $\eta = 0.0001$ b) $\eta = 0.001$ c) $\eta = 0.005$ d) $\eta = 0.01$ e) $\eta = 0.05$ f) $\eta = 0.1$ seconds.	35
Figure 18.	Nominal PI controller vs. Robust control methodology step responses for a) $\eta = 0.0001$ b) $\eta = 0.001$ c) $\eta = 0.005$ d) $\eta = 0.01$ e) $\eta = 0.05$ f) $\eta = 0.1$ seconds.	39

Figure 19.	Noise, $w(k)$ introduced into the control loop.	41
Figure 20.	Noise distributions.	41
Figure 21.	Typical step response with 5% noise, 0.0001 sec RTT delay.	41
Figure 22.	Performance comparison with noise.	42
Figure 23.	GSM noise vs. delay performance.	43
Figure 24.	Optimal method noise vs. delay performance.	44
Figure 25.	Queuing method noise vs. delay performance.	46
Figure 26.	Robust methodology noise vs. delay performance.	47
Figure 27.	find_beta_sim.mdl	55
Figure 28.	lqr_dc_delay_tweak.mdl	64
Figure 29.	queue_comp3_sub10.mdl	68
Figure 30.	robust_delay1_K3.mdl	72
Figure 31.	dkitgui setup screen.	87
Figure 32.	Sample DK iteration summary.	88

LIST OF TABLES

Table 1.	DC motor parameters.	17
Table 2.	Optimal β gains for various time delays.	26
Table 3.	Performance comparison of four methodologies for different RTT network delays.	30
Table 4.	Queue lengths for various delays.	34
Table 5.	Optimal robust controller for each delay.	37
Table 6.	Peak μ values for controller generated during each DK iteration.	37
Table 7.	GSM noise results.	43
Table 8.	Optimal method noise results.	45
Table 9.	Queuing method noise results.	46
Table 10.	Robust methodology noise results.	48
Table 11.	Pros and Cons of the four NBC methodologies.	50

CHAPTER 1

INTRODUCTION

A recent trend in control systems is to incorporate a network communication medium into the control loop [1], termed network-based control (NBC). One such medium under investigation is the Internet because of its attractive advantages such as: affordability, widespread usage, and availability. For industrial electronics and factory automation areas, networked control systems (NCS) provide a promising solution by reducing wiring costs, wiring complexity, and streamlining the use of assets. However, for an Internet Protocol (IP)-based NCS, the network medium introduces several difficulties that must be resolved for successful, practical use of NBC. The major challenge is the IP network-induced delay effect on the control system. It is well known that the delay introduced by the network can degrade performance and destabilize the control system. Thus, most IP-based control applications have been limited to non time-sensitive applications. It is important to develop strategies to compensate or alleviate the network delay effect on real-time applications. Conventional methods for network-based control address constant network delays. The Internet introduces stochastic time delays such that these methods may not be applicable. Several advanced techniques have been presented [2-5] that compensate or alleviate the stochastic network delay, potentially enough to be used in critical real-time applications. NBC combined with these techniques aim to eliminate the detrimental delay effect and prevent destabilization of the closed loop control system in an IP network environment.

Network-based control systems experience network delay from the controller to the plant and from the plant to the controller. Let us denote the time delay experienced from

the controller to the actuator of the plant as τ^{ca} , and the time delay from the sensor of the plant to the controller as τ^{sc} . Figure 1 shows the typical structure of a network-based control system. The terms for NCS and NBC systems are interchangeable. For this thesis, the delays associated with processing time are assumed constant and much smaller than τ^{ca} and τ^{sc} , and therefore can be lumped into those terms to simplify the problem.

Additional assumptions include [1]:

- The network traffic cannot be overloaded.
- Network transmissions are error-free.
- Every packet has the same constant length.
- Every dimension of the output measurement or the control signal can be packed into one single packet.

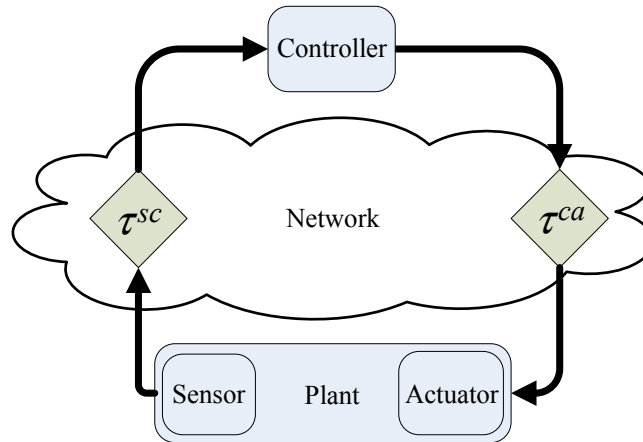


Figure 1. Structure of a network-based control system.

Stochastic network delays present difficulties when formulating the NBC problem. Because of this, network delay has been approximated in numerous ways, such as periodic delays [6], exponential distributions[5], or by using Markov models [7]. Once a satisfactory model has been established, NBC can be applied on a variety of applications. The real-time NBC technology has several potential applications including iSpace [8], teleoperation [9], aiding the elderly, military applications, and other times when a human presence is not feasible.

Various methodologies have been investigated with the task of maintaining system stability and limiting performance degradation caused from the network delay. This thesis selects four methods that have shown optimistic results for viable use as a real-time network-based control scheme. Preliminary results on a DC motor application show their performance on the compensation of the network delay for each of these methods.

The four methods are:

- 1) Gain scheduling middleware (GSM) method [5, 10]
- 2) Optimal stochastic control methodology [4, 11]
- 3) Queuing methodology [3, 12]
- 4) Robust control methodology [2, 13, 14]

Gain scheduling middleware (GSM) retains the nominal controller for the non-delay case and inserts a middleware that modifies the effective gain applied based on the network delay data it continuously monitors. The optimal stochastic method approaches the problem as a Linear-Quadratic-Gaussian (LQG) problem where the LQG gain matrix is optimally chosen based on the network delay statistics. The queuing method is a popular method that turns the networked control system into a time-invariant system to alleviate the delays. The

robust control method considers the delays as multiplicative perturbations on the system and uses robust control to minimize the effect of the perturbations and maintain system performance.

This thesis is structured as follows: Chapter 2 will present and describe each of the methodologies. Chapter 3 will introduce the DC motor model and the performance measure used for the simulations. Chapter 4 will report the results of each method. Chapter 5 addresses the performance when noise is added to the system. Finally, Chapter 6 will conclude the thesis with closing remarks, pros and cons of each methodology, and suggest areas of future work. The simulation files can be found in the appendix.

CHAPTER 2

BRIEF DESCRIPTION OF FOUR NBC METHODS

2-1. Gain scheduling middleware (GSM)

Gain scheduling middleware (GSM) uses a previously designed controller in addition to its middleware to compensate for the network delay [5, 15, 16]. Redesigning a control can be cost prohibitive. Therefore, by using GSM the original controller is retained and its output is modified by the GSM middleware to compensate for the network delay.

The basic components of GSM, shown in Figure 2, are as follows [15].

- 1) *Network Traffic Estimator*: The function of the network traffic estimator is to estimate the current network conditions. It monitors a probing packet to find the roundtrip network measurements that are characterized by q . The variable q is then used by the feedback preprocessor and the gain scheduler to estimate the current network delays.
- 2) *Feedback Preprocessor*: The function of the feedback preprocessor is to gather the sensor data of the remote system and perform any functions on the data before passing it to the controller. These functions can include filtering noise or predicting the state of the remote system.
- 3) *Gain Scheduler*: The function of the gain scheduler is to modify the controller's output with respect to the current network conditions characterized by q as reported by the network traffic estimator. The algorithm to modify the gain will depend on the controller and remote system.

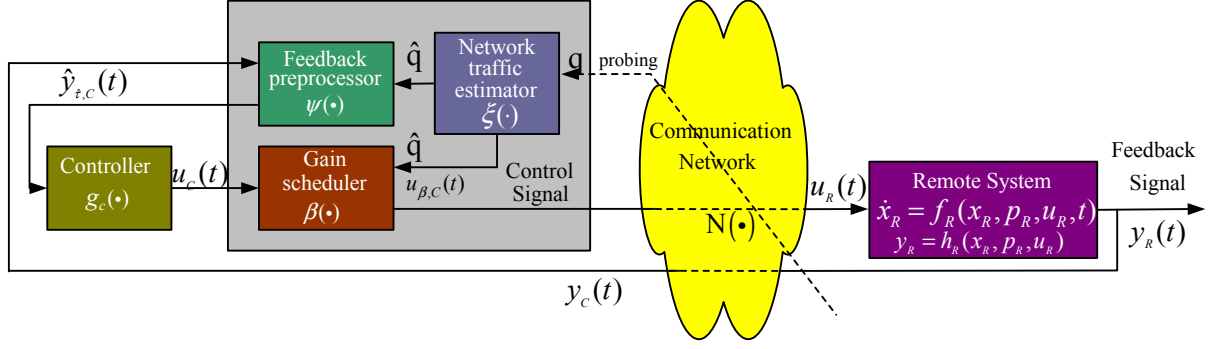


Figure 2. Structure of GSM.

The GSM methodology is adapted to the particular system of interest. In this thesis, GSM is applied on a networked PI controller for a DC motor.

2-2. Optimal stochastic methodology

The optimal stochastic control methodology treats a networked control system with random network delays as a Linear Quadratic Gaussian (LQG) problem. This method was first proposed by Nilsson [4]. To control the system, the methodology assumes that $\tau^{sc} + \tau^{ca} < T$, in addition to the previous assumptions. Without this condition, the control signals could arrive at the actuator out of order, making analysis much more difficult. In addition, if this assumption is violated, the method cannot guarantee system performance.

The system dynamics are assumed to be linear and can be described by:

$$\begin{aligned} \dot{x}(t) &= Ax(t) + Bu(t) + v(t) \\ y(t) &= Cx(t) + w(t) \end{aligned} \quad (1)$$

where $x(t) \in R^n$, $u(t) \in R^m$, and $v(t), w(t) \in R^n$. A and B are matrices of appropriate dimensions, $u(t)$ is the controlled input, and $v(t)$ and $w(t)$ are uncorrelated Gaussian white

noises with zero means [4]. This system can be discretized into kT sampling periods resulting with the following dynamics:

$$\begin{aligned} x(k+1) &= \Phi x(k) + \Gamma_0(\tau^{sc}, \tau^{ca})u(k) + \Gamma_1(\tau^{sc}, \tau^{ca})u(k-1) + v(k), \\ y(k) &= Cx(k) + w(k) \end{aligned} \quad (2)$$

where

$$\begin{aligned} \Phi &= e^{AT} \\ \Gamma_0(\tau^{sc}, \tau^{ca})u(k) &= \int_0^{T-\tau_k^{sc}-\tau_k^{ca}} e^{As} ds B \quad , \\ \Gamma_1(\tau^{sc}, \tau^{ca})u(k-1) &= \int_{T-\tau_k^{sc}-\tau_k^{ca}}^T e^{As} ds B \end{aligned} \quad (3)$$

and τ^{sc} is the network delay from the plant output (sensor) to the controller; τ^{ca} is the network delay from the controller to the plant input (actuator), and T is the sampling rate of the sensor [11].

The goal of the optimal stochastic method is to solve the control problem by minimizing the cost function in the case that full state information is available.

$$J(k) = E\left\{x^T(N)Q_N x(N)\right\} + E\left\{\sum_{k=0}^{N-1} \begin{bmatrix} x(k) \\ u(k) \end{bmatrix}^T Q \begin{bmatrix} x(k) \\ u(k) \end{bmatrix}\right\}, \quad (4)$$

where $E\{\bullet\}$ is the expected value, and Q_N and Q are weighting matrices. The optimal control law that minimizes the cost function is:

$$u_k^* = -K^*(\tau^{sc}) \begin{bmatrix} x_k \\ u_{k-1}^* \end{bmatrix}, \quad (5)$$

where K^* is the optimal gain matrix after solving the formulated LQG problem [4].

In practice, a table for $K_{[0,\infty]}(\tau^{sc})$, where $K_{[0,\infty]}(\bullet)$ is the optimal gain matrix for values of τ^{sc} from zero to infinity, is calculated offline and then the value of $K(\tau^{sc})$ is

interpolated from the tabular values in real-time. This is done because of the calculation intensive process of finding $K^*(\tau_k^{sc})$.

This method can be simplified into a suboptimal method that has been shown to give results close to the optimal method but uses less calculations during runtime [11]. The control law for this method is given by:

$$u_k = -K[\Phi_k^p \quad \Gamma_k^p] \begin{bmatrix} x_k \\ u_{k-1} \end{bmatrix}, \quad (6)$$

where

$$\begin{aligned} \Phi_k^p &= e^{A(\tau_k^{sc} + E\tau_k^{ca})} \\ \Gamma_k^p &= \int_0^{\tau_k^{sc} + E\tau_k^{ca}} e^{As} ds B. \end{aligned} \quad (7)$$

$E\tau_k^{ca}$ is the expected value of τ^{ca} . K is the optimal LQG gain found from the system without delay. Thus the non-delay gain, K , is modified by a matrix containing the delay information for the current step and a prediction of the delay to be experienced from the controller back to the plant. This operation can be seen as a prediction from the state estimate at the current time, kT , to a state estimate when the control signal is applied at the plant [11]. This is an alternative to having the delay information embedded in the K gain matrix. This approach will be used for this thesis because of its near optimal performance and reduced calculation time. The cost function to minimize remains the same as in (4).

2-3. Queuing methodology

The purpose of adding queues into a networked control system is to reshape the random time delays into deterministic delays. As an effect, the system becomes time-invariant. One such method is proposed by Luck and Ray [3, 12]. They propose to use an observer to estimate the plant states and a predictor to compute the predictive control. The past measurement data and the control data are stored in a First-In-First-Out (FIFO) queue, defined as Q_1 and Q_2 of sizes r and α , respectively. The structure of a network control system utilizing the queuing methodology is shown in Figure 3 [1].

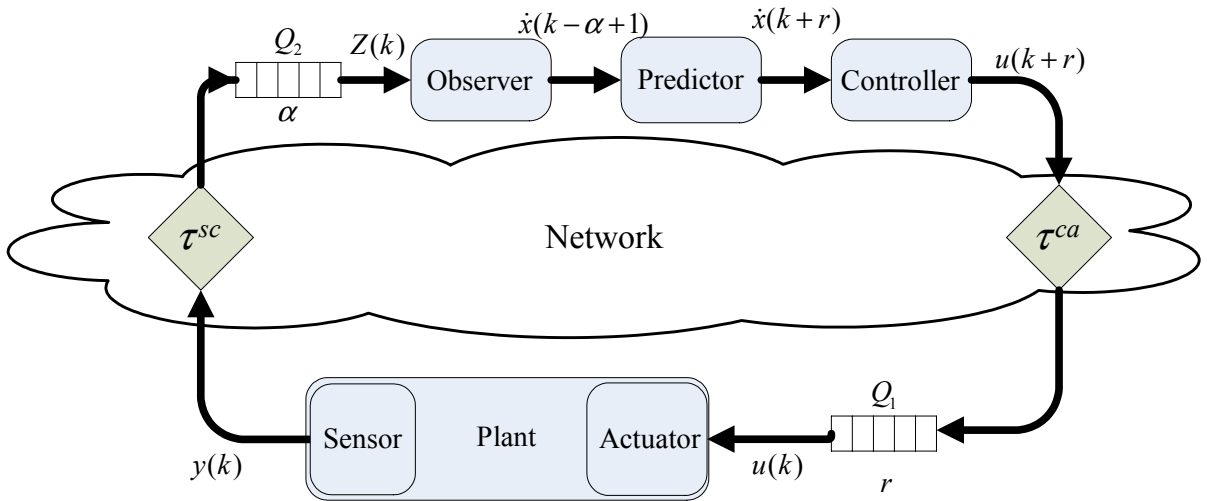


Figure 3. Configuration of networked control system utilizing queuing methodology.

The steps for applying this methodology are listed below [1]:

- Using the set of past measurements $Z(k) = \{y(k-\phi), y(k-\phi-1), \dots\}$ in Q_2 , where $\phi \leq \alpha$ is the current number of packets in Q_2 , the observer estimates the plant state $\hat{x}(k-\alpha+1)$.

- The predictor uses $\hat{x}(k - \alpha + 1)$ to predict the future state $\hat{x}(k + r)$.
- The controller computes the predictive control $u(k + r) = K(\hat{x}(k + r))$, where K is the controller gain, and then sends $u(k + r)$ to be stored in \mathcal{Q}_1 .

The model of the plant must be very precise since the performance of the observer and predictor depend on it. For a closed-loop NBC system, the various components are defined below.

Plant model:

$$x_{k+1} = Ax_k + Bu_k, y_k = Cx_k; \quad (8)$$

Observer model:

$$z_{k+1|1} = Az_{k|1} + Bu_k + L_k(y_k - Cz_{k|1}); \quad (9)$$

δ -step predictor:

$$z_{k+1|\delta} = Az_{k|\delta-1} + Bu_k \text{ for } \delta \geq 2; \quad (10)$$

Predictive control:

$$u_k = \Gamma_k z_{k|\delta} \text{ for a fixed } \delta \geq 2; \quad (11)$$

where $z_{k|\delta} := \hat{x}_{k|k-\delta}$ is the estimation of x_k given the measurement history

$Z_{k-\delta} = \{Z_{k-\delta}, Z_{k-\delta-1}, Z_{k-\delta-2}, \dots\}$, $\{A, B, C\}$ are the realization of the plant system, and the estimation error:

$$e_k = x_k - z_{k|1}. \quad (12)$$

Γ_k and L_k represent the proper controller and observer gains, respectively. δ represents the number of sample times in a given delay time, or

$$\delta = \frac{\tau^{sc} + \tau^{ca}}{T}. \quad (13)$$

Then, the closed loop system equation can be expressed as

$$\begin{bmatrix} x_{k+1} \\ e_{k-\delta+1} \end{bmatrix} = \begin{bmatrix} A + B\Gamma_k & -B\Gamma_k\Lambda_k \\ 0 & A - L_{k-\delta}C \end{bmatrix} \begin{bmatrix} x_k \\ e_{k-\delta} \end{bmatrix}, \quad (14)$$

where

$$\Lambda_k := \begin{cases} \prod_{j=1}^{\delta} (A - L_{k-\delta+j-1}C) + \\ \prod_{i=1}^{\delta-1} \left[A^{i-1} L_{k-1} C \times \prod_{j=1}^{\delta-i-1} (A - L_{k-\delta+j-1}C) \right] & \text{if } \delta \geq 2 \\ I & \text{if } 0 \leq \delta < 2. \end{cases} \quad (15)$$

and the plant model is assumed to be exact [3]. Proof of this proposition can also be found in [3].

The schematic diagram of a δ -step delay compensation algorithm is shown in Figure 4, where δ represents the delay length in terms of sampling time steps. The lengths of the queues determine the number of predictor blocks needed to compensate for the delay.

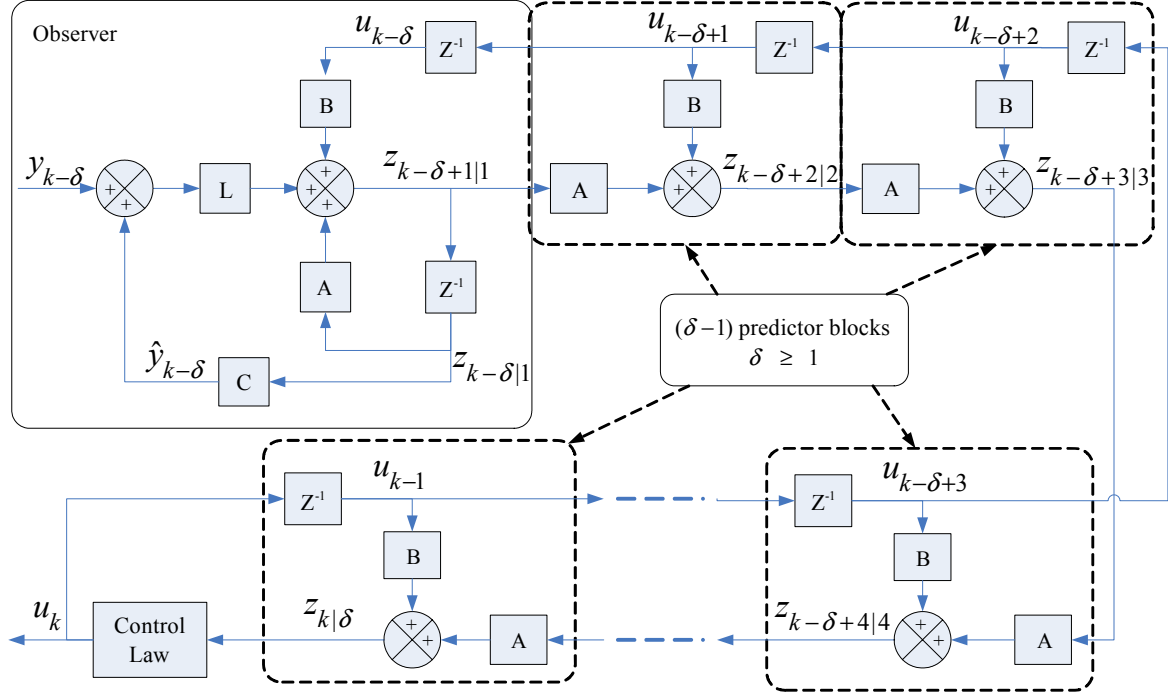


Figure 4. Delay compensation algorithm with predictor blocks.

If $\tau^{sc} = \tau^{ca} = \delta$, then the sensor data received at the controller at time k has already experienced a one time step delay, τ^{sc} . That is, it was generated at time $t(k-1)$. In addition, the controller's gain must also be delivered through the network experiencing another one step delay, τ^{ca} . This will result in the control signal being received at the plant at time $t(k+1)$. Therefore, the controller must estimate the plant state for the time $t(k+1)$ at time k so that it can send $u(k+1)$ through the network to be the correct input, $u(k)$, for the plant at $t(k+1)$. This scenario describes a $\delta=2$ compensation algorithm. Figure 5 illustrates this concept.

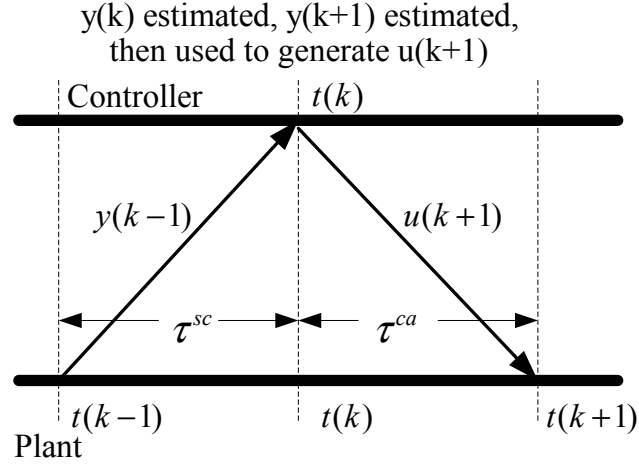


Figure 5 Timing diagram for $\tau^{sc} = \tau^{ca} = \delta$.

As mentioned before, the performance of the methodology depends greatly on the accuracy of the model and observer. This structure allows for variable delays as long as the following constraints are met: (1) $\tau^{sc} + \tau^{ca} = \text{known constant}$, (2) the sensor and controller sampling instants are synchronized with no time skew [3].

2-4. Robust Control

The robust control methodology models the two network delays, τ^{sc} and τ^{ca} , as simultaneous multiplicative perturbations. Using H_∞ and μ -synthesis, the perturbations represent the modeling uncertainties associated with the network delays. The feedback control system can be modeled in the frequency domain. The purpose of robust control is to maintain prescribed performance levels under specified levels of uncertainty [13].

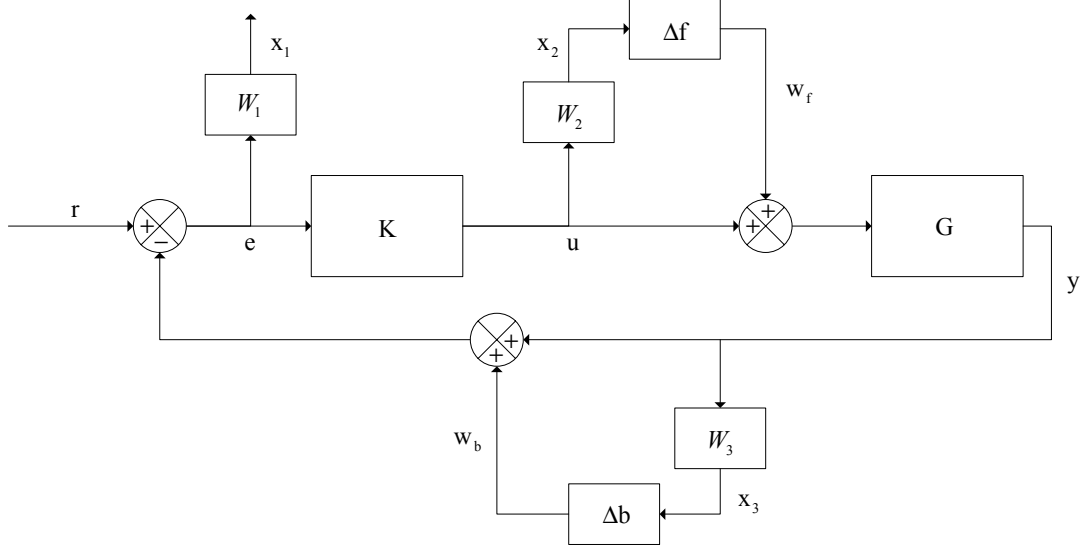


Figure 6. Controller-Plant Modeling with Network Delays.

The controller-plant model with network delays as perturbations is shown in Figure 6. Δf and Δb represent the network delays τ^{ca} and τ^{sc} , respectively. The delay uncertainty in the frequency domain, given as exponential distribution $e^{-\tau_{\max}\Delta s}$, where $-1 < \Delta < 1$, can be approximated using a modified Pade approximation with Δ complex [17]:

$$e^{-\tau_{\max}\Delta s} \approx 1 - \frac{\tau_{\max}s}{1 + \frac{\tau_{\max}}{2}s} \Delta = 1 + W(s)\Delta, \quad (16)$$

where $|\Delta| \leq 1$ and

$$W(s) = \frac{\tau s}{1 + \tau s/3.465}. \quad (17)$$

Here, Δ can be either Δf and Δb , and τ can be either τ^{ca} and τ^{sc} . $W(s)$ is suggested by [17] as an appropriate uncertainty weight that covers the uncertain delay. The factor 3.465 is selected is based on designer preference [1]. It will be used for both W_2 and W_3 shown in

Figure 6. W_1 represents the performance weight for the system in Figure 6 and is determined by the error signal. It is presented here as a low-pass filter as suggested from [14]:

$$W_1(s) = \frac{0.9}{s^2 + s + 0.9}. \quad (18)$$

The problem is solved using the H_∞ and μ -synthesis toolboxes in Matlab [18]. First, the problem illustrated in Figure 6 must be formulated into the H_∞ framework shown in Figure 7, where K is the controller gain to be determined by DK iteration during the H_∞ process.

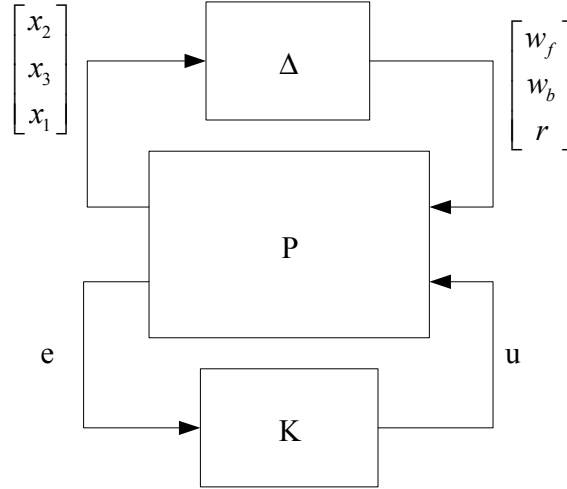


Figure 7. Robust analysis for a general interconnection system.

The matrix, P , contains the interconnection structure. $\Delta = \text{diag}[\Delta f, \Delta b, \Delta p]$, where Δp is a fictitious uncertainty block used to incorporate the H_∞ performance objective. The interconnection matrix, P , can be found as:

$$\begin{bmatrix} x_2 \\ x_3 \\ x_1 \\ e \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & W_2 \\ W_3 G & 0 & 0 & W_3 G \\ -W_1 G & -W_1 & W_1 & -W_1 G \\ -G & -1 & 1 & -G \end{bmatrix} \begin{bmatrix} w_f \\ w_b \\ r \\ u \end{bmatrix}, \quad (19)$$

where W_1 , W_2 , and W_3 are the weights discussed previously, G is the plant dynamics, and w_f and w_b are introduced to represent the inputs to the system when the perturbations are omitted during the open-loop formulation of the problem and can be seen in Figure 6. The necessary condition for robust performance, then, is $\|P, K\|_\infty < 1$. This is often measured by the peak μ value, or $\mu(P, K) < 1$. Robust control tries to minimize the effect of a disturbance on the system as seen by the error signal. This performance measure is given by $\mu(P, K)$, which is desired to be driven to zero. If $\mu(P, K) = 0$, this would indicate that any size disturbance would have zero effect on the error, hence displaying ideal robust stability. A suitable controller K found through DK iteration and μ -synthesis.

CHAPTER 3

CASE STUDY: DC MOTOR

3-1. System Description

The DC motor is governed by the following equations:

$$\begin{aligned} \dot{i}_a &= -\frac{R}{L}i_a - \frac{K_B}{L}\omega_w + \frac{1}{L}e_a \\ \dot{\omega}_w &= \frac{K}{J}i_a - \frac{B}{J}\omega_w + \frac{1}{J}F \end{aligned} \quad (20)$$

where e_a is the armature winding input voltage, L is the armature winding inductance, R is the armature winding resistance, i_a is the armature winding current, K_B is the back EMF constant, K is the torque constant, J is the moment of inertia of the wheel and rotor, B is the damping coefficient of the wheel and rotor, ω_w is the angular velocity of the wheel, and F is the load torque. Table 1 shows the values used during simulation for these constants that were modified from [10].

Table 1. DC motor parameters.

$K = 2.55\text{e-}3 \text{ N-m/A}$	$R = 6.43 \text{ } \Omega$
$K_B = .255\text{e-}3 \text{ V-sec/rad}$	$L = 28.8\text{e-}3 \text{ H}$
$B = 0.1\text{e-}3 \text{ N-m-sec/rad}$	$J = 3.53\text{e-}6 \text{ Kg-m}^2$

The sampling time, T , for the controller is chosen to be 0.1ms. This is to capture the dynamics of the electrical circuit, whose time constant is:

$$\gamma_e = \frac{L}{R} = 4.48 \text{ msec} . \quad (21)$$

The load torque is assumed to be zero during the simulations. (20) can be arranged into the standard state space form shown below:

$$\begin{bmatrix} \dot{i} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} -R/L & -K_b/L \\ K/J & -b/J \end{bmatrix} \begin{bmatrix} i \\ \omega \end{bmatrix} + \begin{bmatrix} 1/L \\ 0 \end{bmatrix} u . \quad (22)$$

During the simulations, the plant is considered to be described by (22) that is sampled every kT seconds. The plant output is assumed to be fully observable.

A PI controller will be used that has been tuned for nominal performance in the non-delay case for the controller of the system. The control voltage, e_a , is assumed to be bounded by ± 12 volts. The optimal PI gains for the non-delay case are determined by the root locus approach and must satisfy the following design requirements:

- Relative damping ratio: $\zeta = 0.707$;
 - Percentage overshoot: $P.O. \leq 5\%$;
 - Settling time: $t_s \leq 0.0321 \text{ sec}$;
 - Rise time: $t_r \leq 0.014 \text{ sec}$;
- (23)

Using the above requirements, $(K_I^*, K_p^*) = (25.572, 0.995)$ will satisfy the conditions above. This will be used as the baseline performance to compare to. The reference speed is set to $r(k) = 1 \text{ rad/sec}$. The nominal performance of the PI controller with $K_I = K_I^*$ and $K_p = K_p^*$ without the communication network in the control loop is shown in Figure 8.

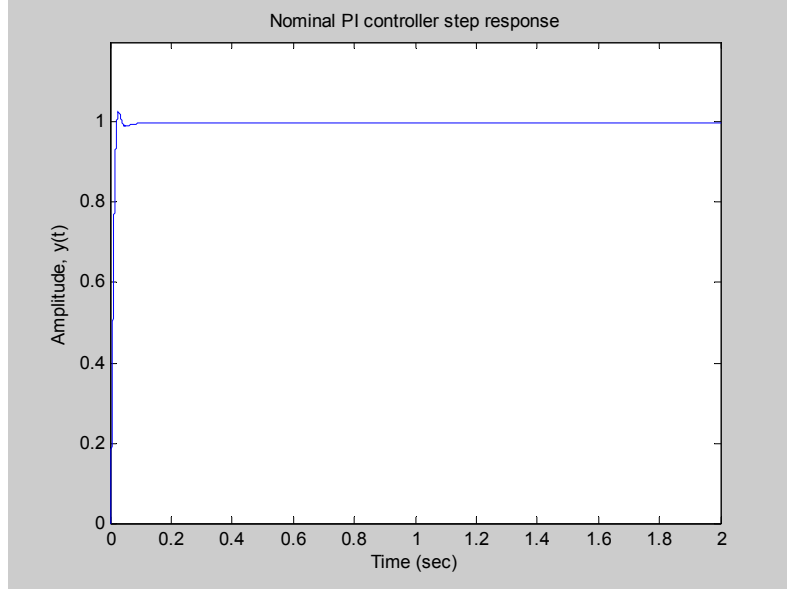


Figure 8. Nominal PI controller step response with $(K_I^*, K_P^*) = (25.572, 0.995)$.

3-2. Performance Characterization

In order to characterize the performance of each methodology, the following performance measure is adopted [5, 15]:

$$J = w_1 J_1 + w_2 J_2 + w_3 J_3, \quad (24)$$

$$J_1 = \begin{cases} (MSE - MSE_0)^2, & MSE > MSE_0 \\ 0, & MSE \leq MSE_0 \end{cases}, \quad (25)$$

$$J_2 = \begin{cases} (P.O. - P.O._0)^2, & P.O. > P.O._0 \\ 0, & P.O. \leq P.O._0 \end{cases}, \quad (26)$$

$$J_3 = \begin{cases} (t_r - t_{r0})^2, & t_r > t_{r0} \\ 0, & t_r \leq t_{r0} \end{cases}, \quad (27)$$

where

$$MSE = \frac{1}{N} \sum_{k=0}^N e^2(k), \quad (28)$$

is the mean-squared error, MSE_0 is the nominal mean-squared error, $P.O._0$ is the nominal percentage overshoot, t_{r0} is the nominal rise time. The weights w_1, w_2 , and w_3 specify the relative significance of J_1 , J_2 , and J_3 , respectively, on the overall system performance. The error, $e(k) = y(k) - r(k)$, is computed by sampling $y(t)$ every kT time instant. The costs J_1 , J_2 , and J_3 penalize any degradation in performance from the nominal case. The nominal performance meets the design specification such that $MSE_0 = 0.003405$, $P.O._0 = 5\%$, and $t_{r0} = 0.014$. MSE_0 is determined by simulation or by experiment. In the nominal non-delay case, the cost function in (24) will equal zero. J_1 penalizes tracking error. J_2 penalizes higher overshoot. J_3 penalizes the slower response time. The weights w_1, w_2 , and w_3 are found by simulation using the GSM method with $\beta = [0, 2]$ and no delay with the following equation:

$$w_i = \frac{1}{\max(J_i) - \min(J_i)} \quad \text{for } i = 1, 2, 3. \quad (29)$$

When $\beta = 1$, the nominal gain is applied to the system and therefore the cost $J = 0$. Therefore, $\min(J_i) = 0$ for $i = 1, 2, 3$ and (29) reduces to

$$w_i = \frac{1}{\max(J_i)} \quad \text{for } i = 1, 2, 3. \quad (30)$$

For the DC motor parameters listed in Table 1, $w_1 = 17.4572$, $w_2 = 0.0066771$, and $w_3 = 0.8777$. The cost graphs for J_1 , J_2 , and J_3 for $\beta = [0, 2]$ are shown in Figure 9, Figure 10, and Figure 11.

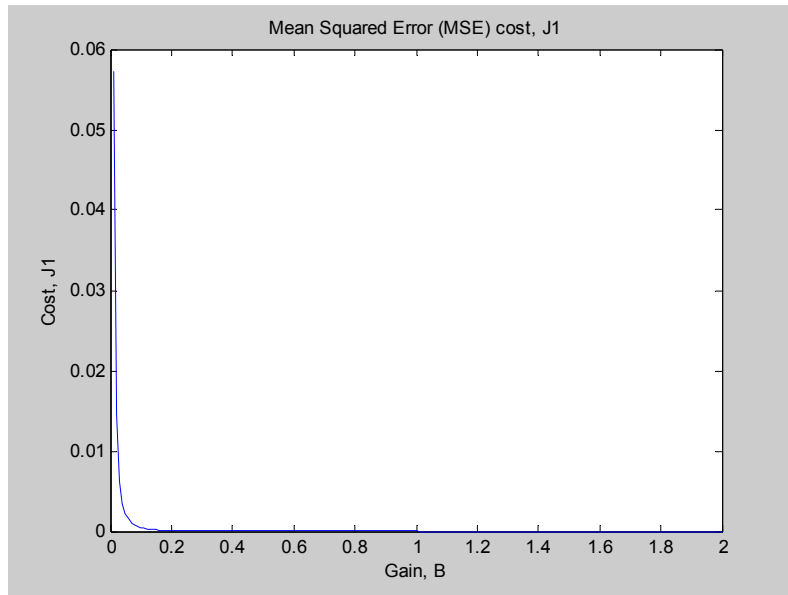


Figure 9. MSE cost vs. β .

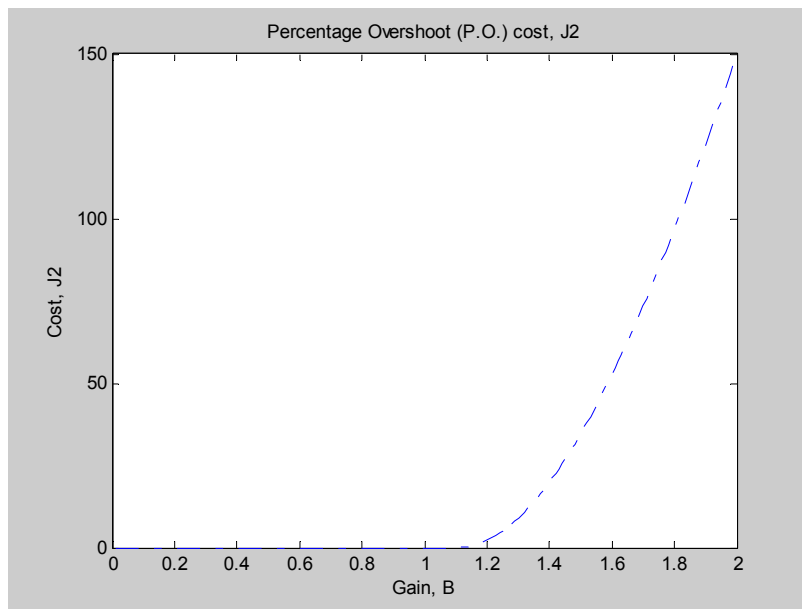


Figure 10. P.O. cost vs. β .

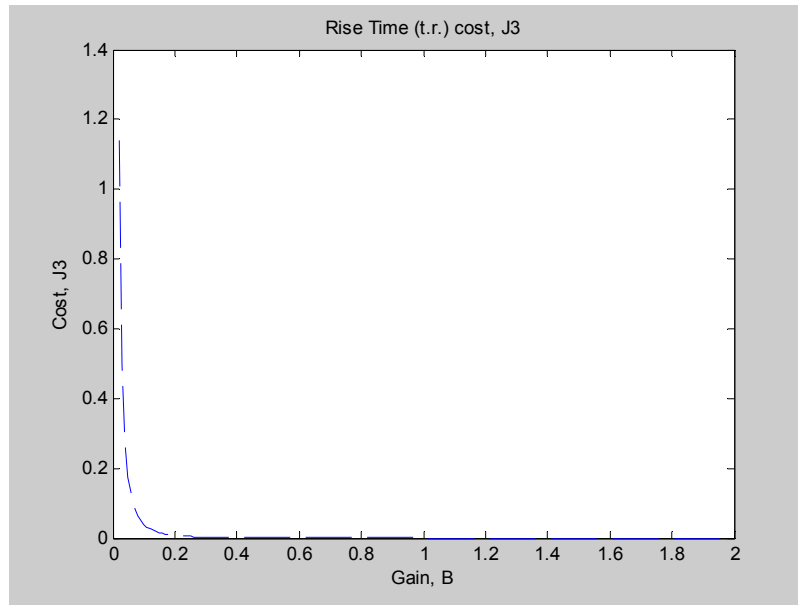


Figure 11. t_r cost vs. β .

3-3. Network Delay

Each methodology has its specific way of modeling the network delay experienced on the system. In general, network delay will be modeled as an exponential distribution unless otherwise noted. The generalized exponential distribution to describe IP network delays is as follows [15]:

$$P[\tau] = \begin{cases} \frac{1}{\phi} e^{-(\tau-\eta)/\phi}, & \tau \geq \eta \\ 0, & \tau < \eta \end{cases}, \quad (31)$$

where η is the minimum transportation delay of the application. The expected value of the RTT delay is $E(\tau) = \phi + \eta$ with variance, $\sigma^2 = \phi^2$, and median, η .

Keeping in mind the DC motor's time constant denoted in (21), $\eta = [0.0001, 0.001, 0.005, 0.01, 0.05, 0.1]$ has been chosen to simulate the RTT IP-network delay and be representative of a variety of minimum network transportation delays possibly experienced. The first two delays should have smaller effects on the system performance as they are less than $0.5\gamma_e$ found in (21) (i.e., can capture the electrical dynamics of the DC motor), while the larger delays should cause more rapid system degradation. A typical distribution of the delay is shown in Figure 12.

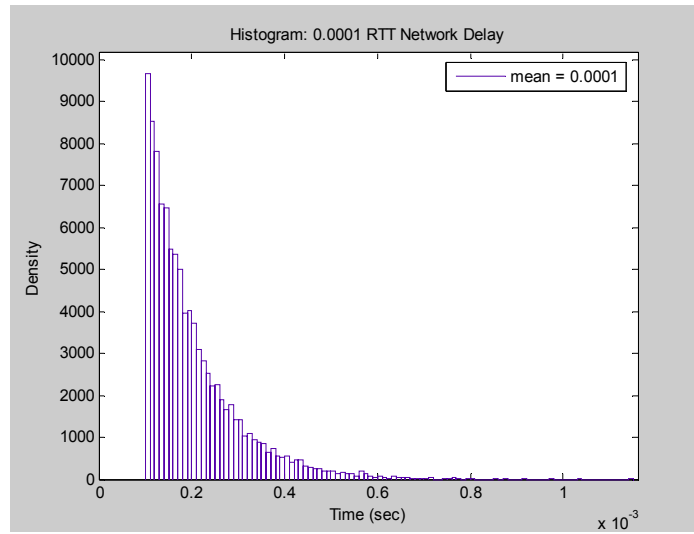


Figure 12. Histogram for $\eta = 0.0001\text{sec}$ RTT delay.

CHAPTER 4

SIMULATION RESULTS

For all simulations, the final time, t_f , is 2 seconds, with a sampling time, T , of 0.0001 seconds. Matlab 7.0 was used to run the simulations. The following sections will present the results for each of the four methods in alphabetical order when applied to the DC motor model described previously. Afterwards, the performance of the methods will be characterized with respect to the other models.

4-1. Gain scheduling middleware (GSM)

4.1.1 DC Motor Application

GSM will be applied on the networked PI controller for a DC motor. The transfer function of the PI controller is given as:

$$G_c(s) = \frac{K_p(s + z_c)}{s}, \quad (32)$$

where $z_c = K_I/K_p$ is a constant. The optimal PI gains for the non-delay case are determined by the root locus approach and satisfy the following design requirements as shown in (23). $(K_I^*, K_p^*) = (25.572, 0.995)$ will satisfy the requirements. With this choice of gains, $z_c = 25.7$. The response at this gain setting will be the baseline performance to compare to. Therefore, the cost function (24) will be minimal for this point.

Let β be the gain scheduling parameter that will modify the gain. In this case, β will modify the PI gain in the following way:

$$\beta G_c(s) = \frac{\beta K_p (s + z_c)}{s}. \quad (33)$$

By keeping z_c a constant, the optimization of β becomes a one-dimensional problem rather than a two-dimensional problem where K_I and K_p were allowed to vary independently.

The optimal β for a given delay is found by minimizing the cost function given in (24). This can be done for several expected delays. The cost, J , from this optimization is shown in Figure 13 and Figure 14. GSM can then use a look-up table to select the appropriate β for the delay measured by the network traffic estimator. Table 2 shows the optimal β for the delay set, η , presented previously using the cost function in (24).

Table 2. Optimal β gains for various time delays.

Delay Time (sec)	0.0001	0.001	0.005	0.01	0.05	0.1
Optimal β	1.008	0.93	0.535	0.355	0.095	0.005

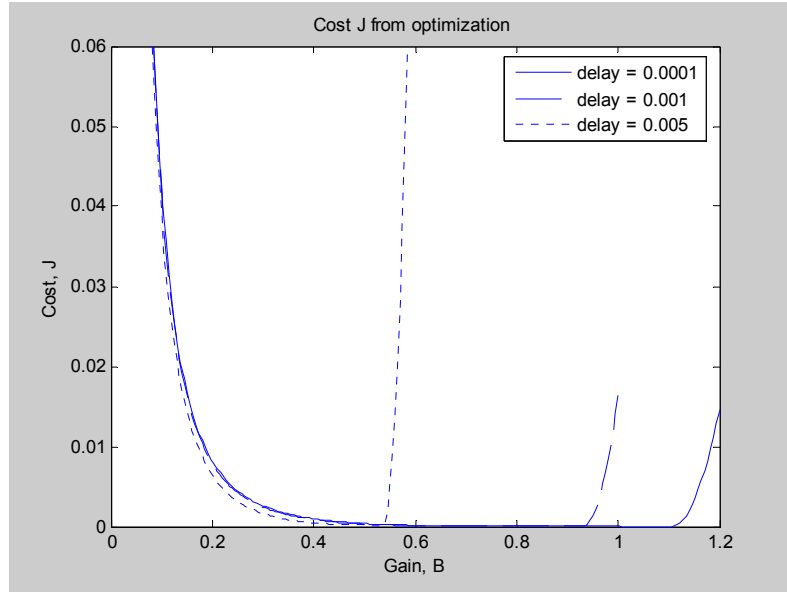


Figure 13. Cost vs. β for $\eta=[0.0001,0.001,0.005]$ network delays.

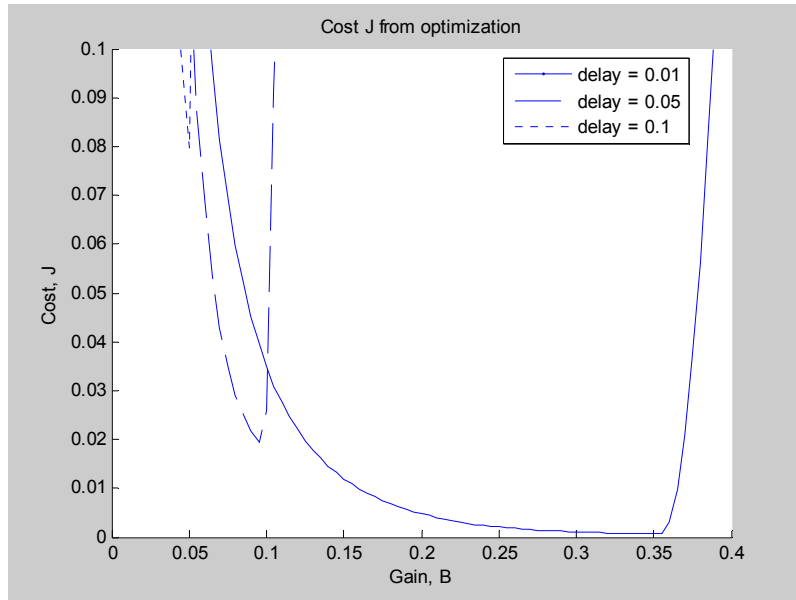


Figure 14. Cost vs. β for $\eta=[0.01,0.05,0.1]$ network delays.

4.1.2 Results

The optimal β decreases as the network delay increases. This lowers the system response time in response to an increase in network delay to ensure stable operation. The cost, J , is more sensitive to variations in delay as the delay time increases, shown in Figure 13 and Figure 14 as the width of the cost curve decreases with higher delays. The step response of the DC motor plant with the nominal PI controller only, and with GSM implemented, in the presence of the set of network delays, η , is shown in Figure 15.

As the network delay increases past 0.01 sec, the controller is unable to stabilize the system. Using the optimal β value found offline, the GSM method is able to maintain system stability at the cost of slower response time. The responses with GSM have fewer oscillations as they approach the reference value. In all cases, the GSM method is able to stabilize to the reference value, while the nominal PI controller fails to stabilize for the 0.05 and 0.01 second delays.

Table 3 shows the performance comparison between the nominal case and all four methods. For all delays, GSM equals or outperforms the uncompensated nominal PI controller.

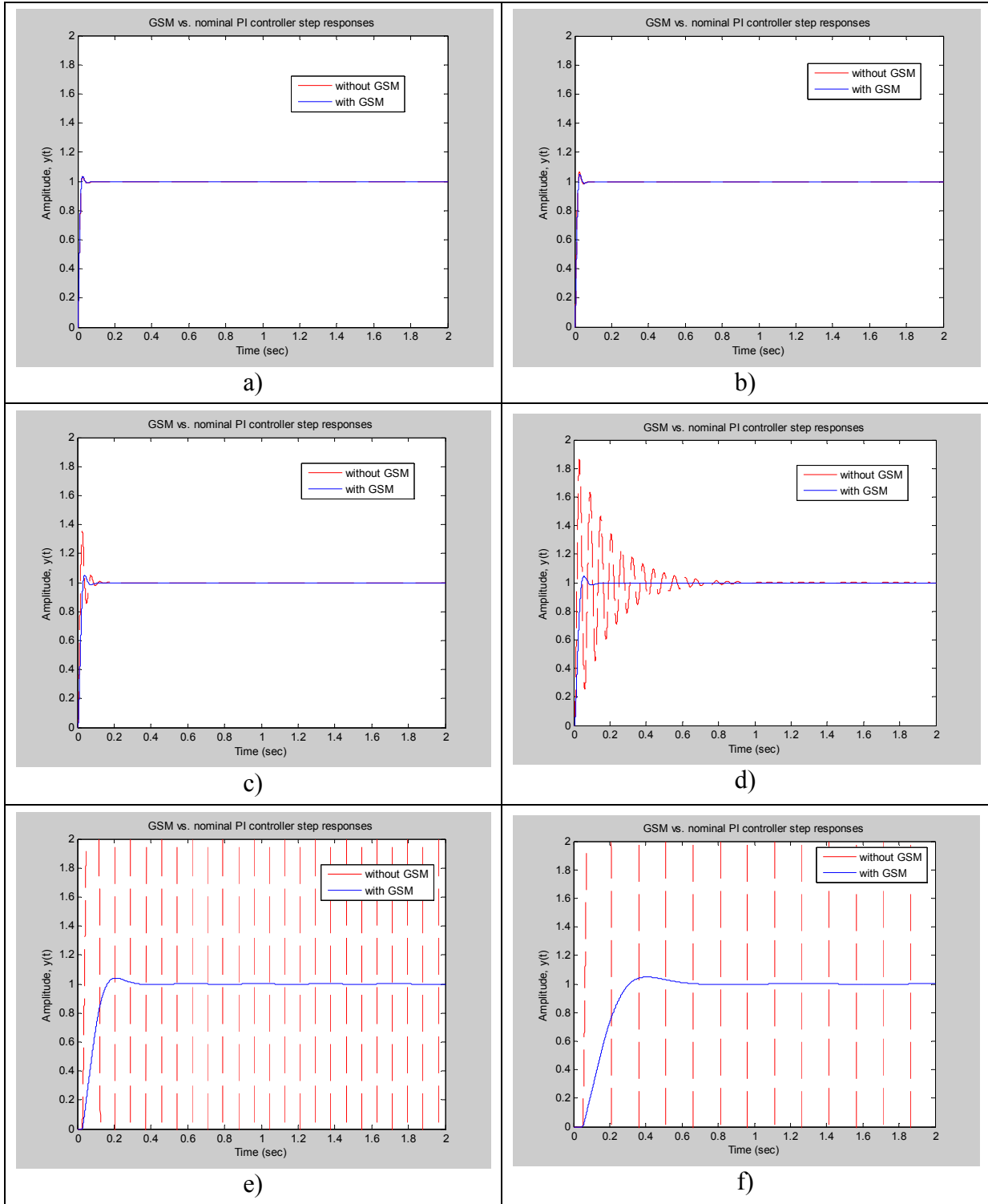


Figure 15. Nominal PI controller vs. GSM step responses for a) $\eta = 0.0001$ b) $\eta = 0.001$ c) $\eta = 0.005$ d) $\eta = 0.01$ e) $\eta = 0.05$ f) $\eta = 0.1$ seconds.

Table 3. Performance comparison of four methodologies for different RTT network delays.

	Delay Time (sec)					
Control Scheme	0.0001	0.001	0.005	0.01	0.05	0.1
Nominal	4.68E-09	0.016480118	6.183768417	43.88451303	3.95E+57	2.65E+45
GSM	0	1.53E-06	1.45E-04	6.74E-04	0.019338165	0.079725567
Robust	0.247174844	0.001999007	0.016218814	0.037692917	0.505627477	6.074238664
LQG	0	0.417990105	30.57058632	4.01E+02	1.05E+05	3.61E+07
Queuing	0	0	0	0	0	0

4-2. Optimal stochastic method

4.2.1 DC motor application

The constraining assumption of the optimal stochastic method is that $\tau^{sc} + \tau^{ca} < T$. Since τ^{sc} and τ^{ca} must be measured and used in the calculation of the LQG gain, timestamps are added to each packet in order to measure the delay experienced. The standard LQG problem regulates the plant output around zero. In this case, it is desired for the system to track a reference input. In order to formulate the regulator as a tracking problem, the output, $y(k)$, must be compared to the reference signal, $r(k)$. The goal is then to drive the error between the output and the reference to zero. A common practice is to add an integrator to the error signal, $e(k) = y(k) - r(k)$, to drive it to zero [19]. The reformulated LQG problem can then be solved using the Matlab Control System toolbox.

For the DC motor, the LQG control objective is to minimize the cost function:

$$J(u) = \int_0^{\infty} (x^T Q x + u^T R u + 2x^T N u) dt, \quad (34)$$

where $Q = \text{diag}(1e6, 10, 10)$, $R = 1$, and $N = 0$, for the augmented plant description with the integrator on the error signal, where the states are defined as $x = \{e, i, \omega\}^T$. Q , R , and N are determined experimentally by finding the LQR gain whose performance best mirrors that of the PI controller during a non-delay case trial. This puts the largest penalty on the error while also keeping the control values within the acceptable limits of $\pm 12V$. The white noises $v(t)$ and $w(t)$ are assumed to be zero in order to properly compare the performance of each methodology. The assumption is that the network delay is the dominating factor of the system performance and stability.

Using the problem formulation above, the LQG controller is designed without taking any delays into account. This yields a gain $K = [-1000, 12.404, 7.2913]$.

4.2.2 Results

Table 3 shows the comparison of (24) between the nominal PI controller and the LQG controller. The step response comparisons are shown for the LQG controller versus the nominal PI controller in Figure 16. It performs very well for 0.0001 sec delay since the major assumption of $\tau^{sc} + \tau^{ca} < T$ is met. Notice that it does manage to lower the cost at the highest delays of 0.05 and 0.1 seconds even though the tracking performance fails to stabilize. The performance degrades faster than the nominal case since T is held constant at 0.0001, while the delay is increased. This invalidates the original assumption.

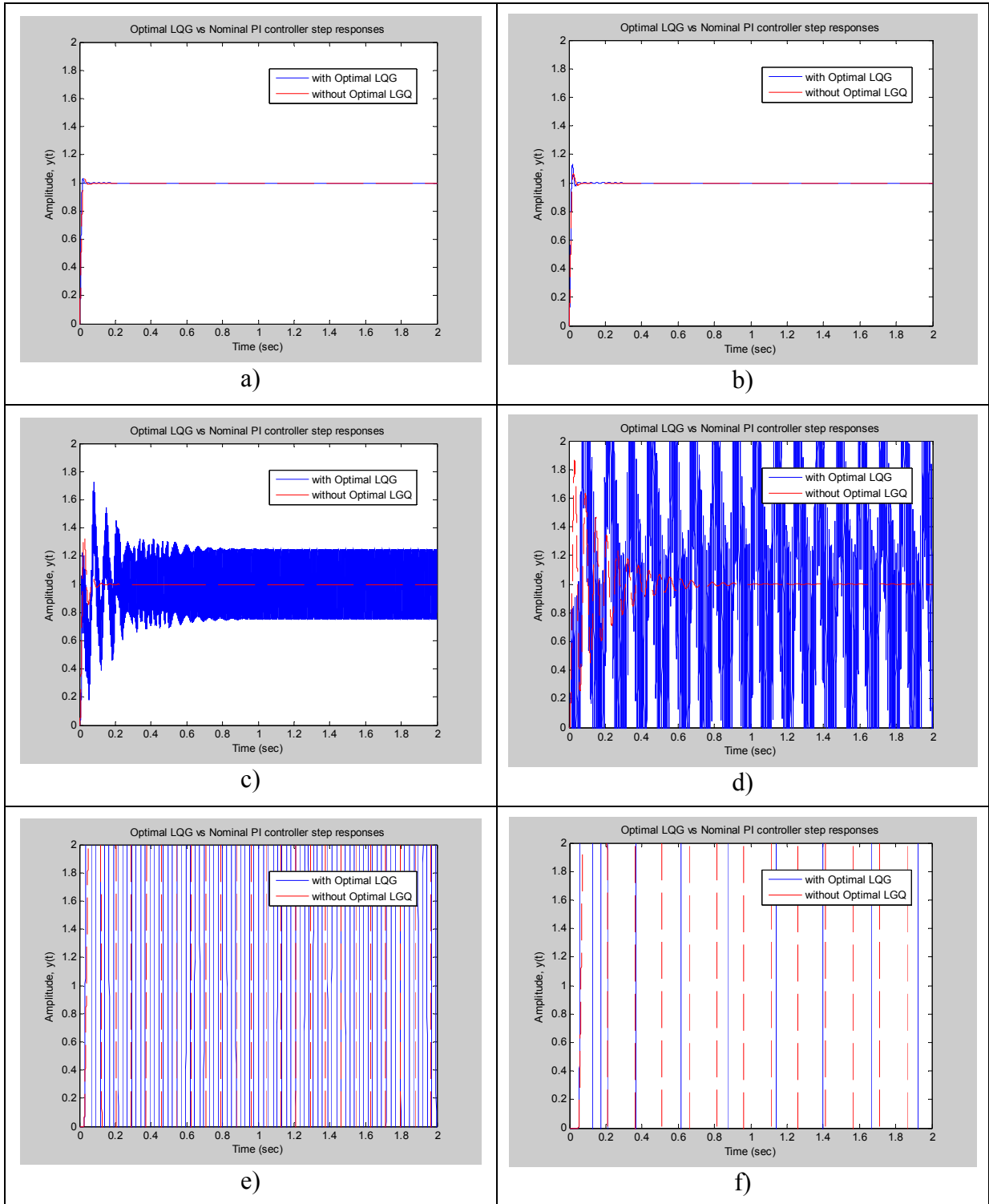


Figure 16. Nominal PI controller vs. Optimal stochastic method step responses for a) $\eta = 0.0001$ b) $\eta = 0.001$ c) $\eta = 0.005$ d) $\eta = 0.01$ e) $\eta = 0.05$ f) $\eta = 0.1$ seconds.

4-3. Queuing methodology

4.3.1 DC motor application

Since full state information is available, the observer block shown in Figure 3 and Figure 4 is omitted. Thus for a δ -step delay, where δ is the delay measured in terms of the number of sample times, δ predictor blocks will be needed. For example, the 0.001 RTT delay will require $0.001/0.0001$ or 10 predictor blocks. It should be noted that the queuing method depends on the accuracy of the model and observer. Since the model is accurate and no observer is used, we should expect to see very good results.

4.3.2 Results

The results of the queuing method versus the nominal PI controller are shown in Figure 17. Note that the responses for each delay are identical. This is because of the perfect model and full state information. Table 3 shows the costs associated with the queuing methodology for the different delays. As noted before, because of the perfect model accuracy, the predictor block is able to predict the states perfectly regardless of the delay. However, the queue length increases as the delay increases, as shown in Table 4. This could provide difficulties when implementing the queuing methodology for a real-time application since the computation time could be significant. The queuing methodology works well when the delay is bounded by a constant and the plant model is known. Also, it is more feasible for time sensitive applications when the sample time is close to the delay time to limit the number of predictor blocks required. These multiple calculations required for the predictor blocks could introduce more delay than actually experienced if done in real-time.

Table 4. Queue lengths for various delays.

Delay (sec)	Queue Length
0.0001	1
0.001	10
0.005	50
0.01	100
0.05	500
0.1	1000

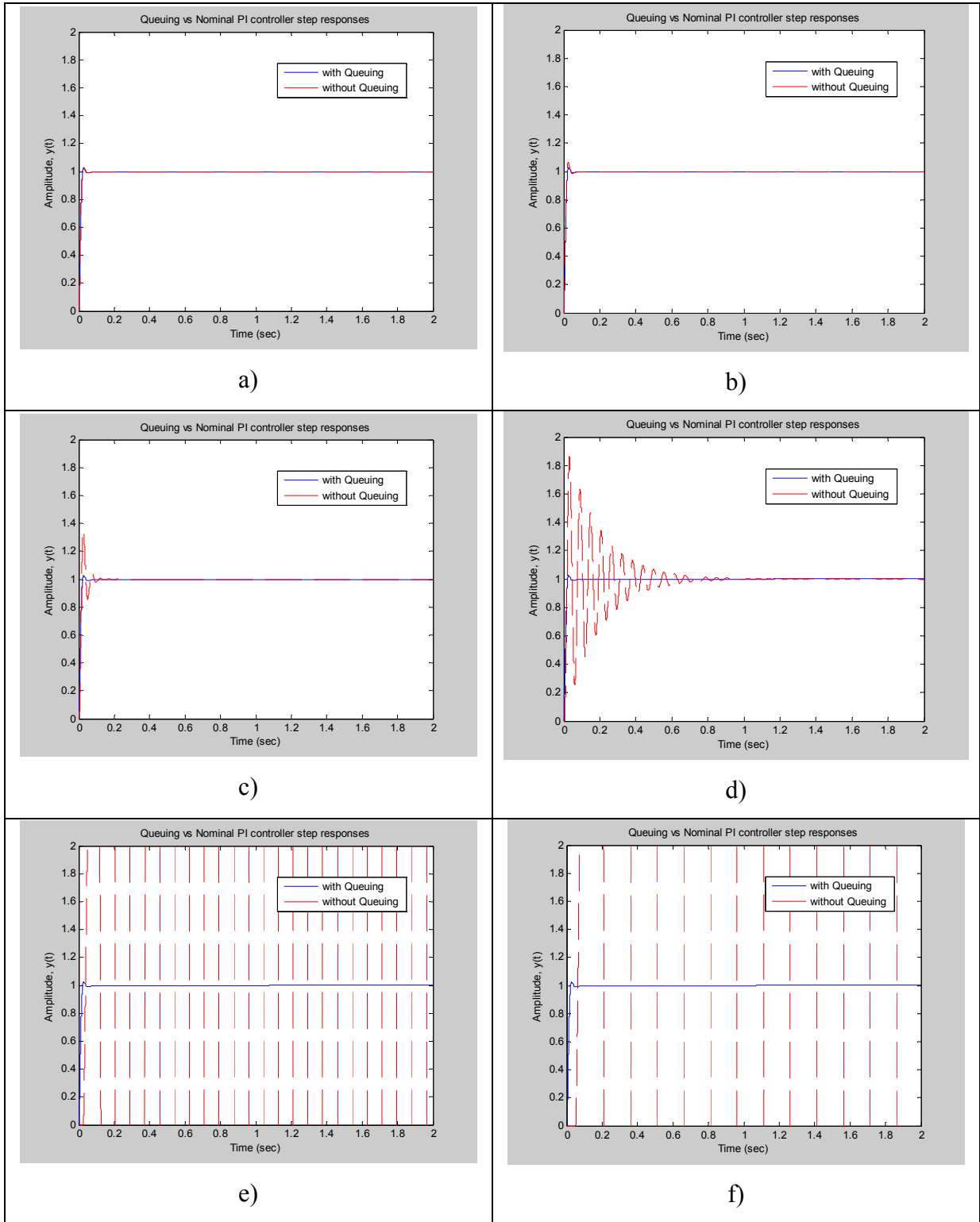


Figure 17. Nominal PI controller vs. Queuing methodology step responses for a) $\eta = 0.0001$ b) $\eta = 0.001$ c) $\eta = 0.005$ d) $\eta = 0.01$ e) $\eta = 0.05$ f) $\eta = 0.1$ seconds.

4-4. Robust control methodology

4.4.1 DC motor application

G represents the plant dynamics and are given for the DC motor in (22). These will be used to form the interconnection matrix P . By experimentation, the performance weight, W_1 , was found to be

$$W_1(s) = \frac{25}{s^2 + 20s + 1}. \quad (35)$$

This gives satisfactory performance for all delays. W_2 and W_3 are given in (17).

Next the system is analyzed using the *dkitgui* tool in Matlab. This tool uses DK iteration to generate controllers for the system and analyze the robustness using μ -synthesis. For the DC motor, the frequency range 0.001 to 1000 with 1000 points was used. The uncertainty structure is a 2x2 matrix. Eight iterations were completed for each delay time, generating eight controllers. Each one was tested for optimal performance based on the performance measure in (24). The best one was chosen for the final trials. The controller chosen also had the lowest peak μ value given by the *dkitgui* tool. A lower μ value indicates a smaller effect of a disturbance (i.e. network delay) on the system.

Overall, the robust methodology does a great job as compensating for network delay. The major benefit is that the controller does not need a priori knowledge of the network delays, nor does it need the past delay measurements as many of the other methods do. It does, however, require that the weights be carefully set such that they cover the uncertain delays for satisfactory performance.

4.4.2 Results

The optimal robust controller for each delay is shown in Table 5 . The number indicates during which iteration the optimal performance, or lowest cost, was achieved. Thus for the first delay of 0.0001 sec, the 4th D-K iteration controller achieved the best results and was, therefore, used for the final simulation runs. In all cases but $\eta = 0.1\text{sec}$, the controller with the lowest peak μ value produced the best performance. The peak μ values for each of the eight controllers generated for each delay time can be found in Table 6.

Table 5. Optimal robust controller for each delay.

Delay time (sec)	0.0001	0.001	0.005	0.01	0.05	0.1
Controller number	4	7	8	7	3	1

Table 6. Peak μ values for controller generated during each DK iteration.

Optimal controller values are shown in *italic*.

Delay time (sec)	D-K iteration number							
	1	2	3	4	5	6	7	8
0.0001	0.029	0.012	0.013	<i>0.012</i>	0.012	0.013	0.018	0.012
0.001	0.111	0.046	0.044	0.046	0.044	0.044	<i>0.044</i>	0.044
0.005	0.317	0.109	0.108	0.106	0.123	0.111	0.109	<i>0.107</i>
0.01	0.516	0.163	0.155	0.157	0.155	0.157	<i>0.155</i>	0.157
0.05	1.9963	0.641	<i>0.641</i>	1.1826	1.9359	0.941	0.934	0.975
0.1	2.9939	1.4372	2.6053	5.1912	3.0666	3.6556	3.4601	3.3631

The step responses comparing the robust control method and the nominal PI controller are shown in Figure 18. The cost for the 0.0001 sec delay is larger than the 0.001 delay because the D-K iteration tries to optimize its control to the H_∞ performance measure that differs from the one given in (24). Aside from that result, the cost increases as the delay time increases, and on the whole, the robust method compensates for the delay as shown by the decreased cost from the nominal case.

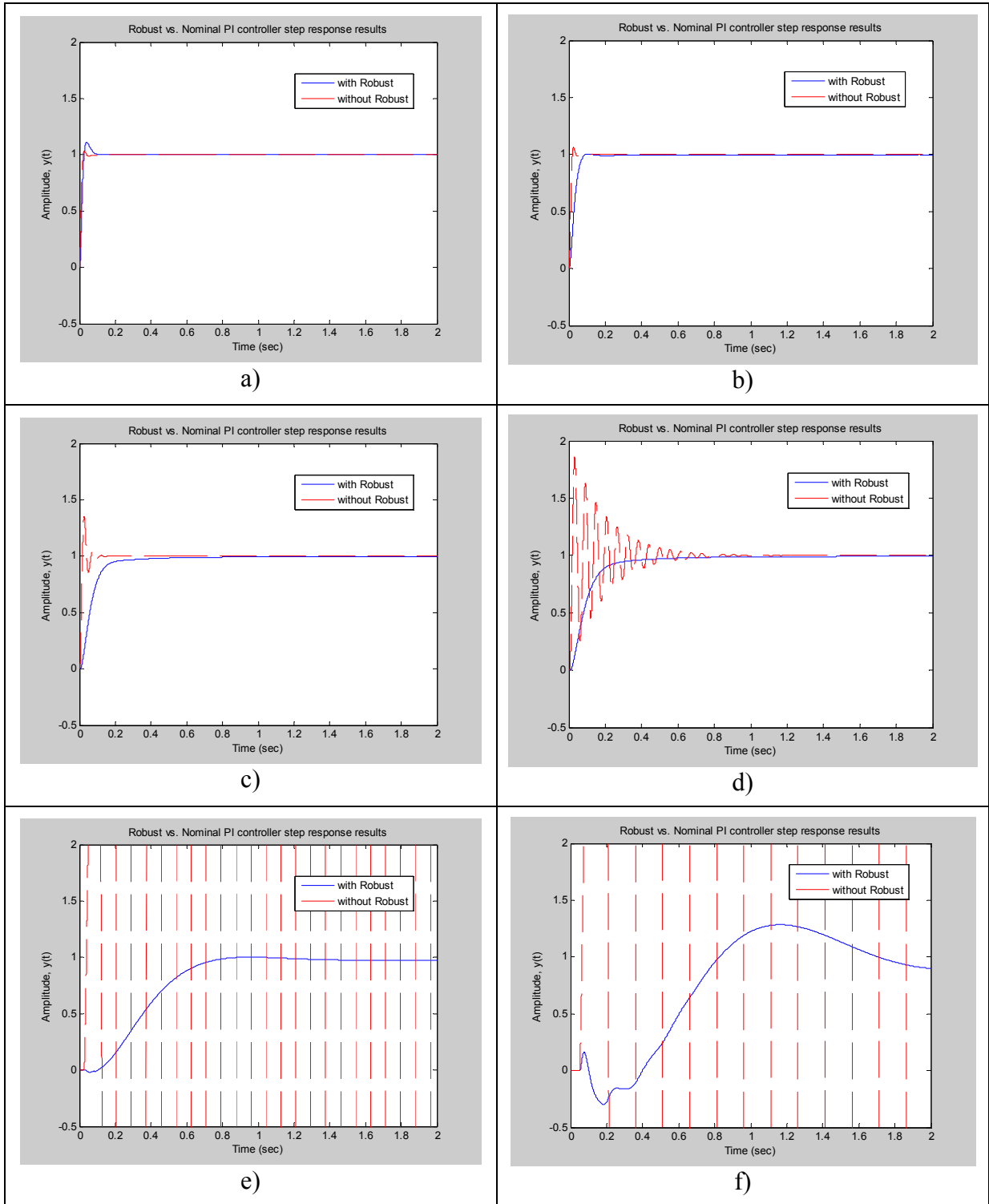


Figure 18. Nominal PI controller vs. Robust control methodology step responses for a) $\eta = 0.0001$ b) $\eta = 0.001$ c) $\eta = 0.005$ d) $\eta = 0.01$ e) $\eta = 0.05$ f) $\eta = 0.1$ seconds.

CHAPTER 5

NOISE

The previous results were conducted under ideal conditions. The performance of each method was also investigated under noisy conditions. Noise was introduced into the control loop at the sensor measurement to simulate modeling errors (both state and measurement). Figure 19 shows how the noise, $w(k)$, is introduced into the system. The noise is Gaussian white noise with the zero mean and a given variance, $v(k) = G(\mu, \sigma^2)$. The mean, μ , is set to zero and the variance, σ^2 , was chosen so that the distribution effected between 0 and 10% of the final reference value of 1 rad/sec. This was determined by $\Psi = 3\sigma$, where Ψ stands for the noise level with respect to the final value, and σ is the standard deviation. 3σ was chosen as it accounts for 99.7% of a normal distribution. Thus for $\psi \in \{0.5\% \quad 1\% \quad 2\% \quad 5\% \quad 10\%\}$, the variance used is

$$\sigma^2 \in \left\{ 2.7778\text{e-}06, 1.1111\text{e-}05, 4.4444\text{e-}05, \right\}. \quad (36)$$

The distributions of the variances are shown in Figure 20. The performance is measured with the same performance measure described earlier in (24). With noise added, the typical step response is shown in Figure 21.

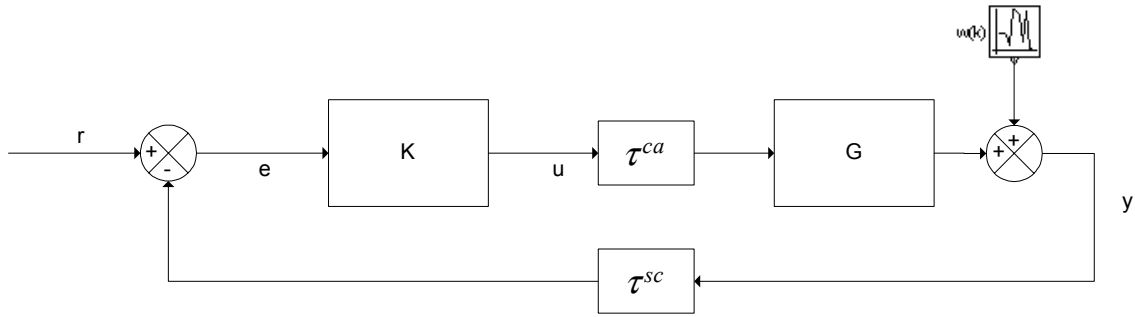


Figure 19. Noise, $w(k)$ introduced into the control loop.

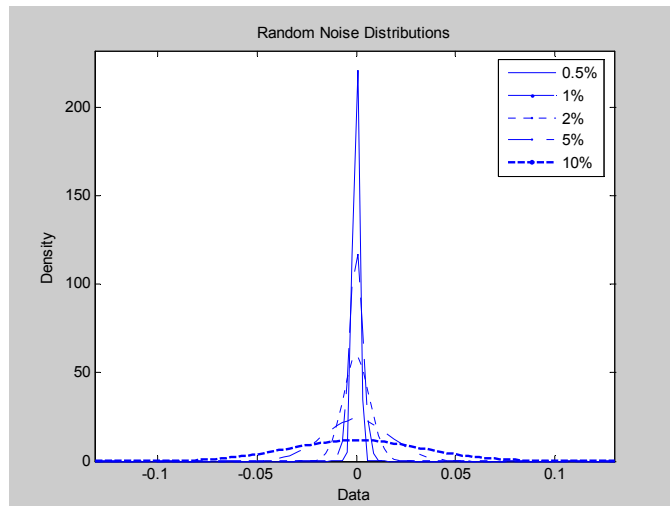


Figure 20. Noise distributions.

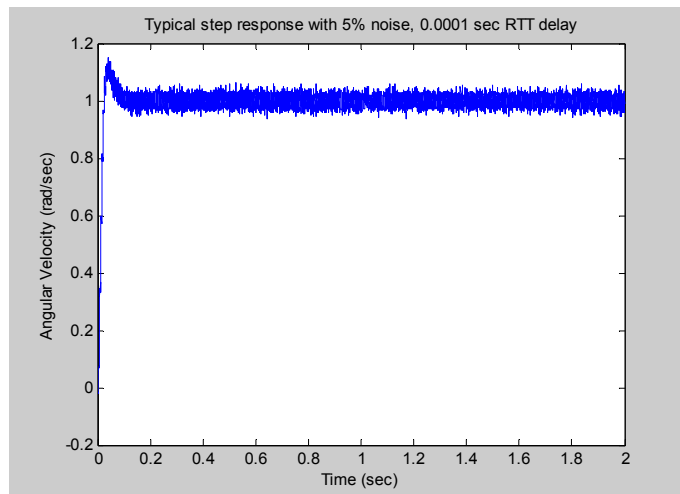


Figure 21. Typical step response with 5% noise, 0.0001 sec RTT delay.

It is expected that the introduction of noise will decrease the performance of the system. As the noise increases, the performance should decrease which will be indicated by a higher cost measured by (24). Figure 22 shows the results for each method with respect to RTT delay and the noise level.

In general, as the delay and noise increase, so does the cost J . The queuing and GSM methods have the lowest cost with respect to noise or network delay.

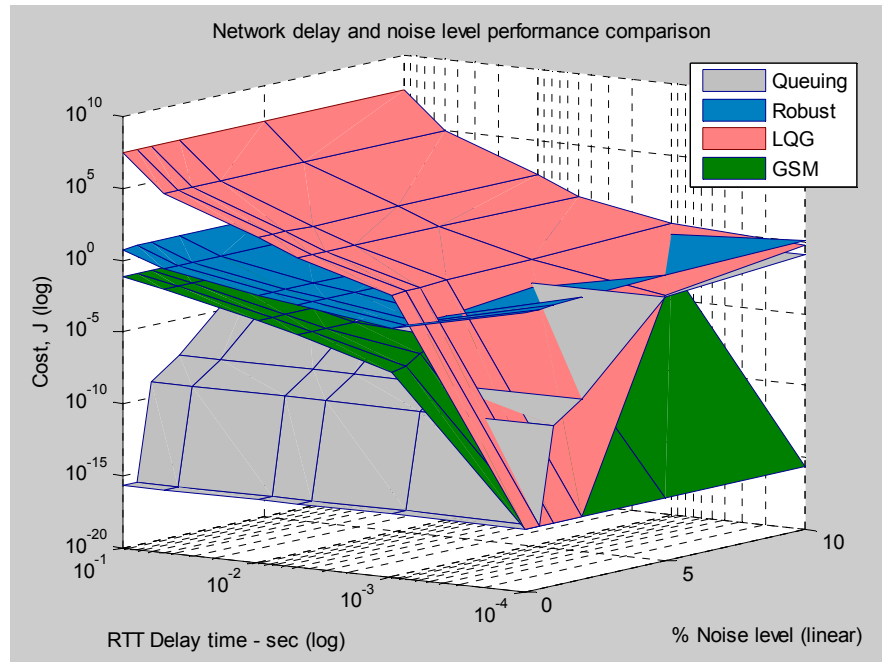


Figure 22. Performance comparison with noise.

5-1. GSM

Figure 23 shows the performance of the GSM methodology with noise. For small delays (i.e. under 0.01 sec), the noise had a larger effect on the performance than for larger delays. In the case of larger delays, the noise had a much smaller effect on the performance

than the delay, as shown by cost values very similar regardless of the noise level. However, the total cost at even the worse condition is still much lower than the nominal PI controller.

Table 7 gives the cost, J , for each trial.

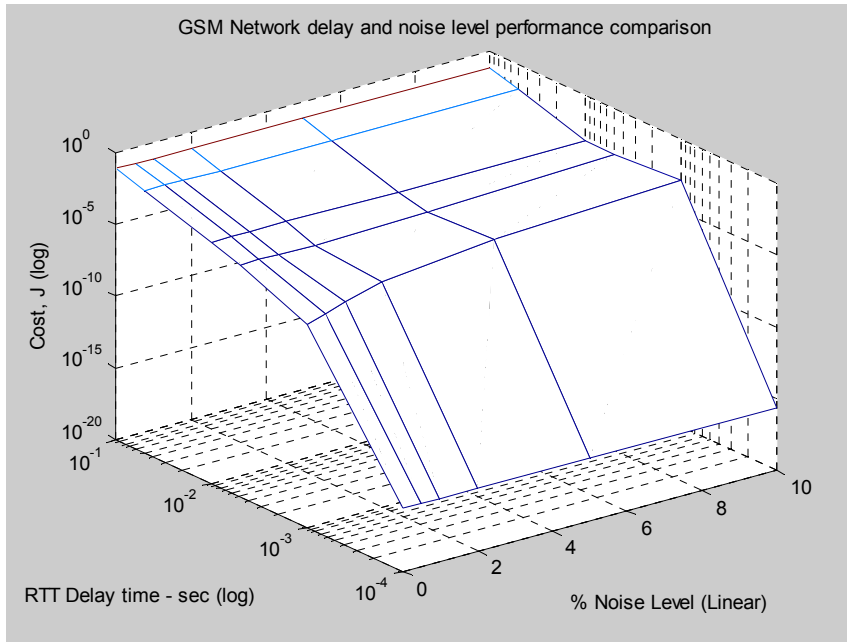


Figure 23. GSM noise vs. delay performance.

Table 7. GSM noise results.

	Noise level					
Delay	0%	0.5%	1%	2%	5%	10%
0.0001	0	0	0	0	0	0
0.001	1.53E-06	3.0506E-06	1.1818E-05	5.2972E-05	3.6717E-04	1.5607E-03
0.005	1.45E-04	1.4359E-04	1.4355E-04	1.4347E-04	2.0716E-04	6.4517E-04
0.01	6.74E-04	6.7411E-04	6.7402E-04	6.7387E-04	6.713E-04	6.6879E-4
0.05	0.019338165	0.019338	0.019337	0.019336	0.019333	0.019315
0.1	0.079725567	0.079723	0.07972	0.079714	0.079668	0.07964

5-2. Optimal stochastic method

Figure 24 presents the performance of the optimal stochastic method with noise. As with the GSM method, the noise effect is more apparent at low delays, 0.001 sec and smaller, than at higher delays. This is partly due to the noisy results of the optimal method even without noise, as shown in Figure 16. Those results are due to the fact that $\tau^{sc} + \tau^{ca} \not\leq T$, where $T = 0.0001\text{sec}$ in this case. Therefore, additional noise will have little effect on the performance of the optimal methodology. However, for where $\tau^{sc} + \tau^{ca} \leq T$ holds true, increased noise causes an increase in cost which indicates a decrease in performance. Table 8 gives the cost, J , for each trial.

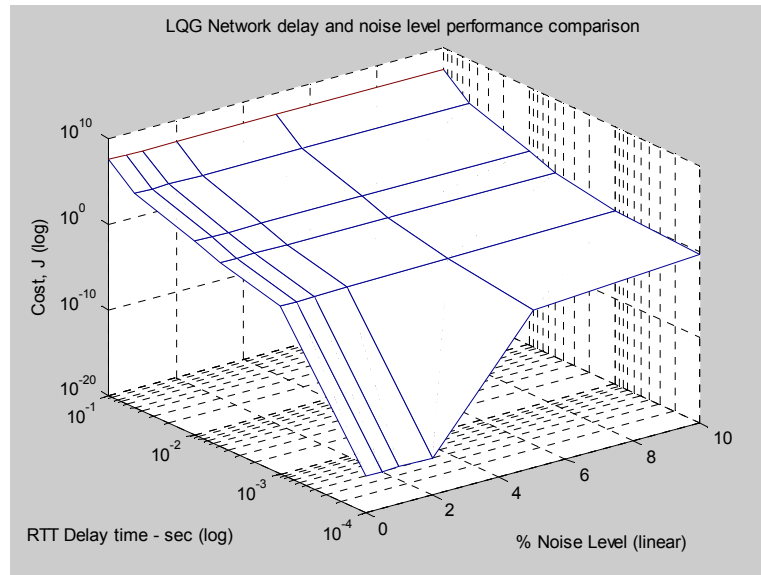


Figure 24. Optimal method noise vs. delay performance.

Table 8. Optimal method noise results.

	Noise level					
Delay	0%	0.5%	1%	2%	5%	10%
0.0001	0	0	0	0	0.028246	0.55458
0.001	0.41799	0.44648	0.49424	0.59705	0.96373	1.769
0.005	30.5706	29.902	30.972	31.712	33.688	24.951
0.01	401.5	427.79	388.85	358.34	383.15	398.74
0.05	1.05E+05	1.0497E+05	1.0506E+05	1.0474E+05	1.0507E+05	1.0528E+05
0.1	3.61E+07	3.6069e+07	3.6068e+07	3.6066e+07	3.606e+07	3.6049e+07

5-3. Queuing method

Figure 25 presents the queuing method's results with respect to noise levels and network delay. As mentioned earlier, because of the high accuracy of the model, the queuing method is able to accurately predict the plant's state regardless of the delay length. Thus an increase in delay has no effect on the performance, as the cost does not rise. However, as noise is introduced, the inputs to the predictor blocks are no longer accurate. This inaccurate prediction results in an increase in the cost as the noise level increases. Therefore, system performance degrades as noise levels increase. Similar to the GSM method, the total cost of the queuing method at the worst condition is still low compared to the nominal PI controller. Table 9 gives the cost, J , for each trial.

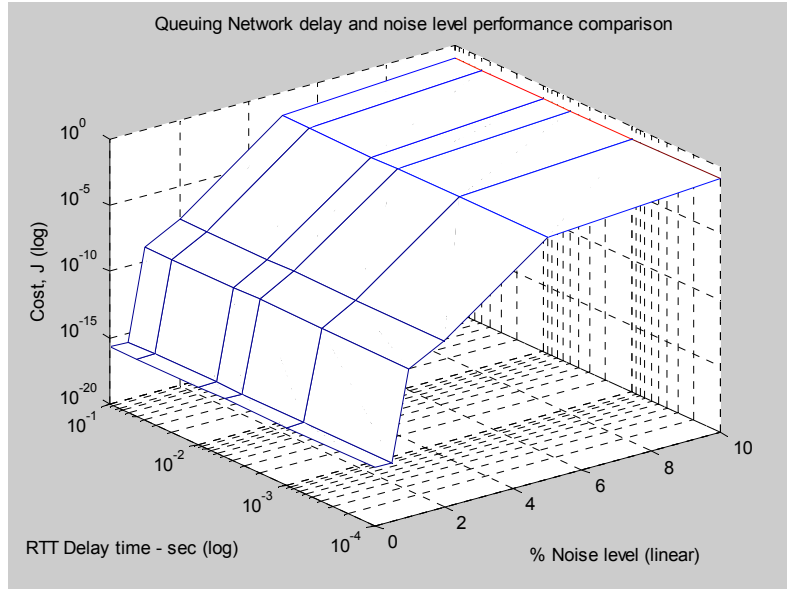


Figure 25. Queuing method noise vs. delay performance.

Table 9. Queuing method noise results.

	Noise level					
Delay	0%	0.5%	1%	2%	5%	10%
0.0001	0	0	1.1135E-09	3.3109E-08	0.018747	0.14272
0.001	0	0	1.2453E-09	3.4573E-08	0.019001	0.12165
0.005	0	0	1.4832E-09	3.7668E-08	0.017853	0.12
0.01	0	0	1.4457E-09	3.7625E-08	0.016157	0.1236
0.05	0	0	1.2905E-09	3.4895E-08	0.017255	0.11824
0.1	0	0	1.2792E-09	3.4681E-08	0.017255	0.12436

5-4. Robust method

Figure 26 shows the performance of the robust methodology for various noise levels and network delays. Unlike the other methods, regardless of the delay an increase in the noise level resulted in an increase in the cost, just as an increase in the network delay resulted in an increase in the cost. Thus, performance degrades as both the network delay and noise levels increase. However, the increase in the cost, J , at each noise level was less than other methods as the network delay increased, signifying the methods robustness to disturbances. Table 10 gives the cost, J , for each trial.

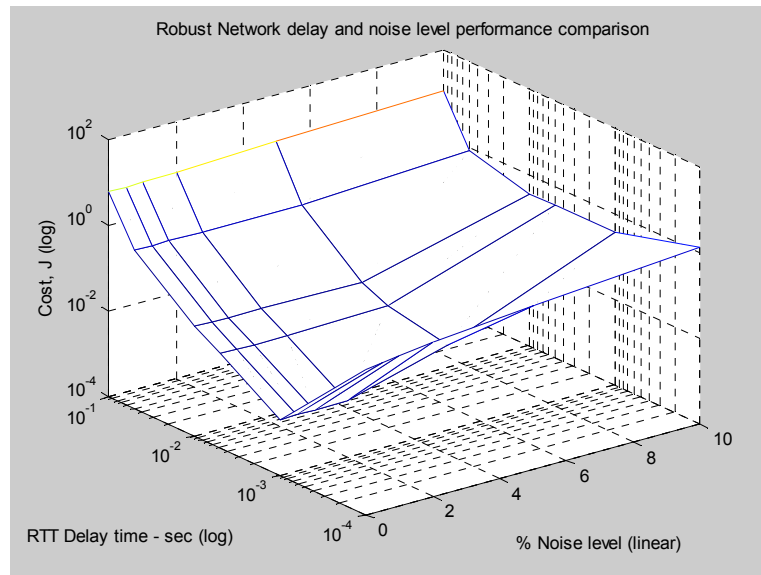


Figure 26. Robust methodology noise vs. delay performance.

Table 10. Robust methodology noise results.

	Noise level					
Delay	0%	0.5%	1%	2%	5%	10%
0.0001	0.247174844	0.27621	0.31253	0.38946	0.6814	1.3254
0.001	0.001999007	0.0020004	0.0020107	0.0020259	0.009546	0.35354
0.005	0.016218814	0.015888	0.015803	0.015502	0.018892	0.35448
0.01	0.037692917	0.037065	0.036059	0.034803	0.034417	0.34935
0.05	0.505627477	0.50487	0.50263	0.50045	0.48733	0.83974
0.1	6.074238664	6.1986	6.3444	6.7151	7.9475	10.544

CHAPTER 6

CONCLUSION

Four methodologies were investigated for use with real-time applications in a NBC setting. The simulation results have shown promising results to apply these methodologies on real-time network-based control systems. The GSM and queuing methods presented the best performance for various delays and noise levels, as compared to the networked nominal PI controller. The robust methodology also showed a significant improvement over the nominal PI controller. The optimal stochastic method showed increased performance over its limited range set by its delay assumption. Each of the top two performers have certain drawbacks, namely: GSM has considerable off-line calculations that must be done in order to schedule the gain for the network traffic conditions, while the queuing method depends significantly on the plant model's accuracy. For critical real-time applications, the appropriate method must be chosen based on the application. All methods have shown the ability to stabilize the closed loop control system and minimize the detrimental delay effect in an IP network environment.

6-1. Benefits Table

The purpose of this section is to provide the benefits and drawbacks of the methodologies so that one may be chosen that best fits the application. Table 11 lists the pros and cons of each methodology. Gain scheduling middleware is best when the original controller cannot be redesigned and β can be determined for a wide range of network delays.

The optimal stochastic method is best when the RTT network delay is less than the sampling time. The queuing methodology is best when an accurate plant model is available and network delays are relatively constant. The robust methodology is best when the network delay varies slightly around relatively constant value since the robust controller only need to be designed once.

Table 11. Pros and Cons of the four NBC methodologies.

	Pros	Cons
Gain scheduling middleware (GSM)	<ul style="list-style-type: none"> • Superb performance for wide range of network delays. • Uses existing controller, eliminating the need for a costly redesign. • Easily integrated into the system. 	<ul style="list-style-type: none"> • Performance comes at the cost of slower response time. • Considerable offline calculations required to find optimal β. • Must have knowledge of the expected range of delays that are to be encountered by the system. • No optimal method of choosing β for changing network conditions.
Optimal stochastic (LQG) methodology	<ul style="list-style-type: none"> • Simplistic implementation. • Sub-optimal approach yields results close to optimal approach. • Excellent performance when $\tau^{sc} + \tau^{ca} < T$. 	<ul style="list-style-type: none"> • Strict sampling time and delay requirements. • Very noisy if RTT delay exceeds sampling time. • Requires a linear system. • Calculation intensive to find $K^*(\tau_k^{sc})$.
Queuing methodology	<ul style="list-style-type: none"> • Network delay becomes transparent with accurate plant model. • Straight forward implementation. • Scalable if network delay is known. 	<ul style="list-style-type: none"> • Performance depends on accurate plant model and observer. • Must dynamically create predictor blocks for current delay conditions. • Significant amount of calculations if sampling time is much less than the network delay.

Table 11, continued

	Pros	Cons
Robust methodology	<ul style="list-style-type: none"> • Each controller can handle delays outside its original specifications. • Noise does not affect performance significantly. • Able to stabilize the system for a wide range of delays. 	<ul style="list-style-type: none"> • Weights for perturbations must be selected carefully to get optimal performance. • H_∞ performance measure differs from desired performance measure. • Several D-K iterations may be required to find best controller.

6-2. Future Work

These methodologies showed promising results for time-sensitive applications during the simulations with a DC motor. Future work would apply these methods on an actual real-time NBC system such as iSpace [8]. In addition, most of the methodologies need some way to select the optimal controller design for the stochastic network delay currently experienced.

REFERENCES

- [1] Y. Tipsuwan and M.-Y. Chow, "Control Methodologies in Networked Control Systems," *Control Engineering Practice*, vol. 11, pp. 1099-1111, 2003.
- [2] F. Goktas, "Distributed control of systems over communication networks," in *Department of computer and information science*. Philadelphia: University of Pennsylvania, 2000.
- [3] R. Luck and A. Ray, "An observer-based compensator for distributed delays," *Automatica*, vol. 26, pp. 903-908, 1990.
- [4] J. Nilsson, "Real-time control systems with delays," in *Department of Automatic Control*. Lund, Sweden: Lund Institute of Technology, 1998, pp. 138.
- [5] Y. Tipsuwan, "Gain scheduling for networked control system." Raleigh NC: North Carolina State Univ., 2003, pp. xiv, 160 p.
- [6] 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.
- [7] J. B. Nilsson, B., "LQG control over a Markov communication network," presented at Decision and Control, 1997., Proceedings of the 36th IEEE Conference on, 1997.
- [8] W.-L. Lueng, R. Vanijjirattikhan, Z. Li, L. Xe, T. Richards, B. Ayhan, and M.-Y. Chow, "Intelligent space with time sensitive applications," presented at IEEE/ASME International Conference on Advanced Intelligent Mechatronics, Monterey, California USA, 2005.
- [9] R. Vanijjirattikhan, M.-Y. Chow, P. Szemes, and H. Hashimoto, "Mobile Agent Gain Scheduler Control in Inter-Continental Intelligent Space," presented at The 2005 IEEE International Conference on Robotics and Automation (ICRA05), Barcelona, Spain, 2005.
- [10] Y. Tipsuwan and M.-Y. Chow, "Gain adaptation of mobile robot for compensating QoS deterioration," presented at IECon'02, Sevilla, Spain, 2002.
- [11] J. Nilsson, B. Bernhardsson, and B. Wittenmark, "Stochastic analysis and control of real-time systems with random time delays," *Automatica*, vol. 34, pp. 57-64, 1998.

- [12] R. Luck and A. Ray, "Experimental Verification of a delay compensation algorithm for integrated communication and control systems," *Int. J. Contr.*, vol. 59, pp. 1357-1372, 1994.
- [13] F. Goktas, J. M. Smith, and R. Bajcsy, "Telerobotics over communication networks," presented at Decision and Control, 1997., Proceedings of the 36th IEEE Conference on, 1997.
- [14] F. Goktas, J. M. Smith, and R. Bajcsy, "mu-synthesis for distributed control systems with network-induced delays," presented at Decision and Control, 1996., Proceedings of the 35th IEEE, 1996.
- [15] Y. Tipsuwan and M.-Y. Chow, "Gain scheduler middleware: a methodology to enable existing controllers for networked control and teleoperation - part I: networked control," *Industrial Electronics, IEEE Transactions on*, vol. 51, pp. 1218-1227, 2004.
- [16] Y. Tipsuwan and M.-Y. Chow, "On the gain scheduling for networked PI controller over IP network," *Mechatronics, IEEE/ASME Transactions on*, vol. 9, pp. 491-498, 2004.
- [17] P. Lundstrom, Z. Wang, and S. Skogestad, "Representation of uncertain time delays in the H-inf framework," *International Journal of Control*, vol. 59, pp. 627-638, 1994.
- [18] G. J. Balas, J. C. Doyle, K. Glover, A. Packard, and R. Smith, "mu-Analysis and Synthesis Toolbox: User's Guide," *The Mathworks Inc*, 1991.
- [19] *Example: LQG Design for Set Point Tracking*: The Mathworks Inc, 2005.

APPENDIX

A1. GSM

A1.1. GSM Simulink diagram

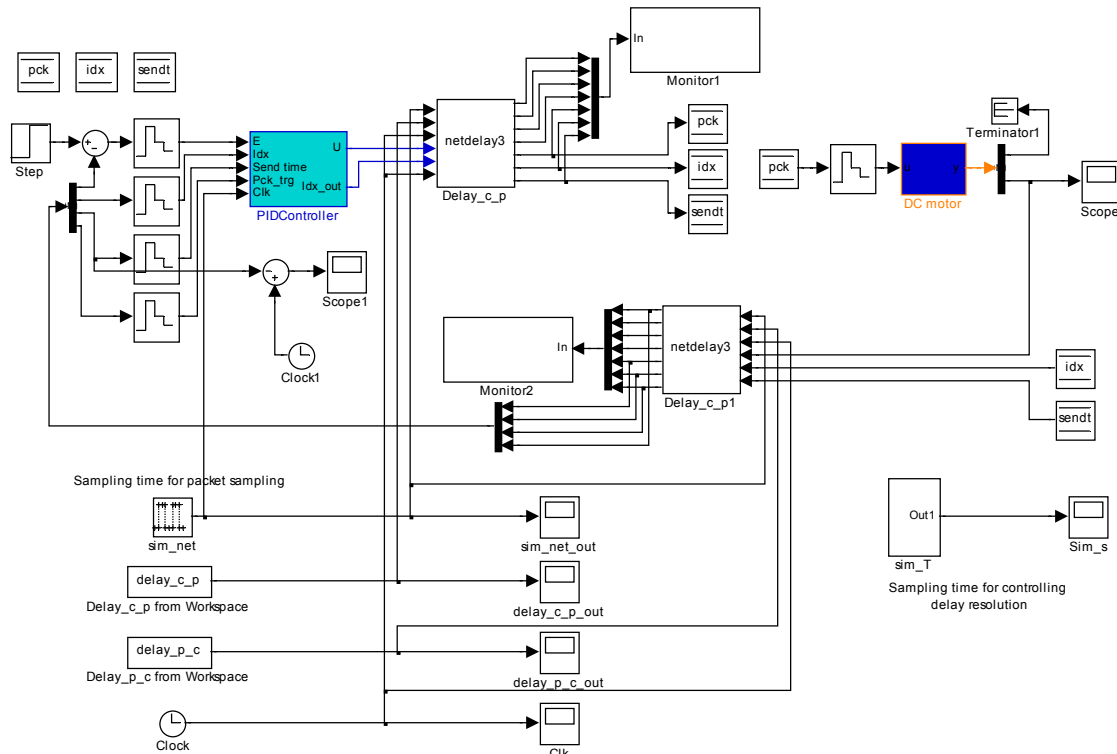


Figure 27. find_beta_sim.mdl

A1.2. GSM Simulink files

A1.2.1 *find_beta_costs.m*

```
% Optimization from 0-2s
% Using forward approximation for Integral gain
% Using exponential delay variation from median delay
% Run with find_beta_sim.mdl
% modified from yodyium's by tyler 3/28/05
```

```
clear all
close all
```

```
% Delay median to consider
%delay_mat=[0.01:0.01:0.1] 0.2 0.3 0.4 0.5];
delay_mat=[.0001 .001 .005 .01 .05 .1]; %6 delays
%delay_mat=[0.00001];
```

```

delay_len=length(delay_mat);

% Define phi (parameter of exponential distribution - variance)
%phi_mat=[0 [0.001:0.001:0.02] 0.03 0.04 0.05 0.06 0.07];
phi_mat=[0];
phi_len=length(phi_mat);

% Initialize variables and create cost matrix
gain_rel=200;%gain_rel=100;
cost1_mse=zeros(gain_rel,phi_len,delay_len); %cost for MSE
cost2_po=zeros(gain_rel,phi_len,delay_len); %cost for Percentage overshoot
cost3_tr=zeros(gain_rel,phi_len,delay_len); %cost for rise time (define as time to go from 10% to 90% of final
value)

Ki_mat=zeros(gain_rel,phi_len,delay_len);
Kp_mat=zeros(gain_rel,phi_len,delay_len);

% PI controller

Kp=.995;
zc = 25.7;
Ki=zc*Kp;

%Ratio Ki/Kp=1/0.0526=19

% Simulation time
final_t=2
t=[0:0.0001:final_t];
N=length(t);

% Searching bound of beta gain
%bound=[1.16 0.677 0.483 0.378 0.313 0.261 0.229 0.205 0.185 0.168];
%bound=[8.18 4.36 3.08 2.36 1.93 1.73 1.55 1.39 1.26 1.16 0.677 0.483 0.378 0.313];
bound=[1.2 1 1 1 1 1];

% Set sampling time in the model
% Controller and plant sampling time

% ZOH at controller
set_param('find_beta_sim/zoh1','Sample time',num2str(0.0001));
% controller sampling time
set_param('find_beta_sim/PIDController','h',num2str(0.0001));
% plant sampling time
set_param('find_beta_sim/zoh2','Sample time',num2str(0.0001));

%set others
set_param('find_beta_sim/zoh3','Sample time',num2str(0.0001));
set_param('find_beta_sim/zoh4','Sample time',num2str(0.0001));
set_param('find_beta_sim/zoh5','Sample time',num2str(0.0001));

% Simulation sampling time (representing delay resolution)
set_param('find_beta_sim/sim_T','Tsim',num2str(0.00001));

% Network transmission sampling time

```

```

set_param('find_beta_sim/sim_net','Sample time',num2str(0.00005));
set_param('find_beta_sim/Delay_c_p from Workspace','Sample time',num2str(0.00005));
set_param('find_beta_sim/Delay_p_c from Workspace','Sample time',num2str(0.00005));

% MSE nominal point
%J1_nom=0.00595492255222;
J1_nom=.003405;

% Percentage overshoot nominal point
J2_nom=5;

% Rise time nominal point
%J3_nom=0.117;
J3_nom=0.0140;

for delay_count=1:delay_len %set RTT delay median and vary phi

    delay_count
    for phi_count=1:phi_len %set phi and vary beta
        delta_t=bound(delay_count)/gain_rel;
        beta=0; %edited from 0
        delay=delay_mat(delay_count)/2;

        % Generate exponential random delay from for delay_p_c and delay_c_p
        if (phi_mat(phi_count)>0) %if there is delay variance, create random delay
            t_delay=[0:0.00005:final_t]';
            t_delay_len=length(t_delay);
            delay_v_cp=random('exp',phi_mat(phi_count),t_delay_len,1);
            delay_v_pc=random('exp',phi_mat(phi_count),t_delay_len,1);
            delay_c_p=[t_delay (delay+delay_v_cp)];
            delay_p_c=[t_delay (delay+delay_v_pc)];
        else
            delay_c_p=[0 delay]; %else delay is RTT/2
            delay_p_c=[0 delay];
        end

        phi_count
        for gain_count=1:gain_rel %edited from 1:gain_rel

            gain_count
            Ki_sim=(beta+delta_t)*Ki;
            Kp_sim=(1/zc)*Ki_sim;
            beta+delta_t
            set_param('find_beta_sim/PIDController','Ki',num2str(Ki_sim));
            set_param('find_beta_sim/PIDController','Kp',num2str(Kp_sim));

            Ki_mat(gain_count,phi_count,delay_count)=Ki_sim;
            beta=beta+delta_t;
            [T,X,Y]=sim('find_beta_sim',t);

            Y=X(:,2); %error!!! not y,
            N=length(Y);
            N_mat(gain_count,delay_count)=N;
            % Compute Costs
            % Cost_1

```

```

% Mean-squared error
err=Y-1;
msr=(err'*err)/N;
if (msr>J1_nom)
    cost1_mse(gain_count,phi_count,delay_count)=(msr-J1_nom)^2;
else
    cost1_mse(gain_count,phi_count,delay_count)=0;
end

% Cost_2
% Percentage Overshoot
count=1;
flag_po=0;
po=0;
while (count<=N)
    if (Y(count)>1)
        po=(max(Y)-1)*100;
        break;
    end
    count=count+1;
end
if (po>J2_nom)
    cost2_po(gain_count,phi_count,delay_count)=(po-J2_nom)^2;
else
    cost2_po(gain_count,phi_count,delay_count)=0;
end

% Cost_3
% Rise time
count=1;
flag_st=0;
flag_ed=0;
start_time=0;
end_time=inf;
while (count<=N)
    if (Y(count)>=0.1)&(flag_st==0)
        start_time=T(count);
        flag_st=1;
    end

    if (Y(count)>=0.9)&(flag_ed==0)
        end_time=T(count);
        flag_ed=1;
        break
    end
    count=count+1;
end
rise_time=end_time-start_time;
if (rise_time>J3_nom)
    cost3_tr(gain_count,phi_count,delay_count)=(rise_time-J3_nom)^2;
else
    cost3_tr(gain_count,phi_count,delay_count)=0;
end
end
end

```

```

end
Kp_mat=(1/zc)*Ki_mat;
save cost_betas cost1_mse cost2_po cost3_tr Ki_mat Kp_mat Kp Ki

```

A1.2.2 *find_weights.m*

```

% Optimization from 0-10s
% Using forward approximation for Integral gain
% Using exponential delay variation from median delay
% Run with find_beta_sim.mdl
% modified from yodyium's by tyler 3/28/05

clear all
close all

% Delay median to consider
%delay_mat=[0.01:0.01:0.1] 0.2 0.3 0.4 0.5;
delay_mat=[.0001];
delay_len=length(delay_mat);

% Define phi (parameter of exponential distribution - variance)
%phi_mat=[0 [0.001:0.001:0.02] 0.03 0.04 0.05 0.06 0.07];
phi_mat=[0];
phi_len=length(phi_mat);

% Initialize variables and create cost matrix
gain_rel=200;%gain_rel=100;
cost1_mse=zeros(gain_rel,phi_len,delay_len); %cost for MSE
cost2_po=zeros(gain_rel,phi_len,delay_len); %cost for Percentage overshoot
cost3_tr=zeros(gain_rel,phi_len,delay_len); %cost for rise time (define as time to go from 10% to 90% of final
value)

Ki_mat=zeros(gain_rel,phi_len,delay_len);
Kp_mat=zeros(gain_rel,phi_len,delay_len);

% PI controller

Kp=.995;
zc = 25.7;
Ki=zc*Kp;

%Ratio Ki/Kp=1/0.0526=19

% Simulation time
final_t=2
t=[0:0.0001:final_t];
N=length(t);

% Searching bound of beta gain
%bound=[1.16 0.677 0.483 0.378 0.313 0.261 0.229 0.205 0.185 0.168];
%bound=[8.18 4.36 3.08 2.36 1.93 1.73 1.55 1.39 1.26 1.16 0.677 0.483 0.378 0.313];
bound=[2 2 1 1 1 1 1 1 1 1 0.677 0.483 0.378 0.313];

```

```

% Set sampling time in the model
% Controller and plant sampling time

% ZOH at controller
set_param('find_beta_sim/zoh1','Sample time',num2str(0.0001));
% controller sampling time
set_param('find_beta_sim/PIDController','h',num2str(0.0001));
% plant sampling time
set_param('find_beta_sim/zoh2','Sample time',num2str(0.0001));

%set others
set_param('find_beta_sim/zoh3','Sample time',num2str(0.0001));
set_param('find_beta_sim/zoh4','Sample time',num2str(0.0001));
set_param('find_beta_sim/zoh5','Sample time',num2str(0.0001));

% Simulation sampling time (representing delay resolution)
set_param('find_beta_sim/sim_T','Tsim',num2str(0.00001));

% Network transmission sampling time
set_param('find_beta_sim/sim_net','Sample time',num2str(0.00005));
set_param('find_beta_sim/Delay_c_p from Workspace','Sample time',num2str(0.00005));
set_param('find_beta_sim/Delay_p_c from Workspace','Sample time',num2str(0.00005));

% MSE nominal point
%J1_nom=0.00595492255222;
J1_nom=.003405;

% Percentage overshoot nominal point
J2_nom=5;

% Rise time nominal point
%J3_nom=0.117;
J3_nom=0.014;

for delay_count=1:delay_len %set RTT delay median and vary phi

    delay_count
    for phi_count=1:phi_len %set phi and vary beta
        delta_t=bound(delay_count)/gain_rel;
        beta=0; %edited from 0
        delay=delay_mat(delay_count)/2;
        %delay = 0;
        % Generate exponential random delay from for delay_p_c and delay_c_p
        if (phi_mat(phi_count)>0) %if there is delay variance, create random delay
            t_delay=[0:0.00005:final_t];
            t_delay_len=length(t_delay);
            delay_v_cp=random('exp',phi_mat(phi_count),t_delay_len,1);
            delay_v_pc=random('exp',phi_mat(phi_count),t_delay_len,1);
            delay_c_p=[t_delay (delay+delay_v_cp)];
            delay_p_c=[t_delay (delay+delay_v_pc)];
        else
            delay_c_p=[0 delay]; %else delay is RTT/2
            delay_p_c=[0 delay];
        end
    end
end

```

```

phi_count
for gain_count=1:gain_rel %edited from 1:gain_rel

    gain_count
    Ki_sim=(beta+delta_t)*Ki;
    Kp_sim=(1/zc)*Ki_sim;
    beta+delta_t
    set_param('find_beta_sim/PIDController','Ki',num2str(Ki_sim));
    set_param('find_beta_sim/PIDController','Kp',num2str(Kp_sim));

    Ki_mat(gain_count,phi_count,delay_count)=Ki_sim;
    beta=beta+delta_t;
    [T,X,Y]=sim('find_beta_sim',t);

    Y=X(:,2);
    N=length(Y);
    N_mat(gain_count,delay_count)=N;
    % Compute Costs
    % Cost_1
    % Mean-squared error
    err=Y-1;
    msr=(err'*err)/N;
    if (msr>J1_nom)
        cost1_mse(gain_count,phi_count,delay_count)=(msr-J1_nom)^2;
    else
        cost1_mse(gain_count,phi_count,delay_count)=0;
    end

    % Cost_2
    % Percentage Overshoot
    count=1;
    flag_po=0;
    po=0;
    while (count<=N)
        if (Y(count)>1)
            po=(max(Y)-1)*100;
            break;
        end
        count=count+1;
    end
    if (po>J2_nom)
        cost2_po(gain_count,phi_count,delay_count)=(po-J2_nom)^2;
    else
        cost2_po(gain_count,phi_count,delay_count)=0;
    end

    % Cost_3
    % Rise time
    count=1;
    flag_st=0;
    flag_ed=0;
    start_time=0;
    end_time=inf;
    while (count<=N)

```

```

        if (Y(count)>=0.1)&(flag_st==0)
            start_time=T(count);
            flag_st=1;
        end

        if (Y(count)>=0.9)&(flag_ed==0)
            end_time=T(count);
            flag_ed=1;
            break
        end
        count=count+1;
    end
    rise_time=end_time-start_time;
    if (rise_time>J3_nom)
        cost3_tr(gain_count,phi_count,delay_count)=(rise_time-J3_nom)^2;
    else
        cost3_tr(gain_count,phi_count,delay_count)=0;
    end
end
end
end
Kp_mat=(1/zc)*Ki_mat;
save finding_weights cost1_mse cost2_po cost3_tr Ki_mat Kp_mat

```

A1.2.3 *find_weights_part2.m*

```

load finding_weights.mat
close all;
beta_range = 0.01:.01:2;

w1 = 1/max(cost1_mse)
w2 = 1/max(cost2_po)
w3 = 1/max(cost3_tr(2)) %first one is inf

figure
plot(beta_range,cost1_mse);
figure
plot(beta_range,cost2_po,'-');
figure
plot(beta_range,cost3_tr,'--');

save finding_weights -append w1 w2 w3

```

A1.2.4 *test_findcost.m*

```

load finding_weights w1 w2 w3
load cost_betas

%-----Yod's weights
%w1 = 1.64902;
%w2 = .00833;

```



```

%w3 = .01395;

Kp=.995;
zc = 25.7;
Ki=zc*Kp;

costJ = w1*cost1_mse + w2*cost2_po + w3*cost3_tr;

scale = 1/200;
reduced = scale:scale:2;

phi_len = 1; %must set to last simulations parameters
delay_len = 6; %must set to last simulation parameters
min_costs = zeros(phi_len,delay_len);
opt_beta = zeros(1,2,phi_len,delay_len);
bound=[1.2 1 1 1 1 1];

for phi = 1:phi_len

    for med_delay = 1:delay_len

        temp = costJ(:,phi,med_delay);
        temp1=find(temp==min(temp));
        %temp_len = length(temp1);
        min_costs(phi,med_delay)=temp1(1);
        opt_beta(1,1,phi,med_delay)= bound(med_delay)*min_costs(phi,med_delay)/200;
        opt_beta(1,2,phi,med_delay)= min(costJ(:,phi,med_delay));
    end

end

save optimal_beta_d5 opt_beta costJ

```



```

%A = [0 1; 0 -b/J];
%B = [0; 1/J];
%C = [1 0; 0 1];
%D = [0;0];
%-----

%-----electrical ss--vinput-----
A = [-R/L -Kb/L; K/J -b/J];
B = [ 1/L; 0];

C = [1 0; 0 1];
D = [0;0];
%-----

[mm,jj]=eig(A,'nobalance');
invmm = inv(mm);

sys_dc = ss(A,B,C,D);

A_aug = [zeros(1,2) -1; zeros(2,1) A];
B_aug = [0 ; B];

% LQR gain synthesis
Qn = blkdiag(1000000,10,10);

Rn = [1];

[K_lqr, S1, e1] = lqr(A_aug,B_aug,Qn,Rn);

max_step=0.0001;
disp('initialization done')

w1 = 17.4572;
w2 = 0.0066771;
w3 = 0.8777;

delay_set = [0.0001 0.001 0.005 0.01 0.05 0.1];

delay_c_p_tune = 0;
delay_p_c_tune = 0;

delay_len = length(delay_set);

cost1_mse=zeros(delay_len,1); %cost for MSE
cost2_po=zeros(delay_len,1); %cost for Percentage overshoot
cost3_tr=zeros(delay_len,1); %cost for rise time (define as time to go from 10% to 90% of final value)

set_param('lqr_dc_delay_tweak/zoh','Sample time','max_step');
set_param('lqr_dc_delay_tweak/zoh1','Sample time','max_step');
set_param('lqr_dc_delay_tweak/Unit Delay','Sample time','max_step');

final_t=2
t=[0:max_step:final_t];

```

```

N=length(t);

results = zeros(N,delay_len);
% MSE nominal point
%J1_nom=0.00595492255222;
J1_nom=.003405;

% Percentage overshoot nominal point
J2_nom=5;

% Rise time nominal point
%J3_nom=0.117;
J3_nom=0.0140;

for delay_count=1:delay_len %set RTT delay median and vary phi

    sysd=c2d(sys_dc,delay_set(delay_count));
    [Ad,Bd,Cd,Dd]=ssdata(sysd);

    delay_count
    real_delay = delay_set(delay_count)/2;
    dt = delay_set(delay_count);
    phi = mm*[exp(jj(1,1)*dt) 0; 0 exp(jj(2,2)*dt)]*invmm;
    gamma = mm*[(((1/jj(1,1))*exp(jj(1,1)*dt))-1) 0; 0 (((1/jj(2,2))*exp(jj(2,2)*dt))-1)]*invmm*B;
    set_param('lqr_dc_delay_tweak/phi','Gain','Ad');
    set_param('lqr_dc_delay_tweak/gamma','Gain','Bd');

    delay_c_p = [real_delay];
    delay_p_c = [real_delay];

    [T,X] = sim('lqr_dc_delay_tweak',t);
    N=length(Y); %angular velocity
    results(:,delay_count) = Y(:,2); %store results

Y=Y(:,2);
% Compute Costs
% Cost_1
% Mean-squared error
err=Y-1;
msr=(err'*err)/N;
if (msr>J1_nom)
    cost1_mse(delay_count,1)=(msr-J1_nom)^2;
else
    cost1_mse(delay_count,1)=0;
end

% Cost_2
% Percentage Overshoot
count=1;
flag_po=0;
po=0;
while (count<=N)
    if (Y(count)>1)
        po=(max(Y)-1)*100;
        break;

```

```

        end
        count=count+1;
    end
    if (po>J2_nom)
        cost2_po(delay_count,1)=(po-J2_nom)^2;
    else
        cost2_po(delay_count,1)=0;
    end

    % Cost_3
    % Rise time
    count=1;
    flag_st=0;
    flag_ed=0;
    start_time=0;
    end_time=inf;
    while (count<=N)
        if (Y(count)>=0.1)&(flag_st==0)
            start_time=T(count);
            flag_st=1;
        end

        if (Y(count)>=0.9)&(flag_ed==0)
            end_time=T(count);
            flag_ed=1;
            break
        end
        count=count+1;
    end
    rise_time=end_time-start_time;
    if (rise_time>J3_nom)
        cost3_tr(delay_count,1)=(rise_time-J3_nom)^2;
    else
        cost3_tr(delay_count,1)=0;
    end
end

costJ = w1*cost1_mse + w2*cost2_po + w3*cost3_tr
save lqg_run cost1_mse cost2_po cost3_tr costJ results w1 w2 w3 Qn Rn K_lqr

```

A3. Queuing methodology

A3.1. Simulink diagram

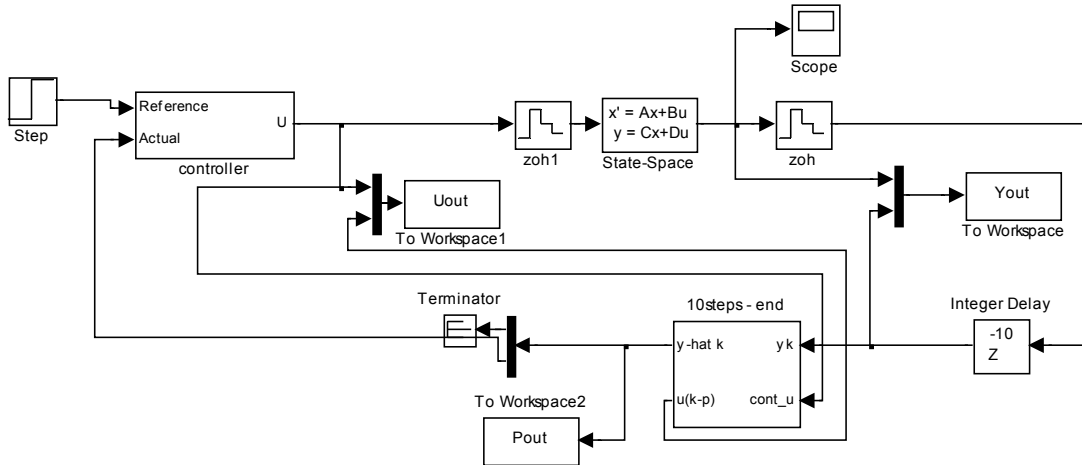


Figure 29. queue_comp3_sub10.mdl

A3.2. Simulink files

A3.2.1 *Setup_test_sub10_init.m*

```
clear all;
close all;
```

```
b = 0.1e-3;
R = 6.43;
J = 3.53e-6;
K = 2.55e-3;
Kb = 0.255e-3;
L = 28.88e-3;
```

```
w1 = 17.4544;
w2 = 0.0021;
w3 = 0.8807;
```

```
%-----voltage input
%A = [0 1; 0 -b/J-(K*Kb)/(J*R)];
%B = [0; K/(J*R)];
%C = [1 0; 0 1];
%D = [0; 0];
%-----
```

```
%-----electrical ss--vinput-----
```

```

A = [-R/L -Kb/L; K/J -b/J];
B = [ 1/L; 0];

C = [1 0; 0 1];
D = [0;0];
%-----
sys_dc = ss(A,B,C,D);
Ts=0.0001;
sysd=c2d(sys_dc,Ts);
[Ad,Bd,Cd,Dd]=ssdata(sysd);

Kp=.995;
zc = 25.7;
Ki=zc*Kp;

%delay = [0.0001 0.001 0.005 0.01:.001:.02];
delay = [0.001];
delay_len = length(delay);

set_param('queue_comp3_sub10/controller','Kp',num2str(Kp));
set_param('queue_comp3_sub10/controller','Ki',num2str(Ki));
set_param('queue_comp3_sub10/zoh','Sample time',num2str(0.0001));
set_param('queue_comp3_sub10/zoh1','Sample time',num2str(0.0001));
set_param('queue_comp3_sub10/Integer Delay','NumDelays',num2str(delay/.0001));

disp('init 10 done')

```

A3.2.2 ***get_results1.m***

```

clear all;
close all;

w1 = 17.4572;
w2 = 0.0066771;
w3 = 0.8777;

delay_len = 6;

cost1_mse=zeros(delay_len,1); %cost for MSE
cost2_po=zeros(delay_len,1); %cost for Percentage overshoot
cost3_tr=zeros(delay_len,1); %cost for rise time (define as time to go from 10% to 90% of final value)

cost_values=zeros(3,6);

% Simulation time
final_t=2
max_step = 0.0001; %set fundamental sampling time
t=[0:0.0001:final_t];
N=length(t);

results=zeros(N,delay_len);
input_u = zeros(N,delay_len);

```

```

%predictor = zeros(N,delay_len);
% MSE nominal point
%J1_nom=0.00595492255222;
J1_nom=.003405;

% Percentage overshoot nominal point
J2_nom=5;

% Rise time nominal point
%J3_nom=0.117;
J3_nom=0.0140;

load queue_comp3_sub1.mat
delay_count = 1;
calculations

clear rt_Yout rt_Pout rt_tout rt_Uout

load queue_comp3_sub10.mat
delay_count = 2;
calculations

clear rt_Yout rt_Pout rt_tout rt_Uout

load queue_comp3_sub50.mat
delay_count = 3;
calculations

clear rt_Yout rt_Pout rt_tout rt_Uout

load queue_comp3_sub100.mat
delay_count = 4;
calculations

clear rt_Yout rt_Pout rt_tout rt_Uout

load queue_comp3_sub500.mat
delay_count = 5;
calculations

clear rt_Yout rt_Pout rt_tout rt_Uout

load queue_comp3_sub1000.mat
delay_count = 6;
calculations

clear rt_Yout rt_Pout rt_tout rt_Uout

costJ = w1*cost1_mse+w2*cost2_po+w3*cost3_tr
cost_values

%save extend_queue_runs cost1_mse cost2_po cost3_tr costJ results input_u

```


A3.2.3 *calculations.m*

```
Y=rt_Yout(:,2);
N=length(Y);
results(:,delay_count)=Y;
input_u(:,delay_count)=rt_Uout(:,1);
predictor(:,delay_count)=rt_Pout(:,2);
% Compute Costs
% Cost_1
% Mean-squared error
err=Y-1;
msr=(err'*err)/N;
if (msr>J1_nom)
    cost1_mse(delay_count,1)=(msr-J1_nom)^2;
else
    cost1_mse(delay_count,1)=0;
end

% Cost_2
% Percentage Overshoot
count=1;
flag_po=0;
po=0;
while (count<=N)
    if (Y(count)>1)
        po=(max(Y)-1)*100;
        break;
    end
    count=count+1;
end
if (po>J2_nom)
    cost2_po(delay_count,1)=(po-J2_nom)^2;
else
    cost2_po(delay_count,1)=0;
end

% Cost_3
% Rise time
count=1;
flag_st=0;
flag_ed=0;
start_time=0;
end_time=inf;
while (count<=N)
    if (Y(count)>=0.1)&(flag_st==0)
        start_time=rt_tout(count);
        flag_st=1;
    end

    if (Y(count)>=0.9)&(flag_ed==0)
        end_time=rt_tout(count);
        flag_ed=1;
        break
    end
end
```



```
% Copyright 1991-2004 MUSYN Inc. and The MathWorks, Inc.
% $Revision: 1.7.2.3 $
```

```
clear all
close all
```

```
b = 0.1e-3;
R = 6.43;
J = 3.53e-6;
K = 2.55e-3;
Kb = 0.255e-3;
L = 28.88e-3;
```

```
wf = 0.0001/2; %time delay forward
wb = 0.0001/2; %time delay backward
```

```
delay_c_p = [wf]'; %big delay
%delay_c_p = [0.02 0.060 0.01]'; %big delay_1, more stable
%delay_c_p = [0.012 0.0040 0.006]'; % used to generate K1-K4
%delay_c_p = [0 0 0]';
%[0.00010948 0.00025971 0.00033965]';
```

```
%[.30 .40 .50]';
```

```
delay_p_c = [wb]'; %big delay
%delay_p_c = [0.030 0.02 0.050]'; %big delay_1, more stable
%delay_p_c = [0.0030 0.0076 0.010]'; %used to generate K1-K4
%delay_p_c = [0 0 0]';
%[4.7523e-005 0.00018463 3.9042e-005]';
```

```
%[.20 .30 .40]';
```

```
delay_c_p_tune = 0;
delay_p_c_tune = 0;
```

```
max_step=0.0001;
```

```
%-----electrical ss--vinput-----
A = [-R/L -Kb/L; K/J -b/J];
B = [ 1/L; 0];
```

```
C = [0 1];
D = [0];
```

```
%-----
```

```
%-----torque ss--v input-----
%A = [0 1; 0 ((-b/J)-((K*Kb)/(J*R)))];
%B = [0; K/(J*R)];
%C = [1 0;0 1];
%D = [0;0];
```

```
%-----

%P    = pss2sys([0 10 1 0;-10 0 0 1;1 10 0 0;-10 1 0 0],2);
G = pss2sys([A B; C D],2);
```

```
W1 = nd2sys(25*[0 0 1],[1 2*10 1]);
%W1 = nd2sys([0 0 100],[1 50 100]);
%W1 = nd2sys([0 0 .9],[1 1 .9]);
W2 = nd2sys([wf 0], [wf/3.465 1]);
W3 = nd2sys([wb 0], [wb/3.465 1]);
%w_del1 = nd2sys([1 4],[1 200],10);
%w_del2 = nd2sys([1 24],[1 200],10/3);
%w_n    = nd2sys([1 25],[1 6000],12/5);
%W_n    = daug(w_n,w_n);
%w_p    = nd2sys([1 4],[1 0.02],1/2);
%W_p    = daug(w_p,w_p);

%systemnames = 'P w_del1 w_del2 W_p W_n';
systemnames = 'G W1 W2 W3';
%inputvar    = '[z1; z2; d{2}; nois{2}; u{2}]';
inputvar     = '[wf; wb; r; u]';
%outputvar   = '[u(1); u(2); P + W_p; P + W_p + W_n]';
outputvar    = '[W2; W3; W1; r - wb - G]';
%input_to_P  = '[u(1) + w_del1; u(2) + w_del2]';
input_to_G   = '[u + wf]';
%input_to_w_del1 = '[ z1 ]';
input_to_W1  = '[ r - wb - G ]';
input_to_W2  = '[ u ]';
input_to_W3  = '[ G ]';

cleanupysic  = 'yes';
sysoutname   = 'G_ss';
sysic
```

```
disp('G_ss updated');
%clear P w_del1 w_del2 w_n W_n w_p W_p
```

A4.2.2 *d1_after.m*

```
% Optimization from 0-2s
% Using forward approximation for Integral gain
% Using exponential delay variation from median delay
% Run with find_beta_sim.mdl
% modified from yodyium's by tyler 3/28/05

%dkitgui parameters = uncertainty = 2, 1x1, 1 meas. 1 control. 0.01-100
%,100 points

[ka,kb,kc,kd]=unpck(K1d1); %decompose controller you want from dkitgui

cost1_mse=zeros(8,1); %cost for MSE
```

```

cost2_po=zeros(8,1); %cost for Percentage overshoot
cost3_tr=zeros(8,1); %cost for rise time (define as time to go from 10% to 90% of final value)
costJ = zeros(8,1);
% Simulation time
final_t=2
max_step = 0.0001; %set fundamental sampling time
t=[0:0.0001:final_t];
N=length(t);

results_d1 = zeros(N,8);
input_u = zeros(N,8);

%set controller parameters
set_param('robust_delay1_K3/K','A','ka');
set_param('robust_delay1_K3/K','B','kb');
set_param('robust_delay1_K3/K','C','kc');
set_param('robust_delay1_K3/K','D','kd');

% MSE nominal point
%J1_nom=0.00595492255222;
J1_nom=.003405;

% Percentage overshoot nominal point
J2_nom=5;

% Rise time nominal point
%J3_nom=0.117;
J3_nom=0.0140;

w1 = 17.4572;
w2 = 0.0066771;
w3 = 0.8777;

cont_num = 1 % controller number
[T,X,Y]=sim('robust_delay1_K3',t);

Y=delay1_K3;
N=length(Y);
results_d1(:,cont_num) = delay1_K3;
input_u(:,cont_num) = control;
% Compute Costs
% Cost_1
% Mean-squared error
err=Y-1;
msr=(err*err)/N;
if (msr>J1_nom)
    cost1_mse(cont_num,1)=(msr-J1_nom)^2;
else
    cost1_mse(cont_num,1)=0;
end

% Cost_2
% Percentage Overshoot
count=1;

```

```

flag_po=0;
po=0;
while (count<=N)
    if (Y(count)>1)
        po=(max(Y)-1)*100;
        break;
    end
    count=count+1;
end
if (po>J2_nom)
    cost2_po(cont_num,1)=(po-J2_nom)^2;
else
    cost2_po(cont_num,1)=0;
end

% Cost_3
% Rise time
count=1;
flag_st=0;
flag_ed=0;
start_time=0;
end_time=inf;
while (count<=N)
    if (Y(count)>=0.1)&(flag_st==0)
        start_time=T(count);
        flag_st=1;
    end

    if (Y(count)>=0.9)&(flag_ed==0)
        end_time=T(count);
        flag_ed=1;
        break
    end
    count=count+1;
end
rise_time=end_time-start_time;
if (rise_time>J3_nom)
    cost3_tr(cont_num,1)=(rise_time-J3_nom)^2;
else
    cost3_tr(cont_num,1)=0;
end

costJ(cont_num,1) = w1*cost1_mse(cont_num,1)+w2*cost2_po(cont_num,1)+w3*cost3_tr(cont_num,1);
%end controller 1

% start controller 2 -----

cont_num = 2

[ka,kb,kc,kd]=unpck(K2d1); %decompose controller you want from dkitgui

%set controller parameters
set_param('robust_delay1_K3/K','A','ka');
set_param('robust_delay1_K3/K','B','kb');

```

```

set_param('robust_delay1_K3/K','C','kc');
set_param('robust_delay1_K3/K','D','kd');

[T,X,Y]=sim('robust_delay1_K3',t);

Y=delay1_K3;
N=length(Y);
results_d1(:,cont_num) = delay1_K3;
input_u(:,cont_num) = control;

% Compute Costs
% Cost_1
% Mean-squared error
err=Y-1;
msr=(err'*err)/N;
if (msr>J1_nom)
    cost1_mse(cont_num,1)=(msr-J1_nom)^2;
else
    cost1_mse(cont_num,1)=0;
end

% Cost_2
% Percentage Overshoot
count=1;
flag_po=0;
po=0;
while (count<=N)
    if (Y(count)>1)
        po=(max(Y)-1)*100;
        break;
    end
    count=count+1;
end
if (po>J2_nom)
    cost2_po(cont_num,1)=(po-J2_nom)^2;
else
    cost2_po(cont_num,1)=0;
end

% Cost_3
% Rise time
count=1;
flag_st=0;
flag_ed=0;
start_time=0;
end_time=inf;
while (count<=N)
    if (Y(count)>=0.1)&(flag_st==0)
        start_time=T(count);
        flag_st=1;
    end

    if (Y(count)>=0.9)&(flag_ed==0)
        end_time=T(count);
        flag_ed=1;
    end
end

```

```

        break
    end
    count=count+1;
end
rise_time=end_time-start_time;
if (rise_time>J3_nom)
    cost3_tr(cont_num,1)=(rise_time-J3_nom)^2;
else
    cost3_tr(cont_num,1)=0;
end

costJ(cont_num,1) = w1*cost1_mse(cont_num,1)+w2*cost2_po(cont_num,1)+w3*cost3_tr(cont_num,1);
%end controller 2

%start controller 3
cont_num = 3

[ka,kb,kc,kd]=unpck(K3d1); %decompose controller you want from dkitgui

%set controller parameters
set_param('robust_delay1_K3/K','A','ka');
set_param('robust_delay1_K3/K','B','kb');
set_param('robust_delay1_K3/K','C','kc');
set_param('robust_delay1_K3/K','D','kd');

[T,X,Y]=sim('robust_delay1_K3',t);

Y=delay1_K3;
N=length(Y);
results_d1(:,cont_num) = delay1_K3;
input_u(:,cont_num) = control;
% Compute Costs
% Cost_1
% Mean-squared error
err=Y-1;
msr=(err'*err)/N;
if (msr>J1_nom)
    cost1_mse(cont_num,1)=(msr-J1_nom)^2;
else
    cost1_mse(cont_num,1)=0;
end

% Cost_2
% Percentage Overshoot
count=1;
flag_po=0;
po=0;
while (count<=N)
    if (Y(count)>1)
        po=(max(Y)-1)*100;
        break;
    end
    count=count+1;
end
if (po>J2_nom)

```



```

        cost2_po(cont_num,1)=(po-J2_nom)^2;
    else
        cost2_po(cont_num,1)=0;
    end

    % Cost_3
    % Rise time
    count=1;
    flag_st=0;
    flag_ed=0;
    start_time=0;
    end_time=inf;
    while (count<=N)
        if (Y(count)>=0.1)&(flag_st==0)
            start_time=T(count);
            flag_st=1;
        end

        if (Y(count)>=0.9)&(flag_ed==0)
            end_time=T(count);
            flag_ed=1;
            break
        end
        count=count+1;
    end
    rise_time=end_time-start_time;
    if (rise_time>J3_nom)
        cost3_tr(cont_num,1)=(rise_time-J3_nom)^2;
    else
        cost3_tr(cont_num,1)=0;
    end

    costJ(cont_num,1) = w1*cost1_mse(cont_num,1)+w2*cost2_po(cont_num,1)+w3*cost3_tr(cont_num,1);
    %end controller 3

    %start controller 4
    cont_num = 4

    [ka,kb,kc,kd]=unpck(K4d1); %decompose controller you want from dkitgui

    %set controller parameters
    set_param('robust_delay1_K3/K','A','ka');
    set_param('robust_delay1_K3/K','B','kb');
    set_param('robust_delay1_K3/K','C','kc');
    set_param('robust_delay1_K3/K','D','kd');

    [T,X,Y]=sim('robust_delay1_K3',t);

    Y=delay1_K3;
    N=length(Y);
    results_d1(:,cont_num) = delay1_K3;
    input_u(:,cont_num) = control;
    % Compute Costs
    % Cost_1
    % Mean-squared error

```

```

err=Y-1;
msr=(err'*err)/N;
if (msr>J1_nom)
    cost1_mse(cont_num,1)=(msr-J1_nom)^2;
else
    cost1_mse(cont_num,1)=0;
end

% Cost_2
% Percentage Overshoot
count=1;
flag_po=0;
po=0;
while (count<=N)
    if (Y(count)>1)
        po=(max(Y)-1)*100;
        break;
    end
    count=count+1;
end
if (po>J2_nom)
    cost2_po(cont_num,1)=(po-J2_nom)^2;
else
    cost2_po(cont_num,1)=0;
end

% Cost_3
% Rise time
count=1;
flag_st=0;
flag_ed=0;
start_time=0;
end_time=inf;
while (count<=N)
    if (Y(count)>=0.1)&(flag_st==0)
        start_time=T(count);
        flag_st=1;
    end

    if (Y(count)>=0.9)&(flag_ed==0)
        end_time=T(count);
        flag_ed=1;
        break
    end
    count=count+1;
end
rise_time=end_time-start_time;
if (rise_time>J3_nom)
    cost3_tr(cont_num,1)=(rise_time-J3_nom)^2;
else
    cost3_tr(cont_num,1)=0;
end

costJ(cont_num,1) = w1*cost1_mse(cont_num,1)+w2*cost2_po(cont_num,1)+w3*cost3_tr(cont_num,1);
%end controller 4

```

```

%start controller 5

cont_num = 5

[ka,kb,kc,kd]=unpck(K5d1); %decompose controller you want from dkitgui

%set controller parameters
set_param('robust_delay1_K3/K','A','ka');
set_param('robust_delay1_K3/K','B','kb');
set_param('robust_delay1_K3/K','C','kc');
set_param('robust_delay1_K3/K','D','kd');

[T,X,Y]=sim('robust_delay1_K3',t);

Y=delay1_K3;
N=length(Y);
results_d1(:,cont_num) = delay1_K3;
input_u(:,cont_num) = control;
% Compute Costs
% Cost_1
% Mean-squared error
err=Y-1;
msr=(err'*err)/N;
if (msr>J1_nom)
    cost1_mse(cont_num,1)=(msr-J1_nom)^2;
else
    cost1_mse(cont_num,1)=0;
end

% Cost_2
% Percentage Overshoot
count=1;
flag_po=0;
po=0;
while (count<=N)
    if (Y(count)>1)
        po=(max(Y)-1)*100;
        break;
    end
    count=count+1;
end
if (po>J2_nom)
    cost2_po(cont_num,1)=(po-J2_nom)^2;
else
    cost2_po(cont_num,1)=0;
end

% Cost_3
% Rise time
count=1;
flag_st=0;
flag_ed=0;
start_time=0;
end_time=inf;

```

```

while (count<=N)
    if (Y(count)>=0.1)&(flag_st==0)
        start_time=T(count);
        flag_st=1;
    end

    if (Y(count)>=0.9)&(flag_ed==0)
        end_time=T(count);
        flag_ed=1;
        break
    end
    count=count+1;
end
rise_time=end_time-start_time;
if (rise_time>J3_nom)
    cost3_tr(cont_num,1)=(rise_time-J3_nom)^2;
else
    cost3_tr(cont_num,1)=0;
end

costJ(cont_num,1) = w1*cost1_mse(cont_num,1)+w2*cost2_po(cont_num,1)+w3*cost3_tr(cont_num,1);
%end controller 5

%start controller 6
cont_num = 6
[ka,kb,kc,kd]=unpck(K6d1); %decompose controller you want from dkitgui

%set controller parameters
set_param('robust_delay1_K3/K','A','ka');
set_param('robust_delay1_K3/K','B','kb');
set_param('robust_delay1_K3/K','C','kc');
set_param('robust_delay1_K3/K','D','kd');

[T,X,Y]=sim('robust_delay1_K3',t);

Y=delay1_K3;
N=length(Y);
results_d1(:,cont_num) = delay1_K3;
input_u(:,cont_num) = control;
% Compute Costs
% Cost_1
% Mean-squared error
err=Y-1;
msr=(err*err)/N;
if (msr>J1_nom)
    cost1_mse(cont_num,1)=(msr-J1_nom)^2;
else
    cost1_mse(cont_num,1)=0;
end

% Cost_2
% Percentage Overshoot
count=1;
flag_po=0;
po=0;

```

```

while (count<=N)
    if (Y(count)>1)
        po=(max(Y)-1)*100;
        break;
    end
    count=count+1;
end
if (po>J2_nom)
    cost2_po(cont_num,1)=(po-J2_nom)^2;
else
    cost2_po(cont_num,1)=0;
end

% Cost_3
% Rise time
count=1;
flag_st=0;
flag_ed=0;
start_time=0;
end_time=inf;
while (count<=N)
    if (Y(count)>=0.1)&(flag_st==0)
        start_time=T(count);
        flag_st=1;
    end

    if (Y(count)>=0.9)&(flag_ed==0)
        end_time=T(count);
        flag_ed=1;
        break
    end
    count=count+1;
end
rise_time=end_time-start_time;
if (rise_time>J3_nom)
    cost3_tr(cont_num,1)=(rise_time-J3_nom)^2;
else
    cost3_tr(cont_num,1)=0;
end

costJ(cont_num,1) = w1*cost1_mse(cont_num,1)+w2*cost2_po(cont_num,1)+w3*cost3_tr(cont_num,1);

%end controller 6

%start controller 7
cont_num = 7
[ka,kb,kc,kd]=unpck(K7d1); %decompose controller you want from dkitgui

%set controller parameters
set_param('robust_delay1_K3/K','A','ka');
set_param('robust_delay1_K3/K','B','kb');
set_param('robust_delay1_K3/K','C','kc');
set_param('robust_delay1_K3/K','D','kd');

[T,X,Y]=sim('robust_delay1_K3',t);

```

```

Y=delay1_K3;
N=length(Y);
results_d1(:,cont_num) = delay1_K3;
input_u(:,cont_num) = control;
% Compute Costs
% Cost_1
% Mean-squared error
err=Y-1;
msr=(err'*err)/N;
if (msr>J1_nom)
    cost1_mse(cont_num,1)=(msr-J1_nom)^2;
else
    cost1_mse(cont_num,1)=0;
end

% Cost_2
% Percentage Overshoot
count=1;
flag_po=0;
po=0;
while (count<=N)
    if (Y(count)>1)
        po=(max(Y)-1)*100;
        break;
    end
    count=count+1;
end
if (po>J2_nom)
    cost2_po(cont_num,1)=(po-J2_nom)^2;
else
    cost2_po(cont_num,1)=0;
end

% Cost_3
% Rise time
count=1;
flag_st=0;
flag_ed=0;
start_time=0;
end_time=inf;
while (count<=N)
    if (Y(count)>=0.1)&(flag_st==0)
        start_time=T(count);
        flag_st=1;
    end

    if (Y(count)>=0.9)&(flag_ed==0)
        end_time=T(count);
        flag_ed=1;
        break
    end
    count=count+1;
end
rise_time=end_time-start_time;

```

```

    if (rise_time>J3_nom)
        cost3_tr(cont_num,1)=(rise_time-J3_nom)^2;
    else
        cost3_tr(cont_num,1)=0;
    end

costJ(cont_num,1) = w1*cost1_mse(cont_num,1)+w2*cost2_po(cont_num,1)+w3*cost3_tr(cont_num,1);
%end controller 7

%start controller 8
cont_num = 8

[ka,kb,kc,kd]=unpck(K8d1); %decompose controller you want from dkitgui

%set controller parameters
set_param('robust_delay1_K3/K','A','ka');
set_param('robust_delay1_K3/K','B','kb');
set_param('robust_delay1_K3/K','C','kc');
set_param('robust_delay1_K3/K','D','kd');

[T,X,Y]=sim('robust_delay1_K3',t);

Y=delay1_K3;
N=length(Y);
results_d1(:,cont_num) = delay1_K3;
input_u(:,cont_num) = control;
% Compute Costs
% Cost_1
% Mean-squared error
err=Y-1;
msr=(err'*err)/N;
if (msr>J1_nom)
    cost1_mse(cont_num,1)=(msr-J1_nom)^2;
else
    cost1_mse(cont_num,1)=0;
end

% Cost_2
% Percentage Overshoot
count=1;
flag_po=0;
po=0;
while (count<=N)
    if (Y(count)>1)
        po=(max(Y)-1)*100;
        break;
    end
    count=count+1;
end
if (po>J2_nom)
    cost2_po(cont_num,1)=(po-J2_nom)^2;
else
    cost2_po(cont_num,1)=0;
end
end

```

```

% Cost_3
% Rise time
count=1;
flag_st=0;
flag_ed=0;
start_time=0;
end_time=inf;
while (count<=N)
    if (Y(count)>=0.1)&(flag_st==0)
        start_time=T(count);
        flag_st=1;
    end

    if (Y(count)>=0.9)&(flag_ed==0)
        end_time=T(count);
        flag_ed=1;
        break
    end
    count=count+1;
end
rise_time=end_time-start_time;
if (rise_time>J3_nom)
    cost3_tr(cont_num,1)=(rise_time-J3_nom)^2;
else
    cost3_tr(cont_num,1)=0;
end

costJ(cont_num,1) = w1*cost1_mse(cont_num,1)+w2*cost2_po(cont_num,1)+w3*cost3_tr(cont_num,1);
% end controller 8

costJ_d1 = costJ
save d1_results1 costJ_d1 results_d1 mu_d1 K1d1 K2d1 K3d1 K4d1 K5d1 K6d1 K7d1 K8d1 input_u
results = results_d1;

plot_figs

```

A4.2.3 *find_cost.m*

```

load d1_results
load d2_results
load d3_results
load d4_results
load d5_results
load d6_results

costJ=zeros(6,2); % min cost, controller number

costJ(1,1) = min(costJ_d1);
costJ(1,2) = find(costJ_d1 == min(costJ_d1));

costJ(2,1) = min(costJ_d2);
costJ(2,2) = find(costJ_d2 == min(costJ_d2));

```



```
costJ(3,1) = min(costJ_d3);
costJ(3,2) = find(costJ_d3 == min(costJ_d3));
```

```
costJ(4,1) = min(costJ_d4);
costJ(4,2) = find(costJ_d4 == min(costJ_d4));
```

```
costJ(5,1) = min(costJ_d5);
costJ(5,2) = find(costJ_d5 == min(costJ_d5));
```

```
costJ(6,1) = min(costJ_d6);
costJ(6,2) = find(costJ_d6 == min(costJ_d6));
```

```
costJ
```

```
save cost_controller costJ
```

A4.3. DKitgui

A4.3.1 Setup

muTools(1): DK Iteration Setup - DC motor

Window

Open-Loop IC: S: y4, u4, x6

<Controller>:

SIGNAL DIMENSIONS

Block Structure

Uncertainty Structure

of Blocks:

#	Row	Col	Fac
1	<input type="text" value="1"/>	<input type="text" value="1"/>	<input type="text" value="1"/>
2	<input type="text" value="1"/>	<input type="text" value="1"/>	<input type="text" value="1"/>

Performance Structure

of Errors:

of Disturbances:

Feedback Structure

of Measurements:

of Controls:

Uncertainty:

Performance:

Feedback:

Omega:

Frequency Range

☒ Logspace ☐ Custom

Low: High: # Points:

Identifiers

<Iteration Suffix>:

<Iteration Name>:

Figure 31. dkitgui setup screen.

A4.3.2 Sample results

DK Iteration Summary				
Iteration #	1	2	3	4
Total D Order	0	20	20	20
Controller Order	6	26	26	26
Gamma Achieved	0.993	0.243	0.221	0.253
Peak Mu Value	0.358	0.228	0.204	0.238
<div><<< >>></div>				

Figure 32. Sample DK iteration summary. Iteration 3 gave the best results shown by lowest Peak Mu Value.