Title:	Generalized Predictive and Neural Generalized Predictive Control of Aerospace Systems
Principal Investigator:	Dr. Atul G. Kelkar Mechanical and Nuclear Engineering Department Kansas State University Durland 341 Manhattan, KS 66506
Cooperative Grant No.:	NCC-1225
Technical Monitor:	Ms. Pamela J. Haley M/S 406, Dynamics and Control Branch NASA Langley Research Center Hampton, VA 23681-0001
Total Project Duration:	05/16/96 -05/15/00
Report Duration:	05/16/96-05/15/00

Summary

This report provides a comprehensive summary of the research work performed over the entire duration of the co-operative research agreement NCC-1-225 between NASA Langley Research Center and Kansas State University. The cooperative agreement which was originally for the duration the three years was extended by another year through no-cost extension in order to accomplish the goals of the project.

A detailed final report is enclosed in the form of a thesis. This thesis summarizes the findings and also suggests possible future directions for the continuation of the research in the area of GPC and NGPC. The enclosed thesis is the outcome of the research conducted under cooperative agreement NCC-1225.

ABSTRACT

The research work presented in this thesis addresses the problem of robust control of uncertain linear and nonlinear systems using Neural network-based Generalized Predictive Control (NGPC) methodology. A brief overview of predictive control and its comparison with Linear Quadratic (LQ) control is given to emphasize advantages and drawbacks of predictive control methods. It is shown that the Generalized Predictive Control (GPC) methodology overcomes the drawbacks associated with traditional LQ control as well as conventional predictive control methods. It is shown that in spite of the model-based nature of GPC it has good robustness properties being special case of receding horizon control. The conditions for choosing tuning parameters for GPC to ensure closed-loop stability are derived. A neural network-based GPC architecture is proposed for the control of linear and nonlinear uncertain systems. A methodology to account for parametric uncertainty in the system is proposed using on-line training capability of multi-layer neural network. Several simulation examples and results from real-time experiments are given to demonstrate the effectiveness of the proposed methodology.

TABLE OF CONTENTS

LIST OF FIGURES iv

CHAPTER 1 Introduction 1

1.1 Preface 1

1.2 Organization of thesis 2

CHAPTER 2 Generalized Predictive Control 4

2.1 Predictive controllers revisited 4

2.2 Receding horizon principle 8

2.3 Predictive control versus linear quadratic control 13

2.4 Generalized Predictive Control 16

2.5 Modeling of the plant and disturbances 21

2.5.1 System modeling 21

2.5.2 Disturbances modeling 23

2.6 Derivation of Generalized Predictive Control law 29

CHAPTER 3 Neural Networks in Control Applications 36

3.1 Neural network overview 36

3.2 Benefits of neural networks 38

3.3 Model of a neuron 41

3.4 Model of a layer 44

3.5 The network architecture 46

3.6 Neural network as input/output estimators 48

3.6.1 Introduction to I/O estimator using multilayer neural networks 48

3.6.2 Neural network as function estimator 50

3.6.3 Back-propagation algorithm 50
3.6.4 Back-propagation training 53
3.6.5 Neural networks as dynamic plant estimator 56
3.6.6 Tapped-time delay network architecture for modeling dynamic plant 58
3.6.7 Training procedure for neural network 60
3.6.8 Neural network as predictor 63
CHAPTER 4 Neural Generalized Predictive Control (NGPC) 66
4.1 Stability of GPC control law 66
4.2 Neural Generalized Predictive Control (NGPC) 76
4.2.1 Introduction 76
4.2.2 Basic NGPC algorithm 78
4.2.3 NGPC cost function with actuator constraints 79
4.3 Cost Function Minimization (CFM) 80
4.3.1 Brief review of Newton-Raphson algorithm 81
4.3.2 CFM algorithm 82
4.3.3 Neural network architecture for system modeling 86
CHAPTER 5 Simulation and real time control experiment 89
5.1 Introduction 89
5.2 Pitch axis control of a fighter aircraft 89
5.3 Control of slewing link with flexible joint 99
5.4 Mathematical modeling of a rotary flexible joint system 102
5.4.1 Derivation of the joint stiffness 102
5.4.2 Dynamic analysis 103

5.5 Experimental set-up and definition of control problems 107

5.6 Real-time experiment results 109

5.7 Observations 123

5.8 Nonlinear mass-spring-damper system 126

CHAPTER 6 Conclusions 132

Reference: 135

LIST OF FIGURES

Fig 2.1 Schematic of a model based control system 4

Fig 2.2 Receding horizon predictive control 9

Fig 2.3 GPC block diagram 20

Fig 3.1 Model of a neuron 42

Fig 3.2 Sigmoid function with different ' a ' 44

Fig 3.3 Layer of a neural network 45

Fig 3.4 Multilayer feedforward neural network 47

Fig 3.5 Network architecture for modeling a dynamic system 58

Fig 3.6 Block diagram representation of a time delay network 60

Fig 3.7 Discrete plant 61

Fig 3.8 Block diagram of a neural network wired to learn the dynamics of a plant 63

Fig 3.9 Network prediction for k=2 65

Fig 4.1 Block diagram of NGPC 78

Fig 4.2 Neural network with embedded linear model 87

Fig 5.1 NGPC online training block diagram 91

Fig 5.2 Case a): Embedded model identical to plant 93

Fig 5.3 Case b): +30% uncertainty in ω_n of the plant, no on-line adaptation 94

Fig 5.4 Case c): +30% uncertainty in ω_n of the plant, on-line training is on 95

Fig 5.5 Case d): -30% uncertainty in ω_n of the plant, no on-line Adaptation 96

Fig 5.6 Case e): -30% uncertainty in ω_n of plant, on-line training is on 96

Fig 5.7 Steady-state error performance: $\Delta\omega_n = 0\%$, no adaptation 97

Fig 5.8 Steady-state error performance: $\Delta\omega_n = +30\%$, no adaptation 97

Fig 5.9 Steady-state error performance: $\Delta\omega_n = +30\%$, online adaptation 98

Fig 5.10 Steady-state error performance: $\Delta\omega_n = -30\%$, no adaptation 98

Fig 5.11 Steady-state error performance: $\Delta\omega_n = -30\%$, online adaptation 99

Picture 5.1 Slewing link with flexible joint 100

Fig 5.12 Rotary flexible joint 101

Fig 5.13 System geometry 101

Fig 5.14 Poles-zeros map & open-loop step response 106

Fig 5.15 Real-time step response 107

Fig 5.16 Step response: exact embedded model 111

Fig 5.17 Step response: +104% uncertainty, without online adaptation 112

Fig 5.18 Step response: +104% uncertainty, with online adaptation 113

Fig 5.19 Tracking performance: exact embedded model 114

Fig 5.20 Tracking performance: +69% uncertainty, without online adaptation 115

Fig 5.21 Tracking performance: +69% uncertainty, with online adaptation 116

Fig 5.22 Tracking performance: +25% uncertainty, without online adaptation 117

Fig 5.23 Tracking performance: +25% uncertainty, with online adaptation 118

Fig 5.24 Tracking performance: +104% uncertainty, without online adaptation 119

Fig 5.25 Tracking performance: +104% uncertainty, with online adaptation 120

Fig 5.26 Tracking performance with different NGPC parameter 122

Fig 5.27 System response: +104% uncertainty, with online adaptation 124

Fig 5.28 System response: +104% uncertainty, with online adaptation 125

Fig 5.29 No pre-training, no online adaptation 128

Fig 5.30 Using pre-training, no online adaptation 129

Fig 5.31 No pre-training, but using online adaptation 130

Fig 5.32 Using pre-training and online adaptation..... 131

CHAPTER 1 Introduction

1.1 Preface

One way to classify controllers is model-based and model-independent controllers. Model-based controllers rely on the explicit plant model information to compute the control signal whereas model-independent controllers do not depend on plant model to compute the control signal. Model-independent controllers can provide inherent robustness to modeling errors and parametric perturbations as they do not depend on the plant model. Their performance, however, can get greatly affected by such perturbations. Model-based controllers, on the other hand, are sensitive to modeling as well as parametric uncertainties; however, they can deliver better performance with proper tuning. Predictive controllers belong to a class of model-based controllers and have been proved to be very effective for control of dynamic systems. Traditionally, in past, model-based controllers have suffered from lack of proof of stability and performance robustness. Their recent popularity in control community, however, is due to significant advances in their theoretical and experimental research on stability and performance robustness. The work presented in this thesis focuses on the use of Generalized Predictive Control (GPC) methods, which use neural network-based plant model for real-time control of dynamic systems. GPC belongs to a special class of predictive control method, namely, Receding Horizon Control (RHC). GPC overcomes the drawbacks of traditional model-based controllers by providing guaranteed closed-loop stability with acceptable level of performance. The research work reported herein exploits the nonlinear mapping and on-line learning capabilities of neural network and

stability properties of GPC to achieve effective real-time control of unstable and non-minimum phase dynamic systems. Since theoretical framework of GPC parallels to a large extent to Linear Quadratic (LQ) framework, a comparative assessment is presented between RHC and LQ methodologies wherever appropriate.

1.2 Organization of thesis

The organization of this thesis is as follows:

In the second chapter, basic framework of GPC is presented along with a brief introduction of GPC and some remarks on their comparison with LQ controllers. It is shown that GPC controllers belong to a class of model-based controllers, namely, Receding Horizon Control (RHC) and that GPC is essentially a dynamic version of RHC. Since RHC paradigm parallels very close to an LQ paradigm, a comparison between LQ and RHC controllers is given to point out the differences between the two. It is shown that RHC has numerous advantages over its LQ counterparts. It is also shown that the disadvantages arising owing to model-based control strategies are minimized by the use of receding horizon control. GPC, being special case of RHC, inherits the basic characteristics of RHC and offers additional benefits due to it own architecture. The basic derivation of GPC control law for LTI system with known plant model is also given.

The third chapter gives the review of neural networks and presents some of their potential applications in the control system design. In particular, it is shown that how neural networks can be used as estimators or predictors in the control system applications. In GPC framework, neural networks find their use for system modeling. It is shown that the tapped-delay architecture of the neural networks allows them to model

dynamic systems. A procedure to train neural network as dynamic plant estimator is also given.

In Chapter 4, a methodology to integrate neural network into GPC framework to yield Neural network-based GPC (NGPC) architecture is given. The controller block diagram for NGPC architecture is presented. The cost function minimization algorithm used in NGPC framework is described in detail. The methodology to use neural network for modeling and prediction in the control of linear and nonlinear uncertain systems is given.

Chapter 5 focuses on numerical simulations and real-time experiments. The results presented in this chapter validate NGPC algorithm and analytical development given in the preceding chapters. Numerous simulation and experimental results are presented in this chapter, which support the proposed methodology for the control of uncertain systems.

Finally, Chapter 6 gives some concluding remarks and lists possible directions for future research.

CHAPTER 2 Generalized Predictive Control

This chapter will review background material pertinent to the research work presented in this thesis and lay theoretical foundation for ensuing chapters. The basics of predictive control are first revisited and their strengths and weaknesses are reviewed in comparison with traditional LQ control methods. The basic principle RHC is presented and the concept of GPC is introduced. Finally, modeling of plant and disturbances is discussed followed by the derivation of GPC control law.

2.1 Predictive controllers revisited

Predictive controllers belong to a class of model-based controllers. A typical schematic of model-based controllers is shown in Fig 2.1. As the name suggests, predictive controllers are based on the control methodology, which uses prediction of the future outputs of the system to compute the control signal. The future outputs are predicted using the best possible model available to the designer and a known sequence of control signals. The error between the desired output trajectory and the predicted output is used to compute the desirable control signal. All predictive controllers are essentially based on this basic idea.

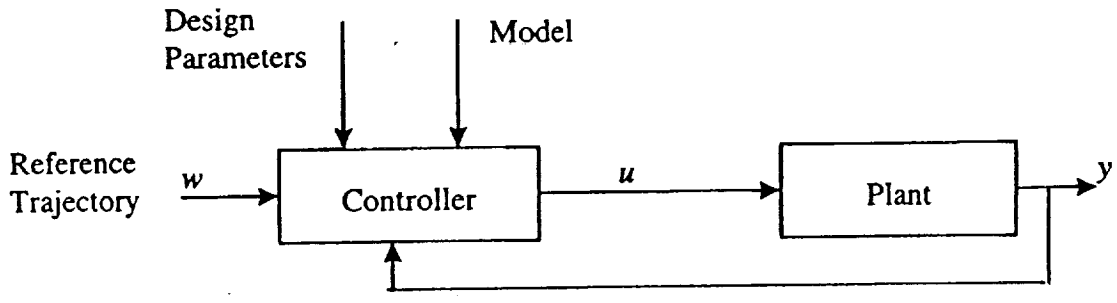


Fig 2.1 Schematic of a model based control system

In Fig 2.1, u denotes the controller output, y denotes the plant output and w denotes the reference trajectory or the desired plant output. In the case of model-based controllers, such as pole-placement controllers or linear-quadratic (LQ) controllers, there is a two-step process in the calculation of the control input u. For example, in LQ control the first step involves obtaining controller parameters using the known model of the plant and solving an optimization problem, and in the second step the controller parameters are used to obtain a state feedback type control law.

In general, if the system is linear, there are no constraints on the input/output, and the desired system output (or reference trajectory) is simple, then all of the above-mentioned model-based controllers yield approximately the same results. That is, one method is no 'better' than another. This is due to the fact that these methods yield linear controllers that after some manipulations have similar structure and have sufficient number of degrees of freedom. The underlying methodology to determine the controller parameters is, however, different. The difference between the methods is in the design parameters that are used to obtain the desired behavior of the control system. For example, in LQ control, the weighting matrices in the performance function are the design parameters used to obtain the desired behavior of the system. On the other hand, in the pole-placement method the closed-loop pole locations are considered as the design parameters. The design specifications of control system, however, are seldom defined in terms of the closed-loop pole locations or weighting matrices. As a result, it is necessary to transform the design specifications into the requirements on the design parameters. For LQ and pole-placement controllers such a transformation is usually difficult to obtain because the relation between the design parameters and the design specifications is, in

general, highly nonlinear. Consequently, theoretically possible results may not be obtained in practice. This, of course, limits the practical use of such methods. Therefore, one can say that the results obtained in practice are not only determined by what is theoretically possible but also by the simplicity of the transformation of design specifications into the requirements on the design parameters.

The predictive control concept was introduced simultaneously by Richalet [1] and Cutler and Ramaker [2] in the late seventies. One of the attractive features of predictive controllers is that they are relatively easy to tune. Other noteworthy features of predictive controllers that make them more attractive in many applications are summarized below:

1. The application of predictive control is not restricted to single-input, single-output (SISO) systems. Predictive controllers can be used for and applied to multi-input, multi-output (MIMO) systems. The structure of the predictive controllers makes it easy to extend their application from SISO systems to MIMO systems (See [3]).
2. In contrast to LQ and pole-placement controllers, predictive controllers can also be derived for nonlinear systems. The only difference being the nonlinear system model is then used explicitly to design the controller (See [4], [5]).
3. Predictive control is the only methodology that can handle system constraints in a systematic way during the design of the controller. Since, in real life, systems have constraints, this feature is rather important and is believed to be one of the most attractive aspects of predictive controllers.
4. Predictive control methodology is an open architecture. That is, within the framework of predictive control there are many ways to design a predictive controller. As a

result, over ten different type of predictive controllers, each with different properties, have been proposed in the literature over the last decade.

5. The predictive control methodology can be used to control a wide variety of systems. It can be used to control 'simple' as well as 'hard to control' systems; for example, systems with large time delays, systems with non-minimum phase zeros, and systems that are open loop unstable.
6. The architecture of predictive control allows feed-forward compensation of measurable disturbances and/or reference trajectories in a natural way.

Predictive controllers also have some drawbacks. Being model-based controllers, predictive controllers require a model of the system. In general, the controller design process involves two steps: system modeling and control law design. Predictive control provides the solution only for the controller design part. The model of the process must be obtained by some other methods. Another disadvantage arises due to the open architecture of the predictive control concept. As a result of such an open architecture, many different types of predictive controllers can be obtained each having different properties. Although, at first glance, the differences between these controllers seem rather small, these 'small' differences can yield very different behavior. As a result, the selection of the type of predictive controller that should be used to solve a particular problem becomes a difficult task. Therefore, a unified approach to predictive controller design is needed which allows treatment of each problem within the same framework and results in a significant reduction in the design costs.

Given below are different types of predictive controllers resulting from minor variations in the basic predictive control algorithm. The differences lie in the design

parameters used, computational techniques used, implementation aspects, and performance function used:

1. GPC (Generalized Predictive Control) [6], [7]
2. DMC (Dynamic Matrix Control) [8]
3. EPSAC (Extended Prediction Self-Adaptive Control) [9]
4. PFC (Predictive Functional Control) [10]
5. EHAC (Extended Horizon Adaptive Control) [11]
6. UPC (Unified Predictive Control) [12], [13]

2.2 Receding horizon principle

Most of the predictive controllers listed above are based on the principle of receding horizon. This section will describe control methodology based on this principle and cite notable differences with LQ methodology. As stated before, receding horizon principle forms basis for GPC methodology which is the main control technique used in this research. In order to illustrate receding horizon principle, a discrete-time formulation is used since it naturally lends itself for digital implementation on real life hardware. However, it is to be noted that it is possible to design predictive controllers in continuous time also [14].

Consider a SISO system with input u and output y; Fig 2.2 shows time histories of u and y in a typical predictive control scenario. The time scales in part (a), (b), (c), and (d) are time scales relative to the sample k, which denotes the present. The time scales shown at the bottom of Fig 2.2 are absolute time scales. First, consider figures 2.2(b) and 2.2(d) and suppose that the current time is denoted by sample k which corresponds to the

absolute time t. Further $u(k), y(k)$ and $w(k)$ denote the controller output, the system output, and the desired system output at sample k, respectively.

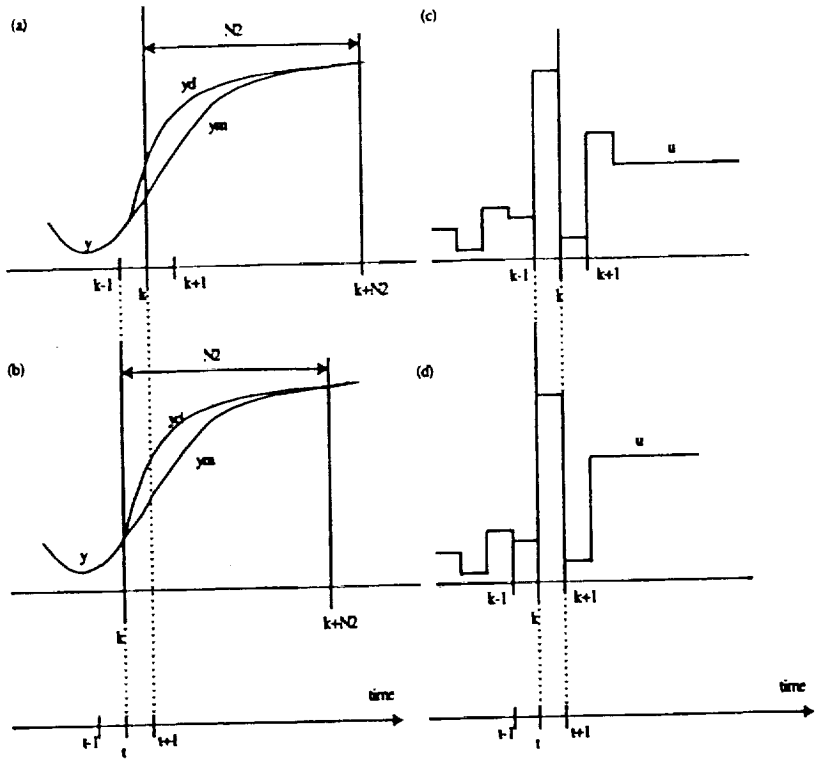


Fig 2.2 Receding horizon predictive control

Now, define:

$$\mathbf{u} = [u(k), \cdots, u(k+N_2-1)]^T$$
$$\hat{\mathbf{y}} = [\hat{y}(k+1), \cdots, \hat{y}(k+N_2)]^T$$
$$\mathbf{y_d} = [y_d(k+1), \cdots, y_d(k+H_p)]^T$$

where N_2 is the prediction horizon and the symbol ^ denotes estimation. $\mathbf{u}$ is the controller output, $\hat{\mathbf{y}}$ is the predicted system output, and $\mathbf{y_d}$ is the desired system output.

Predictive controller calculates a future controller output sequence $\mathbf{u}$ such that $\hat{\mathbf{y}}$ is 'close' to the desired system output $\mathbf{y}_d$. This desired system output is often called the reference trajectory and it can be an arbitrary sequence of points. However, in general, the response of a first or second order system model is used as the reference trajectory.

Now, suppose that instead of using the entire controller output sequence so determined to control the system in the next N_2 samples, only the first element of this controller output sequence (i.e., $u(k)$) is used and at the next sample, the whole procedure is repeated using the latest measured information. This method is called Receding Horizon Control (RHC). Assuming that there are no disturbances and no modeling errors the predicted system output $\hat{y}(k+1)$ predicted at the time t is exactly equal to the system output $y(k)$ measured at time $t+1$. Now, again, a future controller output sequence is calculated such that the predicted system output is 'close' to the reference trajectory. In general, this controller output sequence is different from the one obtained at the previous sample, as is illustrated by Fig 2.2(c). The reason for using the receding horizon approach is that this allows us to compensate for future disturbances or modeling errors. For example, due to the disturbances or modeling errors the predicted system output $\hat{y}(k+1)$ predicted at time t is not equal to the system output $y(k)$ measured at $(t+1)$. Then, it is intuitively clear that at time $(t+1)$ it is better to start the predictions from the measured system output rather than from the system output predicted at time previous sample. The predicted system output is now corrected for disturbances and modeling errors. A feedback mechanism is activated. As a result of the receding horizon approach the horizon over which the system output is predicted shifts one sample into the future at every sample instant.

An outline of a complete process used to determine the control signal is as follows. Determination of a future controller output sequence is based on the minimization of a meaningful performance index such as the one given below with respect to control input signal u :

$$J = \sum_{i=1}^{N_2} (\hat{y}(k+i) - y_d(k+i))^2$$

An optimal control sequence $u^*(k)$ is given by:

$$u^*(k) = \arg \quad \min J_{u(k)}$$

Predicted output, $\hat{y}$, used in J is based on the plant model information and the future control sequence available at current time. It is to be noted that the form of J used is different for different types of predictive controllers. As is evident, obtaining $u^*(k)$ is an optimization problem and the existence of global minimization as well as closed-form solution for $u^*(k)$ depends on the form of J. When J has a quadratic form, a nice analytical solution is possible in the case of linear systems with no constraints. For nonlinear systems and/or nonlinear performance function J, numerical optimization is necessary. For convenience, usually quadratic performance function that best reflects the objectives is used.

Note that when u is obtained by such minimization controllers outputs do not have any structure as in the case of LQ controllers where u is assumed to be of the form: $u(k) = -G(k)x(k)$, where $G(k)$ is the vector of controller parameters. In predictive control, such a priori assumption about the structure of the controller is not made and hence it is possible to account for system constraints in a more systematic way. For

example, in the case of constraints on the inputs (like saturation), an optimal control $u^*(k)$ is given by the solution of the following problem:

$$u^*(k) = \arg \quad \min J_{u(k)}$$

subject to:

$$u_{\min} \le u(k+i-1) \le u_{\max} \qquad 1 \le i \le N_2$$

In this case, however, analytical solution is not available and iterative numerical optimization is used.

Conceptually, RHC is a simple method to synthesize a feedback control law for linear and nonlinear systems. Although the method, if desired, can be used to synthesize approximately the state-space LQ feedback with a guaranteed stabilizing property, it has extra feature that makes it particularly attractive in the GPC setting. Since RHC strategy involves a horizon made up of only a finite number of time steps, the RHC input can be sequentially calculated online by existing optimization routines so as to minimize a performance index and satisfy hard constraints; for example, bounds on the time evolution of the input and the state. RHC is most suitable for slow linear and nonlinear systems, where it is possible to solve constrained optimization control problems on line. Receding horizon control was first proposed to relax the computational shortcomings of steady-state linear quadratic (LQ) control. In RHC, the current controller output $u(k)$ and the state of the system is obtained by determining over an N_2 step horizon, the controller output sequence $u_{[k,k+N_2]}$ which is optimal in a constrained LQ sense, and the whole optimization procedure is repeated for next sampling instant. That is, at every sampling instant, the plant is fed by the first element of the controller output sequence and the

subsequent $N_2 - 1$ elements are discarded. Next section points out main differences between LQ and predictive control methodologies.

2.3 Predictive control versus linear quadratic control

As it has been shown previously, predictive controllers, LQ controllers, and pole-placement controllers belong to a class of model-based controllers. Moreover, since LQ controllers and predictive controllers share a similar framework which involves minimization of a criterion function to compute the control signal, one can expect that the LQ control be very closely related to the predictive control.

Because predictive controllers are in general discrete-time controllers, only discrete LQ controllers are considered. Further, only the SISO case is considered for simplicity. In order to show the similarities and differences between predictive control and LQ control, a brief discussion of discrete-time LQ control is in order. A state-space approach is considered for this purpose.

In discrete time LQ control, the process is given by:

$$\mathbf{x}(k+1) = \mathbf{A}\mathbf{x}(k) + bu(k)$$
$$y(k) = c^T\mathbf{x}(k)$$

where $\mathbf{x}(k)$ denotes the state of the process and $\mathbf{A}$, b and c are the process parameters. In order to calculate the controller output the following criterion function is minimized:

$$J = \sum_{k=0}^{N} \mathbf{x}^T(k)\mathbf{Q}\mathbf{x}(k) + Ru^2(k)$$

where N is the terminal sampling instant, $\mathbf{Q}$ is a weighting matrix and R is a weighting factor. Hence, similar to predictive controller design, a criterion function is minimized over a horizon. However, in contrast to predictive controllers the receding horizon

approach is not employed. This is a fundamental difference. The criterion function is minimized only once (at k=0), resulting in an optimal controller output sequence from which the controller output used to control the process is taken at every sample. Hence, future disturbances and modeling errors can not be taken into account. However, it is shown in, for example, [15] that the controller output sequence determined in this way can also be generated by the following linear state feedback:

$$u(k) = -\mathbf{k}^T(k)\mathbf{x}(k)$$

The feedback vector $\mathbf{k}$ is time varying despite the fact that the process is time invariant. However as k increases $\mathbf{k}$ becomes constant. This is one of the reasons why in practice the solution for $N \to \infty$ is used. A time-invariant state feedback controller is then obtained which, because of the feedback, attenuates the effect of disturbances and modeling errors on the behavior of the control system. Disturbance can also be taken into account explicitly by introducing disturbances on the states and on the output of the system. Then, a stochastic optimization problem must be solved. This problem is usually solved by applying the separation theorem which states that the stochastic optimization problem can be separated into two parts: find a state estimator which gives the best estimates of the states from the observed outputs, and find the optimal state feedback law which is now a feedback from the estimated states. The latter problem is again a LQ problem (a deterministic optimization problem) and is solved by minimization of the LQ criterion function in which the real states are replaced by their estimates.

Another advantage of using $N \to \infty$ is that the nominal closed-loop system (the closed-loop system assuming that the model is identical to the process) is guaranteed to

be stable. The feedback vector $\mathbf{k}$ for $N \to \infty$ can be obtained by solving the algebraic Riccati equation (ARE).

For finite N, the optimal controller output sequence can be found by using dynamic programming. First, the optimal controller output $u(N)$ is determined, then $u(N-1)$, and so on. It now follows from the Bellman's principle [16] that once the controller output sequence $u(0),\cdots,u(N)$ minimizing:

$$J = \sum_{k=0}^{N} \mathbf{x}^T(k)\mathbf{Q}\mathbf{x}(k) + Ru^2(k)$$

has been found, minimization of

$$J = \sum_{k=j}^{N} \mathbf{x}^T(k)\mathbf{Q}\mathbf{x}(k) + Ru^2(k)$$

under the assumption that $u(0),\cdots u(j-1)$ have been supplied to the process and there are neither disturbances nor modeling errors, yields a controller output $u(j),\cdots,u(N)$ which is equal to that obtained when minimizing the LQ criterion function. Knowing this, finite horizon LQ control can also be obtained by using a receding horizon framework where the prediction horizon is decreased by one at every sampling instant. Similar to predictive controllers, an optimal controller output sequence is calculated over the prediction horizon and only the first controller output in this sequence is used to control the process. At the next sampling instant, the situation changes. Again, an optimal controller output sequence is calculated but now over an interval with length $N-1$ keeping the end of the interval constant (namely at time $t+N$). Assuming that there are no disturbances and no modeling errors, it follows from Bellman's optimality principle that the optimal controller output sequence found at t+1 is identical to the one found at time t. When the

prediction horizon goes to infinity this difference between LQ and predictive control vanishes, as shown in [17].

In conclusion it can be stated that for a finite prediction horizon the fundamental difference between predictive control and finite horizon LQ control is the receding horizon approach with a fixed length prediction horizon employed by predictive controller in contrast to a decreasing length prediction horizon employed by LQ controllers.

An important disadvantage of LQ control is that, in general, it is quite difficult to translate the design specification into weighting matrices in the criterion function because the states of a discrete-time process are usually artificial and hence they are not directly related to the true states of the process. Rule of thumb methods on how to choose the criterion parameters are not readily available. An important advantage of infinite horizon LQ controllers is that, under some quite general conditions, the closed-loop system is guaranteed to be stable. Usually this claim can not be made for finite horizon predictive controllers.

2.4 Generalized Predictive Control

Generalized Predictive Control (GPC) methodology belongs to a class of receding horizon-based predictive control techniques. GPC has been analyzed and implemented successfully in various process control industries since the end of the 1970's. GPC can systematically take into account the real plant constraints in real-time. One main feature of GPC is that, it can control non-minimum phase plants, open-loop unstable plants and plants with variable or unknown dead time. GPC is a special case of RHC. In particular,

GPC is a type of RHC in which compensator is dynamic and has certain stabilizing properties inherited from RHC.

The basic structure of predictive control methods consists of a predictor model and an optimization algorithm that minimizes a particular cost function. The choice of different prediction model and optimization algorithm leads to different predictive control techniques. Initially, the prediction models were simply generated from step response values at the sampling instants. For GPC, a model that takes into account the effect of noise is used. This model is called Controlled Auto-Regressive Integrated Moving-Average (CARIMA) model which in standard terminology would be called as Auto-Regressive Integrated Moving Average eXogenous input (ARIMAX) model. This input/output model is given by:

$$A(q^{-1})y(t) = B(q^{-1})u(t) + C(q^{-1})e(t) \tag{2.1}$$

where q^{-1} denotes the back shift operator, so $A(q^{-1})$, $B(q^{-1})$ and $C(q^{-1})$ are polynomials in q^{-1}. The $A(q^{-1})$ and $B(q^{-1})$ polynomials define the plants dynamics, which are the poles and zeros of the plants respectively. Typically, in linear system, the b_0 coefficient in the $B(q^{-1})$ polynomial is set to be zero. In the model used here, the b_0 term will be learned. The $C(q^{-1})$ polynomial defines the dynamics of the sensor noise.

GPC is a predictive control technique that uses a long-range prediction cost function. At each sampling instant GPC uses predicted values from the predictor model to minimize a cost function that takes into account predicted tracking errors and control signals. Part of the success of these techniques is due to the fact that instead of choosing

GPC predicts the performance of the plant. There are several rules of thumb to select N_2. Clarke shows that the rising time of the plant model will be suggestive [6].

Since the cost function is quadratic, analytic solutions of the minimization are possible using one of the prediction models. The analytic solution leads to a standard implementation of a feedback digital controller. In practice, an advantage of Model Based Predictive Control (MBPC) techniques is that they can also handle implementation constraints such as limits on actuator amplitudes. In this case, the cost function is minimized on-line. The slow time constants of process control have made it possible for a variety of numerical techniques to be used in real-time control.

An on-line GPC algorithm could be implemented as follows:

1. $[ym(n+N_1) \quad ym(n+N_1+1) \quad \cdots \quad ym(n+N_2)]^T$ reference trajectory is generated. If the future trajectory of $ym(n)$ is unknown, keep $ym(n)$ constant for the future trajectory. For real-time minimization of the cost function, the reference trajectory is usually smoothed using a reference model.

2. If the first time, start with an initial control input vector, $[u(n+1) \quad u(n+2) \quad \cdots \quad u(n+N_2)]^T$, equal to the zero vector, else start with the previously calculated control input vector. Generate predicted output vector of the plant, $[yn(n+N_1) \quad yn(n+N_1+1) \quad \cdots \quad yn(n+N_2)]^T$, using the plant model.

3. A new sequence of control inputs, $[u(n+1) \quad u(n+2) \quad \cdots \quad u(n+N_2)]^T$, is calculated that minimizes the cost function.

4. Repeat steps 2 and 3 until desired minimization is achieved.

5. Send the first control input, to the plant.

6. Repeat entire process for each time step.

Figure 2.3 shows the block diagram of the above process:

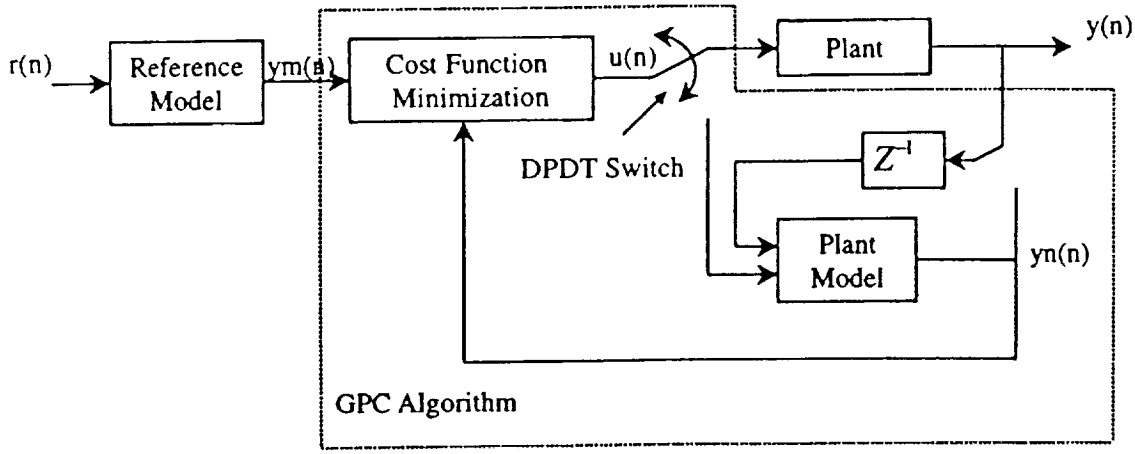


Fig 2.3 GPC block diagram

The cost function minimization (CFM) algorithm evaluates steps 3 and 4. The double pole double throw (DPDT) switch is kept in the down position while doing prediction. Having this switch lets the plant model to use previous measured plant outputs for future predictions and in the down position adaptation of the plant model can be performed.

A standard GPC algorithm is limited by the use of a linear predictor model. In many applications, the plant nonlinearities are more prevalent and the plant dynamics are faster than in the process industries. One way GPC can handle nonlinear plants is to use many linearized plant models about a set of operating points. If the plant is highly nonlinear, the set of operating points can be very large. Another technique involves developing a nonlinear model, which is an approximation of the dynamics of the actual nonlinear plant. If the approximation is incorrect, the accuracy of the model will be reduced. Ideally, a general-purpose nonlinear black-box estimator can serve the purpose.

Neural networks have been shown to be good nonlinear dynamic system estimators. By using neural network as predictor for GPC, the ability of controller to make predictions for a general nonlinear plant is improved. The use of neural networks as plant estimators will be addressed in more detail in the next chapter.

2.5 Modeling of the plant and disturbances

One important requirement of any control system is the stability of the closed loop system. In most cases, the controller design is based on the plant model information, which is approximate and does not account for the real life disturbances that are likely to affect the operation of the system. In view of this, a more realistic goal of the control system would be to ensure the closed loop stability in the presence of modeling errors, parametric uncertainties and unknown disturbances. The controllers designed using GPC methodology are capable of handling above mentioned uncertainties and disturbances while delivering required performance characterized by cost function to be minimized. The choices of plant model as well as disturbance model is an important aspect of the control design methodology in any model-based controller, and hence, the following subsections are devoted to this topic. The treatment given below refers to the linear systems.

2.5.1 System modeling

The linear system to be controlled can be represented by the following transfer function model in discrete domain:

$$y(k) = \frac{q^{-d} B(q^{-1})}{A(q^{-1})} u(k-1) \tag{2.3}$$

Now introduce the following identity:

$$\frac{1}{\hat{A}} = M_i + q^{-i}\frac{N_i}{\hat{A}} \Rightarrow M_i\hat{A} = 1 - q^{-i}N_i \tag{2.7}$$

where M_i has a degree less than or equal to $i-1$ and N_i is of degree $n_A - 1$. Eq.(2.7) is called a Diophantine equation whose solution can be computed manually using long division. Rewrite the Eq.(2.7) as,

$$\hat{y}(k+i) = \frac{q^{-d}\hat{B}}{\hat{A}}u(k+i-1) + N_i[y(k) - \hat{y}(k)] \tag{2.8}$$

The first part of Eq.(2.8) is prediction fully relying on the model, while the second part of Eq.(2.8) corrects for modeling errors or disturbances present on the output of the process. Obviously, if there are neither modeling errors or any kind of disturbances, the second part is equal to zero and the i-step-ahead predictor coincides with Eq.(2.5).

2.5.2 Disturbances modeling

Thus far, disturbance of the system has not been explicitly taken into account. In order to take this disturbance into account while predicting the output of the process, the disturbance must also be modeled. For this purpose, the model Eq.(2.3) is extended with a disturbance term $\xi(k)$ which represents the totality of all disturbance and without loss of generality is assumed to be located at the output of the process:

$$y(k) = \frac{q^{-d}B(q^{-1})}{A(q^{-1})}u(k-1) + \xi(k) \tag{2.9}$$

Our goal is to obtain prediction of the process output at $t = k+i$. Because this prediction depends on the disturbance characteristics, two classes of disturbance are considered hereafter.

Deterministic Disturbance

$$\xi(k) = \frac{C(q^{-1})}{D(q^{-1})} v(k) \tag{2.10}$$

where $v(k)$ is a signal that can be measured but not predicted and C and D are monic polynomials with degree n_C and n_D respectively. Prediction of $\xi(k)$ at $t = k+i$ is realized by substituting $k+i$ for k in Eq.(2.10):

$$\xi(k+i) = \frac{C(q^{-1})}{D(q^{-1})} v(k+i) \tag{2.11}$$

Because, by assumption, $v(k)$ cannot be predicted it is not possible to calculate $\xi(k+i)$ exactly. However, because of the filter $\frac{C(q^{-1})}{D(q^{-1})}$, $\xi(k+i)$ not only depends on the unknown $v(k+1),\cdots,v(k+i)$ but also on $v(k), v(k-1)\cdots$ which are assumed to be known. In order to separate Eq.(1.11) in future and past signals, the following Diophantine equation is used:

$$\frac{C}{D} = E_i + q^{-i}\frac{F_i}{D} \tag{2.12}$$

where E_i and F_i are polynomials with degree:

$$n_{E_i} = i-1 \qquad \text{If } n_D > 0$$

$$n_{E_i} = \min(i-1, n_C) \qquad \text{If } n_D = 0$$

$$n_{F_i} = \max(n_C - i, n_D - 1)$$

Note that a negative degree of a polynomial implies that the polynomial does not exist and may be replaced by zero. Using Eq.(2.12), Eq.(2.11) can be rewritten as:

$$\xi(k+i) = E_i v(k+i) + \frac{F_i}{D} v(k) \tag{2.13}$$

The first term of Eq.(2.13) involves future values of $v(k)$ and is thus unknown. The second term of Eq.(2.13) is known because, by assumption, $v(k)$ can be measured. Note that this second term is the future response of $\xi(k)$ if $v(k+i)=0$ for $i \geq 1$. In order to be able to calculate $\xi(k+i)$ an assumption for $v(k+i)$ must be made for $i \geq 1$. Hence, some *a priori* of $v(k)$ must be available.

Stochastic Disturbance

A stochastic disturbance appearing on the output of the process is assumed to be described by:

$$\xi(k) = \frac{C(q^{-1})}{D(q^{-1})} e(k) \tag{2.14}$$

where $e(k)$ is a discrete white noise sequence with zero mean and variance σ_e^2. Note that this disturbance model is quite general and all stationary random processes with rational spectral density can be described by Eq.(2.14) with the roots of C and D inside the unit circle.

The prediction of $\xi(k)$ at $t = k+i$ is realized by substituting $(k+i)$ for k in Eq.(2.14):

$$\xi(k+i) = \frac{C(q^{-1})}{D(q^{-1})} e(k+i) \tag{2.15}$$

Separation of future and past term is again realized by using the Diophantine equation Eq.(2.12):

$$\xi(k+i) = E_i e(k+i) + \frac{F_i}{D} e(k) \tag{2.16}$$

The first term involves future noise and thus is unknown. Although the second term involves the unknown $e(k)$ it can be calculated using data available at $t = k$.

$$\hat{y}(k+i) = \underbrace{G_i u(k+i-\hat{d}-1)}_{future} + \underbrace{\frac{H_i}{\hat{A}} u(k-1) + \frac{F_i}{C_i}[y(k)-\hat{y}(k)]}_{past} \tag{2.20}$$

where $\hat{y}(k)$ is given by Eq.(2.18). Because the degree of G_i is less than or equal to $i-\hat{d}-1$, the first term of Eq.(2.20) involves future controller outputs only. The other terms in Eq.(2.20) do not depend on the future controller outputs and hence are fully determined at $t=k$.

For convenience, the i-step-ahead predictor Eq.(2.20) for $i=\hat{d}+1,\ldots,H_p$ (Prediction Horizon) can be rewritten in the matrix notation yielding:

$$\hat{y} = Gu + H\bar{u} + Fc \tag{2.21}$$

where:

$$\hat{y} = [\hat{y}(k+\hat{d}+1),\cdots,\hat{y}(k+N_2)]^T$$
$$u = [u(k),\cdots,u(k+H_p-\hat{d}-1)]^T$$
$$\bar{u} = [\bar{u}(k-1),\bar{u}(k-2),\cdots]^T$$
$$c = [c(k),c(k-1),\cdots]^T$$

with

$$\bar{u}(k) = \frac{u(k)}{\hat{A}}$$

$$c(k) = \frac{y(k)-\hat{y}(k)}{\hat{C}}$$

and the dimensions of $\hat{y},u,\hat{u}$, and c are given by:

$$[\hat{y}] = (N_2-\hat{d})\times 1$$

$$[u] = (N_2-\hat{d})\times 1$$

$$[\bar{u}] = (\max_i(n_{H_i})+1)\times 1$$

$$[c] = (\max_i(n_F) + 1) \times 1$$

The matrices G, H, and F are built up of the elements of the polynomials G_i, H_i, and F_i, respectively:

$$G = \begin{bmatrix} g_0 & 0 & \cdots & 0 \\ g_1 & g_0 & \ddots & \vdots \\ \vdots & & \ddots & 0 \\ g_{H_p - \hat{d}-1} & \cdots & \cdots & g_0 \end{bmatrix} \qquad [G] = (N_2 - \hat{d}) \times (N_2 - \hat{d})$$

$$H = \begin{bmatrix} H_{\hat{d}+1} \\ \vdots \\ H_i \\ \vdots \\ H_{N_2} \end{bmatrix}$$

$$F = \begin{bmatrix} F_{\hat{d}+1} \\ \vdots \\ F_i \\ \vdots \\ F_{N_2} \end{bmatrix}$$

where g_i denotes the jth element of G_i. Note that $i \geq \hat{d} + 1$ since prediction of $y(k+1), \cdots, y(k+d)$ does not make sense because these values cannot be influenced by the future controller output sequence u. Note also that G is lower triangular.

As already mentioned, since the disturbance model is assumed to be described by Eq.(2.10) or Eq.(2.14), it can be taken into account simply by adding predictions of these disturbances to Eq.(2.21):

$$\hat{y} = Gu + H\tilde{u} + Fc + \xi \tag{2.22}$$

where:

The generalized predictive control law can be derived by the minimization of the criterion function Eq.(2.24) subject to Eq.(2.25) with respect to the controller output sequence over the control horizon $N_u : u(k),\ldots,u(k+N_1-1)$. Optimization of criterion function J with respect to the control vector $\mathbf{u}$ satisfies the following stationary condition:

$$\mathbf{g} = \frac{\partial J}{\partial \mathbf{u}} = 0$$

where $\mathbf{g}$ denotes the gradient and the symbol ∂ denotes the partial derivative. If $\mathbf{u}^*$ is the stationary point then $\mathbf{u}^*$ is a local minimum. If the Hessian $\mathbf{J}$ is positive definite with respect to $\mathbf{u}$, then the local minimum is the global minimum. The Hessian is given by:

$$\mathbf{J} = \frac{\partial^2 J}{\partial \mathbf{u}^2}$$

In order to calculate the gradient of (Eq.(2.24)) with respect to $\mathbf{u}$, the criterion function Eq.(2.24) is rewritten in the matrix notation as:

$$J = (\hat{\mathbf{y}}^* - \mathbf{y}_d^*)^T(\hat{\mathbf{y}}^* - \mathbf{y}_d^*) + \lambda \mathbf{u}^{*T}\mathbf{u}^* \tag{2.26}$$

where:

$$\mathbf{y}_d^* = [y_d(k+N_1),\ldots,y_d(k+N_2)]^T$$
$$\hat{\mathbf{y}}^* = [\hat{y}(k+N_1),\ldots,\hat{y}(k+N_2)]^T$$
$$\mathbf{u}^* = [u^*(k),\ldots,u^*(k+N_2-\hat{d}-1)]^T$$
$$u^*(k) = \Delta u(k)$$

Introduce the vector $\overline{\mathbf{u}}$:

$$\overline{\mathbf{u}} = [u(k),\ldots,u(k+N_u-1)]^T \tag{2.27}$$

Note that $\bar{\mathbf{u}}$ contains only those elements of the controller output sequence that need to be calculated. The other elements over the prediction horizon must satisfy Eq.(2.25). The gradient of Eq.(2.26) with respect to $\bar{\mathbf{u}}$ becomes:

$$\frac{\partial J}{\partial \bar{\mathbf{u}}} = 2\frac{\partial \hat{\mathbf{y}}^*}{\partial \bar{\mathbf{u}}}(\hat{\mathbf{y}}^* - \mathbf{w}^*) + 2\lambda \frac{\partial \mathbf{u}^*}{\partial \bar{\mathbf{u}}}\mathbf{u}^* \tag{2.28}$$

where the partial derivatives $\frac{\partial \hat{\mathbf{y}}^*}{\partial \bar{\mathbf{u}}}$ and $\frac{\partial \mathbf{u}^*}{\partial \bar{\mathbf{u}}}$ in equation Eq.(2.28) need to be determined.

Relationship between $\mathbf{u}^*$ and $\bar{\mathbf{u}}$

The relationship between $\mathbf{u}$ and $\bar{\mathbf{u}}$ can be derived by solving for $u(k+N_u),\ldots,u(k+N_2-\hat{d}-1)$ from Eq.(2.25):

$$\Delta u(k+i-1) = 0, 1 \le N_u < i \le N_2 - \hat{d} \tag{2.29}$$

Now using the following Diophantine equation, the necessary relationship between $\mathbf{u}$ and $\bar{\mathbf{u}}$ can be obtained:

$$\frac{1}{\Delta} = E_{i-N_u} + q^{-i+N_u}\frac{F_{i-N_u}}{\Delta} \Rightarrow \Delta E_{i-N_u} = 1 - q^{-i+N_u}F_{i-N_u} \tag{2.30}$$

where the degree of F_{i-N_u} is given by $n_\Delta - 1(=0)$.

It follows from Eq.(2.30) using Eq.(2.29):

$$u(k+i-1) = F_{i-N_u}u(k+N_u-1),\ 1 \le N_u < i \le N_2 - \hat{d} \tag{2.31}$$

Separation of the future and the past term is given by using the relationship:

$$F_{i-N_u} = G_{i-N_u} + q^{-N_u}H_{i-N_u} \tag{2.32}$$

where the degrees of G_{i-N_u} and H_{i-N_u} are given by $n_G = \min(N_u, n_\Delta) - 1$ and $n_H = n_\Delta - N_u - 1$, respectively.

Using Eq.(2.32) in Eq.(2.31) yields:

$$u(k+i-1) = G_{i-N_u} u(k+N_u-1) + H_{i-N_u} u(k-1) \tag{2.33}$$

with $1 \le N_u < i \le N_2 - \hat{d}$. Note that from Eq.(2.32) and Eq.(2.30), it can be easily shown that because $\Delta = 1 - q^{-1}$, the $F_{i-N_u} = 1, G_{i-N_u} = 1$, and the $H_{i-N_u} = 0$.

Now the relationship between $\mathbf{u}$ and $\bar{\mathbf{u}}$ can be given in a matrix notation as:

$$\mathbf{u} = \mathbf{M}\bar{\mathbf{u}} + \mathbf{N}\tilde{\mathbf{u}} \tag{2.34}$$

where $\mathbf{M}$ is a matrix of dimension $(N_2 - \hat{d}) \times N_u$:

$$\mathbf{M} = \begin{bmatrix} 1 & 0 & & \cdots & & 0 \\ 0 & 1 & & & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & \\ & & & & 1 & 0 \\ 0 & & \cdots & & 0 & 1 \\ 0 & \cdots & 0 & g_{1,n_G} & \cdots & g_{1,0} \\ \vdots & & \vdots & \vdots & & \vdots \\ 0 & \cdots & 0 & g_{j,n_G} & \cdots & g_{j,0} \end{bmatrix} \begin{matrix} \left.\begin{matrix} \\ \\ \\ \\ \end{matrix}\right\} N_u \\ \left.\begin{matrix} \\ \\ \\ \\ \end{matrix}\right\} N_2 - N_u - \hat{d} \end{matrix}$$

$\mathbf{N}$ is a matrix of dimension $(N_2 - \hat{d}) \times (n_\Delta - N_u)$:

$$\mathbf{N} = \begin{bmatrix} 0 & \cdots & 0 \\ \vdots & & \\ 0 & \cdots & 0 \\ h_{1,0} & \cdots & h_{1,n_H} \\ \vdots & & \vdots \\ h_{j,0} & \cdots & h_{j,n_H} \end{bmatrix} \begin{matrix} \left.\begin{matrix} \\ \\ \end{matrix}\right\} N_u \\ \left.\begin{matrix} \\ \\ \\ \end{matrix}\right\} N_2 - N_u - \hat{d} \end{matrix}$$

where $j = N_2 - N_u - \hat{d}$, and $\tilde{\mathbf{u}}$ is given by:

$$\tilde{\mathbf{u}} = [u(k-1), \cdots, u(k+N_u-n_\Delta)]^T$$

Note that if $N_u = N_2 - \hat{d}$, then $\mathbf{M} = I$ and $\mathbf{N} = 0$.

Next, the relationship between $\mathbf{u}^*$ and $\mathbf{u}$ is determined:

$$u^*(k+i-1) = \Delta u(k+i-1) \qquad 1 \le i \le N_2 - \hat{d} \tag{2.35}$$

Separation of future and past elements is obtained by using:

$$\Delta = \Phi_i + q^{-i}\Omega_i \tag{2.36}$$

where Φ_i and Ω_i are polynomials of degree $\min(i-1, n_\Delta)$ and $(n_\Delta - i)$ respectively. Also Φ_i is a monic polynomial.

Using Eq.(2.36) in Eq.(2.35) yields

$$u^*(k+i-1) = \Phi_i u(k+i-1) + \Omega_i u(k-1) \tag{2.37}$$

Note that the $\Omega_i = 0$, for $i > 1$ and when $i = 1$, $\Omega_i = -1$, so the relationship between $\mathbf{u}^*$ and $\mathbf{u}$ can be obtained as:

$$\mathbf{u}^* = \Phi \mathbf{u} + \Omega u(k-1)$$

where Φ is a lower triangular matrix of dimension $(N_2 - \hat{d}) \times (N_2 - \hat{d})$ and Ω is a matrix of dimension $(N_2 - \hat{d}) \times 1$.

So, the relationship between $\mathbf{u}^*$ and $\bar{\mathbf{u}}$ can be described as:

$$\mathbf{u}^* = \Phi \mathbf{M} \bar{\mathbf{u}} + \Omega u(k-1) \tag{2.38}$$

The partial derivative $\dfrac{\partial \mathbf{u}^*}{\partial \bar{\mathbf{u}}}$ now becomes:

$$\frac{\partial \mathbf{u}^*}{\partial \bar{\mathbf{u}}} = \mathbf{M}^T \Phi^T$$

Relationship between $\mathbf{y}^*$ and $\bar{\mathbf{u}}$

The partial derivative $\dfrac{\partial \hat{\mathbf{y}}^*}{\partial \bar{\mathbf{u}}}$ can be calculated by using the prediction model of the system Eq.(2.22):

$$\hat{\mathbf{y}}^{\bullet} = G\mathbf{u} + H\tilde{u} + Fc + \xi \tag{2.39}$$

where:

$$\hat{\mathbf{y}}^{\bullet} = [\hat{y}(k+N_1),\ldots,\hat{y}(k+N_2)]^T$$
$$\mathbf{u} = [u(k),\ldots,u(k+N_2-\hat{d}-1)]^T$$

Using the Eq.(2.37) and Eq.(2.39), the relationship between $\mathbf{y}^{\bullet}$ and $\overline{\mathbf{u}}$ is given by:

$$\hat{\mathbf{y}}^{\bullet} = G\mathbf{M}\mathbf{u} + H\tilde{u} + Fc + \xi \tag{2.40}$$

and hence :

$$\frac{\partial \hat{\mathbf{y}}^{\bullet}}{\partial \overline{\mathbf{u}}} = \mathbf{M}^T G^T$$

The predictive control law

The gradient Eq.(2.28) becomes:

$$\frac{\partial J}{\partial \overline{\mathbf{u}}} = 2\mathbf{M}^T(G^T G + \lambda\Phi^T\Phi)\mathbf{M}\overline{\mathbf{u}} + 2\mathbf{M}^T G^T(H\tilde{u} + Fc + \xi - \mathbf{w}^{\bullet} + \lambda\Phi^T\Omega u(k-1)) \tag{2.41}$$

and the Hessian $\mathbf{J}$ is:

$$\mathbf{J} = 2\mathbf{M}^T(G^T G + \lambda\Phi^T\Phi)\mathbf{M} \tag{2.42}$$

Assuming that the Hessian is singular, which means the global minimum of cost function has been archived, this minimum value of J with respect to $\overline{\mathbf{u}}$ can be obtained by setting the gradient Eq.(2.41) equal to zero and solving for $\overline{\mathbf{u}}$:

$$\overline{\mathbf{u}} = [\mathbf{M}^T(G^T G + \lambda)\mathbf{M}]^{-1}\mathbf{M}^T G^T(\mathbf{w}^{\bullet} - H\tilde{u} - Fc - \xi - \lambda\Phi^T\Omega u(k-1)) \tag{2.43}$$

Note that the matrix to be inverted is of dimension $N_u \times N_u$. Hence, a small control horizon is preferable for numerical reason in order to save computation time. In many practical situations the control horizon can be chosen small like $N_u \le n_A + 1$ and hence the inverse can be calculated easily for low-order systems. The first element of $\overline{\mathbf{u}}$ which

is equal to $u(k)$ is used to control the system. All other elements are not used and need not be calculated. Then, the Eq.(2.43) can be reduced to:

$$u(k) = v^T \mathbf{y}_d^* - h^T \tilde{u} - f^T c - v^T \xi - \lambda z^T u(k-1) \tag{2.44}$$

where:

$$v^T = x^T G^T \qquad 1\times(N_2 - N_1 + 1) \tag{2.45}$$

$$x^T = e_1^T R_v^{-1} \mathbf{M}^T \qquad 1\times(N_2 - \hat{d}) \tag{2.46}$$

$$e_1^T = [1,0,\ldots,0]^T \qquad 1\times N_u \tag{2.47}$$

$$R_v = \mathbf{M}^T (G^T G + \lambda \Phi^T \Phi)\mathbf{M} \qquad N_u \times N_u \tag{2.48}$$

$$h^T = v^T H \tag{2.49}$$

$$f^T = v^T F \tag{2.50}$$

$$z^T = x^T \Phi^T \Omega \qquad 1\times 1 \tag{2.51}$$

From the control law of Eq.(2.44), it can be seen that the difference between the model prediction and the system output is calculated by vector c, and the disturbance from the unknown source, ξ, which has influence on the output of the system is also measured and included in the calculation of the control signal. So, one can say that the prediction error and the disturbance is accounted for in the predictive control law. The results presented in this chapter will be used in the development of NGPC control design methodology given in later chapters.

CHAPTER 3 Neural Networks in Control Applications

This chapter gives background of neural network pertinent to the research presented in this thesis and describes architecture and training procedures for neural network for learning plant dynamics.

3.1 Neural network overview

Work on Artificial Neural Networks, commonly referred as "Neural Networks" has been motivated right from its inception by the recognition that the brain computes in an entirely different way from the conventional digital computer. Investigators from across the scientific spectrum, including neurobiologists, psychologists, physicists, computer scientists and engineers, have been attracted to this field. Neurobiologists and psychologists are interested in real neural networks and particularly in understanding the physiological and psychological functioning of the human neural system. The interest of neuroscientists is consequently in the stimulus-response characteristics of individual neurons as well as networks of neurons. Psychologists study brain functions at the cognitive and behavioral level and are interested in using neural network based techniques to create detailed models of human behavior. Computer scientists are intrigued by the prospect of designing novel information processing devices. All research communities are exited at the opportunity for cooperative research among scholars of different fields and the prospect of drawing ideas and perspectives from many different disciplines. They all believe that a meaningful theoretical integration of knowledge from different fields is possible and will yield useful solution for many scientific problems.

System theory is scientific discipline that is inherently interdisciplinary in nature and extends from design, development and production on one hand to mathematics on the other. Advances in system theory have been made through a combination of modeling, computation and experimentation, all of which are essential to the study of neural networks. The different areas of current research in system theory including adaptive and learning systems, nonlinear systems, stochastic systems and hierarchical and decentralized systems are directly relevant to the study of dynamic systems in which neural networks are to be used as components.

Nonlinear control theory deals with the modeling and control of nonlinear dynamical systems. When the characteristics of the process to be controlled are either unknown or only partially known, the problem belongs to the domain of nonlinear adaptive control. Linear adaptive control theory, which has been explored for over thirty years, has provided numerous concepts related to the structures that are appropriate for both identifiers and controllers. From a system theoretic point of view artificial neural networks can be considered as convenient parameterizations of nonlinear maps from one finite dimensional space to another. Since methods for training such networks using input-output data are currently well known, they are ideally suited to approximate unknown nonlinear functions in differential or difference equations used to represent nonlinear dynamic systems. From the foregoing comments it is clear that the results in nonlinear control theory, concepts and structures provided by linear control, and the approximating capabilities of neural networks have to be judiciously combined to deal with problems of nonlinear adaptive control that arise in complex dynamic systems.

Neural networks are aggregates of interconnected nerve cells or neurons. The human brain is a neural network consisting of 100 billion neurons in which a single neuron can be connected to upon 10,000 other neurons, so that there are an estimated 1000 trillion connections. Artificial neural networks (ANN) are so named because their design is based on the structure of natural neural networks. The great interest in artificial neural networks stems from the fact that the human brain can learn, remember, think and act purposefully, and building intelligent systems that can model human behavior has been one of the major aims of engineers and computer scientists over the years.

Most of the research in artificial neural networks has been on static feedforward multilayer networks and the recurrent Hopfield network. The major applications of such networks have been in the areas of function approximation, optimization, and the pattern recognition. With the introduction of dynamics and feedback, the scope of applications of multilayer neural networks was significantly increased. Hence, the control of complex systems, using intelligent sensors and controllers based on neural network, will involve the study of dynamical systems and neural networks connected in arbitrary configurations.

3.2 Benefits of neural networks

Neural network derives its computing power through its massively parallel distributed structure its ability to learn and generalize. Generalization refers to the neural network producing reasonable outputs for inputs not encountered during training (learning). These two information-processing capabilities make it possible for neural networks to solve many complex (large-scale) problems that are currently intractable. In

practice. however. neural networks cannot provide the solution by working in independent mode. Rather, they need to be integrated into a consistent system engineering approach. Specifically, a complex problem of interest is decomposed into a number of relatively simple tasks, and neural networks are assigned a subset of tasks (e.g., pattern recognition, associate memory, control) that match their inherent capabilities. It is important to recognize, however, that there is a long way to go (if ever) before a computer architecture can be built to mimic a human brain.

Multilayer neural networks offer the following properties and capabilities that will be useful for us in modeling complex systems in model-based control designs:

1. *Nonlinear mapping*: A neuron with nonlinear activation function acts like a nonlinear mapping device. Consequently, a multilayer neural network, which is made up of large number of interconnection of neurons, forms a nonlinear modeling device. Moreover, the nonlinearity is of a special kind in the sense that it is distributed throughout the network. Nonlinear mapping is a very important property, particularly if the physical system to be modeled is inherently nonlinear.

2. *Learning ability*: A popular paradigm of learning called supervised learning involves the modification of synaptic weights of a neural network by applying a set of labeled training samples or task examples. Each example consists of a unique input signal and the corresponding desired response. The network is presented an example picked at random from the set, and the synaptic weights (free parameter) of the network are modified so as to minimize the difference between the desired response and the actual response of the network produced by the input signal in accordance with an appropriate statistical criterion. The training of the network is repeated for many examples in the set

until the network reaches a steady state, i.e., it reaches a state where there are no further significant changes in the synaptic weights. Thus, the network learns from the examples by learning an input-output mapping for the problem at hand. Such an approach brings to mind the study of nonparametric statistical inference, which is a branch of statistics dealing with model-free estimation. Consider, for example, a pattern classification task, where the requirements is to assign an input signal representing a physical object or event to one of several prespecified categories (classes). In a nonparametric approach to this problem, the requirement is to "estimate" arbitrary decision boundaries in the input signal space for the pattern-classification task using a set of examples, and to do so without invoking a probabilistic distribution model. A similar point of view is implicit in the supervised learning paradigm, which suggests a close analogy between the input-output mapping performed by a neural network and nonparametric statistical inference.

3. *Adaptivity*: Neural network has an inherent capability to adapt its synaptic weights to changes in the input-output model of the surrounding environment. In particular, a neural network trained to model in a specific environment can be retrained to accommodate minor changes in the operating environment conditions. Moreover, when it is operating in a nonstationary environment (i.e., one whose statistics change with time), a neural network can be designed to change its synaptic weights in 'real time'. The natural architecture of a neural network for pattern classification, signal processing, and control application, coupled with the adaptive capability of the network, make it an ideal tool for use in adaptive pattern classification, adaptive signal processing, and adaptive control. In general, the more adaptivity we have in the network modeling the system, the

more robustness will result in performance when the system is required to operate in a nonstationary environment, provided the adaptive system is stable.

4. *Evidential Response*: In the context of pattern classification, a neural network can be designed to provide information not only about which particular pattern to select, but also the confidence in the decision made. This latter information may be used to reject ambiguous patterns, should they arise, and thereby improve the classification performance of the network. This property is not directly used in control applications presented in this work.

5. *Contextual Information*: Knowledge is represented by the very structure and the activation state of a neural network. Every neuron in the network is potentially affected by the global activity of all other neurons in the network. Consequently, a neural network deals with contextual information naturally.

6. *Fault Tolerance*: A neural network, implemented in hardware form, has the potential to be inherently fault tolerant in the sense that its performance is degraded gracefully under adverse operating conditions. For example, if a neuron or its connecting links are damaged, recall of a stored pattern is impaired in quality. However, owing to the distributed nature of information in the network, the damage has to be extensive before the catastrophic failure can occur. The overall response of the network shows degradation in performance rather than resulting in catastrophic failure.

3.3 Model of a neuron

A neuron is an information-processing unit that is fundamental to the operation of a neural network. Figure 3.1 shows a model of a neuron.

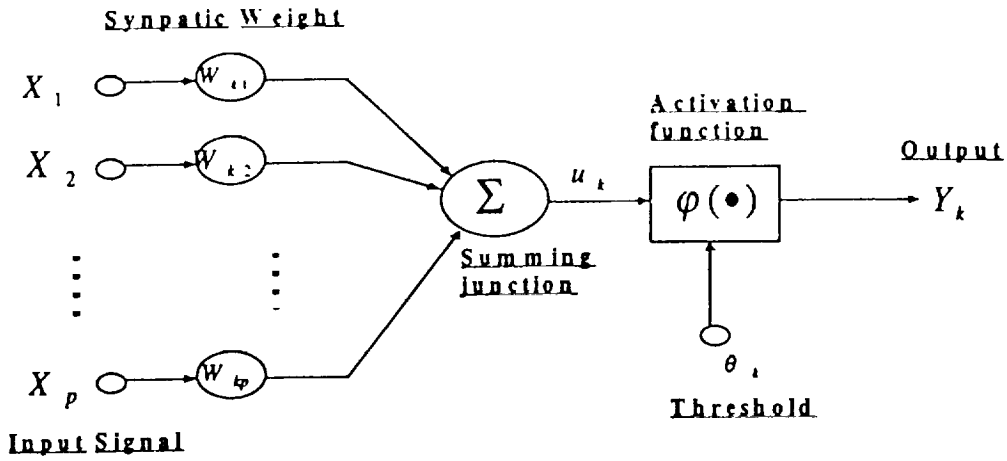


Fig 3.1 Model of a neuron

The three basic elements of a neuron model can be described as follows:

1. A set of synapse or connecting links, each of which is characterized by a weight or strength of its own. Specifically, a signal X_j at the input of synapse j connected to neuron k is multiplied by the synapse weight W_{kj}. It is important to make a note of the ordering in which the subscripts appear W_{kj}. The first subscript refers to the neuron the input is connected to and the second subscript refers to the input to which the weight is applied.

2. An adder for summing the input signals, weighted by the respective weights on the synapse of the neuron. The operation constitutes a weighted linear combination of inputs.

3. An activation function for limiting the amplitude of the output of a neuron. The activation function is also refereed to as a squashing function since it squashes (limits) the permissible amplitude range of the output signal to some finite value.

The model of a neuron shown in Fig 3.1 also includes an externally applied threshold θ_k that has the effect of lowering the net input of the activation function. On the other hands, the net input of the activation function may be increased by employed by a bias term rather than a threshold. The bias is the negative of the threshold.

In mathematical terms, a neuron k can be described by writing the following pair of equations:

$$u_k = \sum_{j=1}^{p} W_{kj} X_j \tag{3.1}$$

and

$$y_k = \varphi(u_k - \theta_k) \tag{3.2}$$

where $X_1, X_2, \cdots, X_p$ are the input signals; $W_{k1}, W_{k2}, \ldots, W_{kp}$ are the synaptic weights of neuron k; u_k is the linearly combined inputs; θ_k is the threshold; $\varphi(\cdot)$ is the activation function; and y_k is the output of the neuron. The activation functions define the output of a neuron in terms of the activity level at its input. We may identify several types of activation functions, for example, threshold function, piecewise-linear function, typically the most common one is sigmoid function, which is defined as a strictly increasing function that exhibits smoothness and asymptotic properties. An example of the sigmoid function is the logistic function, defined by:

$$\varphi(v) = \frac{1}{1 + \exp(-av)} \tag{3.3}$$

where a is the slope parameter of the sigmoid function. By varying this parameter, we obtain sigmoid function of different slopes, as illustrated in Fig 3.2.

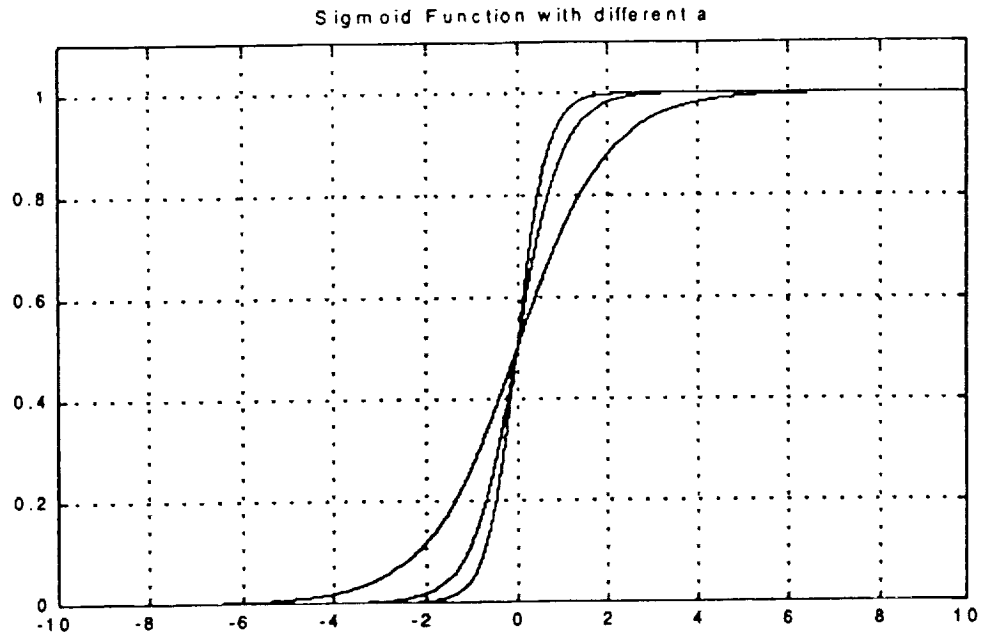


Fig 3.2 Sigmoid function with different 'a'

In fact, the slope at the origin equals a/4. In the limit, as the slope parameter approaches infinity, the sigmoid function becomes simply a threshold function.

3.4 Model of a layer

A layered neural network is a network of neurons organized in the form of layers. A layer consists of a set of nodes. This collection of nodes maps the N^i-dimension vector of inputs to the N^{i+1}-dimension vector of outputs. A schematic of the layer activity is given in Fig 3.3.

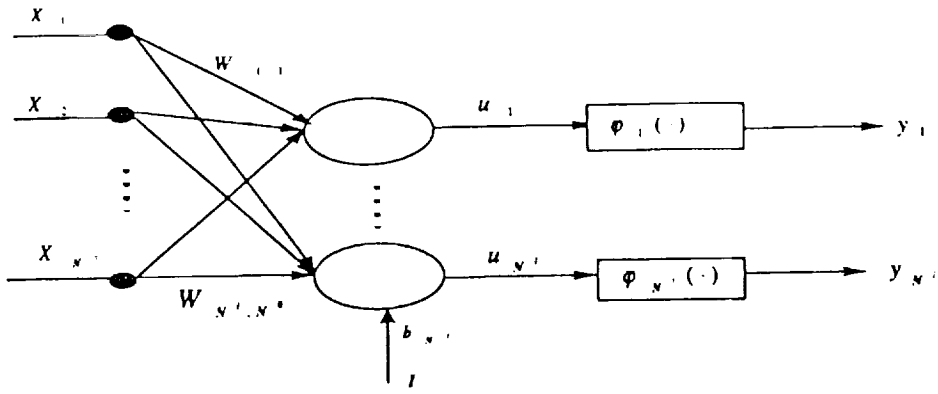


Fig 3.3 Layer of a neural network

In the first stage, the node activities are calculated using Equation (3.1) and (3.2). This activity can be simply represented by the product of the input vector $[X_1, X_2, \cdots, X_{N^0}]^T$ and a weight matrix W, the addition of a bias vector b, and the transformation function φ. The equations that describe this activity are :

$$u_j = \sum_{i=1}^{N^0} W_{j,i} X_i + b_j \tag{3.4}$$

and

$$y_j = \varphi_j(u_j) \qquad \text{For} \quad j = 1,2,\ldots,N^l \tag{3.5}$$

where

X_i is the i^{th} element of the input vector of length N^0,

$W_{j,i}$ is the weight connecting the i^{th} input, X_i, with the j^{th} output y_j,

b_j is a bias input to the j^{th} node,

$\varphi_j(\cdot)$ is the output function of the j^{th} node,

y_j is the output of the j^{th} node.

3.5 The network architecture

The manner in which the neurons of a neural network are structured is intimately linked with the learning algorithm used to train the network. We may therefore speak of learning algorithm (rules) used in the design of the neural networks as being structured. In general, we may identify four different classes of network architectures:

1. Single-Layer Feedforward Networks
2. Multilayer Feedforward Networks
3. Recurrent Networks
4. Lattice Structures

The most commonly used neural network structure in the multilayer feedforward networks draw our attention here. The multilayer feedforward network distinguishes itself by the presence of one or more hidden layers, whose computation nodes are correspondingly called hidden neurons or hidden units. The function of the hidden neurons is to intervene between the external input and the network output. By adding one or more hidden layer, the network is enabled to extract high-order statistics, which is particularly valuable when the size of the input layer is large. A multilayer feedforward neural network can be broken down into three parts:

1. Input Layer
2. Hidden Layer, and
3. Output Layer

The input layer distributes the inputs to the following layer, it doesn't multiply any weights and doesn't process the input vector through a output function. The hidden layers and the output layer consist of processing nodes. Each layer is fully connected via

connection weights to the next layer as shown in Fig 3.4. This network has L layers with N^l nodes on layer l. The input layer is denoted as layer 0.

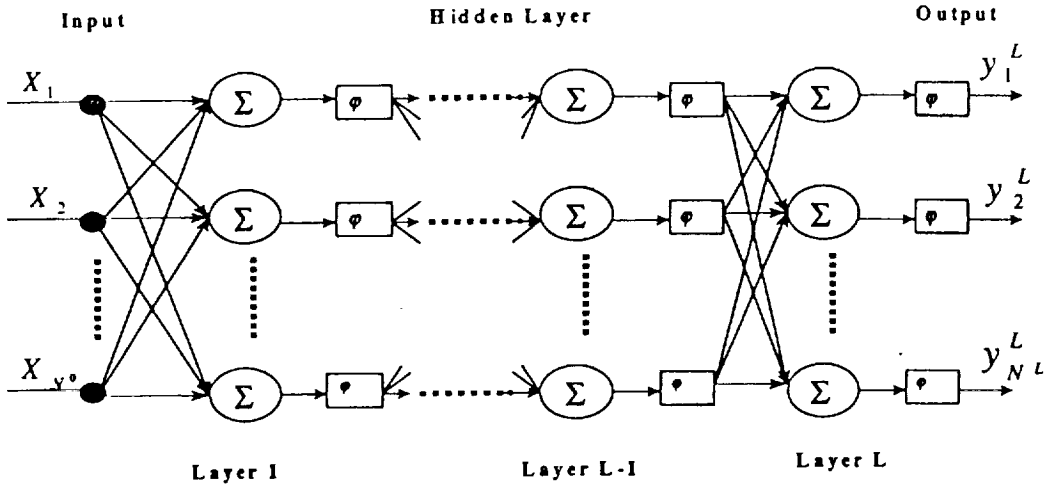


Fig 3.4 Multilayer feedforward neural network

Forward propagation is accomplished by presenting an input vector, $[X_1 \quad X_2 \cdots X_{N^0}]^T$, to the network. This input is fed out to the first hidden layer and propagated through the nodes by evaluation of:

$$u_j^l = \sum_{i=1}^{N^{l-1}} W_{j,i}^l y_i^{l-1} + b_j^l \tag{3.6}$$

$$y_j^l = \varphi_j^l(u_j^l) \tag{3.7}$$

where :

L is the number of layers,

N^l is the number of nodes on the l^{th} layer, $l = 0,1,2,\cdots,L$,

y_j^l is the output of the j^{th} node of the l^{th} layer (if $l = 0$ then $y_j^0 = X_j$),

X_j is the j^{th} element of the input node with a vector of length N^0,

$W_{j,i}^l$ is the j,i^{th} element of the weight of the l^{th} layer with a matrix of size $N^{l+1} \times N^l$,

b_j^l is a bias input to the j^{th} node of the l^{th} layer,

$\varphi_j^l(\cdot)$ is the output function of the j^{th} node of the l^{th} layer.

The output at each layer is fed to the next layer, and the process is repeated until the output layer L is reached. Thus, output signal at layer L is nonlinear function of input signal at layer 0. Next section describes how such an architecture can be used to model dynamic systems.

3.6 Neural network as input/output estimators

3.6.1 Introduction to I/O estimator using multilayer neural networks

Multilayer network have been applied successfully to solve some difficult and diverse problems by training them in a supervised manner with a highly popular algorithm known as error back-propagation algorithm. This algorithm is based on error-correction learning rule. A brief introduction to this type of learning rule will be useful before the actual algorithm structure is introduced in later section.

The error back-propagation process consists of two passes through the different layers of the network: a forward pass and a backward pass. In the forward pass, an input vector is applied to the sensory nodes of the network, and its effect propagates through the network, layer by layer. Finally, a set of outputs is produced as the actual response of

the network. During the forward pass the weights of the network are all fixed. During the backward pass, on the other hand, the weights of the network are all adjusted in accordance with the error-correction rule. Specifically, the actual response of the network is subtracted from a desired (target) response to produce an error signal. This error signal is propagated backward through the network, against the direction of the weights connections, hence the name "error back-propagation". The weights are adjusted so as to make the actual response of the network move closer to the desired response. The learning process performed with the algorithm is called back-propagation learning.

The research interests in Multilayer feedforward networks dates back to the pioneering work of Rosenblatt (1962) on perceptrons and that of Widrow (1962). However, the tool that was missing in those early days of multilayer feedforward networks was what we now call back-propagation learning. The usage of the term "back-propagation" appears to have evolved after 1985. However, the basic idea of back-propagation was first described by Werbos (1974) in his Ph.D. thesis. Subsequently, it was rediscovered by Rumelhart, Hinton, and Williams (1986), and popularized through the publication of the seminal book entitled *Parallel Distributed Processing* (Rumelhart and McClelland, 1986).

The development of the back-propagation algorithm represents a "landmark" in neural networks in that it provides a computationally efficient method for the training of multilayer neural networks. Although it cannot be claimed that the back-propagation algorithm can provide a solution for all solvable problems, it is fair to say that it has put to rest the pessimism about learning in multilayer machines that may have been inferred from the book by Minsky and Papert (1969).

In previous section (1.5), the multilayer feedforward network architecture has been described. The main property of this network is their ability to learn input/output (I/O) relationships. These networks have been shown to be universal function approximators. Adding a time series structure to the static network can convert the network to dynamic system estimator. These features are described in the following sub-sections.

3.6.2 Neural network as function estimator

The problem addressed is as follows:

Given a set of I/O data, $[X_1 \quad X_2 \quad \cdots \quad X_{N^0}]^T$, $[y_1 \quad y_2 \quad \cdots \quad y_{N^L}]^T$ and a multilayer feedforward network, it is required to find a set of weights and biases that would approximate the I/O relationship.

The first question that arises is how good could this approximation be. The research result mentioned in Section 3.6.1 shows that a multilayer feedforward network with one hidden layer can approximate measurable and continuous functions to any desired degree of accuracy. The second question is then how does one obtain the weights and biases for this approximation. This can be accomplished by using a well-known algorithm namely, back-propagation algorithm. Next section gives a brief review of the back propagation algorithm used for tuning weights of neural networks in order to approximate given I/O data.

3.6.3 Back-propagation algorithm

The error signal at the output of neuron j at iteration n (i.e., presentation of the n^{th} train pattern) is determined by :

$$e_j(n) = d_j(n) - y_j(n) \tag{3.8}$$

Define instantaneous value of the squared error of the neuron j as $\frac{1}{2}e_j^2(n)$, and instantaneous sum of squared errors of the network is thus written as :

$$E(n) = \frac{1}{2}\sum_{j \in C} e_j^2(n) \tag{3.9}$$

where the set C includes all the neurons in the output layer of the network. Let N denote the total number of patterns contained in the training set. The average squared error is obtained by summing $E(n)$ over all n and then normalizing with respect to the set size N, i.e.,

$$E_{av} = \frac{1}{N}\sum_{n=1}^{N} E(n) \tag{3.10}$$

Define the network internal activity level $v_j(n)$ which is produced at the input of the neuron j as :

$$v_j(n) = \sum_{i=0}^{p} W_{ji}(n) y_i(n) \tag{3.11}$$

where p is the total number of inputs (excluding the threshold) applied to neuron j. Hence the function signal $y_j(n)$ appearing at the output of neuron j at iteration n is :

$$y_j(n) = \varphi_j(v_j(n)) \tag{3.12}$$

The back-propagation algorithm applies a correction $\Delta W_{ji}(n)$ to the weight $W_{ji}(n)$, which is proportional to the instantaneous gradient $\frac{\partial E(n)}{\partial W_{ji}(n)}$. According to the chain rule, the gradient can be rewritten as follows:

$$\frac{\partial E(n)}{\partial W_{ji}(n)} = \frac{\partial E(n)\partial e_j(n)\partial y_j(n)\partial v_j(n)}{\partial e_j(n)\partial y_j(n)\partial v_j(n)\partial W_{ji}(n)} \tag{3.13}$$

Using Eq. (3.8) ~ Eq.(3.12) in Eq.(3.13) yields:

$$\frac{\partial E(n)}{\partial W_{ji}(n)} = -e_j(n)\varphi_j'(v_j(n))y_i(n) \tag{3.14}$$

The correction $\Delta W_{ji}(n)$ applied to $W_{ji}(n)$ is defined by the delta rule:

$$\Delta W_{ji}(n) = -\eta \frac{\partial E(n)}{\partial W_{ji}(n)} \tag{3.15}$$

where η is the constant that determines the rate of learning and is called learning-rate parameter of the back-propagation algorithm. The use of the minus sign in Eq.(3.15) accounts for the gradient descent in the weight space. Accordingly, the use of Eq.(3.14) in Eq.(3.15) yields :

$$\Delta W_{ji}(n) = -\eta \delta_j(n) y_i(n) \tag{3.16}$$

where the local gradient $\delta_j(n)$ is itself defined by:

$$\delta_j(n) = e_j(n)\varphi_j'(v_j(n)) \tag{3.17}$$

When a neuron is located in a hidden layer of the network, there is no specified desired response for that neuron. Accordingly, the error signal for a hidden neuron would have to be determined recursively in terms of the error signals of all the neurons to which that hidden neuron is directly connected; this is where the development of the back-

propagation gets complicated. Suppose, a neuron j is directly connected with another neuron k, then using the chain rule, yields:

$$\delta_j(n) = \varphi_j'(v_j(n)) \sum_k \delta_k(n) W_{kj}(n) \tag{3.18}$$

Now the relations that are derived for the back-propagation algorithm can be summarized. First, the correction $\Delta W_{ji}(n)$ applied to the weights connecting neuron i to neuron j is defined by the delta rule as:

$$\begin{pmatrix} Weight \\ correction \\ \Delta W_{ji}(n) \end{pmatrix} = \begin{pmatrix} learning-rate \\ parameter \\ \eta \end{pmatrix} \cdot \begin{pmatrix} local \\ gradient \\ \delta_j(n) \end{pmatrix} \cdot \begin{pmatrix} input\ \ signal \\ of\ \ neuron\ \ j \\ y_j(n) \end{pmatrix} \tag{3.19}$$

Second, the local gradient $\delta_j(n)$ depends on whether neuron j is an output node or a hidden node:

1. If neuron j is an output node, $\delta_j(n)$ equals the product of the derivative $\varphi_j'(v_j(n))$ and the error signal $e_j(n)$, both of which are associated with neuron j;

2. If neuron j is a hidden node, $\delta_j(n)$ equals the product of the associated derivative $\varphi_j'(v_j(n))$ and the weight sum of the δ's computed for the neurons in the next hidden or output layer that are connected to neuron j;

3.6.4 Back-propagation training

Minsky and Papert (1962) pointed out limitations of Rosenblatt's perceptrons. One of these limitations was that there was no learning rule for a multilayer perceptron architecture network. As mentioned in previous Section 3.6.3, the back-propagation algorithm minimizes the root mean square (RMS) error with respect to the weights and

biases of the network. The RMS error is a function of the difference between the desired output and the actual network output over all training pairs. RRM error is given by:

$$RMS = \frac{1}{PN^L}\sqrt{\sum_{p=1}^{P}\sum_{j=1}^{N^L}(y_{jp} - y_{jp}^L)} \tag{3.20}$$

where :

P is the number of input/output training pairs,

L is the number of layers'

N^L is the number of nodes on the output layer L,

y_{jp}^L is the output of the j^{th} node of the L^{th} layer for the p^{th} training pattern

When the minimization of the RMS error is performed, an update rule for the weights and biases is obtained. This update is applied to each input/output training pair. The pattern notation p is dropped for simplicity. Thus, the update equation is:

$$\Delta W_{ji}^l = \eta \delta_j^l y_i^{l-1} \tag{3.21}$$

and

$$\delta_j^l = \begin{Bmatrix} \varphi'^l{}_j(v^l{}_j)\sum_{j=1}^{N^{l+1}}\delta_j^{l+1}W_{ji}^{l+1}, & for \quad l \neq L \\ \varphi'^L{}_j(v_j^L)(y_j - y_j^L), & for \quad l = L \end{Bmatrix} \tag{3.22}$$

where :

L is the number of layers,

N^{l+1} is the number of nodes on the layer l+1,

η is the learning rate,

ΔW_{ji}^l is the change in the i^{th}, j^{th} weight of the l^{th} layer,

y_j^l is the output of the j^{th} node of layer l, (if l=0,then $y_j^l = X_j$),

X_j is the j^{th} element of the input node with a vector of length N^0 .

W_{ji}^l is the j^{th}, i^{th} element of the weight of layer l with a matrix of size $N^{l+1} \times N^l$,

$\varphi_j^{'l}(*)$ is the derivative of the output function for the j^{th} node of layer l with respect to v_j^l .

The biases of the network can be viewed as a weight with an input of one, thus update rule for the biases is the same as the weights with the input to the node equal to one. In the rest of this thesis, all references and comments made about the weights will also apply to the biases.

The weights and biases are typically initialized with small random numbers. The rule used here is to initial these values with uniformly distributed random number between -0.01 and 0.01 divided by the number of weights connecting to the node. This normalization will avoid saturation of a node when the first few input are presented. If a node saturates, the derivative at that point is about zero, and thus very small updating of the weights occur. This initialization has proven successfully when training a network to represent a dynamic system.

The back-propagation algorithm is a special case of a gradient descent algorithm (as described in Section 3.6.3). It is an iterative nonlinear first-order unconstrained optimization technique. The performance of the algorithm depends on the initial weight values, the learning rule parameter, and the way how the weight updates are used.

There are two techniques in the use of the ΔW's. The first technique, called batch learning, obtains the ΔW's for all of the input/output training pairs, averages them

together, and then changes all the weights of the network. The other technique, called on-line learning, updates the weights after obtaining the ΔW for a single input/output pair. The latter technique is more applicable to control systems work because one typically does not have a predefined training set, but a set is generated in real-time while controlling a plant.

The scale factor η, which is known as the learning rate, adjusts the rate of convergence of the network to the desired output. If the rate is small, the weight updates are small and converge to a desired set slowly. If the rate is large, the weight updates are large which could lead to oscillations.

3.6.5 Neural networks as dynamic plant estimator

In previous sections, neural networks are used to approximate static mappings, or static I/O relationships. The same universal approximation property can lead to dynamic plant estimation if an appropriate dynamic structure is added to the neural network. Since neural networks are implemented using digital computers, discrete-time models of dynamic systems will be convenient for analysis and design.

For a network to model a dynamic plant, a special structure must be added to the network to capture the effects of the previous inputs and/or outputs or state information. Several techniques, based on linear filter principles, have been proven to be particularly useful. One technique assigns the input of the network to the states of the plant. This works well when all the states of the plant are measurable. Another technique uses recursion in the network by taking the output of a node and feeding it back to itself. Different variations of this idea can be implemented such as, feeding the output of the

node to the inputs of the other nodes on the same layer or feeding back the output of the network to the input of the network. Making a recurrent network introduces some other problems, such as, a more complex learning rule, slower convergence, and higher sensitivity to stability in learning. Useful neural network models use just the input and output data for control purposes. In this case, the neural network will act as an observer and it will need to have available the inputs and outputs of the plant. To make the neural network a dynamic plant estimator, dynamic structures based on typical linear system identification models can be used. For example, the neural network input could be augmented with:

--- past values of the plant's input, or

--- past values of the plant's input and output, or

--- past values of the predicted outputs and plant's inputs, or

--- past values of the predicted outputs and the plant's input /output

These four structures lead to neural network implementations of various plant models, namely, *Finite Impulse Response* (FIR) model, *Auto-Regressive eXogenous* (ARX) model, *Output Error* (OE) model, *Auto-Regressive Moving-Average eXogenous* (ARMAX) model respectively.

The choice of the model depends on the complexity of the plant to be modeled. When the plant is stable, an FIR model maybe used. This model will require a tapped time delay element for each increment of times for the length of the transient response. If the plant has both low and high frequency modes this model iscomputationally slow. The ARX model requires far less tapped time delays than the FIR model. When sensor noise

$$\begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_{n_d+1} \\ u_{n_d+2} \\ u_{n_d+3} \\ \vdots \\ u_{d_d+n_n+1} \end{bmatrix} \equiv \begin{bmatrix} u(n) \\ u(n-1) \\ \vdots \\ u(n-n_d) \\ y(n-1) \\ y(n-2) \\ \vdots \\ y(n-d_d) \end{bmatrix} \tag{3.23}$$

where n_d is the number of input nodes associated with u not counting $u(n)$, and d_d is the number of input nodes associated with $y(n-1)$.

For the following development, the neural network will have one hidden layer containing several hidden nodes that use a general activation function $\varphi(*)$. A single layer is chosen because it has been shown that single hidden layer neural networks can approximate a measurable set to any desired degree of accuracy. The output node uses a linear output function for scaling the network. This function has a slope of 1.

The equation for this network architecture is:

$$v_j(n) = \sum_{i=0}^{n_d} \{W_{j,i+1} X(n-i)\} + \sum_{i=1}^{d_d} \{W_{j,i+n_d+1} y(n-i)\} + b_j \tag{3.24}$$

and

$$yn(n) = \sum_{j=1}^{h_d} \{W_j \varphi_j(v_j(n))\} + b \tag{3.25}$$

where :

$yn(n)$ is the output of the network at time n,

$\varphi_j(*)$ is the activation function for the j^{th} node of the hidden layer,

h_d is the number of hidden nodes in the hidden layer,

W_j is the weight connecting the j^{th} hidden node to the output node,

$W_{j,i}$ is the weight connecting the i^{th} input node to the j^{th} hidden node,

b_j is the bias on the j^{th} hidden node,

b is the bias on the output node.

It will be convenient to represent Fig 3.5 with a single block representation as shown in Fig 3.6:

Fig 3.6 Block diagram representation of a time delay network

The delay nodes for the network are assumed to be contained within the block diagram and are not shown as inputs. An alternative input may be defined to capture the plant dynamics. Using the network output instead of the plan output for the delayed inputs converts the static network into a recurrent network.

3.6.7 Training procedure for neural network

The network shown in Fig 3.6 is ready to be placed into its learning environment. The plant to be learned is a non-linear dynamic plant. The plant is sampled by using a digital to analog converter (D/A) at the input of the plant followed by a zero order hold (ZOH) circuit. This signal is then feed to the plant. The output of the plant is then fed through an analog to digital converter (A/D). This process is shown in Fig 3.7.

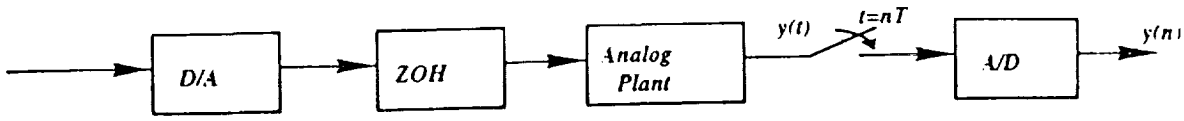


Fig 3.7 Discrete plant

The objective of the network is to predict the plant's performance for a given input at time $t = nT$, where t is the current time in seconds, n is an integer representing discrete time, and T is the sampling interval.

The choice of T, the number of tapped time delays n_d and d_d, type of model (OE or AMX) and the type of excitation signal are important variables for proper modeling of the plant. The sampling interval T is chosen to satisfy Nyquist frequency estimated from the bandwidth of the plant. Specifically one should choose T to be about 20 to 40 times the highest frequency. To slow sampling will limit the ability to model the high frequency modes. Too fast sampling can cause the modeling of a minimum phase plant to be a non-minimum phase one. There is also a problem with the computer resolution. The fast we sample the less the difference is on the output of the plant, *y(n)*, between sampling points. This difference determines the magnitude of the weights for the zeros dynamics, that is the weights associated with *u(n)* and its delays, and the resolution of the weights for the poles dynamics, that is the weights associated with *y(n-1)* and its delays. The fast we sample, the smaller the weights are for the zero dynamics and the smaller resolution for the poles dynamics.

The choice of the number of delays n_d and d_d is based on the order of the plant and additional delays required for capturing unmodeled dynamics. The choice of the

number of hidden nodes is still an open problem. Some thumb rules can be used based on experience.

The choice of whether the plant's output or the network's output is used for the input is of particular importance. The use of the plant's output results in a static network, the ARX model, and thus the learning algorithm is as defined in the previous section. The problem with this configuration occurs when there is significant sensor noise. This could introduce bias error in the network parameters (i.e., weights and biases). When there is significant sensor noise, it is necessary to use the output of the network for learning. This converts the network to a recurrent network (the OE model) and requires a recurrent algorithm. Since recurrent learning algorithm for a general recurrent network is computationally prohibitive a zero order approximation is typically used. The zero order approximation is the same algorithm described in Section 3.6.4. Since this is an approximation, the convergence is slower and less stable. It is recommended that this configuration be used only when the former does not produce good results. In this thesis, we assume the sensor noise is negligible, and therefore use the ARX model for training the network.

The initialization of the weights is done as specified in Section 3.6.4 except when the network is initialized with a nominal linear plant model. Then a correlation of the networks' weights and a discrete linear model of the plant is needed.

Training a network to be a dynamic system estimator is accomplished in the same manner as in Section 3.6.4. The training procedure can be described as follows. First, a forward pass through the network process the input $u(n)$, $y(n-1)$, and their past values, giving the current output $y(n)$. Second, the error signal is formed by subtracting the neural

$$v_j(n+k) = \sum_{i=0}^{n_d} W_{j,i+1} u(n+k-i) + \sum_{i=1}^{\min(k,d_d)} (W_{j,n_d+i+1}\, yn(n+k-i)) + \sum_{i=k+1}^{d_d} (W_{j,n_d+i+1}\, y(n+k-i)) + b_j \tag{3.27}$$

The second summation of Eq.(3.27) introduces the predicted outputs. This feeds back the network output, *yn*, for k or d_d times, whichever is smaller. The last summations of Eq.(3.27) handles the previous values of the plant output, *y*.

Here is an example demonstrating the network prediction:

Consider a network with input nodes consisting of *u(n)* and two previous inputs (i.e., $n_d = 2$), three previous outputs (i.e., $d_d = 3$), two hidden nodes (i.e., $h_d = 2$), and one output node.

Suppose that a 2-step prediction needs to be found, that is, the network needs to predict the output at times *n+1* and *n+2*. Fig 3.9 gives the pictorial representation of how this is achieved.

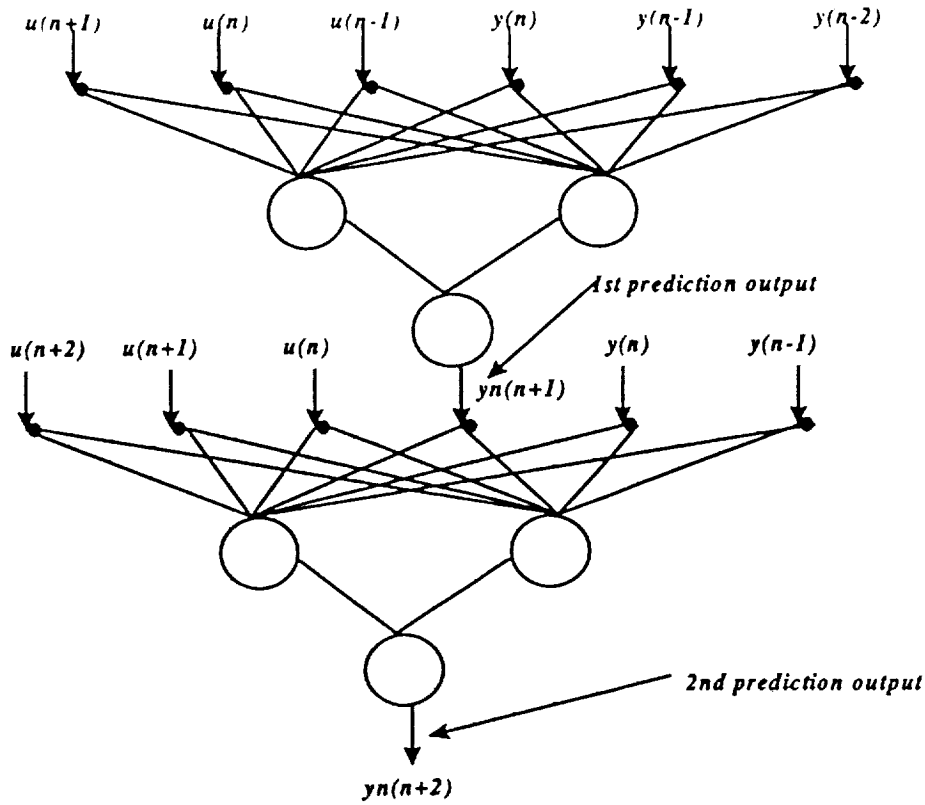


Fig 3.9 Network prediction for k=2

The example above depicts a prediction of the plant output for k=2. To produce the output *yn(n+2)*, inputs *u(n+1)* and *u(n+2)* are needed. The prediction process is started at time *n*, with the initial conditions of $[u(n) \quad u(n-1)]^T$ and $[y(n) \quad y(n-1) \quad y(n-2)]^T$ and the estimated input *u(n+1)*. The output of this process is *yn(n+1)*, which is fed back to the network and the process is repeated to produce the predicted output *yn(n+2)*.

In summary, it was shown in this chapter how a neural network can be used as a predictor for a dynamic system. It will be shown in the next chapter how this predictor configuration of neural network can be effectively used in GPC algorithm for control purposes.

CHAPTER 4 Neural Generalized Predictive Control (NGPC)

In previous two chapters, necessary theoretical background of predictive control and neural networks was given. This chapter gives the analytical and experimental framework of GPC and NGPC for control of dynamic systems.

The first part of this chapter is devoted to the derivation of a stability result for a GPC controller which gives the necessary conditions on the tuning parameters of the controller for closed loop stability. In the latter part, different control architectures are given for neural network-based generalized predictive control of linear and nonlinear systems. For both cases, the control strategies are discussed for two different scenarios: (1) when the plant model is known exactly, and (2) when the plant model is known approximately. For linear systems, the case of parametric uncertainty is addressed in more detail. It is shown that the neural network can be used simultaneously as a 'predictor' and as a adaptor. The function of adaptor is obtained by online training and correction.

4.1 Stability of GPC control law

From the previous discussion in Chapter 2 regarding GPC structure and the control law, it can be seen that the choice of GPC's parameters: N_1, N_u, N_2 and λ play important role in the stability and performance of the closed-loop system. Following theorem gives conditions for selection of GPC tuning parameters, which can ensure the stable tracking behavior of the plant. The performance function used in a simple

performance criteria without penalty on the control input. Note that the conditions in the theorem are only sufficient conditions.

Consider a plant

$$y(k) = \frac{B(q^{-1})}{A(q^{-1})} u(k-1)$$

and performance criteria: $J = \sum_{i=N_1}^{N_2} [y_m(k+i) - y_d(k+i)]^2$, (4.0a)

subject to $\Delta u(k+i-1) = 0$, $N_u \le i \le N_2$. (4.0b)

Theorem 4.1 Consider the cost function given by Eq.(4.0a), except the penalty term related to the control increments, and constraint given by Eq.(4.0b). If the GPC tuning parameters are chosen such that, $N_2 \ge n_A + n_B + d + 1$, $N_1 = n_B + d + 1$, $N_u = n_A + 1$, then, in the absence of disturbances and uncertainties (parametric and modeling), minimization of Eq.(4.0a) under constraint (4.0b) yields a controller that drives output $y(k)$ to follow the reference trajectory given by $\Delta y_d(k+i) = 0$ $(i \ge 1)$ in $n_B + d + 1$ samples.

Proof.

For the sake of convenience it is assumed in the proof that $d = 0$. Further, note that because disturbances and modeling errors are absent, the $\xi(k)$ term in (2.9) will be zero. From Eq. (2.9):

$$\begin{aligned} &A(q^{-1})y(k) = B(q^{-1})u(k-1) \Rightarrow \\ &y(k) = -a_1 y(k-1) - \cdots - a_{n_A} y(k-n_A) + b_0 u(k-1) + \cdots + b_{n_B} u(k-n_B-1) \end{aligned} \quad (4.1)$$

Rewriting Eq.(4.1) with k substituted by $k + n_A + n_B + 2$ and both sides multiplied by $\Delta(= 1 - q^{-1})$ yields:

$$\Delta y(k+n_A+n_B+2) =$$

$$-a_1\Delta y(k+n_A+n_B+1)-\cdots-a_{n_A}\Delta y(k+n_B+2)$$
$$+b_0\Delta u(k+n_A+n_B+1)+\cdots+b_{n_B}\Delta u(k+n_A+1)$$

Now, the plant output settles in n_B+1 samples to a reference trajectory specified by $\Delta y_d(k+i)=0$ if the following conditions are satisfied:

$$y(k+i)=w(k+i) \qquad i=n_B+1 \tag{4.2}$$

$$\Delta y(k+i)=0 \qquad n_B+2\le i\le n_A+n_B+1 \tag{4.3}$$

$$\Delta u(k+i)=0 \qquad n_A+1\le i\le n_A+n_B+1 \tag{4.4}$$

The condition of Eq.(4.4) is satisfied if $N_u=n_A+1$ (remember that, by definition (2.25), $\Delta u(k+i)=0$ for $i\ge N_u$). The conditions of Eq.(4.2) and Eq.(4.3) are satisfied if:

$$y(k+i)=y_d(k+i) \quad n_B+1\le i\le n_A+n_B+1 \tag{4.5}$$

It remains to show that Eq.(4.5) holds if $N_2=n_A+n_B+1$, $N_1=n_B+1$, model is correctly estimated (i.e., $A=\hat{A}, B=\hat{B}$), $d=0$, and $\xi(k)=0$.

Now cost function (2.24) becomes:

$$J=\sum_{i=n_B+1}^{n_A+n_B+1}[\hat{y}(k+i)-y_d(k+i)]^2 \tag{4.6}$$

The minimal value of Eq.(4.6) can be obtained by setting:

$$\hat{y}(k+i)=y_d(k+i) \quad n_B+1\le i\le n_A+n_B+1 \tag{4.7}$$

Now the question to be answered is: is it possible to obtain a controller output sequence such that Eq.(4.7) can actually be satisfied? In order to show that there is a controller output sequence over the control horizon that yields Eq.(4.7), Eq.(4.6) is rewritten in the matrix notation as:

$$J = [\hat{\boldsymbol{y}} - \boldsymbol{y}_d]^T [\hat{\boldsymbol{y}} - \boldsymbol{y}_d] + \lambda \tilde{\boldsymbol{u}}^T \tilde{\boldsymbol{u}}$$

where:

$$\hat{\boldsymbol{y}} = [\hat{y}(k + n_B + 1), \cdots, \hat{y}(k + n_A + n_B + 1)]^T \qquad [\hat{\boldsymbol{y}}] = (n_A + 1) \times 1$$

$$\boldsymbol{y}_d = [y_d(k + n_B + 1), \cdots, y_d(k + n_A + n_B + 1)]^T \qquad [\boldsymbol{y}_d] = (n_A + 1) \times 1$$

$$\tilde{\boldsymbol{u}} = [u(k) - u(k-1), \cdots, u(k + n_A) - u(k + n_A - 1)]^T \qquad [\tilde{\boldsymbol{u}}] = (n_A + 1) \times 1$$

Further, Eq.(2.55) must be satisfied. Now recall the prediction model Eq.(2.22) (with the disturbance being zero):

$$\hat{\boldsymbol{y}} = \boldsymbol{G}\boldsymbol{u} + \boldsymbol{H}\tilde{\boldsymbol{u}} + \boldsymbol{F}\boldsymbol{c} \tag{4.8}$$

where $[\boldsymbol{u}] = N_2 \times 1 = (n_A + n_B + 1) \times 1$. It was shown in Section 2.6 that because Eq.(4.4) must be satisfied, the vector $\boldsymbol{u}$ can be written as:

$$\boldsymbol{u} = \boldsymbol{M}\bar{\boldsymbol{u}} + \boldsymbol{N}\tilde{\boldsymbol{u}}$$

where matrix $\boldsymbol{M}$ and vector $\bar{\boldsymbol{u}}$ is given by:

$$\bar{\boldsymbol{u}} = [u(k), \cdots, u(k + N_u - 1)]^T \qquad [\bar{\boldsymbol{u}}] = N_u \times 1$$

Assuming that Eq.(4.4) holds, Eq.(4.8) can be rewritten as:

$$\hat{\boldsymbol{y}} = \boldsymbol{G}\boldsymbol{M}\bar{\boldsymbol{u}} + \boldsymbol{G}\boldsymbol{N}\tilde{\boldsymbol{u}} + \boldsymbol{H}\tilde{\boldsymbol{u}} + \boldsymbol{F}\boldsymbol{c} \tag{4.9}$$

Now the optimization problem is reduced to solving N_u unknowns $(u(k), \cdots, u(k + N_u - 1))$ from N_u equations. If the matrix inverse that is involved in solving $u(k), \cdots, u(k + N_u - 1)$ from Eq.(4.9) can be calculated, a unique solution exists and hence Eq.(4.7) is obtained when Eq.(4.6) is minimized. By using the assumption that the plant is correctly estimated and that there are no disturbances, Eq.(4.7) becomes:

$$y(k + i) = y_d(k + i) \qquad n_B + 1 \le i \le n_A + n_B + 1$$

and hence, the condition Eq.(4.5) is satisfied. Further, Eq.(4.5) is also satisfied if $N_2 > n_A + n_B + 1$.

The proof with $d \neq 0$ is identical to the proof above in which n_B is replaced by $n_B + d$.

Next, a stability result is presented which considers the cost function used in GPC algorithm which differs from the one used in the earlier result in the term used to penalize control increments over the control horizon.

In order to prove the next result a state-space approach is used for convenience, a discrete time state-space description for a linear system is given by:

$$\mathbf{x}(k+1) = \mathbf{A}\mathbf{x}(k) + \mathbf{b}\Delta u(k)$$
$$y(k) = \mathbf{c}^T\mathbf{x}(k)$$

where $\mathbf{x}(k)$ is the state vector, $\mathbf{A}$ is the system matrix, $\mathbf{b}$ is the control influence matrix, and $\mathbf{c}^T$ is the output matrix. The next theorem gives the conditions for closed-loop stability that need to be satisfied in making choices of GPC horizon parameters. It is assumed that the system is controllable and the transition matrix of the system is non-singular. The assumption of controllability is relaxed to stabilizability in latter stage.

Theorem 4.2 The closed-loop system under finite-horizon GPC control is stable if:

1. the n-state system model $(\mathbf{A},\mathbf{b},\mathbf{c})$ is stabilizable and detectable, and if
2. $N_u = N_1 \geq n$, $N_2 - N_1 \geq n-1$, and $\lambda \to 0$.

Proof

Before the proof of this theorem can be addressed, some additional preliminary results are in order. The result are given in the following five lemmas.

Lemma 1 If the pair $(\mathbf{A},\mathbf{b})$ is completely controllable and $\mathbf{A}$ is non-singular. then for any $N \geq n$ with:

$$\mathbf{k}_0^T = \frac{1}{\lambda}\mathbf{b}^T(\mathbf{A}^T)^N \mathbf{W}_N^{-1}\mathbf{A}^{N+1}$$

where

$$\mathbf{W}_N = \sum \mathbf{A}^i \frac{\mathbf{b}\mathbf{b}^T}{\lambda}(\mathbf{A}^T)^i$$

the feedback control law given by $\Delta u(t) = -\mathbf{k}_0^T\mathbf{x}(t)$ is stabilizing.

Proof of Lemma 1

Consider the system

$$\mathbf{x}(t+1) = \Phi^T\mathbf{x}(t)$$
$$\Phi = \mathbf{A} - \mathbf{A}^{N+1}\frac{\mathbf{b}\mathbf{b}^T}{\lambda}(\mathbf{A}^T)^N\mathbf{W}_N^{-1}$$

and the Lyapunov function $\mathbf{x}^T(t)\mathbf{W}_N\mathbf{x}(t)$. Then it can be shown that

$$\Phi\mathbf{W}_N\Phi^T = -\frac{\mathbf{b}\mathbf{b}^T}{\lambda} - \frac{\mathbf{A}^{N+1}\mathbf{b}}{\lambda}\left[1 - \frac{\mathbf{b}^T(\mathbf{A}^T)^N\mathbf{W}_N^{-1}\mathbf{A}^N\mathbf{b}}{\lambda}\right]\frac{\mathbf{b}^T(\mathbf{A}^T)^{N+1}}{\lambda}$$

From the definition of $\mathbf{W}_N$ and the matrix inversion lemma, the quantity $\Phi\mathbf{W}_N\Phi^T$ is negative semidefinite. It now remains to show that there exists no initial state such that the Lyapunov function is zero for all time or in turn that there exists no state $\mathbf{x}_0$ such that $\mathbf{x}_0^T\Phi^i\mathbf{A}^{N+1}\mathbf{b}$ is zero for all i. From complete controllability, it implies that:

$$\mathbf{x}_0^T\mathbf{A}^{N+1}\left[\mathbf{b},\mathbf{A}\mathbf{b},\mathbf{A}^2\mathbf{b},\ldots,\mathbf{A}^{n-1}\mathbf{b}\right] \neq 0.$$

Recall that complete controllability of the open-loop system $(\mathbf{A},\mathbf{b})$ implies complete controllability of the closed-loop system. This in turn suggests that there exist no state $\mathbf{x}_0$

such that the Lyapunov function can become zero for all time. Φ is therefore a strictly stable matrix and a similarity transformation using the matrix $\mathbf{A}^{N-1}$ completes the proof.

The next set of lemmas removes the restriction of non-singularity of $\mathbf{A}$ and complete controllability.

Lemma 2 If the system $(\mathbf{A},\mathbf{b})$ has a single input and is completely controllable, then the state feedback law of Lemma 1 stabilizes the system irrespective of the non-singularity of matrix $\mathbf{A}$.

Proof of Lemma 2

It is straightforward to obtain a similarity transformation such that

$$\mathbf{S}^{-1}\mathbf{A}\mathbf{S} = \begin{bmatrix} \mathbf{A}_0 & 0 \\ 0 & \mathbf{A}_1 \end{bmatrix}$$

where $\mathbf{A}_0$ is nonsingular and $\mathbf{A}_1$ has only zero eigenvalues. It is then possible to show by direct substitution that the control law of Lemma 1 implies feedback only about the states associated with the non-singular part, as $\mathbf{A}_1$ is nilpotent.

The next Lemma establishes the link between the Kleinman controller and the predictive controller using the state-feedback, where the gains are computed using the appropriate Riccati equation.

Lemma 3 The control law based on iterations of the Riccati equation below and the Kleinman controller are equivalent if:

i. $N_u = N_1 > n$ and $N_2 - N_1 \geq n-1$

ii. $\mathbf{Q}(i) = \mathbf{c}\mathbf{c}^T$ for $i \geq N_1$ and 0 otherwise;

iii. $\lambda(i) =$ very large, for $i \geq N_u = N_1$

$= \lambda = \varepsilon$, vanishingly small, otherwise.

iv. $\mathbf{P}(t+N_2) = \mathbf{c}\mathbf{c}^T$

v. For $i = N_2 - 1$ to 1:

$$\mathbf{P}^*(t+i) = \mathbf{P}(t+i+1) - \frac{\mathbf{P}(t+i+1)\mathbf{b}\mathbf{b}^T\mathbf{P}(t+i+1)}{\lambda(i) + \mathbf{b}^T\mathbf{P}(t+i+1)\mathbf{b}}$$

$$\mathbf{P}(t+i) = \mathbf{Q}(i) + \mathbf{A}^T\mathbf{P}^*(t+i)\mathbf{A}$$

$$\mathbf{k}^T(t) = (\lambda + \mathbf{b}^T\mathbf{P}(t+1)\mathbf{b})^{-1}\mathbf{b}^T\mathbf{P}(t+1)\mathbf{A}$$

$$\Delta u(t) = -\mathbf{k}^T(t)\mathbf{x}(t)$$

Proof of Lemma 3

For the first $N_2 - N_1$ iterations of the algorithm above, the rank of the matrix $\mathbf{P}$ increases progressively from 1 to n (its maximum rank) if $N_2 - N_1 \geq n-1$. This corresponds to the range of points over which the future system errors are included in J. $\mathbf{P}(t+N_1)$ is therefore of full rank. For the next set of iterations $i \leq N_u$ and the Riccati updating equation is:

$$\mathbf{A}\mathbf{P}^{-1}(t+i)\mathbf{A}^T = \mathbf{P}^{-1}(t+i+1) + \frac{\mathbf{b}\mathbf{b}^T}{\lambda}$$

which by the final iteration gives

$$\mathbf{A}^{N_1-1}\mathbf{P}(t+1)^{-1}(\mathbf{A}^T)^{N_1-1} = \mathbf{P}^{-1}(t+N_1) + \sum_{i=0}^{N-2}\mathbf{A}^i\frac{\mathbf{b}\mathbf{b}^T}{\lambda}(\mathbf{A}^T)^i$$

For λ vanishingly small:

$$\mathbf{A}^{N_1-1}\mathbf{P}(t+1)^{-1}(\mathbf{A}^T)^{N_1-1} \approx \sum_{i=0}^{N-2}\mathbf{A}^i\frac{\mathbf{b}\mathbf{b}^T}{\lambda}(\mathbf{A}^T)^i = \mathbf{W}_{N_1-2}$$

It is necessary to show that this leads to the Kleinman law, for which $\mathbf{W}_N$ can be written as:

$$\mathbf{W}_N^{-1} = \left(\mathbf{W}_{N-1} + \frac{\mathbf{A}^N \mathbf{b}\mathbf{b}^T (\mathbf{A}^T)^N}{\lambda} \right)^{-1}$$

Applying the matrix inversion lemma, the Kleinman feedback gain becomes:

$$\mathbf{k}_0^T = (\lambda + \mathbf{b}^T \mathbf{R}^{-1} \mathbf{b})^{-1} \mathbf{b}^T \mathbf{R}^{-1} \mathbf{A}$$

where $\mathbf{W}_{N-1} = \mathbf{A}^N \mathbf{R}^{-1} (\mathbf{A}^T)^N$, $\mathbf{R}^{-1} = \mathbf{P}(t+1)$. It is then easy to see that this is the law based on the Riccati iteration and $N = N_1 - 1$. This implies that a sufficient condition for stability is that $N_1 - 1 \geq n$ or $N_1 > n$.

In the next Lemma, it shows that in the noise-free case there exists a direct mapping between the state-feedback controller and the equivalent input/output controller.

Lemma 4 The input/output GPC control calculation and the optimization using Riccati equations and the state-feedback are equivalent.

Proof of Lemma 4

Consider the GPC calculation for $N_1 = N_2 = N_u = 1$. The feedback control law is given by:

$$\Delta u(t) = \frac{g_1}{g_1^2 + \lambda} (-\hat{y}(t+1 \mid t))$$

which is equivalent to

$$\Delta u(t) = -(\lambda + \mathbf{b}^T \mathbf{c}\mathbf{c}^T \mathbf{b})^{-1} \mathbf{b}^T \mathbf{c}\mathbf{c}^T \mathbf{A}\mathbf{x}(t)$$

by inspection as $\hat{y}(t+1 \mid t) = \mathbf{c}^T \mathbf{A}\mathbf{x}(t)$. For two-stage optimization we have $(N_1 = 1, N_2 = 2, N_u = 2)$

$$\begin{bmatrix} g_1^2+g_2^2+\lambda & g_1g_2 \\ g_1g_2 & g_1^2+\lambda \end{bmatrix}\begin{bmatrix} \Delta u(t) \\ \Delta u(t+1) \end{bmatrix}=\begin{bmatrix} g_1 & g_2 \\ 0 & g_1 \end{bmatrix}\begin{bmatrix} \hat{y}(t+1|t) \\ \hat{y}(t+2|t) \end{bmatrix}.$$

It is straightforward to show that

$$\Delta u(t+1)=-(\lambda+\mathbf{b}^T\mathbf{c}\mathbf{c}^T\mathbf{b})^{-1}\mathbf{b}^T\mathbf{c}\mathbf{c}^T\mathbf{A}\overbrace{(\mathbf{A}\mathbf{x}(t)+\mathbf{b}\Delta u(t))}^{\mathbf{x}(t+1)}$$

and that

$$\Delta u(t)=-(\lambda+\mathbf{b}^T\mathbf{P}\mathbf{b})^{-1}\mathbf{b}^T\mathbf{P}\mathbf{A}\mathbf{x}(t)$$

where

$$\mathbf{P}=\mathbf{c}\mathbf{c}^T+\mathbf{A}^T\mathbf{c}\mathbf{c}^T\mathbf{A}-\mathbf{A}^T\mathbf{c}\mathbf{c}^T\mathbf{b}(\lambda+\mathbf{b}^T\mathbf{c}\mathbf{c}^T\mathbf{b})^{-1}\mathbf{b}^T\mathbf{c}\mathbf{c}^T\mathbf{A}$$

It can be seen that the equations above are none other than the Riccati equations for the two-stage optimization. It is easy to show that the next stages are simply repetitions of the equations above and induction completes the proof.

It can be also be demonstrated that the mapping between the state-feedback gains and the coefficients of the input/output controller is given by:

$$\Delta u(t)=-\mathbf{k}^T\left(\mathrm{O}^{-1}\begin{bmatrix} y(t-n+1) \\ \vdots \\ y(t) \end{bmatrix}-\mathrm{O}^{-1}\begin{bmatrix} g_1 & & 0 \\ \vdots & \ddots & \\ g_n & \cdots & g_1 \end{bmatrix}\begin{bmatrix} \Delta u(t-n) \\ \vdots \\ \Delta u(t-1) \end{bmatrix}+\begin{bmatrix} g_1 & & 0 \\ \vdots & \ddots & \\ g_n & \cdots & g_1 \end{bmatrix}\begin{bmatrix} \Delta u(t-n) \\ \vdots \\ \Delta u(t-1) \end{bmatrix}\right)$$

where O is the observability matrix.

Lemma 5 The stability characteristics of GPC control are unaffected by the presence of common factors in the pole/zero polynomials of the system model.

Proof of Lemma 5

Consider the system

$$A(q^{-1})\Delta y(t)=B(q^{-1})\Delta u(t-1)$$

If the model of the system is given by polynomials with common factor $\alpha(q^{-1})$:

$$A(q^{-1})\alpha(q^{-1})\Delta y(t) = B(q^{-1})\alpha(q^{-1})\Delta u(t-1)$$

then the Diophantine equation for j-step-ahead predictor becomes:

$$1 = E_c A\alpha\Delta + q^{-j}F_c$$

Comparing this with the Diophantine identity without the common factor $\alpha(q^{-1})$:

$$1 = EA\Delta + q^{-j}F$$
$$\alpha E_c = E - q^{-j}r$$
$$F_c = rA\Delta + F$$

and the output at time $t+j$ can be written as:

$$y(t+j) = EB\Delta u(t+j-1) + Fy(t)$$

Note that the terms in the predictor equation are the same as for the case of no commom factors and the g-parameters will also remain the same as before. This in turn implies that there will be no singularity in the GPC control calculations in the presence of common factors and that the stability characteristics are unaffected.

Back to proof of Theorem 4.2

The above Lemmas combined provide the proof of Theorem 3.2. Hence for a completely controllable and observable plant, GPC control law provides stability if $N_u = N_1 \geq n, N_2 - N_1 \geq n-1$ and λ is vanishingly small.

4.2 Neural Generalized Predictive Control (NGPC)

4.2.1 Introduction

To use GPC framework for the real time control of non-linear plants, a nonlinear black box estimator is needed. This 'black box' model should be able to estimate the

nonlinear dynamics on line. In addition, the cost function should be integrated with the black box model predictor for improved real time performance.

As shown in Chapter 2, GPC algorithm relies on the model of the plant and, in some cases, of the disturbances. Even though there are some techniques for modeling disturbances and uncertainty, it is not possible to obtain an exact model, which can predict the plant output accurately. In general, even the best available modeling technique can yield the plant model that has uncertainties and errors. This is especially true of nonlinear systems where often times modeling is an extremely difficult task. GPC, being model-based control technique, requires a model of the plant for output prediction. In case of nonlinear systems, availability of such a model could be a problem. Use of neural network can help solve this problem. Neural network can be used to model linear or nonlinear system. In particular, for nonlinear systems, whose model is not known, neural network can be used as online identification tool in concurrence with its function as a predictor for GPC.

As it is shown in Chapter 3, multilayer neural networks offer the properties and capabilities that will meet the controller requirement discussed above. Especially, the properties of neural networks that are attractive in GPC control are: (1) nonlinear mapping, (2) online learning ability, and (3) online adaptivity. Using neural networks as a predictor in GPC enriches the intelligence of the controller behavior, which results in the increased robustness in the system's stability and performance.

Next, an NGPC controller architecture is presented which is used to obtain robust control of uncertain dynamic systems.

4.2.2 Basic NGPC algorithm

In this section, a basic NGPC architecture is presented. The NGPC algorithm is similar to the GPC algorithm where linear predictor model is replaced with a neural network model. To use a neural network as a plant predictor, a training procedure needs to be established. In the controller architecture presented, neural network is integrated with the cost function minimization routine. The cost function to be minimized is the same basic cost function used in conventional GPC algorithm.

The NGPC framework consists of four components, the plant to be controlled, a reference model that specifies the desired performance of the plant, a neural network that models the plant, and the Cost Function Minimization (CFM) algorithm that determines a sequence of future inputs needed to obtain a desired plant performance. The NGPC algorithm consists of the CFM block and the neural network block. Fig 4.1 shows the block diagram of the NGPC system.

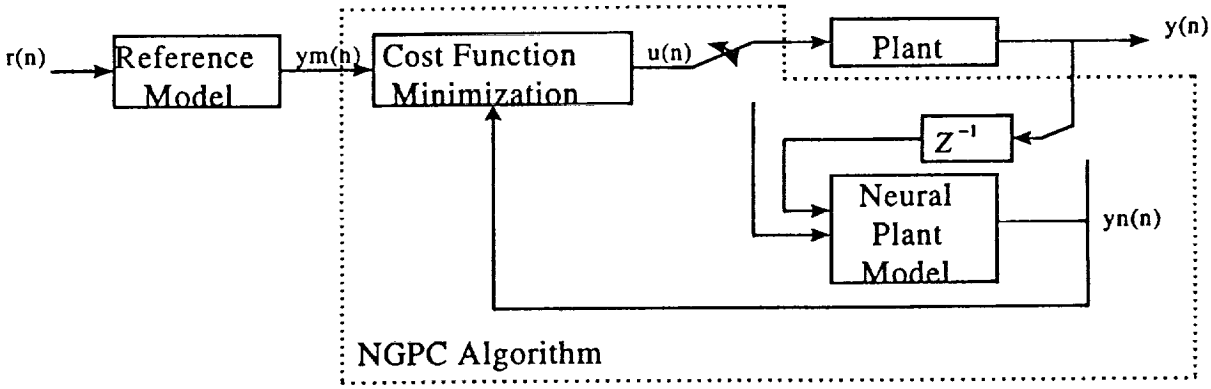


Fig 4.1 Block diagram of NGPC

The NGPC algorithm works as follows: reference input $r(n)$ is presented to the reference model. This reference model produces a tracking reference signal, $ym(n)$,

which is used as the desired output and is passed on to the CFM block. The CFM algorithm produces a control signal, which could be sent to the plant as well as the model of the plant as the input signal. The double-pole double-throw switch is set alternatively to feed the plant or the neural network model. The switch is set to the plant when the CFM algorithm has converged to the best input sequence, $u(n)$, which minimizes the specified cost function. Between the samples the switch is set to the neural network plant model. The CFM algorithm uses this model for prediction of the future outputs which are needed in the calculation of the future control input, $u(n+1)$. Once the cost function is minimized and a sequence of future control input is obtained the first input is passed on to the plant and the process is repeated for the next time constant.

4.2.3 NGPC cost function with actuator constraints

One of the advantages of GPC or NGPC is that the real plant constraints can be easily taken into account in the cost function. One commonly occurring actuator constraint is the saturation which limits the amplitude of the control signal. Adding the input constraint function to the basic cost function results in:

$$J = \sum_{j=N_1}^{N_2}[ym(n+j) - yn(n+j)]^2 + \sum_{j=1}^{N_2}\lambda(j)[\Delta u(n+j)]^2$$

$$+ \sum_{j=1}^{N_2}[\frac{s}{u(n+j)+r/2-b} + \frac{s}{r/2+b-u(n+j)} - \frac{4}{r}]$$

where

$\Delta u(n+j) = u(n+j) - u(n+j-1)$,

N_1 is the minimum costing horizon,

N_2 is the maximum costing horizon,

$\lambda(j)$ is a control-weighting sequence,

s is the sharpness of the constraint function,

r is the range for the constraint, and

b is the offset of the range for the constraint.

N_1 specifies the dead time of the plant and N_2 is the horizon. This cost function has three parts. The first part minimizes the error between the model, $ym(n)$, and the neural network, $yn(n)$. The second part minimizes the rate of change for the inputs, $u(n+j)$, with λ as the weighting factor. λ can be tuned to change the penalty on the control input. The input constraint guarantees that the control input will not saturate the actuator. The parameters s, r and b characterize the sharpness, range, and offset of the input constraint functions, respectively. The sharpness, s, controls the shape of the constraint function.

The algorithm used to minimize the cost function is the key to the real-time control. Several algorithms are available, however, most of them have been proven to have inadequate speed for the real time application. The next section develops a real-time minimization procedure based on the Newton-Raphson optimization algorithm, which has been found to give the fastest convergence over other existing methods.

4.3 Cost Function Minimization (CFM)

In Section (3.1), the GPC algorithm has been described with the basic cost function (3.2). When the plant model is represented by a neural network, any linear minimization technique will not work because of the inherent non-linear properties of the

neural network model. In such a case, the use of a non-linear iterative solution is necessary. The existing iterative techniques can be divided into two groups: gradient-based techniques and nongradient-based techniques. Only one of the gradient-based techniques has quadratic converging algorithm. This algorithm, namely, Newton-Raphson, is the fast converging algorithm when measured in terms of iterations. This algorithm could be made faster if the parameters are computationally inexpensive to calculate. It is critical for real-time control that the minimization is done efficiently. Presented here is the Newton-Raphson solution for the augmented cost function minimization problem that is about a order of magnitude more efficient than a first order gradient-based algorithm.

4.3.1 Brief review of Newton-Raphson algorithm

Newton-Raphson is a quadratically converging optimization technique for solving nonlinear equations of the form $f(x) = 0$ where $f(x)$ is nonlinear differentiable function in the space of $x \in R^n$. The derivation of the Newton-Raphson algorithm is as follows:

It starts with initial guess for the solution x, say x_n. Then a Taylor series expansion of $f(x_{n+1})$ is obtained:

$$f(x_{n+1}) \cong f(x_n) + f'(x_n)(\Delta x) + \frac{1}{2!} f''(x_n)(\Delta x)^2 + \frac{1}{3!} f'''(x_n)(\Delta x)^3 + \cdots \tag{4.10}$$

where

$$\Delta x = x_{n+1} - x_n \tag{4.11}$$

Taking the first two terms, $f(x_{n-1}) \cong f(x_n) + f'(x_n)(\Delta x)$, and assuming that $f(x_{n+1}) = 0$, solve for Δx and get $\Delta x = -\dfrac{f(x_n)}{f'(x_n)}$. Substituting this into Eq.(4.11) yields the solution $x_{n+1} = x_n - \dfrac{f(x_n)}{f'(x_n)}$.

To find the local extremum of a function, say $g(x)$, the first derivative of $g(x)$ is taken with respect to x and set it to zero, i.e., $\dfrac{dg(x)}{dx} = 0$. Thus the update rule for the extremum is $x_{n+1} = x_n - \dfrac{g'(x_n)}{g''(x_n)}$.

4.3.2 CFM algorithm

Minimizing the cost function J with respect to the sequence $[u(n+1) \quad u(n+2) \quad \cdots \quad u(n+N_2)]^T$, denoted by U, is accomplished by setting the Jacobian to zero and solving for U. Using Newton-Raphson method, J is minimized iteratively to determine the optimum U. An iterative process yields intermediate values for J which are denoted by $J(k)$. For each iteration of $J(k)$, an intermediate control input vector is also generated and is denoted as:

$$U(k) = \begin{bmatrix} u(n+1) \\ u(n+2) \\ \vdots \\ u(n+N_2) \end{bmatrix}, \quad k = 1,2,\ldots,\#\text{ of iterations.}$$

The Newton-Raphson update rule for $U(k+1)$ is:

$$U(k+1) = U(k) - \left(\frac{\partial^2 J}{\partial U^2(k)}\right)^{-1} \frac{\partial J}{\partial U}(k) \tag{4.12}$$

where the Jacobian is denoted as

$$\frac{\partial J}{\partial U}(k) = \begin{bmatrix} \frac{\partial J}{\partial u(n+1)} \\ \vdots \\ \frac{\partial J}{\partial u(n+N_2)} \end{bmatrix}$$

and the Hessian is denoted as

$$\frac{\partial^2 J}{\partial U^2}(k) = \begin{bmatrix} \frac{\partial^2 J}{\partial u(n+1)^2} & \cdots & \frac{\partial^2 J}{\partial u(n+N_2)\partial u(n+1)} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 J}{\partial u(n+N_2)\partial u(n+1)} & \cdots & \frac{\partial^2 J}{\partial u(n+N_2)^2} \end{bmatrix}$$

To use LU decomposition to solve for the control input vector $U(k+1)$, Eq.(4.12) is rewritten in the form of a system of linear equations, $Ax = b$. This results in :

$$\frac{\partial^2 J}{\partial U^2}(k)(U(k+1) - U(k)) = -\frac{\partial J}{\partial U}(k) \tag{4.13}$$

where

$$\frac{\partial^2 J}{\partial U^2}(k) = A,$$
$$-\frac{\partial J}{\partial U}(k) = -b,$$
$$U(k+1) - U(k) = x$$

After x is calculated, $U(k+1)$ is solved by evaluating $U(k+1) = x + U(k)$. This procedure is repeated until the percent change in each element of $U(k+1)$ is less than some ξ . When solving for x, calculation of each element of the Jacobian and Hessian is needed for each Newton-Raphson iteration. The h^{th} element of the Jacobian is:

$$\frac{\partial J}{\partial u(n+h)} = -2\sum_{j=N_1}^{N_2}[ym(n+j) - yn(n+j)] + 2\sum_{j=1}^{N_2}\lambda(j)[\Delta u(n+j)]\frac{\partial \Delta u(n+j)}{\partial u(n+h)}$$

$$+\sum_{j=1}^{N_2}\delta(h,j)\left[\frac{-s}{(u(n+j)+r/2-b)^2} + \frac{s}{(r/2+b-u(n+j))^2}\right],\ h=1,\ldots,N_2$$

The term $\frac{\partial \Delta u(n+j)}{\partial u(n+h)}$ when expanded and evaluated can be rewritten in terms of the Kronecker delta function[1] as,

$$\frac{\partial u(n+j)}{\partial u(n+h)} - \frac{\partial u(n+j-1)}{\partial u(n+h)} = \delta(h,j) - \delta(h,j-1)$$

The m^{th}, h^{th} element of the Hessian is:

$$\frac{\partial^2 J}{\partial u(n+m)\partial u(n+h)} = 2\sum_{j=N_1}^{N_2}\left\{\frac{\partial yn(n+j)}{\partial u(n+m)}\frac{\partial yn(n+j)}{\partial u(n+h)} - \frac{\partial yn(n+j)}{\partial u(n+m)}\frac{\partial yn(n+j)}{\partial u(n+h)}[ym(n+j) - yn(n+j)]\right\}$$

$$+2\sum_{j=1}^{N_2}\lambda(j)\left\{\frac{\partial yn(n+j)}{\partial u(n+m)}\frac{\partial yn(n+j)}{\partial u(n+h)} + \Delta u(n+j)\frac{\partial yn(n+j)}{\partial u(n+m)}\frac{\partial yn(n+j)}{\partial u(n+h)}\right\}$$

$$+\sum_{j=1}^{N_2}\delta(h,j)\delta(m,j)\left[\frac{2s}{(u(n+j)+r/2-b)^3} + \frac{2s}{(r/2+b-u(n+j))^3}\right],$$

$$h=1,\ldots,N_2$$
$$m=1,\ldots N_2$$

Again, using Delta function, yields:

$$\frac{\partial \Delta u(n+j)}{\partial u(n+h)}\cdot\frac{\partial \Delta u(n+j)}{\partial u(n+m)} = [\delta(h,j) - \delta(h,j-1)][\delta(m,j) - \delta(m,j-1)]$$

The $\frac{\partial \Delta u(n+j)}{\partial u(n+h)}\frac{\partial \Delta u(n+j)}{\partial u(n+m)}$ always evaluates to zero.

---

[1] Here the Kronecker Delta function is defined as $\delta(h,j) = \begin{cases} 1, & if \quad h = j \\ 0, & if \quad h \neq j \end{cases}$

To evaluate the Jacobian and the Hessian, the first and second derivatives of the network outputs with respect to the control input vector are needed. The elements of the Jacobian are obtained by differentiating $yn(n+k)$, with respect to $u(n+h)$, i.e.,

$$\frac{\partial yn(n+k)}{\partial u(n+h)} = \sum_{j=1}^{h_d} W_j \frac{\partial \varphi_j(v_j(n+k))}{\partial u(n+h)} \tag{4.14}$$

Applying the chain rule to the term $\frac{\partial \varphi_j(v_j(n+k))}{\partial u(n+h)}$ results in:

$$\frac{\partial \varphi_j(v_j(n+k))}{\partial u(n+h)} = \frac{\partial \varphi_j(v_j(n+k))}{\partial v_j(n+k)} \frac{\partial v_j(n+k)}{\partial u(n+h)} \tag{4.15}$$

where $\frac{\partial \varphi_j(v_j(n+k))}{\partial v_j(n+k)}$ is the derivative of the output functions and

$$\frac{\partial v_j(n+k)}{\partial u(n+h)} = \sum_{i=0}^{n_d} W_{j,i+1}\delta(k-i,h) + \sum_{i=1}^{\min(k,d_d)} W_{j,i+n_d+1} \frac{\partial yn(n+k-i)}{\partial u(n+h)} \delta_1(k-i,1) \tag{4.16}$$

In Eq.(4.16), the step function, δ_1, was introduced to the second summation. This was added to point out that this summation evaluates to zero for $k-i<1$, thus the partial derivative does not need to be calculated for this condition. The elements of the Hessian are obtained by differentiating Eq.(4.14), Eq.(4.15) and Eq.(4.16) with respect to $u(n+m)$, resulting in:

$$\frac{\partial^2 yn(n+k)}{\partial u(n+h)\partial u(n+m)} = \sum_{j=1}^{h_d} W_j \frac{\partial^2 \varphi_j(v_j(n+k))}{\partial u(n+h)\partial u(n+m)} \tag{4.17}$$

$$\frac{\partial^2 \varphi_j(v_j(n+k))}{\partial u(n+h)\partial u(n+m)} = \frac{\partial \varphi_j(v_j(n+k))}{v_j(n+k)} \frac{\partial^2 v_j(n+k)}{\partial u(n+h)\partial u(n+m)}$$

$$+ \frac{\partial^2 \varphi_j(v_j(n+k))}{v_j(n+k)^2} \frac{\partial v_j(n+k)}{\partial u(n+h)} \frac{\partial v_j(n+k)}{\partial u(n+m)} \tag{4.18}$$

CHAPTER 5 Simulation and real time control experiment

5.1 Introduction

In the previous chapters, background of Neural Network and Generalized Predictive Control was given. The last chapter gave introduction to the control architecture for Neural network-based Generalized Predictive Control methodology. A real time implementation of NGPC was also discussed. This chapter is devoted to the simulation and experimental validation of the robust control strategies using NGPC techniques developed in Chapter 4. For simulation purposes, two example systems are chosen. An experimental validation is obtained for one of these systems for which laboratory setup was possible. The simulation and experimental results show that an NGPC controller framework has great potential for application in robust control of unstable and non-minimum phase linear and nonlinear systems. It is shown that NGPC is very robust to parametric uncertainties and unknown disturbances.

5.2 Pitch axis control of a fighter aircraft

As a first example, longitudinal model of an F-16 fighter aircraft is considered [17]. The problem addressed is that of controlling the pitch rate of the aircraft in the presence of parametric uncertainties. The design model is obtained from a 13^{th} order transfer function model derived for straight and level flight conditions. The poles and zeros of this 13-state transfer function show that cancellation of at least the first six digits has occurred, so the final simplified transfer function can be expected to be a good approximation of the original 13-state transfer function. The first three poles are at the origin and represent an integration of the velocity components that lead to the north, east,

and directional states. These poles are canceled by the zeros at the origin. The poles corresponding to dutch roll, roll mode, and spiral mode are also canceled by the zeros. The altitude pole is not exactly at the origin, and is therefore not canceled precisely because it is coupled to the longitudinal dynamics. The engine pole is canceled exactly because the engine-lag model is driven only by the throttle input. The remaining four poles (phugoid and short-period modes) and three zeros yield the following transfer function:

$$\frac{q}{\delta_e} = \frac{-10.45s(s+0.9871)(s+0.02179)}{(s+1.204 \pm j1.492)(s+0.007654 \pm j0.07812)} \frac{\text{deg/sec}}{\text{deg}} \qquad (5.1)$$

The elevator to pitch-rate transfer function has a dc gain of zero (because of the zero at the origin), indicating that a constant elevator deflection will not sustain a steady pitch rate. If the phugoid poles are canceled with the zero at the origin and the zero at s= -0.02, a *short-period approximation* of the transfer function is obtained:

$$\frac{q}{\delta_e} = \frac{-10.45(s+0.9871)}{s+1.204 \pm j1.492} \frac{\text{deg/sec}}{\text{deg}} \qquad (5.2)$$

Uncertainty modeling:

For simulation studies, the short-period approximation (5.2) will be used as the design model. The difference between the full order longitudinal model and the short-period approximation can be treated as model uncertainty. This type of uncertainty, which represents unmodeled part of the dynamics, can be best represented as an additive uncertainty. Another kind of uncertainty, which arises due to incorrect knowledge of the system parameters, is called parametric uncertainty. In the simulation experiments it will

be assumed that the short-period approximation model is fairly accurate representation of the longitudinal dynamics except that the short-period frequencies could vary between a known range. The uncertainty of this type can be characterized as parametric uncertainty. As explained in Chapter 4, such uncertainty can be dealt with in NGPC architecture by using the methodology given in Section 4.3.

The problem addressed is that of robust tracking of a reference pitch rate signal in the presence of uncertainty in the short-period frequency. It is assumed that the short-period frequency can vary between $\pm 30\%$ from its nominal value. NGPC controller architecture used for this problem is described below.

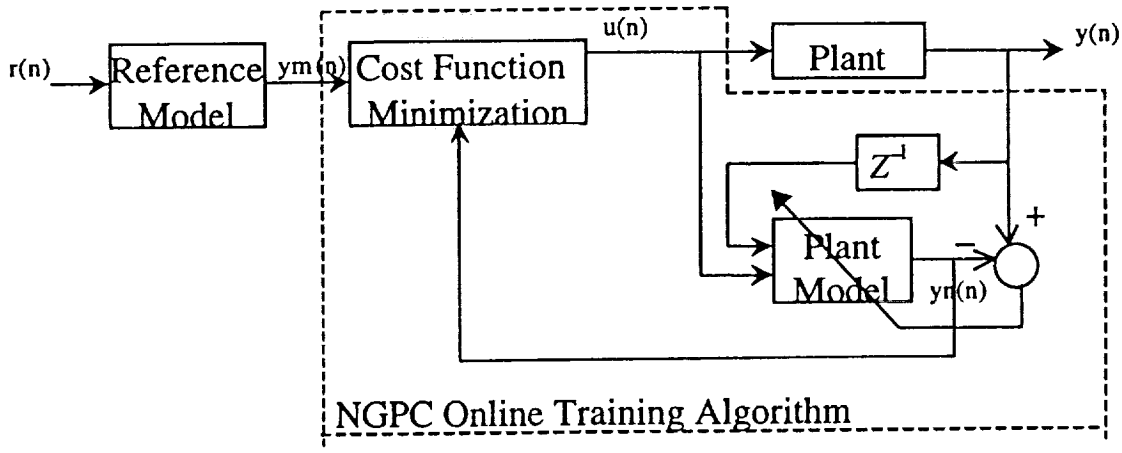
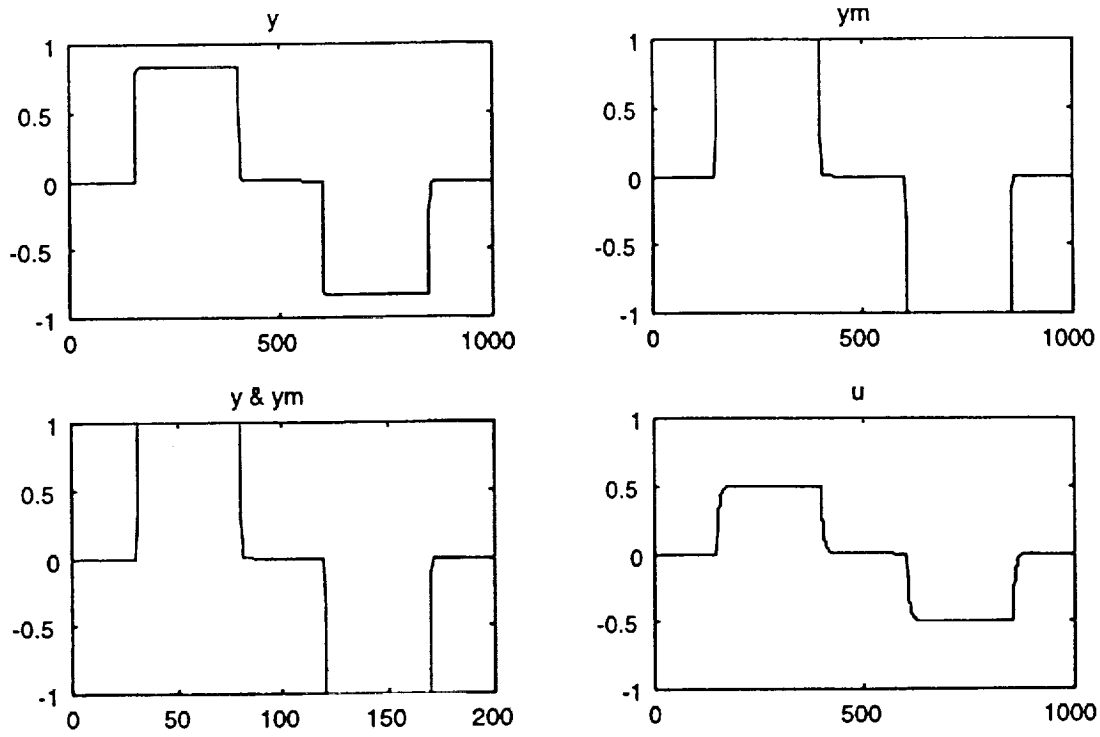


Fig 5.1 NGPC online training block diagram

Controller structure

The block diagram of the closed loop system is as shown in Fig 5.1. The neural network used for prediction is configured such that it embeds the nominal plant model by using fixed weights on the appropriate hidden layer connections (refer Fig 4.2). The activation function of the first hidden node is linear with unity slope. The second hidden

setting of GPC tuning parameters to the values recommended by Theorem 4.2 assures the closed loop stability. The stability is robust to parametric uncertainties.

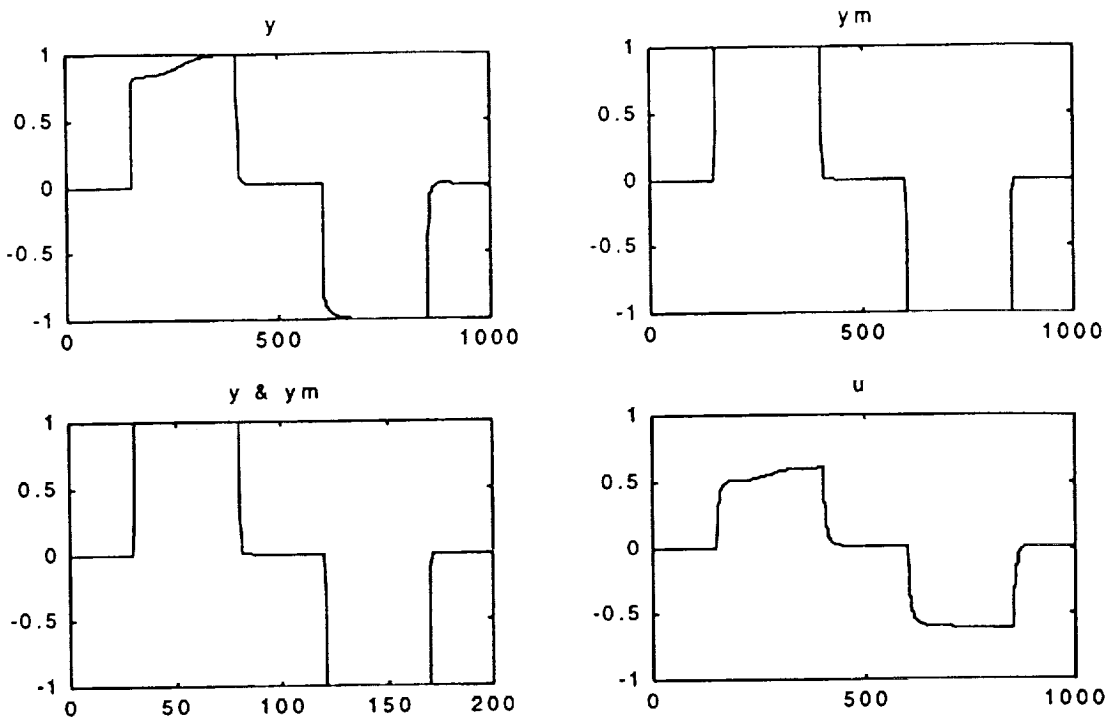


(x axis: time in samples (sampling frequency: 20 Hz))

Fig 5.3 Case b): +30% uncertainty in ω_n of the plant, no on-line adaptation

Case c): +30% uncertainty in ω_n, online adaptation;

For this case, the set-up is similar to case (b) except that the neural network model predictor is allowed to learn the plant dynamics online to compensates for the uncertainty. That means, the weightings on connections to and from second hidden node are allowed to adapt online through back-propagation learning. This has an effect of neural-network model asymptotically converging to the actual (correct) plant model and thereby improving steady-state performance. Fig 5.4 clearly shows this behavior. Note that it takes only half a cycle of the doublet for model to adapt.



(x axis: time in samples (sampling frequency: 20 Hz))

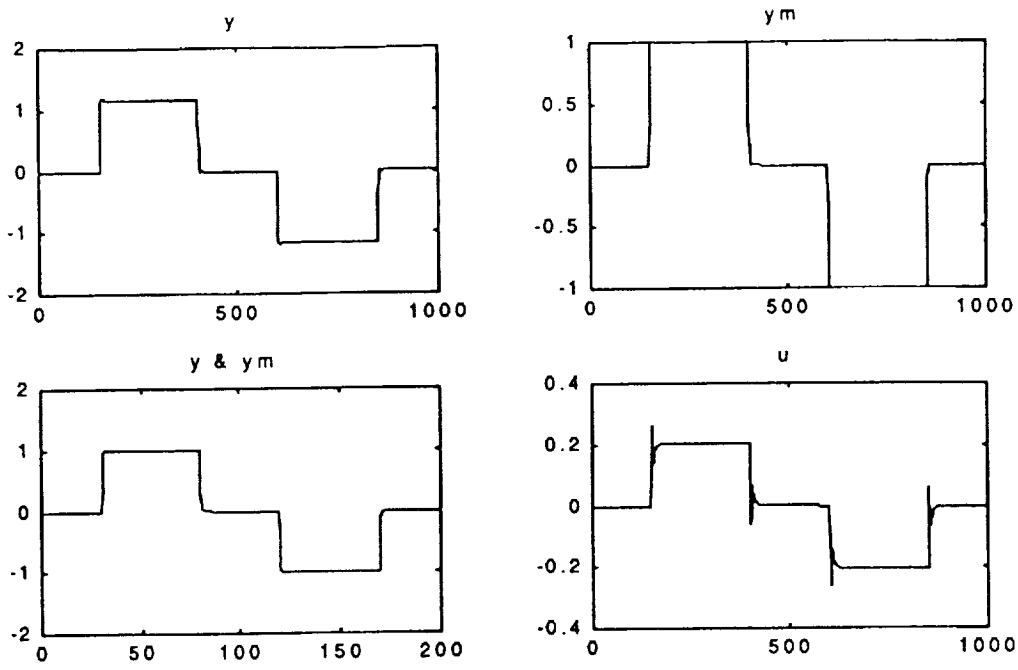
Fig 5.4 Case c): +30% uncertainty in ω_n of the plant, on-line training is on

Case d): -30% uncertainty in ω_n, no online adaptation;

This case is similar to case (b) except that the uncertainty in short-period frequency is taken to be –39%. A similar result as case (b) can be observed in Fig 5.5.

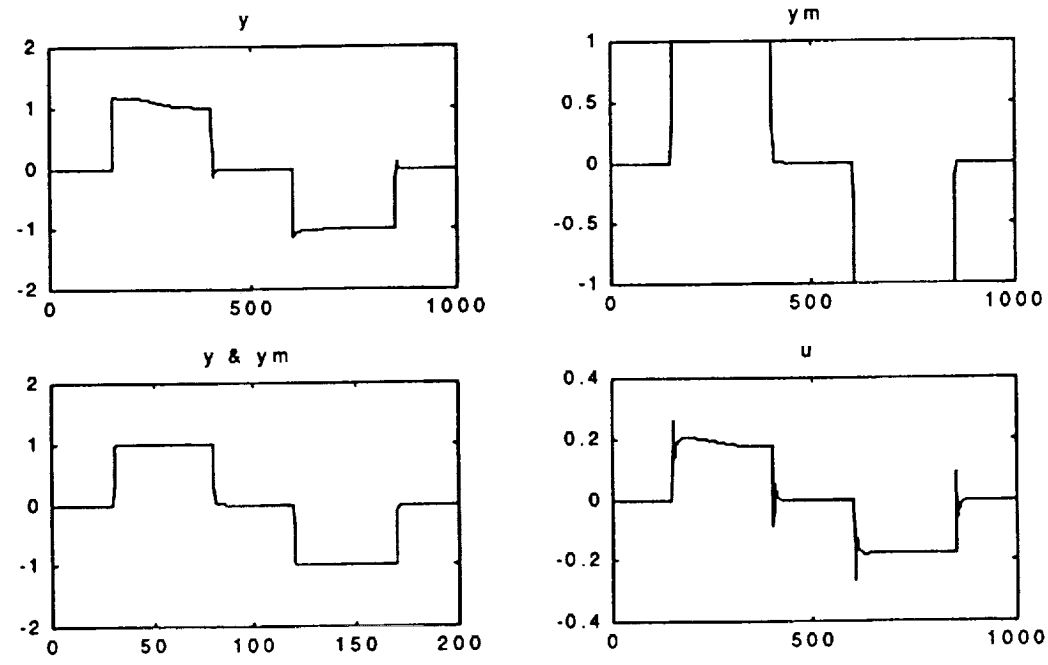
Case e): -30% uncertainty in ω_n, online adaptation;

This case is similar to case (c) except that the uncertainty is –30% in the short-period frequency as opposed to +30%. As expected, the simulation results for this case parallel to these from case (c) and are given in Fig 5.6.



(x axis: time in samples (sampling frequency: 20 Hz))

Fig 5.5 Case d): -30% uncertainty in ω_n of the plant, no on-line Adaptation



(x axis: time in samples (sampling frequency: 20 Hz))

Fig 5.6 Case e): -30% uncertainty in ω_n of plant, on-line training is on

For all the simulations shown in Fig 5.2 through 5.6 the choice of the NGPC tuning parameters was N1= 2, Nu= 2, and N2= 3. Another way to improve performance, in addition to online adaptation, is to tune GPC parameters such as N1, N2 and Nu. It is observed that increasing prediction horizon N2 can yield better performance in many applications. In this example also it was observed that increasing N2 led to better performance in general (see Fig 5.7 through 5.11) for all cases except case (c).

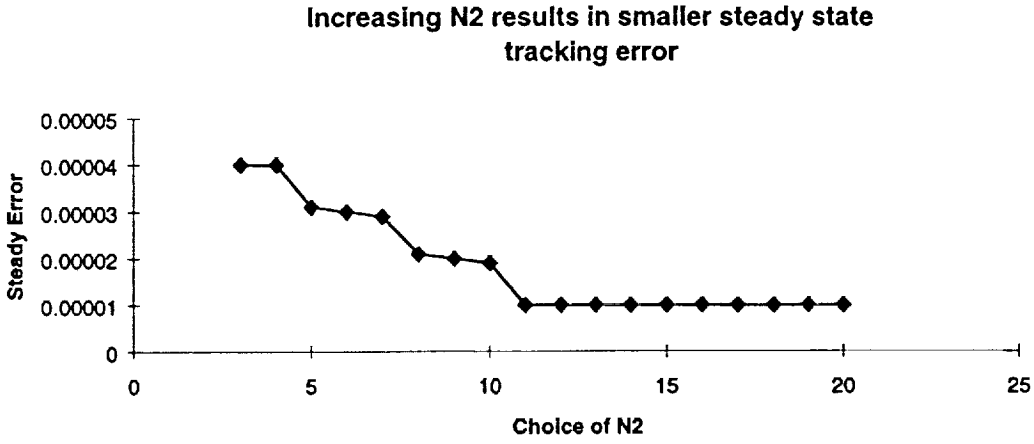


Fig 5.7 Steady-state error performance: $\Delta\omega_n = 0\%$, no adaptation

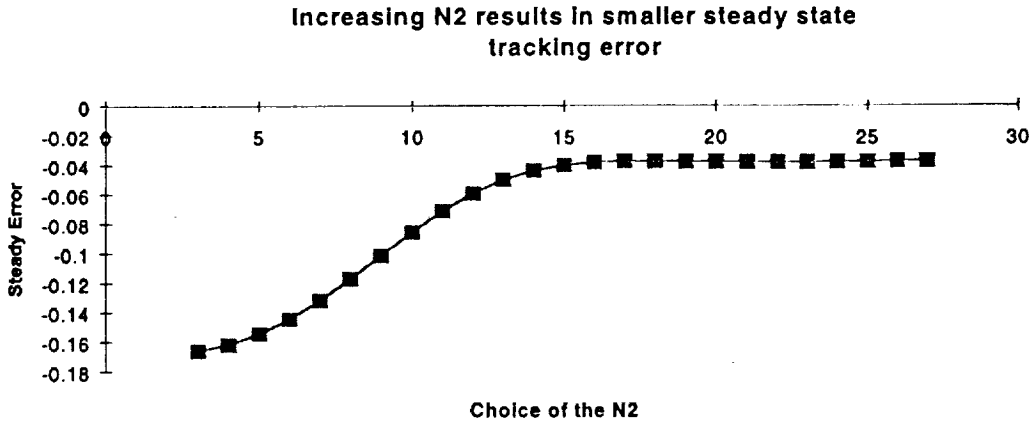


Fig 5.8 Steady-state error performance: $\Delta\omega_n = +30\%$, no adaptation

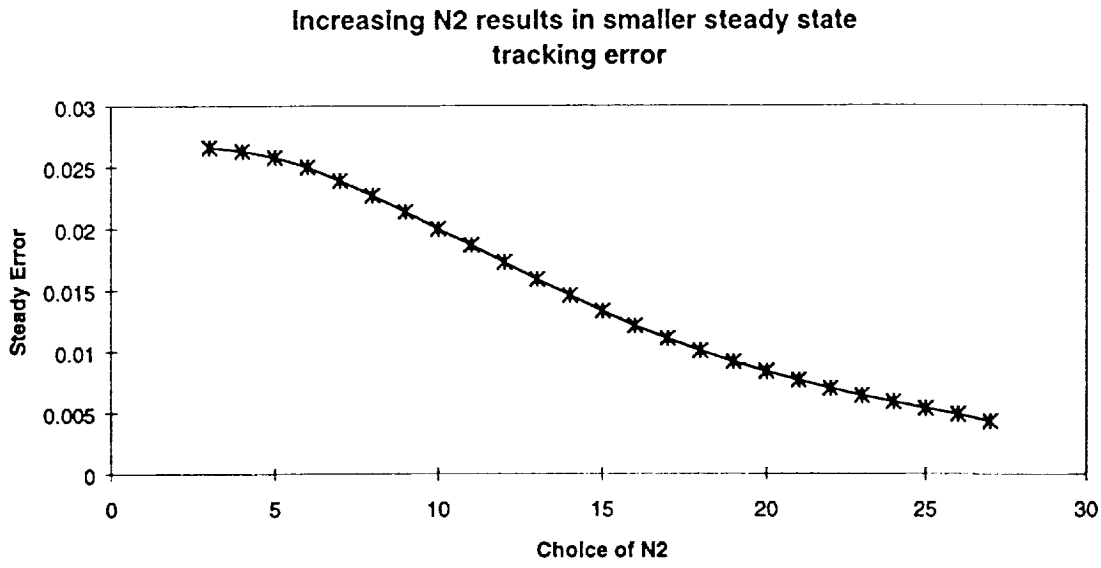
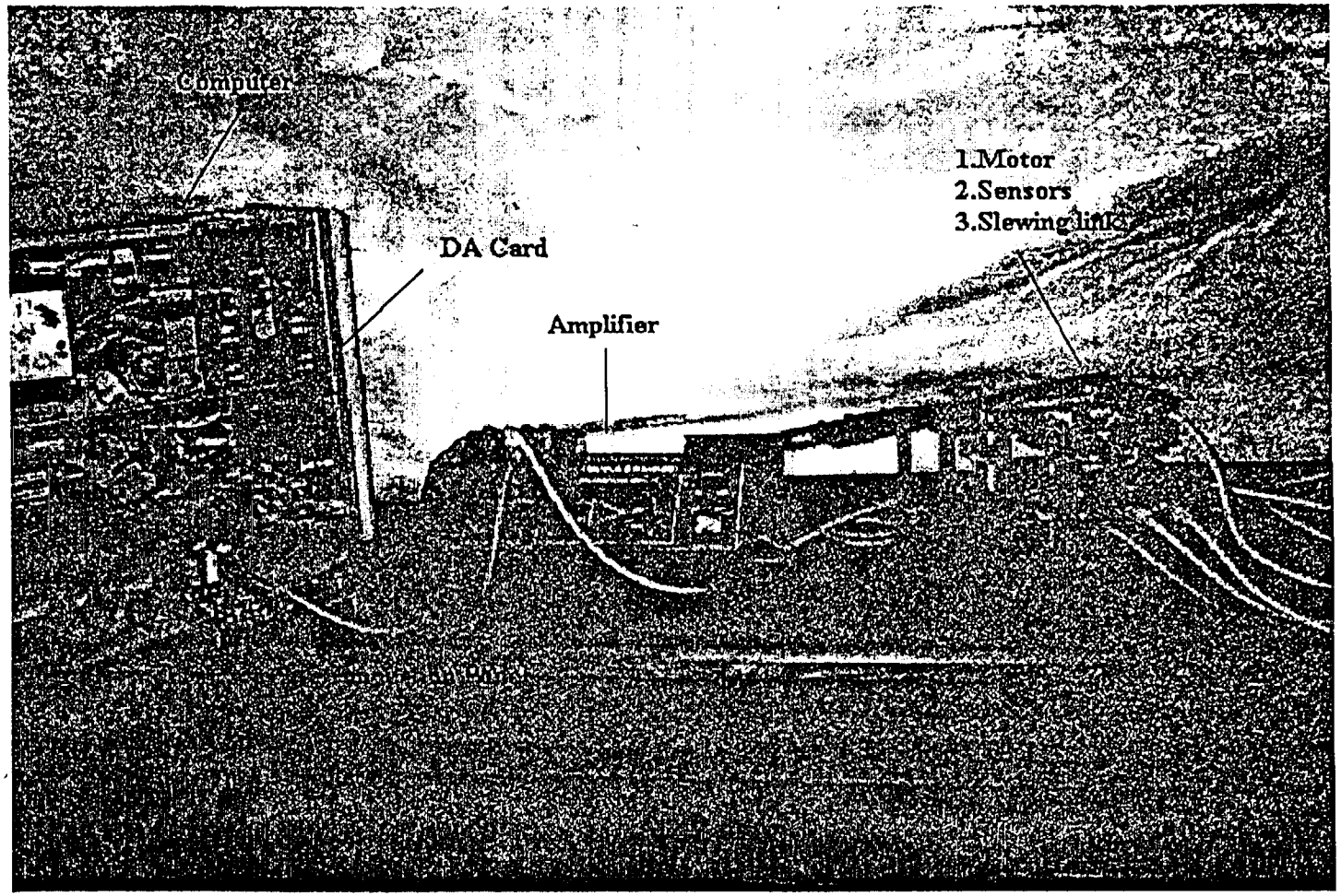


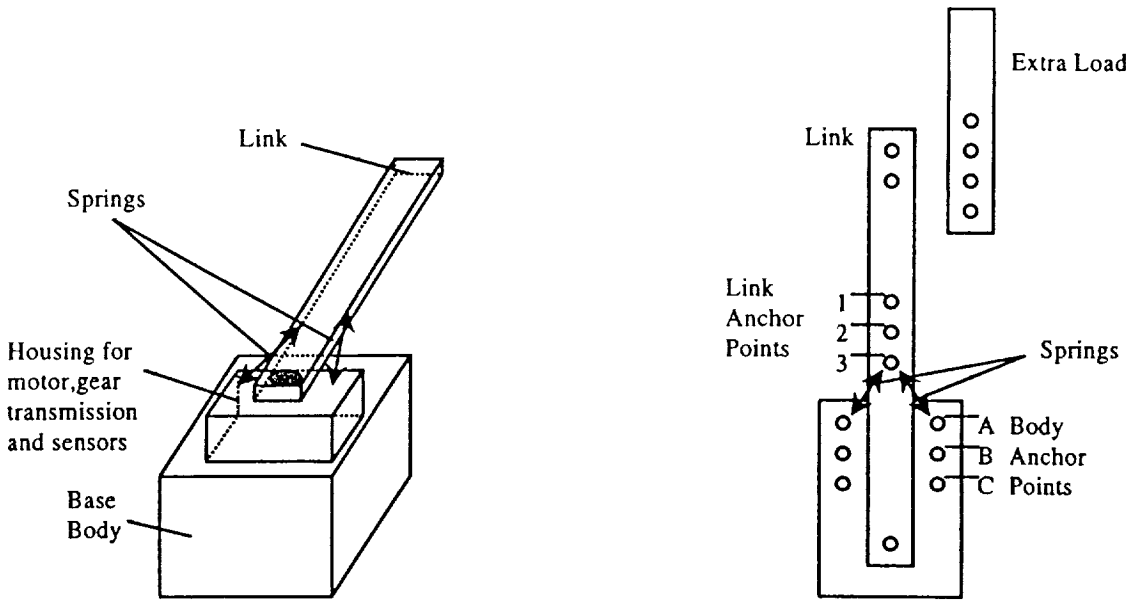
Fig 5.11 Steady-state error performance: $\Delta\omega_n = -30\%$, online adaptation

5.3 Control of slewing link with flexible joint

As a second example, the control of slewing link with flexible joint is considered. Fig 5.12 and Fig 5.13 show the schematic of the apparatus and the geometry of the system, respectively. The apparatus consists of a rotating rigid link with flexible joint (See Picture 5.1). The link is housed on a rectangular box and is driven by a DC motor. The length of the link can be adjusted using anchor points. The joint flexibility is achieved via two identical springs that are anchored to the base body at one end and to the link at the other end. By changing the springs or the anchor points, joint stiffness can be varied. A small extension can be attached at the end of the main link to change the link's inertia properties. The input to the system is the voltage applied to the motor that drives the joint. The output is the angular displacement of the link with respect to the inertial frame.



Picture 5.1 Slewing link with flexible joint



Rotary Flexible Joint Module

Fig 5.12 Rotary flexible joint

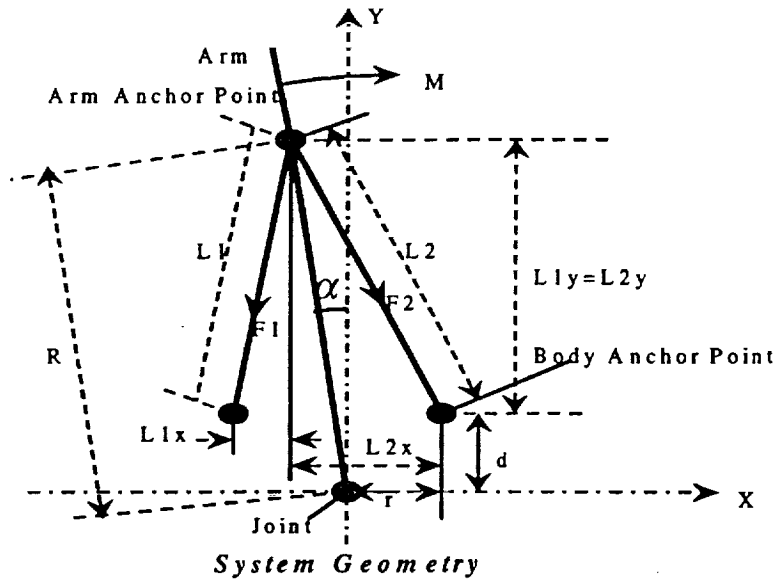


Fig 5.13 System geometry

The rest of this chapter is arranged as follows:

In Section 5.4, derivation of a mathematical model is given for the slewing link with flexible joint. A linearized state space as well as transfer function model is obtained. In Section 5.5, an experimental set-up is given and different control problems are defined. In Section 5.6, simulation as well as experimental results are given. Finally, in Section 5.7, discussion of the results and concluding remarks are given.

5.4 Mathematical modeling of a rotary flexible joint system

5.4.1 Derivation of the joint stiffness

Consider the system geometry shown in Fig 5.12. The link is displaced from the zero angular position such that the spring #1 is stretched to length $L1$ and the spring #2 is stretched to length $L2$. From Fig 5.13, we have the following relationships:

$$L_{1x} = r - R\sin(\theta)$$
$$L_{2x} = r + R\sin(\theta)$$
$$L_{1y} = L_{2y} = R\cos(\theta) - d$$
$$L_1 = \sqrt{(L_{1x}^2 + L_{1y}^2)}$$
$$L_2 = \sqrt{(L_{2x}^2 + L_{2y}^2)}$$

where

r is the base body anchor point along X-axis,

R is the link anchor point along Y-axis,

d is the base body anchor point along Y axis.

Let L be the initial unstretched length of the spring, then the spring force generated by the extension of the spring to length L_i is given by:

$$F_i = K(L_i - L) + F_r$$

where i=1 for spring 1 and i=2 for spring 2, and F_r is the restoring force on each spring. This means the spring will not stretch unless the force F_r is applied. Note that it is assumed that the two springs have identical stiffness K and restoring force F_r.

The force generated by each spring can be decomposed into their x and y components as shown in Fig 5.12:

$$F_{1x} = F_1 L_{1x} / L_1$$
$$F_{1y} = F_1 L_{1y} / L_1$$
$$F_{2x} = F_2 L_{2x} / L_2$$
$$F_{2y} = F_2 L_{2y} / L_2$$

The restoring moment due to these components is given by:

$$M = R\cos(\theta)(F_{2x} - F_{1x}) - R\sin(\theta)(F_{1y} + F_{2y})$$

The above equation is nonlinear and can be linearized about the zero angular position to obtain a linear estimate of the joint stiffness:

$$K_{STIFF} = \frac{dM}{d\theta}\Big|_{\theta=0}$$

The complete expression for K_{STIFF} is given below:

$$K_{STIFF} = \frac{2R}{D^{3/2}}[(Dd - Rr^2)F_r + (D^{3/2}d - DLd + Rr^2L)K]$$
$$D = r^2 + (R-d)^2$$

5.4.2 Dynamic analysis

Let the angular displacement of the motor be given by θ and the relative angle between the arm and the motor be given by α. That means, α is the measurement of the

angular deflection of the arm. The sensed output is the angle $(\theta + \alpha)$. The total inertia of the motor is given by J_{hub} and the total inertia of the arm is given by J_{load}.

Equations of motion

The kinetic and potential energies of the system are given by:

$$\begin{aligned} PE_{spr} &= \frac{1}{2} K_{STIFF} \alpha^2 \\ KE_{hub} &= \frac{1}{2} J_{hub} \dot{\theta}^2 \\ KE_{lod} &= \frac{1}{2} J_{load} (\dot{\theta} + \dot{\alpha})^2 \end{aligned} \tag{5.3}$$

The total kinetic and potential energies are:

$$\begin{aligned} T &= KE_{hub} + KE_{lod} \\ V &= PE_{spr} \end{aligned} \tag{5.4}$$

The Lagrangian of the system then becomes:

$$L = T - V = KE_{hub} + KE_{lod} - PE_{spr} \tag{5.5}$$

The equations of motion using Lagrangian formulation are given by:

$$\begin{aligned} \frac{\partial}{\partial t}\frac{\partial L}{\partial \dot{\alpha}} - \frac{\partial L}{\partial \alpha} &= 0 \\ \frac{\partial}{\partial t}\frac{\partial L}{\partial \dot{\theta}} - \frac{\partial L}{\partial \theta} &= T \end{aligned} \tag{5.6}$$

Substituting for L from (4.5), yields:

$$\begin{aligned} (J_{hub} + J_{load})\ddot{\theta} + J_{load}\ddot{\alpha} &= T \\ J_{load}\ddot{\alpha} + J_{load}\ddot{\theta} + K_{STIFF}\alpha &= 0 \end{aligned} \tag{5.7}$$

The governing equation relating electrical and mechanical variables are given by:

$$V = IR_m + K_m\omega_m = IR + K_m K_g \omega \tag{5.8}$$

where $\omega_m = K_g \omega$, $I = \frac{V}{R} - \frac{K_m K_g}{R}\omega$

and,

$$T = K_g T_m = K_g K_m I = \frac{K_m K_g}{R} V - \frac{K_m^2 K_g^2}{R} \omega \tag{5.9}$$

Substituting for T and solving for the accelerations, we obtain the state space representation as follows:

$$\dot{X} = AX + Bu$$
$$y = CX + Du$$

where,

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & \frac{K_{STIFF}}{J_{hub}} & -\frac{K_m^2 K_g^2}{R J_{hub}} & 0 \\ 0 & -\frac{K_{STIFF}(J_{load} + J_{hub})}{J_{hub} J_{load}} & \frac{K_m^2 K_g^2}{R J_{hub}} & 0 \end{bmatrix} \qquad B = \begin{bmatrix} 0 \\ 0 \\ \frac{K_m K_g}{R J_{hub}} \\ -\frac{K_m K_g}{R J_{hub}} \end{bmatrix}$$

$$C = [1 \quad 1 \quad 0 \quad 0]$$

$$D = [0]$$

and,

$$X = [\theta \quad \alpha \quad \dot{\theta} \quad \dot{\alpha}]$$

There are different possible positions of the anchor points on the arm and the body. The model of the system could be initialized by some default position. For our case, the anchor position [A, 3] was used which means there are two identical springs (#2) connected between arm anchor #3 and body anchor #A. Also J_{load} was selected to be 0.0059 $kg - m^2$. The equivalent spring stiffness was found to be $K_{STIFF} = 1.61 Nm/rad$. With this numerical data, the state-space matrices become:

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 765.12 & -53.02 & 1 \\ 0 & -1039.48 & 53.02 & 0 \end{bmatrix}$$

$$B = \begin{bmatrix} 0 \\ 0 \\ 99.25 \\ -99.25 \end{bmatrix}$$

$$C = [1 \quad 1 \quad 0 \quad 0]$$

$$D = [0]$$

Figure 5.14 shows locations of poles and zeros of the system and the unit step response of the open-loop and closed-loop system in simulation. Fig 5.15 shows the system's real time step response using the same open-loop configuration.

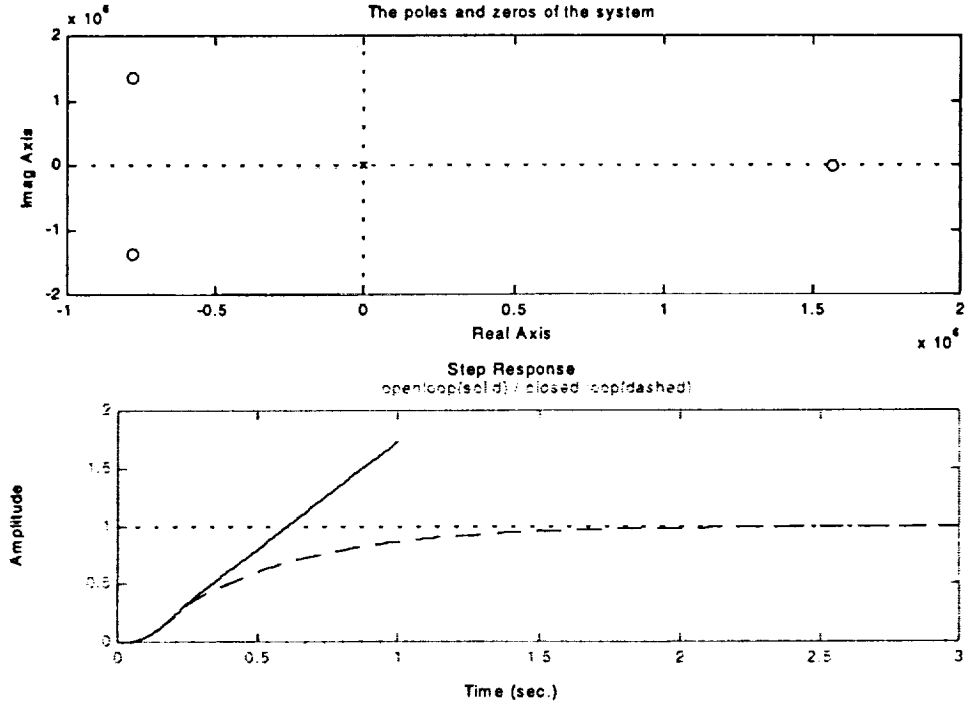
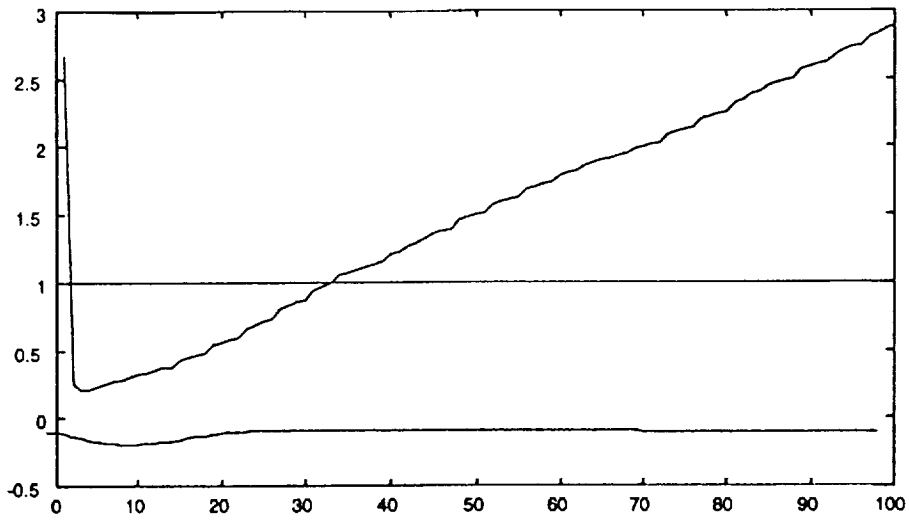


Fig 5.14 Poles-zeros map & open-loop step response



(x axis: time in seconds)

Fig 5.15 Real-time step response

5.5 Experimental set-up and definition of control problems

For testing the closed loop system with an NGPC controller, the performance criteria needs to be established. In this case, the classical criteria such as maximum overshoot (M_P) and steady-state error (ε_{ss}) are used. They are defined as:

$$M_P = (\max_t y(t) - y_{ss}) \times 100\%$$

$$\varepsilon_{ss} = |1 - y_{ss}| \times 100\%$$

where y_{ss} is the steady-state value of the output.

After defining the performance criteria, the type of the test signal (or input signal) is chosen. In order to test the closed-loop system with an NGPC controller, two test signal sequence were considered:

1. A unit step input: A step input equivalent to the angular displacement of 1 radian was used to test the system's rest to rest maneuver;
2. A reduced gap square wave input: A square wave with the magnitude of ± 1 radian was applied to test the system's tracking performance

The next step was to obtain a discrete-time representation of the system to embed the nominal system model into the neural network. The choice of sampling frequency is typically governed by the dominant time constants in the system and the stability considerations. In the subject example, 20 Hz was adequate for discretization. Having obtained a discretized model of the system, it was embedded into neural network by appropriately assigning the weights to the input layer connections using coefficients in the discretized model of system's transfer function. (See Fig 4.2)

If the plant model has uncertainty, online adaptation part of the neural network can be activated as shown in Fig 4.2. In addition to refining the plant model through online learning for better prediction, GPC tuning parameters can also be adjusted for achieving robustness to model uncertainty and disturbances.

In this example, parametric uncertainty were incorporated in two different ways: (1) by changing the spring stiffness, and (2) by changing the link inertia. Following are three different cases for which NGPC controller was tested.

Case (a) ***Plant with uncertainty in spring stiffness***: This was achieved by changing anchor points for the springs. The estimated change in the stiffness value was from 368 *N/m* to 665 *N/m*, i.e., the percentage change of 81%;

Case (b) *Plant with uncertainty in link inertia*: This was achieved by adding additional weight at the end of the link. The change in inertia value was from 0.0059 Nm^2 to 0.0021 Nm^2, i.e., the percentage change of 64%;

Case (c) *Plant with simultaneous uncertainties in spring stiffness and link inertia*: This case is a combination of case (a) and case (b) occurring simultaneously.

The parametric uncertainties in cases (a), (b) and (c) can translate into uncertainties in natural frequency of the system. The nominal value of the frequency is 2.91Hz. The perturbed values of frequency for cases (a), (b) and (c) are: 4.9Hz, 3.62Hz and 5.94Hz, respectively. These are equivalent to percentage changes of +69%, +25% and +104%, respectively.

For experimental verification, the parameters for NGPC control knobs were set to the values consistent with the results of Theorem 4.2. The corresponding settings of NGPC tuning parameters was as follows:

$$N_1 = 4, \quad N_2 = 7, \quad N_u = 4, \quad \text{and} \quad \lambda = 0.05$$

5.6 Real-time experiment results

For experimental validation, two different test inputs were used. The first test input was a unit step input which translates to a re-orientation command for the slewing link. The second test input was chosen to be a doublet, particularly to test the tracking ability of the controller in both directions.

For the first test input, i.e., the unit-step, three different experiments were conducted: (1) with embedded model same as the plant without any known uncertainty in the plant (Fig 5.16); (2) with embedded model being nominal plant model and the actual

plant has uncertainty (case c) with no online adaptation (Fig 5.17); (3) with embedded model being nominal plant model with online adaptation on (Fig 5.18);

For the second test input, following different experiments were conducted:

Exp. 1: Exact Embedded Model (Fig 5.19);

Exp. 2: Uncertainty (case (a)), without online adaptation (Fig 5.20);

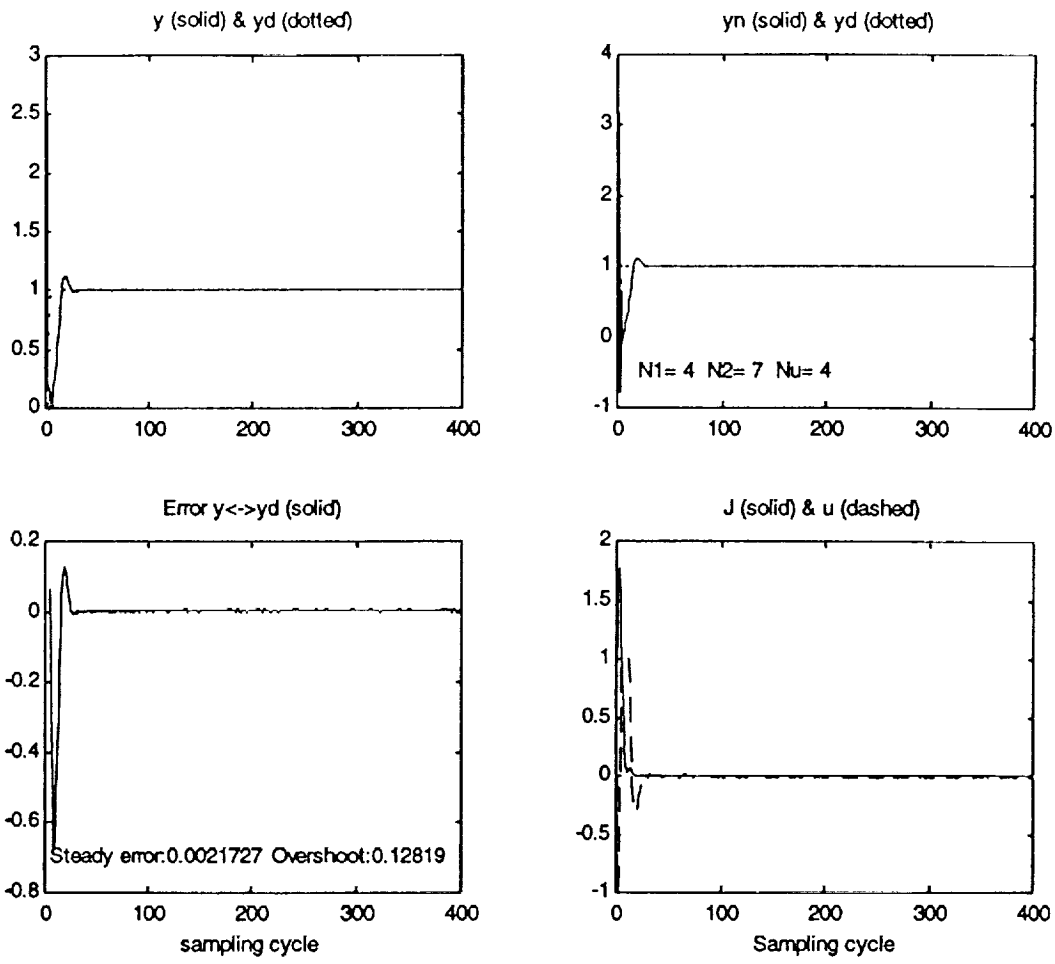
Exp. 3: Uncertainty (case (a)), with online adaptation (Fig 5.21);

Exp. 4: Uncertainty (case (b)), without online adaptation (Fig 5.22);

Exp. 5: Uncertainty (case (b)), with online adaptation (Fig 5.23);

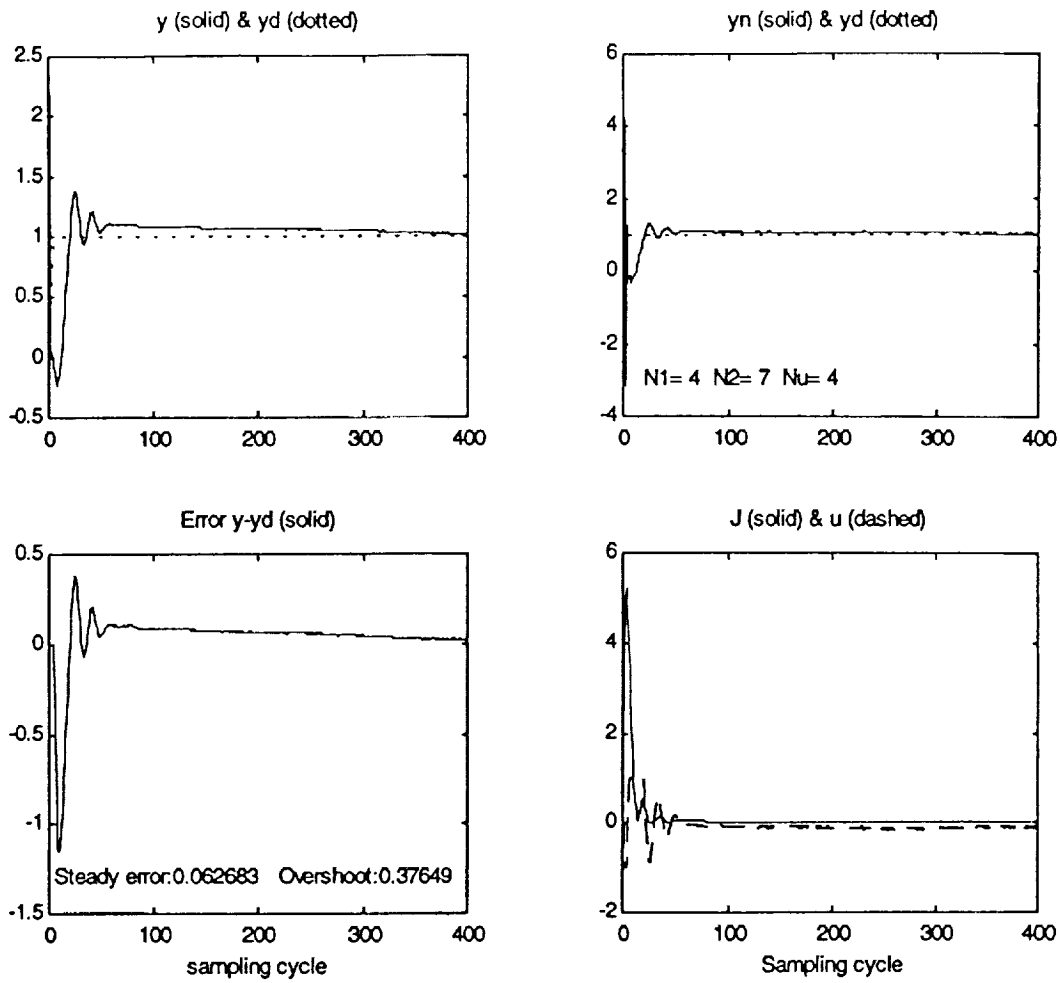
Exp. 6: Uncertainty (case (c)), without online adaptation (Fig 5.24);

Exp. 7: Uncertainty (case (c)), with online adaptation (Fig 5.25);



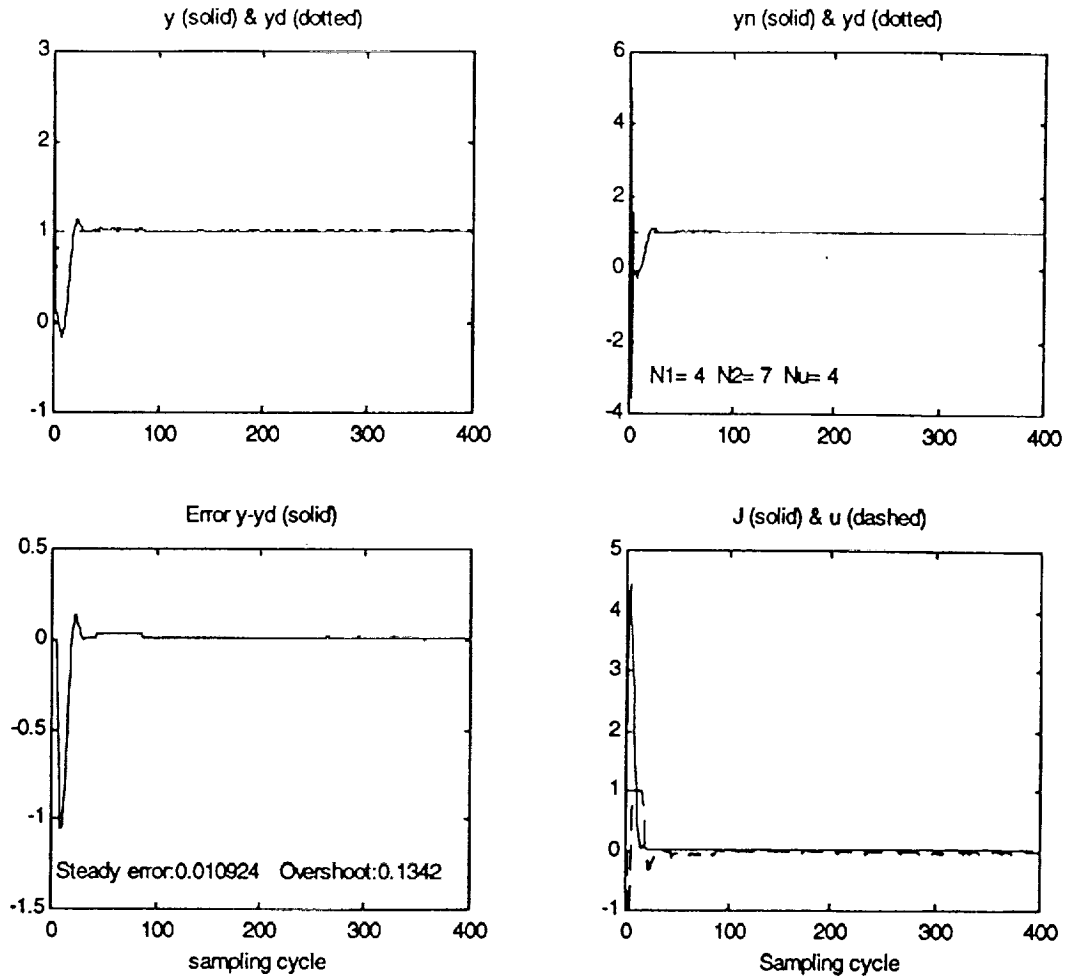
(x axis: time in samples (sampling frequency: 20 Hz))

Fig 5.16 Step response: exact embedded model



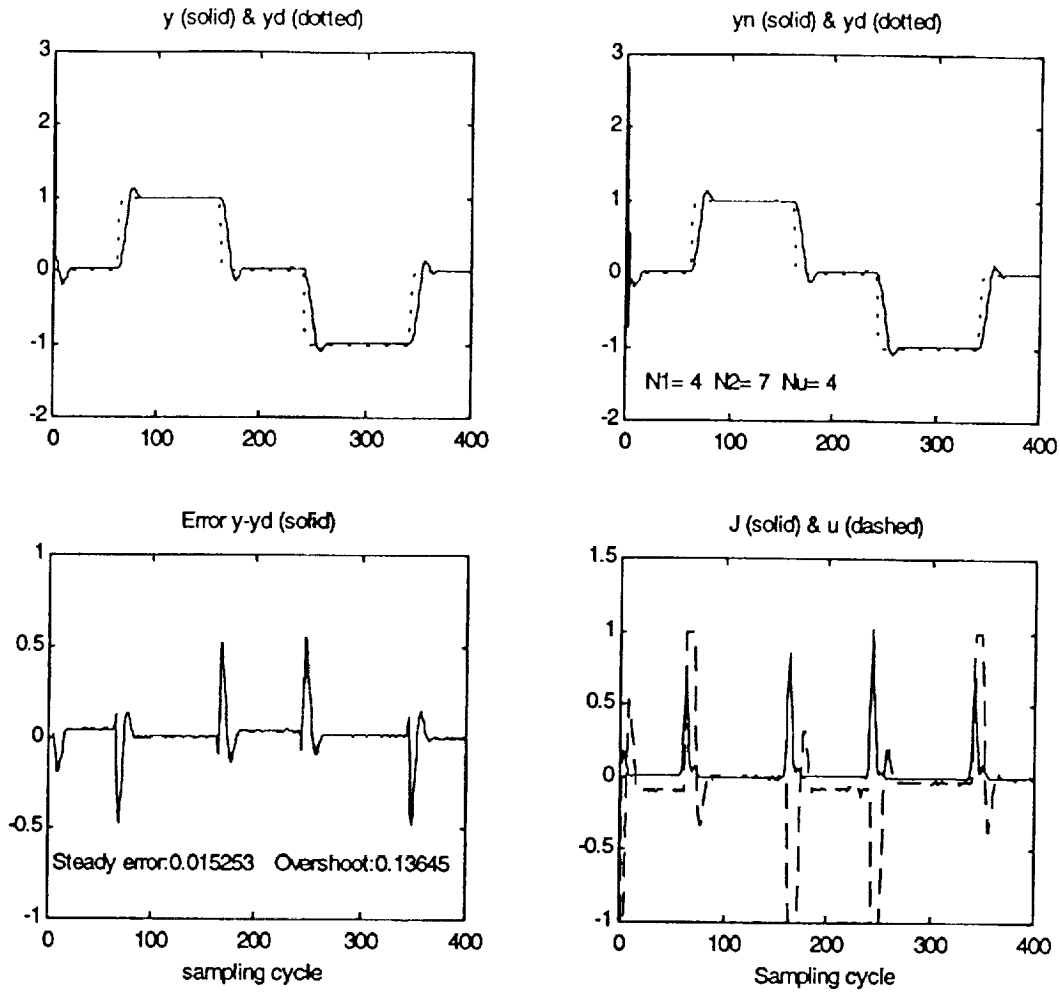
(x axis: time in samples (sampling frequency: 20 Hz))

Fig 5.17 Step response: +104% uncertainty, without online adaptation



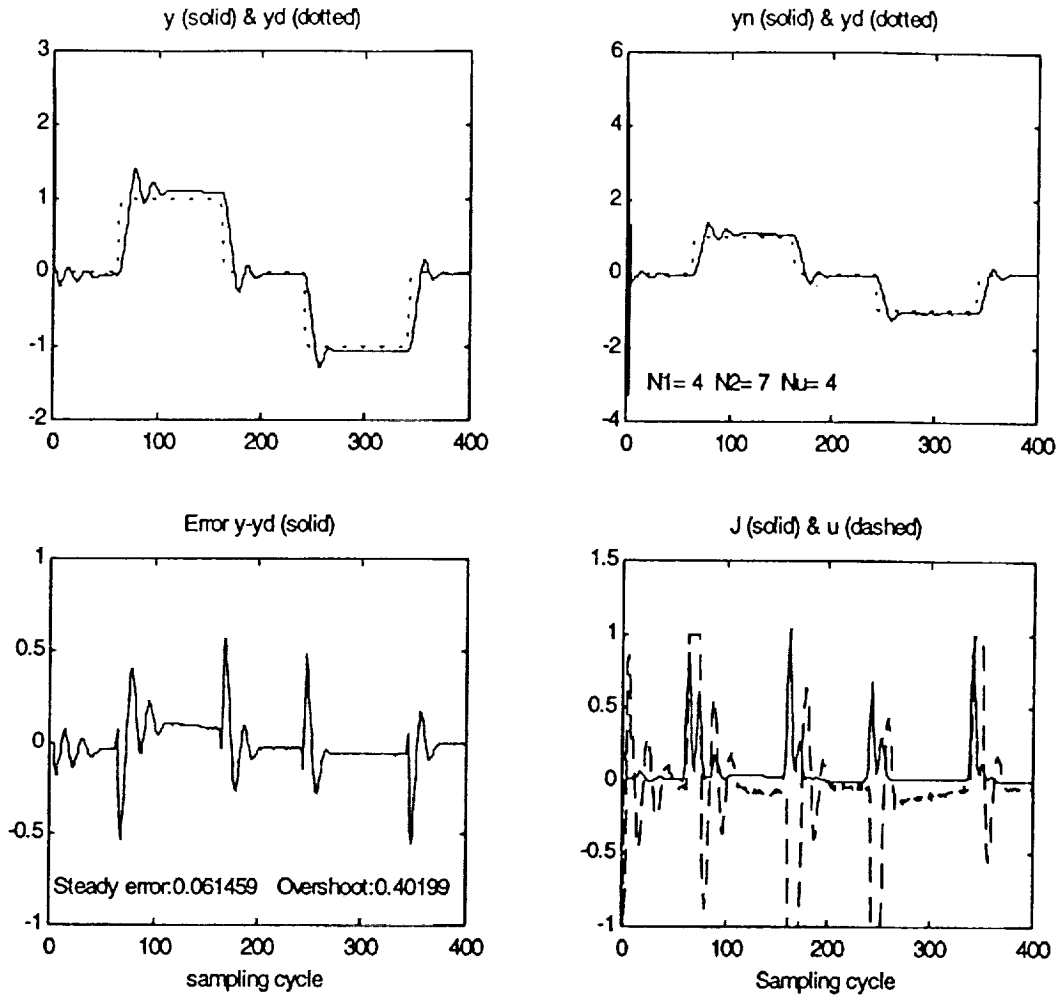
(x axis: time in samples (sampling frequency: 20 Hz))

Fig 5.18 Step response: +104% uncertainty, with online adaptation



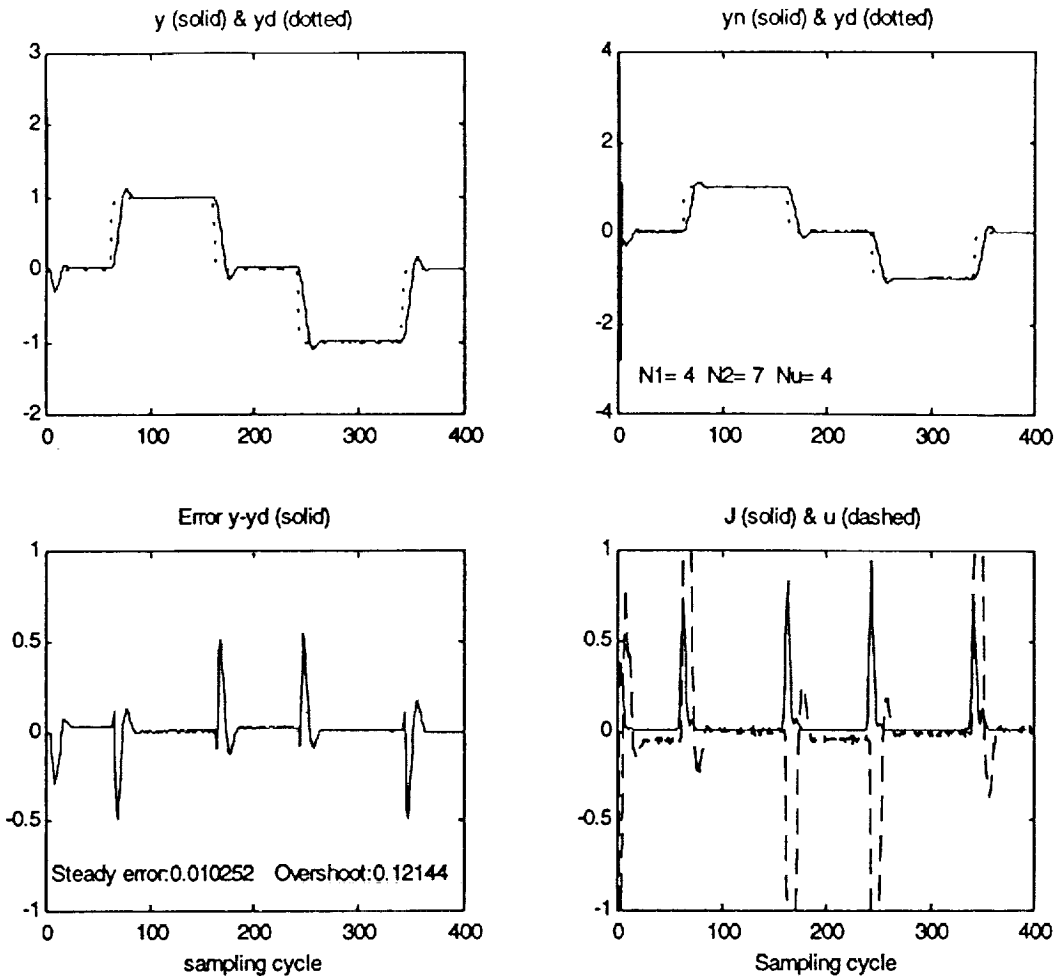
(x axis: time in samples (sampling frequency: 20 Hz))

Fig 5.19 Tracking performance: exact embedded model



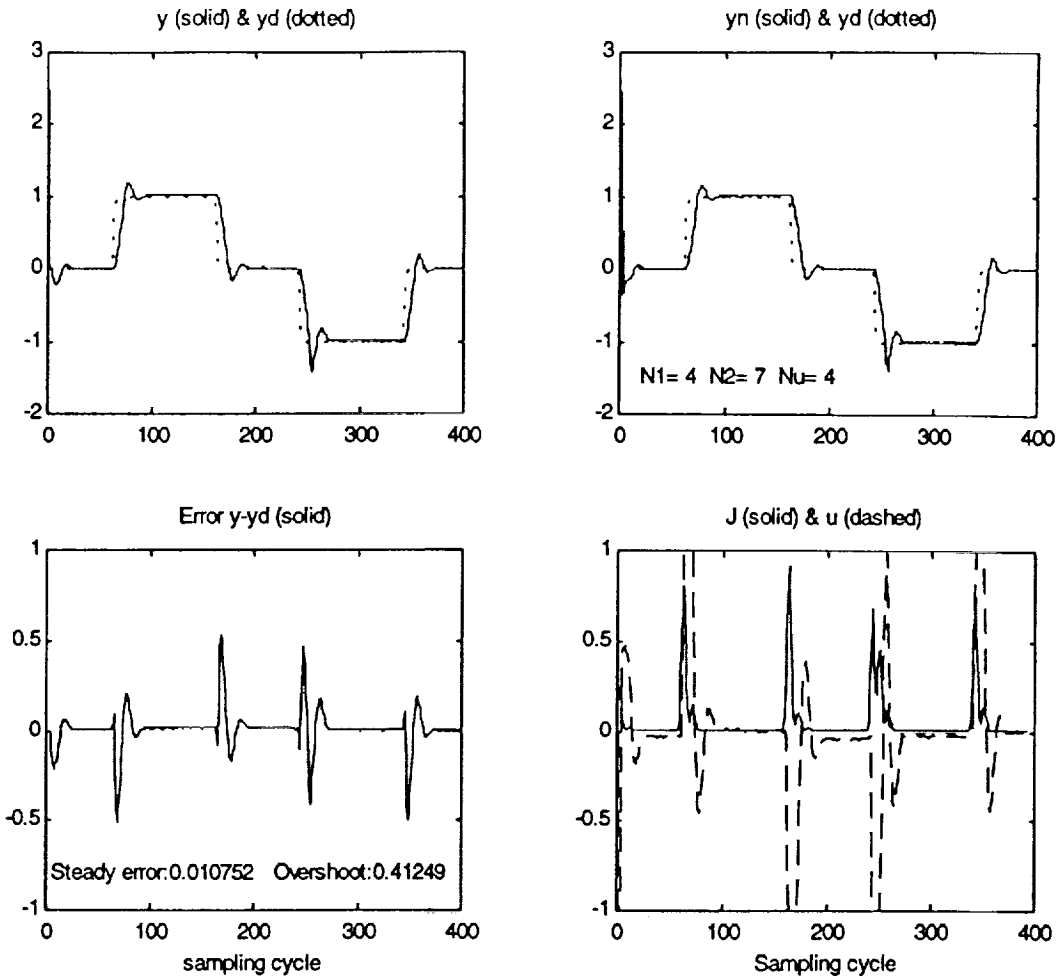
(x axis: time in samples (sampling frequency: 20 Hz))

Fig 5.20 Tracking performance: +69% uncertainty, without online adaptation



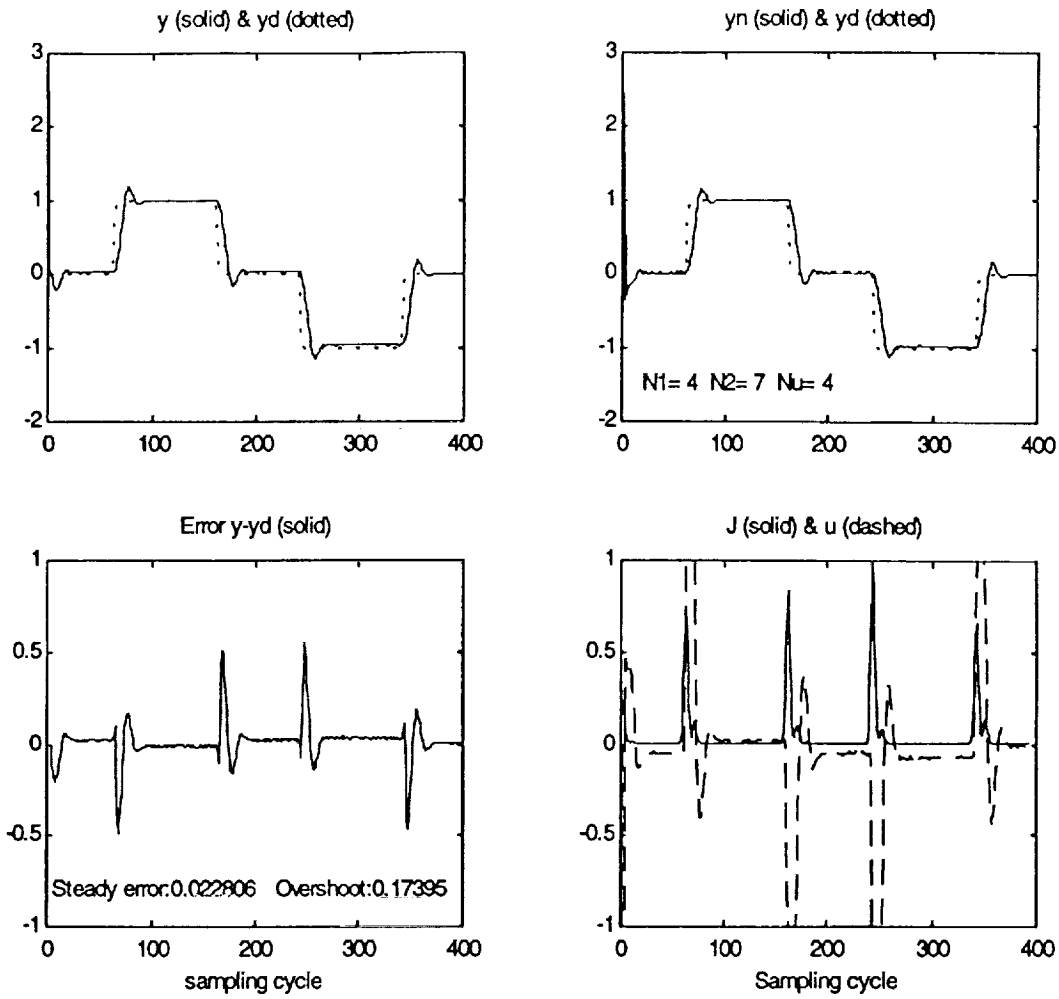
(x axis: time in samples (sampling frequency: 20 Hz))

Fig 5.21 Tracking performance: +69% uncertainty, with online adaptation



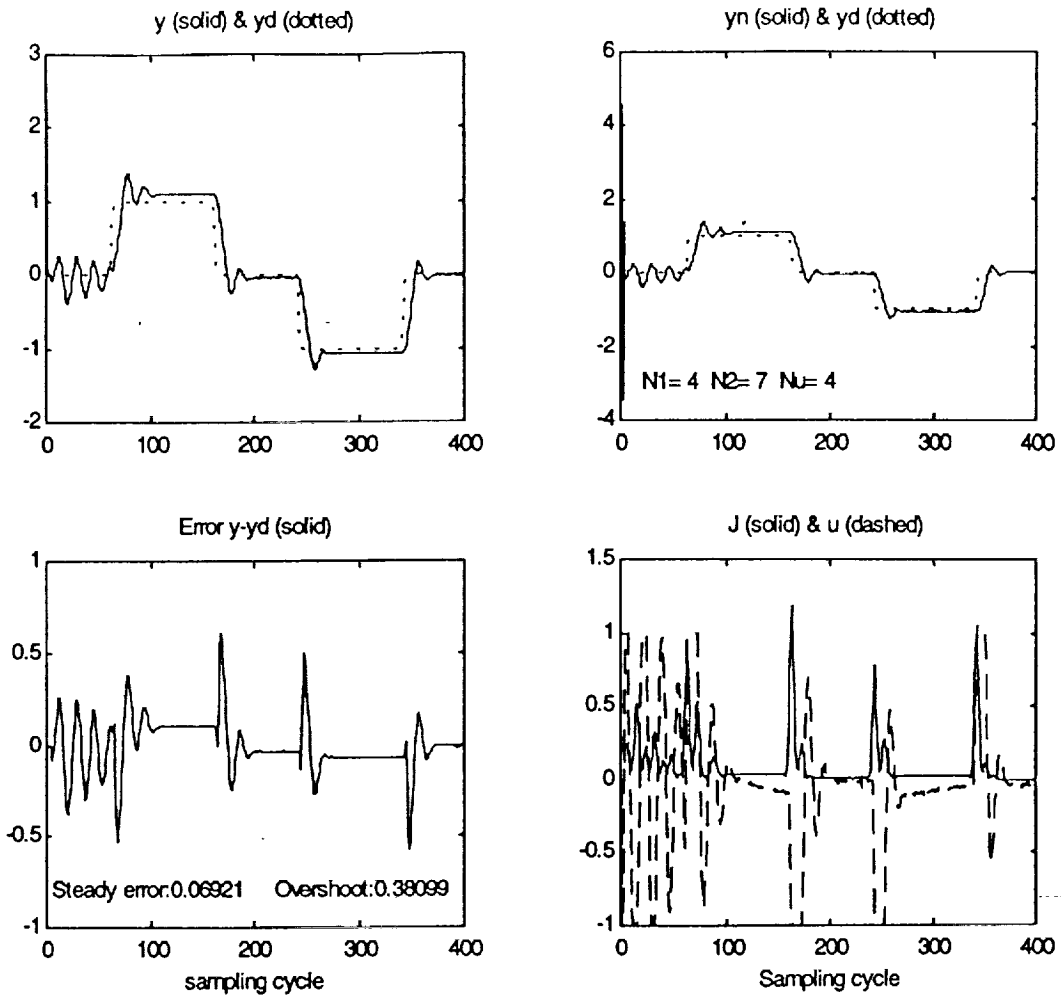
(x axis: time in samples (sampling frequency: 20 Hz))

Fig 5.22 Tracking performance: +25% uncertainty, without online adaptation



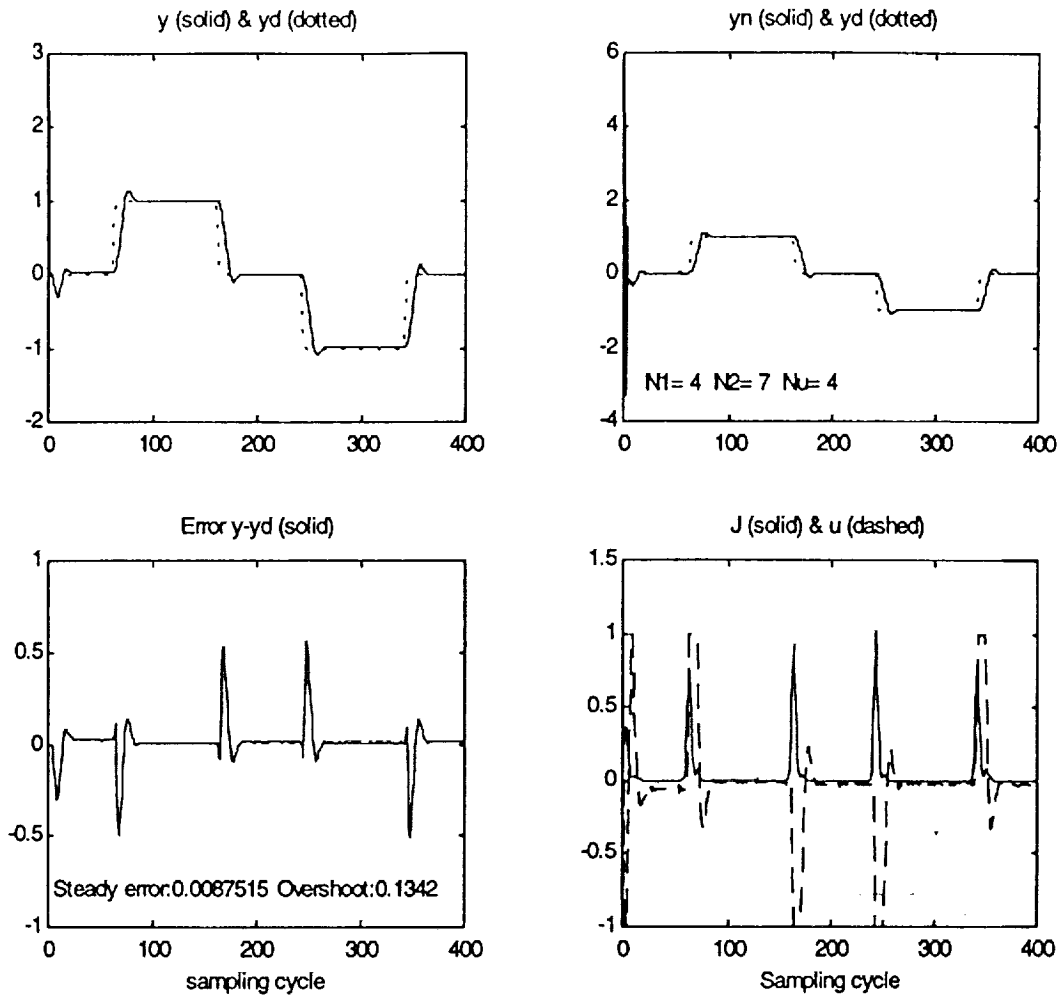
(x axis: time in samples (sampling frequency: 20 Hz))

Fig 5.23 Tracking performance: +25% uncertainty, with online adaptation



(x axis: time in samples (sampling frequency: 20 Hz))

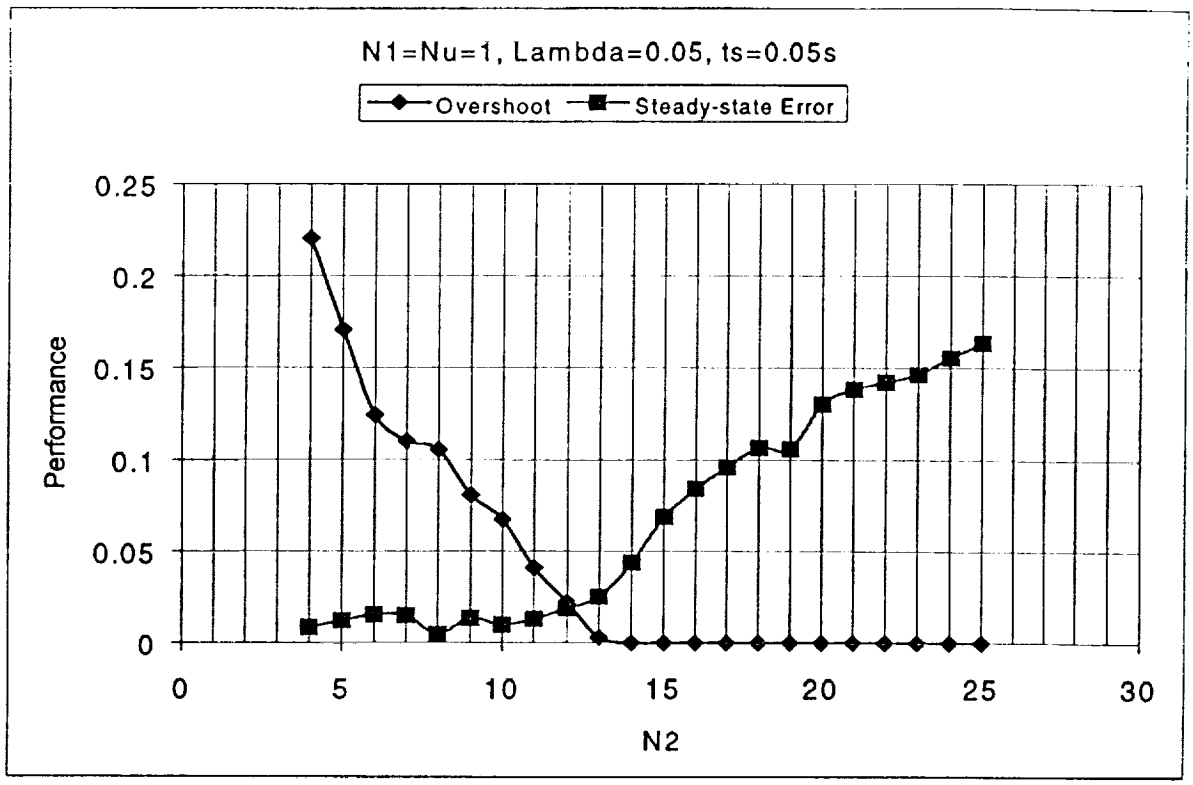
Fig 5.24 Tracking performance: +104% uncertainty, without online adaptation

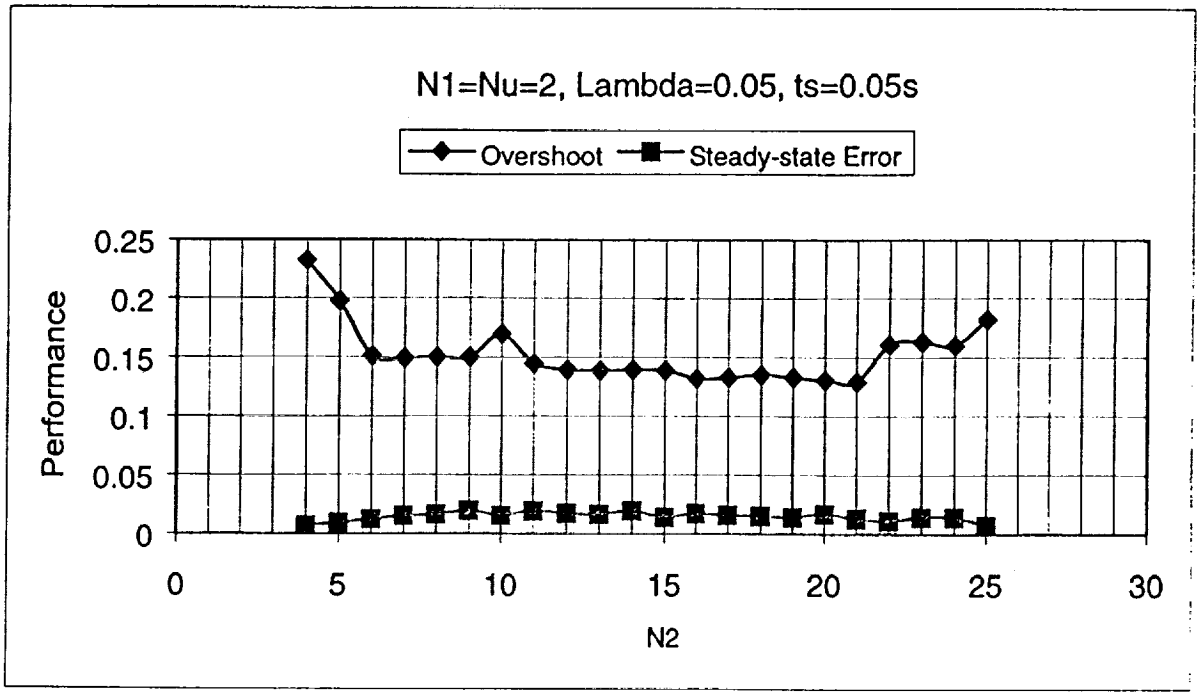


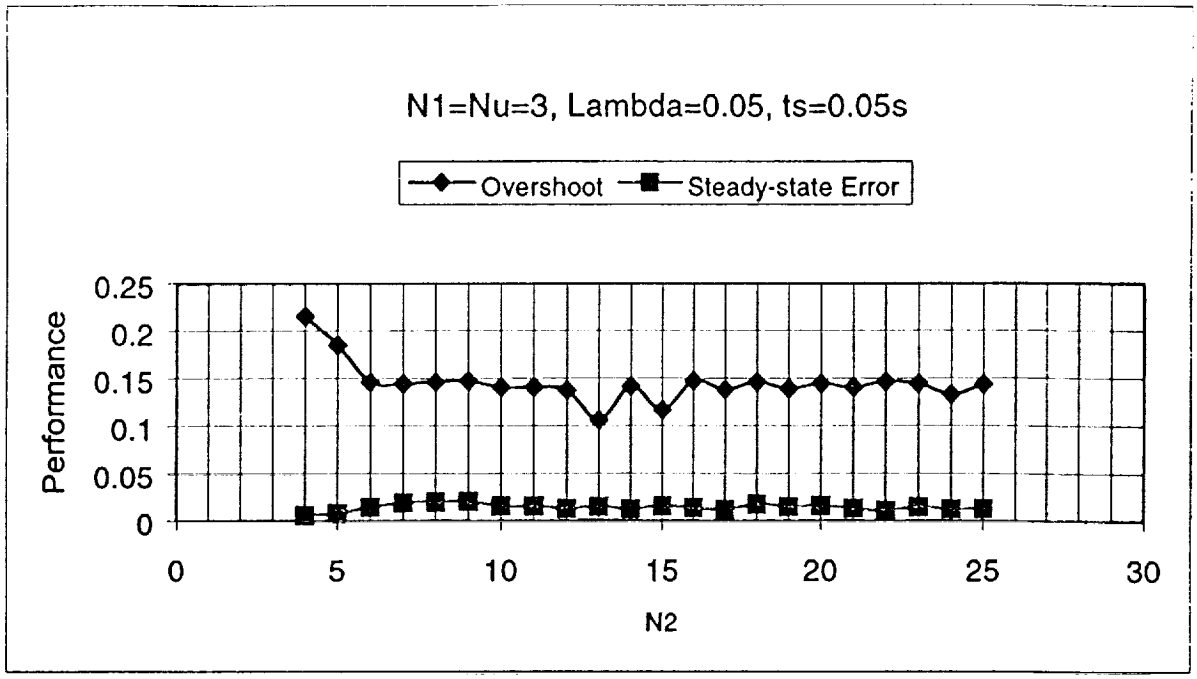
(x axis: time in samples (sampling frequency: 20 Hz))

Fig 5.25 Tracking performance: +104% uncertainty, with online adaptation

In all of the above experiments, NGPC tuning parameters were fixed. As discussed in Section 2.1, one of the attractive features of GPC is that the GPC's parameters can be tuned to improve performance. The following experimental results demonstrate this feature of GPC. The results are given in Fig 5.25, where effect of tuning parameters, N_2 ,is given on the system performance.


N1=Nu=1, Lambda=0.05, ts=0.05s
Overshoot
Steady-state Error
Performance
0.25
0.2
0.15
0.1
0.05
0
0
5
10
15
20
25
30
N2


N1=Nu=2, Lambda=0.05, ts=0.05s
Overshoot
Steady-state Error
Performance
0.25
0.2
0.15
0.1
0.05
0
0
5
10
15
20
25
30
N2



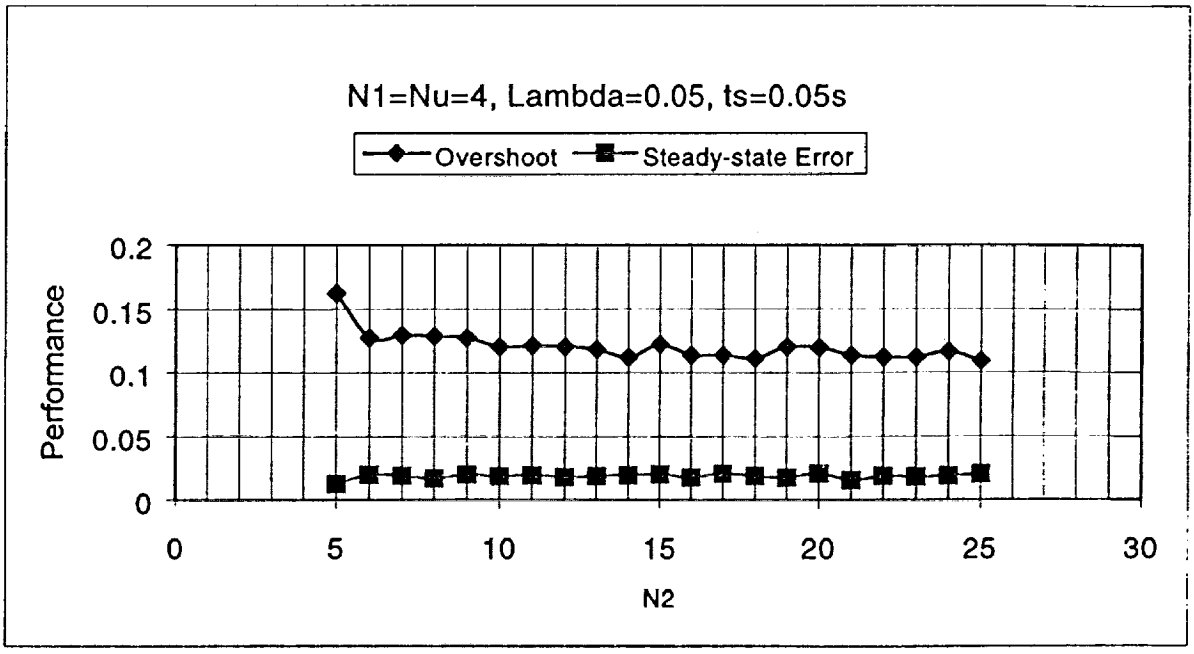


Fig 5.26 Tracking performance with different NGPC parameter

5.7 Observations

In this section, some observations are given based on the series of experiments conducted to validate robustness of GPC and use of NGPC in the control of uncertain systems.

A slewing link with flexible joint was used as a proof-of-concept apparatus for experimental validation. The objective was to control the rotation of the link in the presence of parametric uncertainty under two different test inputs. The test inputs used were unit-step and doublet. For all the experiments, the tuning parameters for GPC were set at the minimum value required for ensuring stability of the closed-loop system.

As seen in Fig 5.15 through 5.17, the unit-step response first deteriorates when uncertainty is introduced but improves again when online adaptation is turned on. The stability of closed-loop system, however, is maintained in all the cases since it is ensured by the choice of GPC tuning parameters.

In the second set of experiments, a doublet was used as reference signal in order to test the robustness of NGPC in tracking. Again, figures from Fig 5.18 through 5.24 show that the stability of NGPC is maintained in all cases and online adaptation greatly enhances the performance as before.

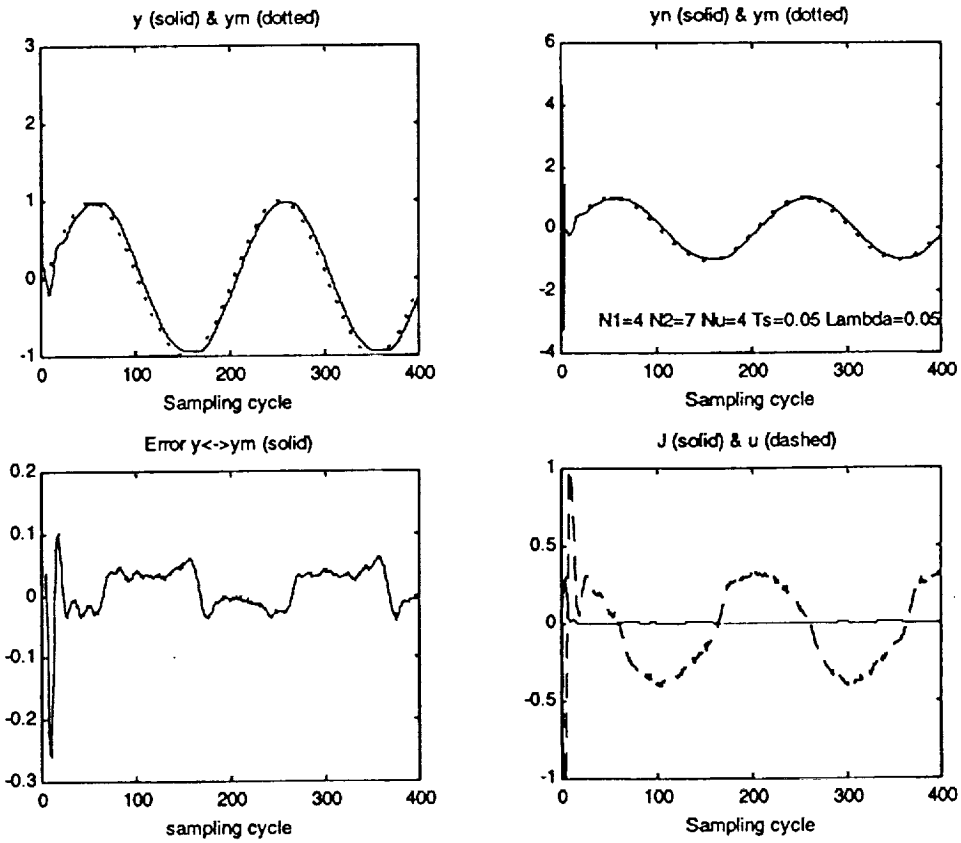
The effect of varying parameter N_2 (prediction horizon) on performance was also studied and from the results shown in Fig 5.25 following observations can be made:

1. The overshoot of the system is improved (i.e., reduced) with increase in N_2;
2. The steady state error of the system is also improved (decreased) with higher values of N_2. The only exception was the case when $N_1 = N_u = 1$. This could

be attributed to the fact that this value of N_1 and N_u is not sufficient to account for the system's delay;

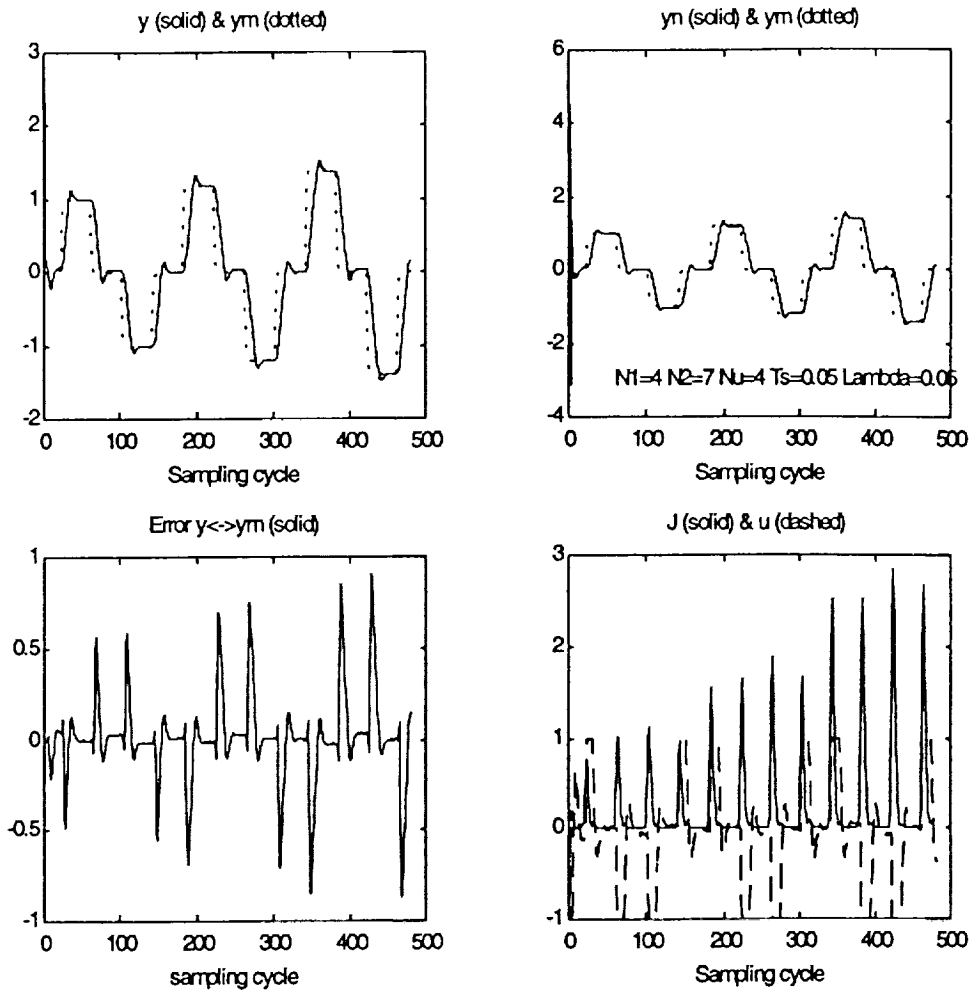
3. After certain value of N_2 (greater than 10), there is no significant change in the performance.

In order to reinforce the choice of tuning parameters, the system was tested with two more test signals: a sinusoidal input and a square wave input with changing amplitude. Also, the uncertainty level was selected to be +104% for experimental validation. The results of these experiments are given in Fig 5.27 and Fig 5.28. In both cases, online adaptation was turned on.



(x axis: time in samples (sampling frequency: 20 Hz))

Fig 5.27 System response: +104% uncertainty, with online adaptation



(x axis: time in samples (sampling frequency: 20 Hz))

Fig 5.28 System response: +104% uncertainty, with online adaptation

Fig 5.27 and Fig 5.28 shows that NGPC can handle inputs with changing frequency as well as amplitudes.

The robustness of GPC controller and online adaptation of neural network are two most important features of NGPC architectures.

5.8 Nonlinear mass-spring-damper system

In the previous section, it was shown that the NGPC architecture can be used effectively for the control of uncertain linear systems. In particular, for the case of large parametric uncertainty, it was shown, using simulations as well as experiments, that NGPC can archive stability robustness with desirable performance in the presence of such uncertainty. In this section, it will be shown that NGPC can be used in the control of nonlinear systems with uncertainty. In the case of nonlinear systems, NGPC has distinct advantage over other methods due to the learning and approximation capabilities of neural network models used for output predictions. A numerical example is given in this section to demonstrate the use of NGPC in the control of nonlinear system. The nonlinear system used is the well-known Duffing equation:

$$\ddot{X} + 2\dot{X} + X + 5X^3 = u \tag{5.10}$$

In Eq. (5.10), the nonlinearity arises from the nonlinear stiffness of the spring and is reflected in the term $5x^3$. For the purpose of robustness analysis, it is assumed that the stiffness of the nonlinear spring has uncertainty, i.e., coefficient of x^3 term in Eq. (5.10) is unknown. The design model (nominal model) is assumed to be given by the following approximation of the actual model:

$$\ddot{X} + \dot{X} + X + 2.5X^3 = u \tag{5.11}$$

In order to control this plant using NGPC, a linear embedded model is used to initialize the network. It can be obtained by using the linear terms in the 'Best Known' model as:

$$\ddot{X} + \dot{X} + X = u \tag{5.12}$$

The control problem that is addressed here to obtain a robust tracking control for the plant (5.10) given the best estimate of the plant model by Eq.(5.11).

There are several types of controller architectures that can be used to accomplish the objection defined above. Different architectures yield different performance. For the purpose of analysis following methodology was used. The neural network in NGPC set-up was used either in 'pre-train' mode or 'no-train' mode. In both cases, the option of on-line adaptation can be used if desired. The combinations of training options and on-line adaptations options give rise to the following four different cases for simulation. (In the case of pre-training the neural network is trained off-line using input-output data set obtained from actual plant model and using nominal plant (design) model as initial configuration)

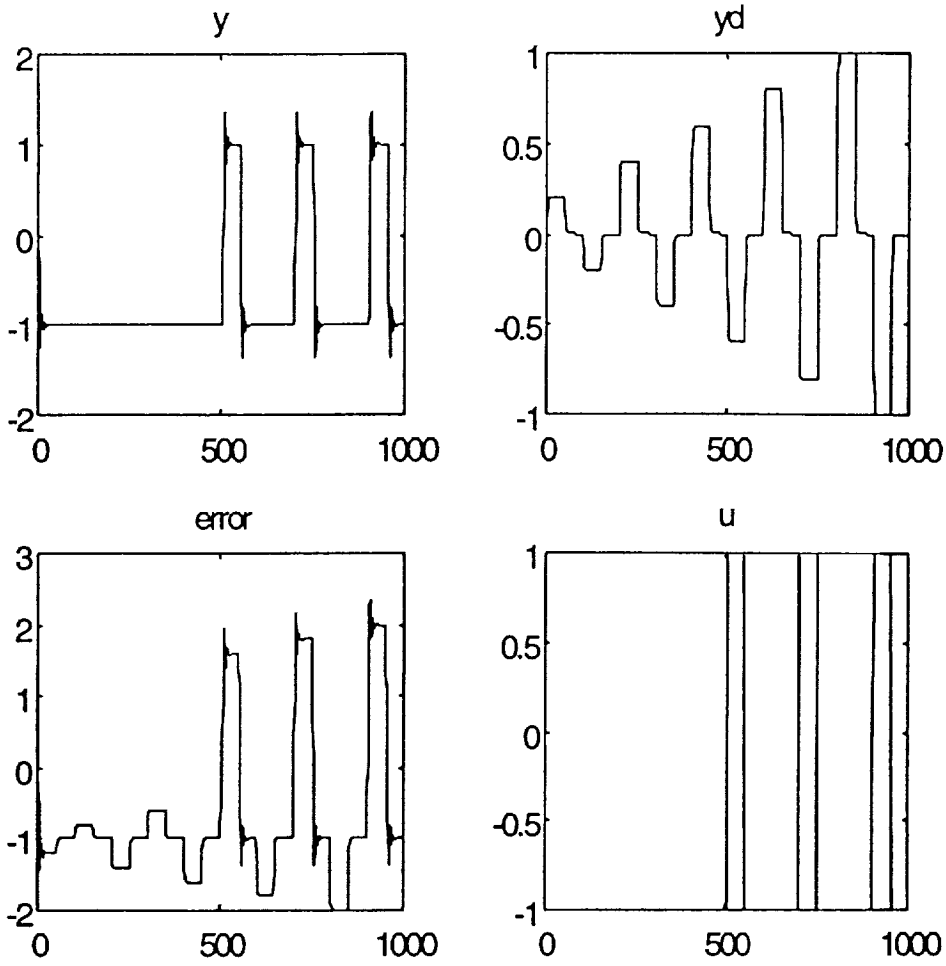
Case 1) No pre-training, no online adaptation (Shown in Fig 5.29)

Case 2) Using pre-training, no online adaptation (Shown in Fig 5.30)

Case 3) No pre-training, but using online adaptation (Shown in Fig 5.31)

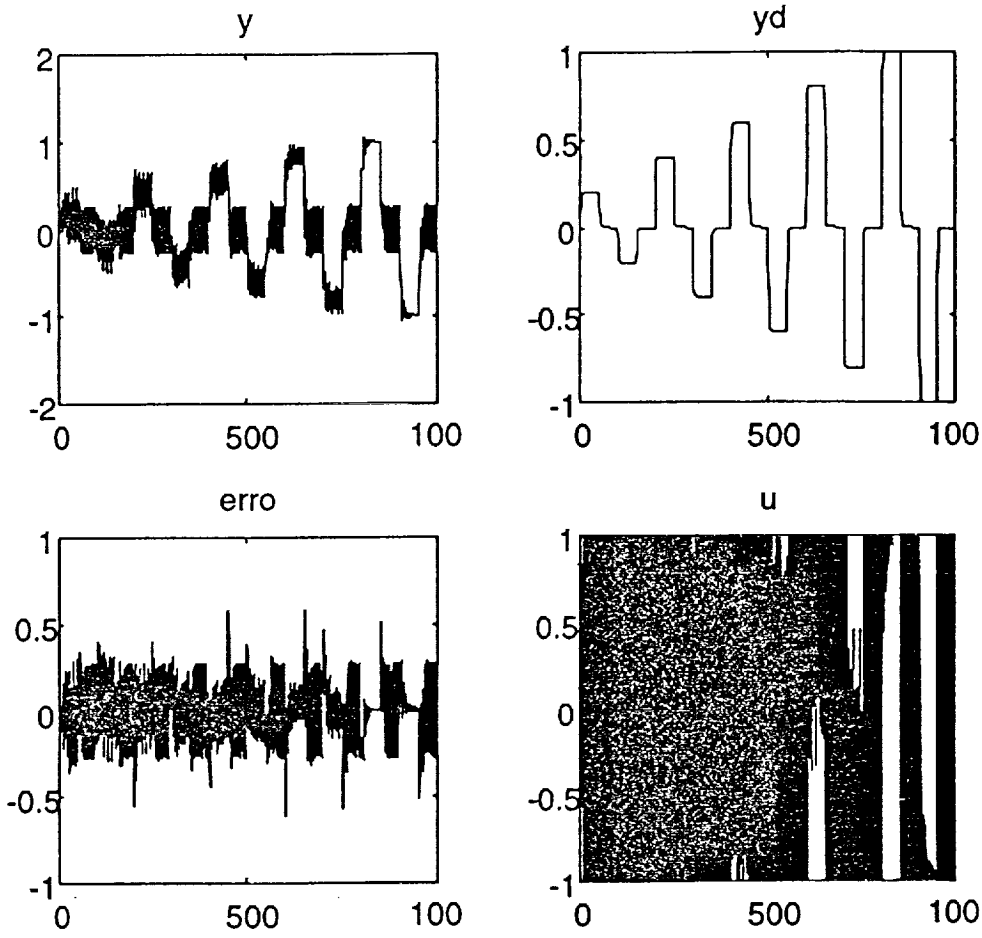
Case 4) Using pre-training and online adaptation (Shown in Fig 5.32)

The following figures show the testing results of above control structure:



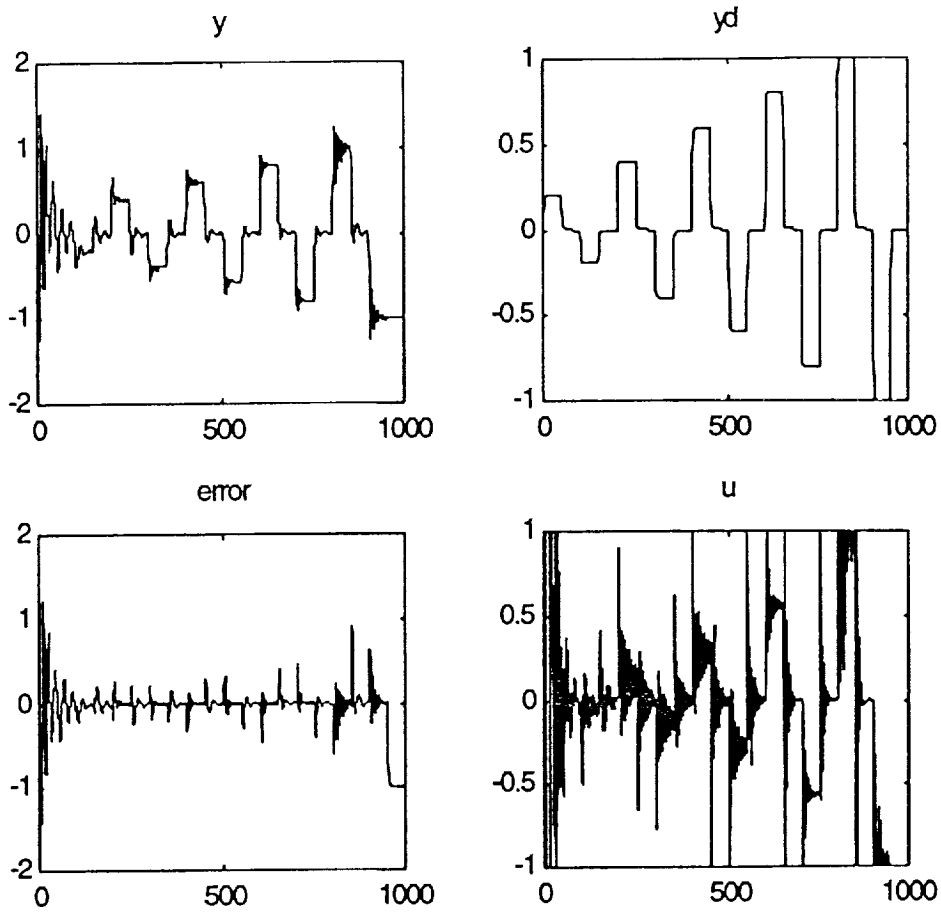
(x axis: time in samples (sampling frequency: 20 Hz))

Fig 5.29 No pre-training, no online adaptation



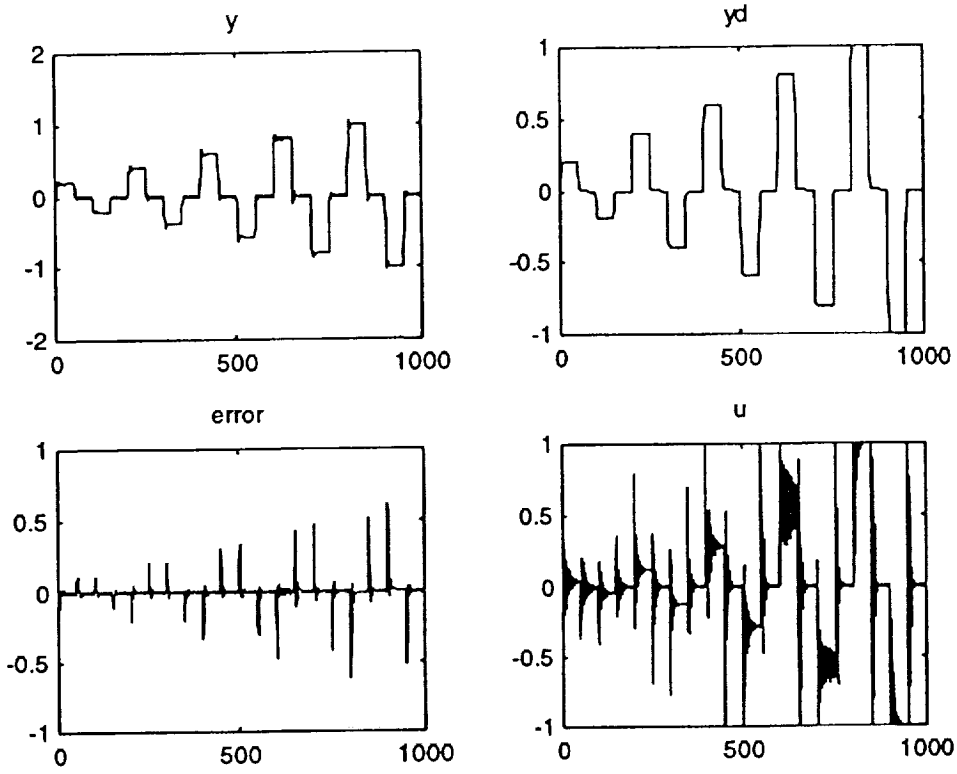
(x axis: time in samples (sampling frequency: 20 Hz))

Fig 5.30 Using pre-training, no online adaptation



(x axis: time in samples (sampling frequency: 20 Hz))

Fig 5.31 No pre-training, but using online adaptation



(x axis: time in samples (sampling frequency: 20 Hz))

Fig 5.32 Using pre-training and online adaptation

As it can be seen from the responses, on-line adaptation gives better tracking than no adaptation. Also, as expected, pre-trained model gives better tracking performance than no pre-training. The best results are obtained when on-line adaptation is used along with pre-trained network.

In summary, this example has demonstrated that NGPC can be used effectively to obtain robust control of nonlinear systems. The choice of GPC tuning parameters for nonlinear systems still remains to be a subject of research.

CHAPTER 6 Conclusions

This chapter is devoted to summarize the results of the research work reported in this thesis. The objective of the research was to obtain robustly stabilizing controllers and controller architectures for the control of uncertain linear and nonlinear systems. The controllers presented in this work are Generalized Predictive Controllers (GPC) and Neural Generalized Predictive Controllers (NGPC) both of which belong to a class of model-based controllers. The work presented here involved analysis and synthesis of controller architectures for real-time control of proof-of-concept laboratory apparatus and several numerical examples. An experimental example and several numerical examples were used to demonstrate the control methodology developed in this thesis. Given below is the chapter-vise summary of the work presented in this thesis.

In Chapter 2, the basic framework of GPC was presented along with a brief introduction of GPC and some remarks on their comparison with Linear Quadratic (LQ) controllers. It was shown that GPC controllers belong to a class of model-based controllers, namely, Receding Horizon Controllers (RHC). GPC was shown to be a dynamic version of RHC. Since RHC paradigm parallels very close to LQ paradigm, a comparison between LQ and RHC controllers was given to point out the differences between the two. It was shown that RHC has numerous advantages over its LQ counterparts. It was also shown that the disadvantages arising owing to the model-based control strategies are minimized by the use of "receding horizon principle". GPC, being a special case of RHC, inherits the basic robustness characteristics of RHC and offers additional benefits due to its own architecture. The basic derivation of GPC control law for LTI systems with known plant model was also given.

The third chapter gave the review of neural networks and presented some of their potential applications in the control system design. In particular, it was shown that how neural networks can be used as estimators or predictors in the control system applications. In GPC framework, neural networks find their use for system modeling. It was shown that the tapped-delay architecture of neural networks allows them to model dynamic systems. A procedure to train neural network as dynamic plant estimator was also given.

In Chapter 4, description of integration of neural network into GPC framework to yield NGPC architecture was given. The controller block diagram for NGPC was also given. The cost function minimization algorithm used in NGPC framework was described. The methodology to use neural network for modeling and prediction in the control of linear and nonlinear uncertain systems was given. The conditions for the closed loop stability were presented. The rules-of-thumb for tuning the NGPC parameters in order to improve system performance were given based on the empirical studies.

The analytical results and control methodology presented in Chapter 2-4 as validated by using several numerical examples and real-time control experiments. The numerical examples include pitch-rate control system for an F-16 fighter aircraft and nonlinear spring-mass-damper system. An experimental validation was performed using real-life hardware system consisting of slewing link with flexible joint. It was shown numerically as well as experimentally that NGPC, which combines GPC and neural network, can be used successfully in robust tracking problems. It was demonstrated that by choosing NGPC tuning parameters to satisfy stability constraints and using on-line

learning capability of neural networks, robust control with acceptable performance can be achieved for linear as well as nonlinear systems.

In summary, it was shown that NGPC offers a great potential for the robust control of uncertain linear and nonlinear systems. In particular, for nonlinear systems very few tools are available in the area of robust control and NGPC seems to be a promising candidate for these applications. Also, with advances in computer technology, the computational overhead in predictive control-based methodology is not a major concern. However, much research needs to be done in the theoretical aspects of the NGPC methodology. In particular, in the case of nonlinear systems most of the results are empirical in nature and much more work is required before strong analytical foundation can be established. In spite of these drawbacks NGPC methodology seems to be an attractive viable option for robust control for years to come. This makes NGPC valuable tool for robust control.

Further research in this area should focus on analytical aspects of NGPC. For example, stability of NGPC with neural network trained either off-line or on-line is an open and challenging problem. The existing mathematical tolls are not adequate to prove stability of such systems in a rigorous manner. Further research is necessary to establish a mathematical framework for addressing robustness issues related to stability as well as performance.

Reference:

[1] J. Richalet, A. Rault, J. L. Testud and J. Papon , 'Model Predictive heuristic control: application to industrial process', Automation, Vol. 14, No. 5, pp.413—428, 1978

[2] C. R. Cutler and B. L. Ramaker, 'Dynamic matrix control – A computer control algorithm', Proceedings JACC, San Francisco, U.S.A, 1980

[3] M. Kinnaert , 'Adaptive generalized predictive controller for MIMO systems', Int. J. Control, Vol. 50, No.1, pp.161—172, 1989

[4] S. Abu el Ata-Doss and M. Fliess, 'Nonlinear predictive control by inversion', Proceedings IFAC Symposium on Nonlinear Control Systems Design, Capri, Italy, 1989

[5] A. R. M. Soeterboek, H. B. Verbruggen and P. P. J. van den Bosch, 'Self-tuning control of processes with signal level and rate constraints', Proceedings 12th IMACS World Congress on Scientific Computation, Paris, France, 1988

[6] D. W. Clarke, C. Mohtadi and P. S. Tuffs, 'Generalized predictive control – Part I, The basic algorithm', Automation, Vol.23, No.2, pp.137—148, 1987

[7] D. W. Clarke, C. Mohtadi and P.S. Tuffs, 'Generalized predictive control – Part II, Extension and interpretations', Automation, Vol.23, No.2, pp.149--160, 1987

[8] R. M. C. De Keyser and A. R. van Cauwenberghe, 'Extended prediction self-adaptive control', Proceedings 7th IFAC Symposium on Identification and System parameter Estimation, York, U.K., pp.125--1260, 1985

[9] J. Richalet, S. Abu el Ata-Doss, Ch. Arber, H. B. Kuntze, A. Jacubasch and W. Schill, 'Predictive function control: Application to fast and accurate robots', Proceedings 10th IFAC World Congress, Munich, F.R/.G.,1987

[10] B. E. Ydstie, 'Extended horizon adaptive control', Proceedings 9th IFAC World Congress, Budapest, Hungary, 1984

[11] A. R. M. Soeterboek, H. B. Verbruggen, P.P.J. van den Bosch and H. Butler, 'Adaptive predictive control – a unified approach', Proceedings Sixth Yale Workshop on Application of Adaptive System Theory, New Haven, U.S.A, 1990

[12] A. R. M. Soeterboek, H. B. Verbruggen, P. P. J. van den Bosch and H. Butler, 'On the unification of predictive control algorithm', Proceedings 29th IEEE Conference on Decision and Control, Honolulu, U.S.A., 1990

[13] P. Gawthrop, Continuous Time Self-Tuning Control, Vol. I and II, Tannton, Research Study Press, 1990

[14] C. L. Phillips and H. Troy Nagle, Jr. Digital Control System Analysis and Design, Prentice Hall, Inc. , Englewood Cliffs, NJ, 1984

[15] R. Bellman, Adaptive Control Process: A Guided Tour, Princeton University Press, Princeton, NJ, 1961

[16] D. W. Clarke and C. Mohtadi, 'Generalized predictive control, LQ, or pole-placement: A unified approach', Proceedings of 25th Conference on Decision and Control, Athens, Greece, pp. 1536—1541, 1986

[17] Brain L. Stevens, Frank L. Lewis Aircraft Control and Simulation Wiley Interscience Publication ,1992

[18] D. Soloway and P. Haley Neural Generalized Predictive Control : A Newton-Raphson Implementation 1996, NASA TM 110244