uOttawa

L'Université canadienne
Canada's university

FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTOCTORALES

uOttawa

L'Université canadienne
Canada's university

FACULTY OF GRADUATE AND POSDOCTORAL STUDIES

Yiwu Jiang

AUTEUR DE LA THÈSE / AUTHOR OF THESIS

M.A.Sc. (Mechanical Engineering)

GRADE / DEGREE

Department of Mechanical Engineering

FACULTÉ, ÉCOLE, DÉPARTEMENT / FACULTY, SCHOOL, DEPARTMENT

Neural Network Based Feedback Linearization Control of an Unmanned Aerial Vehicle

TITRE DE LA THÈSE / TITLE OF THESIS

D. Necsulescu

DIRECTEUR (DIRECTRICE) DE LA THÈSE / THESIS SUPERVISOR

CO-DIRECTEUR (CO-DIRECTRICE) DE LA THÈSE / THESIS CO-SUPERVISOR

EXAMINATEURS (EXAMINATRICES) DE LA THÈSE / THESIS EXAMINERS

M. Liang

J. Sasiadek

Gary W. Slater

LE DOYEN DE LA FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTDOCTORALES /
DEAN OF THE FACULTY OF GRADUATE AND POSTDOCORAL STUDIES

Neural Network Based

Feedback Linearization

Control of an Unmanned Aerial Vehicle

Yiwu Jiang

A thesis submitted to the Faculty of Graduate and Postdoctoral Studies
in partial fulfillment of the requirements for the degree of

MASTER OF APPLIED SCIENCE

in Mechanical Engineering
Ottawa-Carleton Institute for Mechanical and Aerospace Engineering
University of Ottawa
Ottawa, Canada

November 2005

Your file Votre référence
ISBN: 0-494-11306-5
Our file Notre référence
ISBN: 0-494-11306-5

Acknowledgments

Many people have given me help, guidance, and inspiration in my years of graduate study. To all of them I am very grateful.

I am grateful to my friends Zhaoquan Chen, Kang Wen, Tan Wu and Wei Zhang for their encouragement and suggestions on my work. Thanks too to other friends and office-mates, past and present, who have given me the pleasure of their company and companionship.

I thank members of my qualifying examining board — Professor Ming Liang and Professor Jurek Sasiadek — who, through their questions, suggestions, and interest gave me that sense of validation that a graduate student needs to cross the threshold from the classical into the unknown.

I also thank Dr. Bumsoo Kim at Defense Research and Development Canada for his great contribution and support to this research.

To my families I am grateful for many things. They have been encouraging and patient throughout my graduate study, tolerating my neglect with an understanding.

Finally I would like to extend my deepest gratitude to my supervisor Professor Dan Necsulescu. He has given me encouragement, guidance, and most importantly his interest in my research. The depth of his understanding of control theory is an inspiration that I will carry with me into my professional years.

Abstract

In this thesis, a flight control design of an unmanned aerial vehicle (UAV) using neural network based feedback linearization and the output redefinition technique is presented. The UAV model we chose in this research is a nonlinear nonminimum phase system. The output redefinition technique is used in a way such that the resulting system is minimum phase and can be inverted. The nonlinear autoregressive moving average (NARMA-L2) neural network is trained off-line to identify the forward dynamics of the UAV model with the redefined output, and then inverted to force the real output to approximately track the desired trajectory. The results shows that a good tracking performance can be achieved using this control scheme.

Table of Contents

List of Figures

List of Tables

Nomenclature

Acronyms and Abbreviations

C.G	Centre of Gravity
CTOL	Conventional Take-Off and Landing
CV	Controlled Variables
DI	Dynamic Inversion
DOF	Degree of Freedom
GCS	Ground Control Station
GPS	Global Positioning System
ISR	Intelligence, Surveillance and Reconnaissance
LHP	Left-Half of s Plane
LQR	Linear Quadratic Regulator
MIMO	Multi-Input-Multi-Output
MNN	Multi-layer Neural Network
NARMA	Nonlinear Autoregressive Moving Average
NN	Neural Network
MPC	Model Predictive Control
P.I	Performance Index
PID	Proportion plus Integral plus Derivative
RHP	Right-Half Plane
SISO	Single-Input-Single-output
TITO	Two-Input-Two-Output
UAV	Unmanned Aerial Vehicle
UCAV	Unmanned Combat Aerial Vehicle
VTOL	Vertical Take-Off and Landing

Symbols

x_b, y_b, z_b	Body axes of the UAV
x_e, y_e, z_e	Earth axes
$\delta_e, \delta_a, \delta_r$	Control deflections of elevator/ aileron/ rudder
p, q, r	Roll/ pitch/ yaw rate (rad/s)
ϕ, θ, ψ	Roll/ pitch/ yaw angle (rad)
ϕ_d, θ_d, ψ_d	Desired roll/ Pitch/ Yaw angle (rad)
α	Angle of attack (rad)
β	Sideslip angle (rad)
X, Y, Z	Aerodynamic forces along body axes
L, M, N	Aerodynamic moments about body axes
I_x, I_y, I_z	Moments of inertia about body axes (kg-m^2)
V_p	Velocity of the UAV centre of mass (km/s)
u, v, w	Liner velocity components along body axes
G	Weight of the UAV
g	Gravitational acceleration (m/s^2)
y_*, z_*, l_*, m_*, n_*	Dimensional derivatives of aerodynamic forces and moments, where "*" denotes p, q, r, β, Δα, δa, δe ,δr
K_P, K_I, K_D	Proportional, Integral and Derivative Gains of the PID Controller

Chapter 1

Introduction

1.1 An Introduction to UAVs

An unmanned aerial vehicle (UAV) is a type of powered aircraft that does not carry a human pilot, and can fly autonomously or be remotely controlled. Its origins trace back to 1930's when radio controlled pilotless target aircrafts were developed in Britain and US.

In recent years, UAV technology has experienced a rapid growth in popularity. This has been driven by advancements in computing and positioning (e.g. GPS) technologies as well as the benefits UAVs could offer us. Without the human pilots onboard, UAVs are less restricted in the maneuvers they can perform, which allow us to push the flight envelope, i.e. to go faster, to go longer and to be better, and most importantly, to avoid risking human lives in dangerous situations such as battlefields and enemy territories, atrocious weather and environments with noxious gases or smokes. Therefore, UAV has been one of the latest military and force-multiplying technologies pursued by many countries around the world. Initially developed as an alternative for piloted platforms to undertake airborne intelligence, surveillance and reconnaissance (ISR) missions in high risk regions, UAV is now developing towards more complex tasks such as ground strike and air combat.

Although UAV is currently not so popular in the realm of civil aviation compared with its status in military field, it does show a bright future with a myriad of potential applications such as monitoring public infrastructures, meteorological data acquisition, search and rescue, and precision farming, etc. NASA and the aviation industry are

embarking on an effort to determine what regulations and what advancements in UAV technologies are needed to allow UAVs to use routinely the national air space. Resolving these issues will open up the national air space for civilian applications of UAVs. Another important consideration in UAV commercialization is the high crash rate associated with current UAV systems. During the U.S. military campaign in Afghanistan, as many as 12 percent of Predators (an up-to-date combat UAV of US forces) had crashed due to the bad weather conditions and other "technical problems" [1]. Such a high crash rate and the induced huge loss are definitely unacceptable for civilian research and applications.

In the context of control strategies, most UAV systems are falling into the groups of ground controlled systems and semi-autonomous systems, meaning that the human operators in ground control stations (GCS) interfere fully or partially with the UAVs' flight and mission execution by sending commands to or making decisions for the UAV systems [2]. For example, a semi-autonomous UAV requires ground input during critical portions of flight such as take-off, landing, weapons employment, and some evasive maneuvers. Such UAV systems are unable to complete the entire mission autonomously. Therefore, they can only be used within a restricted mission area where communications between UAVs and GCS are well established, which limits the applications of UAVs.

In order to improve the performance of UAV systems, researchers nowadays are working on fully autonomous UAV systems which are able to perform highly complex autonomous behaviors without operator intervention during the mission. These behaviors include autonomous moving-target tracking and following, autonomous obstacle collision avoidance, autonomous takeoff and landing, and multi-UAV collaboration. These capabilities are the determining factors between intelligent and remotely controlled

systems. J.How et al [3, 4] have carried out such researches on their Manned Vehicle-UAV platform. Their preliminary flight tests show that the UAV is capable of executing some simple tasks without human interaction.

1.2 Research Motivation

When talking about the improvement of the performance of UAV systems, one issue often mentioned is the maneuverability of the UAV. Aircraft maneuverability refers to a measure of the aircraft's ability to move freely in the air space. High maneuverability enhances the operational capability of a UAV under constrained environment. For example, it enables the UAV to avoid collision with static ground obstacles and other moving objects in the air. This is extremely important for each UAV in a multi-UAV system to maintain a formation flight or a cooperative flight. The benefit of high maneuverability for unmanned combat aerial vehicle (UCAV) lies in the tightly tracking of moving targets and avoidance of enemy attack from either the surface or the air. According to M. Christopher and A. Jones [5], a UCAV with a +10g acceleration capability can outfly many anti-aircraft missiles, and a capability of +20g will make the UCAV superior to nearly all missiles. Theoretically, due to the removal of the consideration of human pilot tolerance, such a high maneuverability can be achieved by hardware upgrading, such as proper selection of wing shapes, adoption of canards or vector engines, and the use of relaxed stability and fly-by-wire system.

However, the maneuverability is also affected by the software — flight controller. Traditionally, the flight controller is designed in near linear or low-angle-of-attack flight regimes using linear design methods such as PID control, lead-lag compensation, and pole placement, etc. Linear control techniques may work very well in certain trim

conditions, but fail to achieve a good performance when UAVs are flying in nonlinear regions or high-angle-of-attack conditions. Therefore nonlinear flight controllers are required to enhance the piloting capability and performance over the entire flight envelope.

Over the last three decades, feedback linearization or dynamic inversion, frequently used in nonlinear control laws, has been extensively studied and applied to control of a wide variety of nonlinear systems. The idea behind this approach is to algebraically transform a nonlinear system dynamics into a linear one by canceling out the nonlinearities, so that linear control techniques can be applied [6, 7]. Recently, feedback linearization has drawn great attention in flight control especially in designing high maneuverable fighter-aircraft and UAVs [8]. Compared with traditional flight control design which is based on assuming trim conditions, feedback linearization transforms the nonlinear dynamics of an aircraft to an equivalent linear system over the entire flight envelope, and thus the complex gain scheduling is avoided. This alternative method also offers greater generality for reuse across different airframes and greater flexibility for handling changes of an aircraft model.

Although feedback linearization is such a powerful tool for nonlinear control, several shortcomings have hindered its use in flight controls. A primary difficulty is that the knowledge of the nonlinear system dynamics should be precisely known. However, the distributed parameter characteristics of a UAV make it difficult or even impossible to derive an exact nonlinear model. In practice, for computational convenience, researchers usually build a lumped nonlinear UAV model by neglecting the uncertainty of aerodynamic parameters, dynamics of actuators, and other high order terms in equations. This may give a good result in computer based simulations, but may not be successful in

real flight tests. To deal with this problem, an adaptive element should be added to the feedback linearized system to compensate the uncertainties and small perturbations.

Another issue is that the full-envelope nonlinear inversion of a UAV model is computationally intensive, because the nonlinear model is a multi-input-multi-output (MIMO) system and must be inverted in real time [9, 10, 11]. Therefore, a powerful micro-processor and a large volume of RAM are needed for a UAV's on-board computer to accomplish this task.

The third problem is that the exact input-output feedback linearization only valid for minimum phase systems having stable zero-dynamics (a part of system dynamics rendered unobservable from the input-output feedback linearization). Unfortunately, some of UAVs are nonminimum phase systems due to several reasons such as the under-actuation, inherent structural flexibility, small forces produced by control surfaces, and non-collocated actuators and sensors. Therefore, directly applying the feedback linearization algorithm (including neural network based feedback linearization) on a nonminimum phase UAV model will result in an unbounded control signals and make the whole system unstable.

In this thesis, in order to address some of the above mentioned drawbacks, the nonlinear autoregressive moving average (NARMA-L2) neural network based feedback linearization and output redefinition method are proposed to solve the second and the third problems.

1.3 Thesis Organization

This dissertation is organized as follows.

In chapter 2, literature about feedback linearization and neural network control are

reviewed.

In chapter 3, a simplified UAV model is created using Newton-Euler equations.

In chapter 4, several flight control algorithms are studied. The emphases are given to feedback linearization and the output redefinition method solving for the nonominimum phase problem.

In chapter 5, the NARMA-L2 neural network based feedback linearization controller is introduced.

In chapter 6, the NARMA-L2 neural network based feedback linearization control algorithm together with output redefinition for lateral channel control of a UAV is simulated under MATLAB/Simulink®.

Finally, in chapter 7, the results of this research are summarized and a number of ideas and problems for future work are presented.

Chapter 2

Literature Review

In this chapter, we are going to review some recent literature about conventional control, feedback linearization and neural network control methodologies.

2.1 Conventional Control

Traditionally, aircrafts are mathematically modeled as time invariant systems and linearized about trim conditions, so that a linear control methodology, such as PID control, root locus, frequency response or pole placement, etc. can be applied [12, 13]. For example, using the root locus technique, it is possible to select an appropriate gain K of the closed loop system, such that the locations of the closed loop roots in the s-plane correspond to acceptable flying qualities. However, since the aerodynamics of an aircraft varies under different flight conditions, the above linear control techniques tailored for one condition may fail for others. In other words, these techniques provide an operating envelope that is too restrictive to be acceptable for UAVs, which results in the gain scheduling, a control technique widely used in design of automatic flight control system (AFCS) [12, p 298].

The basic idea of gain scheduling is to synthesize a series of dynamic controllers, one for each linearized model of the aircraft around a flight condition, and then "schedule" these controllers according to the actual flight condition. Therefore, gain scheduled controllers tailored for different trim conditions over the entire flight envelope can offer a more aggressive control design than a single linear controller can do. In reference [14], the theory of gain scheduling and its design procedure are reviewed. Reference [15] is an example of application of gain scheduling in UAV control.

Although gain scheduling is successful in most flight control applications, implementation of this technique is extremely tedious and time consuming and often the stability of the resulting system cannot be guaranteed. Most importantly, this method is unable to provide acceptable performance when a rapid maneuver or high-angle-of-attack is desired [6, p 523]. That is because the gain scheduled linear controllers are designed based on a series of trim conditions assuming that the airspeed is slowly varying, while during the aggressive maneuvering, the aircraft is far from the equilibrium due to rapidly varying airspeed. Therefore, to design a UAV controller to achieve a high performance in maneuverability, it is essential to take into account the nonlinearities in the UAV model.

2.2 Feedback Linearization

Since a UAV is inherently a nonlinear system, to improve the system performance its nonlinearities has to be dealt with directly. Feedback linearization is such a nonlinear control technique suitable for a wide range of operating conditions, including high-angle-of-attack and hypervelocity.

The basic idea behind this approach is to transform nonlinear dynamics into a linear form by using state feedback and canceling out the nonlinearities. A clear introduction to feedback linearization can be found in [6] and [7].

Recently, feedback linearization or so called dynamic inversion (DI) has drawn considerable attention in flight control. D. Enns et al (1994) described this tendency and illustrated the application of dynamic inversion for the F-18 high-angle-of-attack research vehicle [8]. They pointed out that feedback linearization uses on-board dynamic models and full-state feedback to globally linearize dynamics of selected controlled variables (CVs), such that simple controllers can be designed to regulate these variables

with desired closed-loop dynamics.

Usually a two time scale separation method is employed when applying dynamic inversion to a flight control system, which means that the whole system inversion is separated into inner and outer loop inversions [16]. This allows the inversion design to be performed without state transformation because the dynamics for each loop are square (number of controls equals the number of degrees of freedom). In reference [17], Z Q. Chen implemented dynamic inversion for both inner loop and outer loop to control the rates and the attitude of a UAV, and results show that dynamic inversion is superior to classical controllers over the whole flight envelope.

However, just like J E. Slotine mentioned in his book [7, p272], feedback linearization has a number of important limitations. One problem is that the model of the nonlinear system must be precisely known. But in practice, an exact model is not available due to different reasons such as unknown parameters, noises, actuator saturation, etc. Therefore, the sensitivity to modeling errors may be particularly severe when the linearizing transformation is poorly conditioned. To solve this problem, robust and adaptive controls are being introduced to feedback linearizable systems to compensate uncertainties [6, p312]. In [8], a method is developed that guarantees the robustness even if the inverse model is not perfect. In [18], a dynamic neural network is proposed to on-line optimize the fixed parameters of the feedback linearization controller. In [19], several nonlinear control laws, such as fuzzy logic, back stepping, neural network and model predictive control (MPC), are used to improve the performance of feedback linearization for nonlinear flight control.

The second problem associated with feedback linearization is that it cannot be used for nonminimum phase systems. Nonminimum phase refers to unstable zero-dynamics,

which leads to an unstable inverse of the system. To overcome this problem, much effort has been made. One method is approximate feedback linearization. This is done by approximating a nonminimum phase system with a minimum phase system, such that a bounded error tracking can be achieved [7, p261]. For a slightly nonminimum phase vertical take-off and landing (VTOL) aircraft, one can obtain an approximate minimum phase model by neglecting the coupling between the rolling moment and lateral acceleration [20, 21]. Similarly, for a slightly nonminimum phase conventional take-off and landing (CTOL) aircraft, neglecting the small forces caused by control surfaces may give a minimum phase model [22]. However, this method is only valid for slightly nonminimum phase systems and a cost is the loss of performance due to the unmodeled dynamics.

Another interesting method is output redefinition [7, p260]. The idea is to redefine the output function $y=h(x)$ so that the resulting zero dynamics is stable. This method is developed based on the fact that the zero dynamics is a property of the plant, the output, and the desired trajectory. Output redefinition has been successfully applied to control flexible manipulators. In [23], outputs are defined near the tip positions of a robot arm, such that the system becomes marginally minimum phase. However, this approach is unable to be applied to aircraft control, because it is difficult to find such a new output position in the body of an aircraft. In the field of flight control, the way to conduct output redefinition is through controlled variable (CV) selections [13, p 488]. In [8], recommendations for properly selecting the CVs are presented. For example, the roll axis variable is chosen as $p+ar$ to represent the roll rate. The similar work is done in [24] to redefine the output, the normal acceleration a_n. These selections of CVs are suitable for most conventional flight regimes and piloting tasks. However, they may need

modifications for high-angle-of-attack or very-low-speed flight. Furthermore, the selection of CVs still relies on trial and error to some extent. In [25], the output is redefined using stable/anti-stable factorization performed on the zero dynamics of a discrete-time nonlinear nonminimum phase system. This is equivalent to moving the positive zero to the left half of s plane in continuous-time domain. But this approach is only valid for a class of nonminimum phase system whose inputs appear linearly in the output function. In [26], a method is proposed to modify the output of the nonlinear aircraft model based on a transformation performed on the Jacobian linearization of the system. This transformation dose not affect the left-half zeros, thus the resulting system is essentially the same as the original one in the frequency range of interest. Using this approach, however, the system performance becomes worse when the frequency of desired output exceeds certain limits. This limitation must be carefully considered in the context of designing tracking controller for high maneuverable aircraft. In summary, the nonminimum phase problem is still an open area to researchers in feedback control. All the methods mentioned above have their merits and limitations.

The third problem which comes with feedback linearization is that the full-envelope nonlinear inversion of a UAV model is computationally intensive, because the nonlinear aircraft model is an MIMO system and must be inverted in real time [9, 10, 11]. Therefore a powerful micro-processor and a large volume of RAM are needed for a UAV's on-board computer to accomplish this task. This difficulty can be alleviated by adopting the neural network technology which will be discussed in the following section.

2.3 Neural Network Control

Neural networks (NNs) is a rapidly growing facet of artificial intelligence, using a

collection of simple processing units that are massively interconnected in order to produce meaningful behavior. It has been proven that a multi-layer neural network (MNN) is capable to learn any complex mappings to any desired degree of accuracy, which makes it a good approximator of the nonlinearities of a nonlinear model or a nonlinear inverse model [27]. Together with its other characteristics, such as parallel processing, generalization, learning and adaptation, and multivariable handling, neural networks have a great promise in control of dynamic systems and have motivated extensive research and development in this area [28].

Due to its significant ability to deal with nonlinearities, MNN has been widely used to compensate the modeling error and uncertainties. In [29, 30], an on-line neural network is applied as an adaptive element to dynamically augment the dynamic inversion controller of an aerial vehicle.

Since neural networks have the potential to model any systems, the use of neural networks in inverse-model-based strategies is highly promising. The method can be described as follows: the neural network is trained to be the inverse dynamic model of a plant, thus the multiplication of neural network and the plant will be an identity matrix and the system output becomes equal to the reference input. The neural network inverse modeling can be separated as direct one and specialized one [28]. For the direct inverse modeling, the system output is used as the input to the network. The network output is compared with the system input and the error is used to train the network. An example of this approach can be found in [9], where a neural network is trained off-line to model the inverse dynamics of an aircraft. Clearly, this approach relies heavily on the fidelity of the inverse model used as a controller, and is very sensitive to modeling errors. This problem can be solved by either adaptive control [9] or on-line learning [10]. For the specialized

inverse modeling, the network inverse model precedes the system and receives as input a training signal with the past system output and past inputs. The error signal used to train the network is the difference between the reference signal and the system output. This approach makes the neural controller have a one-step ahead prediction. An example can be found in [31].

An alternative way to perform an inverse modeling is to train two neural networks to model the forward dynamics of an affine system, and then invert the neural network model to obtain an approximate inverse of the system [32, 33, 34]. This method, or so called NARMA-L2 controller, can be treated as a neural network based feedback linearization. Since the control input is computed algebraically in a NARMA-L2 controller and no on-line learning is needed, the burden of computation can be greatly reduced compared with exact feedback linearization.

In reference [20], a set of neural networks are trained to approximate the Lie derivatives, such that the feedback linearization can be implemented step-by-step using these networks and most importantly the nonlinearity cancellation can be guaranteed. However, when a system is of high order, training a set of neural networks could be time consuming.

Finally, training a neural network to model the inverse dynamics of a nonlinear system may not always succeed even if we have enough neurons in hidden layers. This is due to the difficulty in meeting the constraints on the selection of training data, such as the magnitude, range and duration of the input excitation signals.

Chapter 3

The Nonlinear UAV Model

In this chapter, the dynamics of a UAV model for our research is presented. Several assumptions are made so that it will be easier to examine our control algorithm which is to be discussed in chapter 4 and chapter 5.

3.1 Definition of Coordinate Systems and Angles

Several Cartesian coordinate systems for aircraft modeling are considered in this research. In order to specify the orientation, position and velocity of a UAV with respect to the earth, the earth-axis frame is chosen to be the fixed frame of reference. This coordinate system has its origin fixed to an arbitrary point on the surface of the Earth. The three axes are denoted as x_e , y_e , and z_e, where x_e points to the north, y_e points to the east, and z_e points toward the center of the Earth.

In order to easily express the forces and moments acting on the UAV, the body-axis system, with its origin fixed on the centre of the gravity (C.G.) of the UAV, is usually used. Axes of the system are labelled x_b, y_b and z_b. The axes x_b and z_b lie in the plane of symmetry. x_b is chosen to point to the nose of the UAV. z_b is directed downward. y_b is perpendicular to the x_b-z_b plane and directed out of the right wing.

The orientation of the UAV body axes with respect to earth axes (x_e, y_e, z_e), defined by Euler angles, is shown in Figure 3.1. In the Euler angle system, three successive rotations carry the body axes from alignment with the earth axes to any arbitrary attitude. These three rotations are identified as yaw angle ψ about z_b axis, pitch angle θ about y_b axis, and roll angle ϕ about x_b axis.

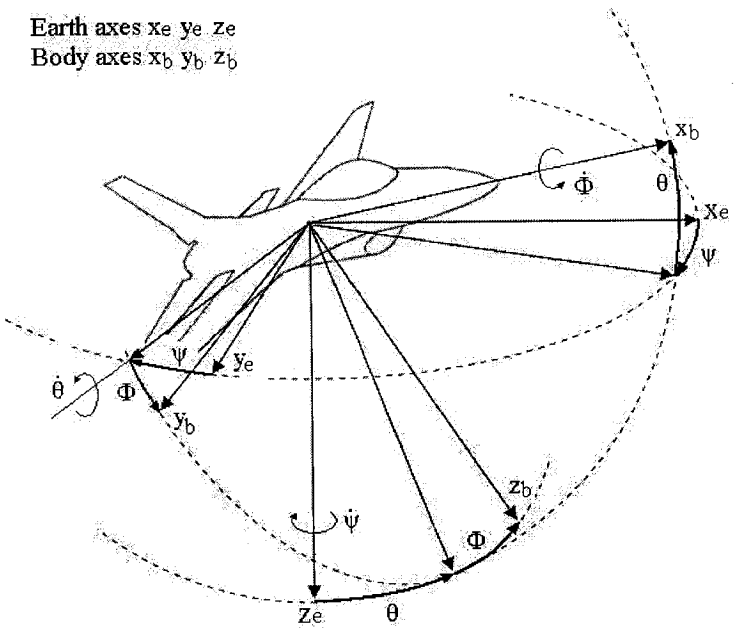

Figure 3.1 Definition of Euler angles

The corresponding matrices for the three rotations are:

$$T_{\psi} = \begin{bmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.1}$$

$$T_{\theta} = \begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix} \tag{3.2}$$

$$T_{\phi} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{bmatrix} \tag{3.3}$$

These three matrices can be combined into a single orientation matrix by matrix multiplication. For the ordering of rotations, ψ followed by θ, followed by ϕ, the matrix (3.1) is premultiplied by matrix (3.2), and that product is premultiplied by matrix (3.3). The single orientation matrix is given as follows.

$$T_{IB} = \begin{bmatrix} \cos\theta\cos\psi & \cos\theta\sin\psi & -\sin\theta \\ \sin\phi\sin\theta\cos\psi - \cos\phi\sin\psi & \sin\phi\sin\theta\sin\psi + \cos\phi\cos\psi & \sin\phi\cos\theta \\ \cos\phi\sin\theta\cos\psi + \sin\phi\sin\psi & \cos\phi\sin\theta\sin\psi - \sin\phi\cos\psi & \cos\phi\cos\theta \end{bmatrix} \tag{3.4}$$

The third coordinate system considered in this thesis is the stability-axis system. The x_s axis is taken as the projection of the velocity vector V_p of the UAV relative to the air mass into the aircraft plane of symmetry. The angle between the x_b axis of the body-axis system and the x_s axis of the stability-axis system is named as angle of attack α shown in Figure 3.2.

Finally the wind-axis system is chosen such that the x_w axis is in the direction of the velocity vector V_p of the UAV relative to the air. The angle between the x_s axis of the stability-axis system and the x_w axis of the wind-axis system is named as sideslip angle β which is also shown in Figure 3.2.

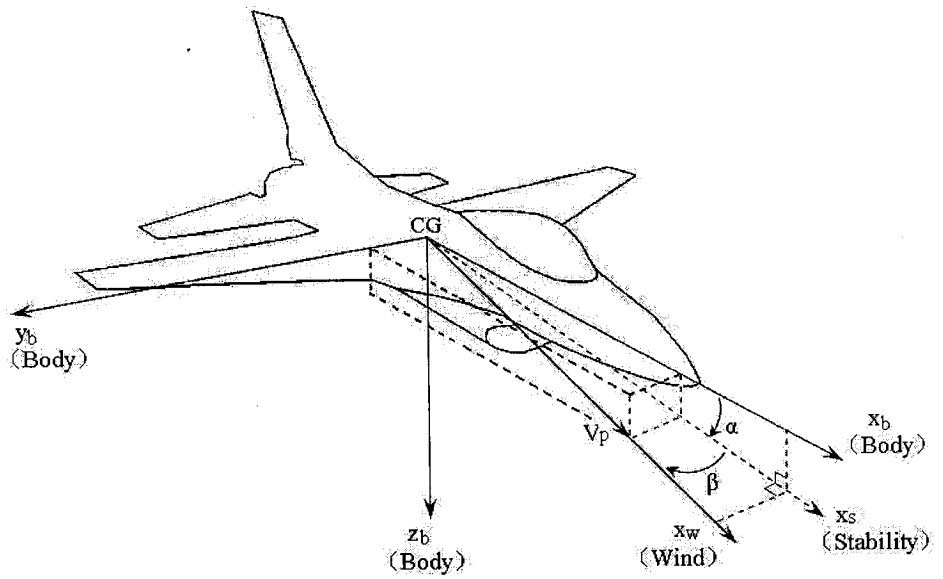

Figure 3.2 Definition of the angle of attack and the sideslip angle [12]

3.2 Definition of Velocities

Figure 3.3 shows the velocities of the UAV. V_p is the velocity of the UAV relative to the air mass. Its components along x_b, y_b and z_b axes of body-axis frame are denoted as u, v and w, respectively. The angular velocities about the body axes are named as roll rate p, pitch rate q and yaw rate r.

In general, the angular velocities p, q and r are not simply the derivatives of Euler angles ϕ, θ and ψ. Their relationship is given by:

$$\begin{aligned} p &= \dot{\phi} - \dot{\psi}\sin\theta \\ q &= \dot{\theta}\cos\phi + \dot{\psi}\cos\theta\sin\phi \\ r &= -\dot{\theta}\sin\phi + \dot{\psi}\cos\theta\cos\phi \end{aligned} \tag{3.5}$$

3.3 Definition of Forces and Moments

The forces acting on a flying aircraft are the aerodynamic force $\mathbf{F}_A$, engine thrust **T** and gravitational force **W**. $\mathbf{F}_A$ has its components X, Y and Z in body-axis system. The engine thrust **T** is assumed coincide with the x_b axis, and has no components along y_b and z_b axes. The gravitational force **W** acts on the centre of mass of the aircraft and points to the centre of the earth.

The aerodynamic moments resolved into the body-axis frame are the roll moment L, pitch moment M and yaw moment N.

The above forces and moments are shown in Figure 3.3.

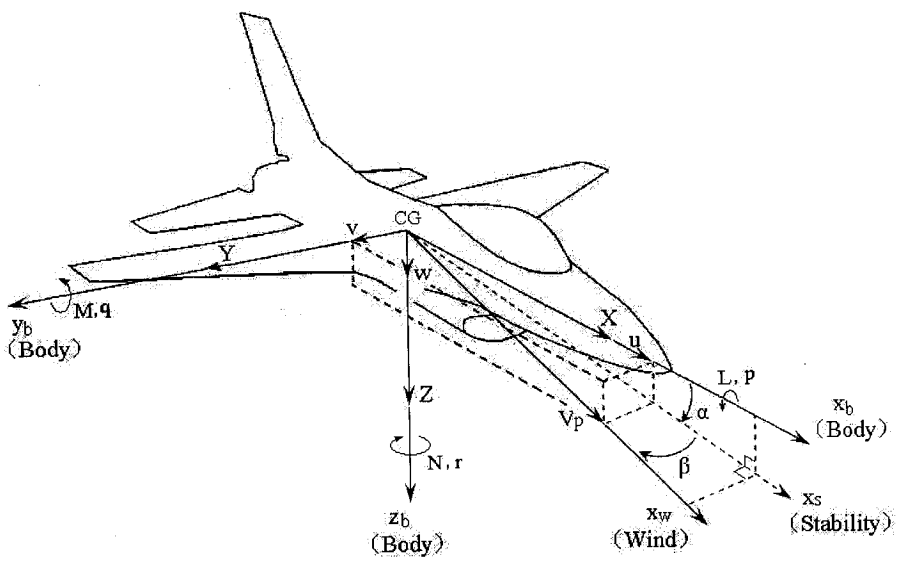

Figure 3.3 Definition of velocities, forces and moments [12]

3.4 Derivation of Motion Equations

In this thesis, the method to derive the motion equations of the UAV closely follows that of the reference [12].

First, the UAV is considered here as a single rigid-body, which means the distance between any points on the UAV does not change in flight. Therefore its motion can be considered to have six degree of freedom. By applying Newton's Second Law to the UAV, the equations of motion can be established in terms of translational and angular accelerations which are caused by the forces and moments discussed above.

According to Newton's second law, the motion equations with respect to the rotating and translating body-axis frame can be deduced as:

$$\boldsymbol{F} = \frac{W}{g}\frac{d}{dt}\boldsymbol{V}_p + \frac{W}{g}(\boldsymbol{\omega} \times \boldsymbol{V}_T) \tag{3.6}$$

$$\boldsymbol{M} = \frac{d}{dt}\boldsymbol{H} + \boldsymbol{\omega} \times \boldsymbol{H} \tag{3.7}$$

where $\boldsymbol{F} = \boldsymbol{i}F_x + \boldsymbol{j}F_y + \boldsymbol{k}F_z$ represents the vector of all external forces; W/g is the vehicle mass; $\boldsymbol{V}_p = \boldsymbol{i}u + \boldsymbol{j}v + \boldsymbol{k}w$ denotes the velocity vector of the UAV; $\boldsymbol{\omega} = \boldsymbol{i}p + \boldsymbol{j}q + \boldsymbol{k}r$ is the angular velocity vector of body axes; $\boldsymbol{M} = \boldsymbol{i}L + \boldsymbol{j}M + \boldsymbol{k}N$ is the vector of total applied torques; and $\boldsymbol{H} = I\boldsymbol{\omega}$ is the angular momentum of the UAV about its c.g, where the inertia matrix I is defined as:

$$I = \begin{bmatrix} I_{xx} & -I_{xy} & -I_{xz} \\ -I_{xy} & I_{yy} & -I_{yz} \\ -I_{xz} & -I_{yz} & I_{zz} \end{bmatrix} \tag{3.8}$$

Since the UAV is symmetrical about x_b-z_b plane, the terms I_{xy} and I_{yz} in (3.8) equal to zero. The term I_{xz} is sufficiently small, and thus allows to be neglected.

Equation (3.6) can be written in a scalar form as follows:

$$\begin{bmatrix} F_x \\ F_y \\ F_z \end{bmatrix} = \frac{W}{g} \begin{bmatrix} \dot{u} + qw - rv \\ \dot{v} + ru - pw \\ \dot{w} + pv - qu \end{bmatrix} \tag{3.9}$$

Since the sum of external forces **F** is composed of aerodynamic force $\mathbf{F}_A$, thrust **T** and gravity $\boldsymbol{W}$, the left-hand side of equation (3.9) can be written as:

$$\begin{bmatrix} F_x \\ F_y \\ F_z \end{bmatrix} = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} + \begin{bmatrix} T \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} W_x \\ W_y \\ W_z \end{bmatrix} \tag{3.10}$$

where $\boldsymbol{T}$ is coincident with x_b axis, and has no components in y_b and z_b axes; W_x, W_y and W_z are three components of gravity $\boldsymbol{W}$ in body-axis frame. They can be calculated using the transformation matrix (3.4) as follows:

$$\begin{bmatrix} W_x \\ W_y \\ W_z \end{bmatrix} = T_{IB} \begin{bmatrix} 0 \\ 0 \\ W \end{bmatrix} = \begin{bmatrix} -W \sin\theta \\ W \sin\phi \cos\theta \\ W \cos\phi \cos\theta \end{bmatrix} \tag{3.11}$$

Substituting (3.10) and (3.11) into (3.9), we have the force equations expressed by:

$$X = (W/g)(\dot{u} - rv + qw) - T + W \sin\theta \tag{3.12}$$

$$Y = (W/g)(\dot{v} - pw + ru) - W \sin\phi \cos\theta \tag{3.13}$$

$$Z = (W/g)(\dot{w} + pv - qu) - W \cos\phi \cos\theta \tag{3.14}$$

Similarly, the moment equation (3.7) can also be transformed into a scalar form:

$$L = I_x \dot{p} + (I_z - I_y) qr \tag{3.15}$$

$$M = I_y \dot{q} - (I_z - I_x) rp \tag{3.16}$$

$$N = I_z \dot{r} + (I_y - I_x) pq \tag{3.17}$$

Equations (3.12-3.17) are the complete equations of motion of the rigid body UAV.

3.5 The Nonlinear UAV Model

In order to derive the UAV model from above motion equations, we shall make the assumption that the velocity V_p is constant during the motion considered; both v and w are sufficiently small, so that one has $u \cong V_p$, $\beta \cong v/V_p$ and $\alpha \cong w/V_p$. Under these

assumptions, equation (3.12) drops out. Let $i_1 = (I_z - I_y)/I_x$, $i_2 = (I_z - I_x)/I_y$, $i_3 = (I_y - I_x)/I_z$, $y=Y/(W/g)V_T$, $z=Z/(W/g)V_T$, $l= L/I_x$, $m = M/I_y$, and $n=N/I_z$. Equations (3.13-3.17) become

$$y = (\dot{v} - pw + ru) - (g/V)\sin\phi\cos\theta \tag{3.18}$$

$$z = (\dot{w} + pv - qu) - (g/V)\cos\phi\cos\theta \tag{3.19}$$

$$l = \dot{p} + i_1 qr \tag{3.20}$$

$$m = \dot{q} - i_2 rp \tag{3.21}$$

$$n = \dot{r} + i_3 pq \tag{3.22}$$

We expend their left-hand side using Taylor series and obtain [35]:

$$\begin{aligned}
y &= \frac{\partial Y}{\partial \beta}\beta + \frac{\partial Y}{\partial \delta_a}\delta_a + \frac{\partial Y}{\partial \delta_r}\delta_r \\
z &= \frac{\partial Z}{\partial \alpha}\Delta\alpha + \frac{\partial Z}{\partial \delta_e}\delta_e \\
l &= \frac{\partial L}{\partial \beta}\beta + \frac{\partial L}{\partial p}p + \frac{\partial L}{\partial q}q + \frac{\partial L}{\partial r}r + \frac{\partial^2 L}{\partial r \partial \alpha}r\Delta\alpha + \frac{\partial^2 L}{\partial \beta \partial \alpha}\beta\Delta\alpha + \frac{\partial^2 L}{\partial \alpha \partial \delta_a}\Delta\alpha\delta_a + \frac{\partial Y}{\partial \delta_a}\delta_a + \frac{\partial Y}{\partial \delta_r}\delta_r \\
m &= \frac{\partial M}{\partial \alpha}\Delta\alpha + \frac{\partial M}{\partial q}q + \frac{\partial M}{\partial \dot{\alpha}}\Delta\dot{\alpha} + \frac{\partial M}{\partial \delta_e}\delta_e \\
n &= \frac{\partial N}{\partial \beta}\beta + \frac{\partial N}{\partial p}p + \frac{\partial N}{\partial q}q + \frac{\partial N}{\partial r}r + \frac{\partial^2 N}{\partial p \partial \alpha}p\Delta\alpha + \frac{\partial^2 N}{\partial \alpha \partial \delta_a}\Delta\alpha\delta_a + \frac{\partial N}{\partial \delta_a}\delta_a + \frac{\partial Y}{\partial \delta_r}\delta_r
\end{aligned} \tag{3.23}$$

To simplify the notation it is customary to make the following substitutions:

$$y_\beta = \frac{\partial Y}{\partial \beta}\ ,\ z_\alpha = \frac{\partial Z}{\partial \Delta\alpha}\ ,\ l_\beta = \frac{\partial L}{\partial \beta}\ ,\ m_\alpha = \frac{\partial M}{\partial \Delta\alpha}\ ,\ n_\beta = \frac{\partial N}{\partial \beta}\ ,\ \ldots$$

Substituting (3.23) into (3.18-3.22) and using equations (3.5), one obtains the nonlinear UAV model in state space form [22]:

$$\begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \\ \Delta\dot{\alpha} \\ \dot{\beta} \\ \dot{\phi} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} l_\beta\beta + l_q q + l_r r + (l_{\beta\alpha}\beta + l_{r\alpha}r)\Delta\alpha + l_p p - i_1 qr \\ \overline{m}_\alpha\Delta\alpha + \overline{m}_q q + i_2 pr - m_{\dot{\alpha}}p\beta + m_{\dot{\alpha}}(g/V)(\cos\theta\cos\phi - \cos\theta_0) \\ n_\beta\beta + n_q q + n_r r + n_{p\alpha}p\Delta\alpha + n_p p - i_3 pq \\ q - p\beta + z_\alpha\Delta\alpha + (g/V)(\cos\theta\cos\phi - \cos\theta_0) \\ y_\beta\beta + p(\sin\alpha_0 + \Delta\alpha) - r\cos\alpha_0 + (g/V)\cos\theta\sin\phi \\ p + q\tan\theta\sin\phi + r\tan\theta\cos\phi \\ q\cos\phi - r\sin\phi \end{bmatrix} + \begin{bmatrix} \bar{l}_{\delta_a} & l_{\delta_r} & 0 \\ 0 & 0 & \overline{m}_{\delta_e} \\ \bar{n}_{\delta_a} & n_{\delta_r} & 0 \\ 0 & 0 & z_{\delta_e} \\ y_{\delta_a} & y_{\delta r} & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} \delta_a \\ \delta_r \\ \delta_e \end{bmatrix}$$

$$= f(x) + g(x)u \tag{3.24}$$

where $\bar{l}_{\delta a} = l_{\delta a} + l_{\alpha\delta_a}\Delta\alpha$, $\bar{m}_\alpha = m_\alpha + m_{\dot{\alpha}}\Delta\alpha$, $\bar{m}_q = m_q + m_{\dot{\alpha}}$, $\bar{m}_{\delta e} = m_{\delta e} + m_{\dot{\alpha}} z_{\delta e}$, $\bar{n}_{\delta a} = n_{\delta a} + n_{\alpha\delta_a}\Delta\alpha$.

The state vector and the input vector are:

$$x = [p \quad q \quad r \quad \Delta\alpha \quad \beta \quad \phi \quad \theta]^T, \qquad u = [\delta_a \quad \delta_r \quad \delta_e]^T$$

where δ_a, δ_r and δ_e denote the deflections of the aileron, the rudder and the elevator.

In reference [22] and [26], the sideslip angle β, Euler angle ϕ and θ are chosen as the outputs. Therefore the UAV model (3.24) has three inputs and three outputs and thus is a square system.

In this thesis, in order to clearly illustrate the nonminimum phase problem associated with this model and verify our control algorithm, (3.24) is further simplified to a single-input-single-output (SISO) system and a two-input-two-output (TITO) system.

By choosing δ_a as the input and β as the output, and setting the other two inputs δ_r and δ_e to zero, system (3.24) becomes:

$$\begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \\ \Delta\dot{\alpha} \\ \dot{\beta} \\ \dot{\phi} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} l_\beta\beta + l_q q + l_r r + (l_{\beta\alpha}\beta + l_{r\alpha} r)\Delta\alpha + l_p p - i_1 qr \\ \bar{m}_\alpha\Delta\alpha + \bar{m}_q q + i_2 pr - m_{\dot{\alpha}} p\beta + m_{\dot{\alpha}}(g/V)(\cos\theta\cos\phi - \cos\theta_0) \\ n_\beta\beta + n_q q + n_r r + n_{p\alpha} p\Delta\alpha + n_p p - i_3 pq \\ q - p\beta + z_\alpha\Delta\alpha + (g/V)(\cos\theta\cos\phi - \cos\theta_0) \\ y_\beta\beta + p(\sin\alpha_0 + \Delta\alpha) - r\cos\alpha_0 + (g/V)\cos\theta\sin\phi \\ p + q\tan\theta\sin\phi + r\tan\theta\cos\phi \\ q\cos\phi - r\sin\phi \end{bmatrix} + \begin{bmatrix} \bar{l}_{\delta a} \\ 0 \\ \bar{n}_{\delta a} \\ 0 \\ y_{\delta a} \\ 0 \\ 0 \end{bmatrix} \delta_a$$

$$y = \beta \tag{3.25}$$

The system (3.25) is an SISO system. Although such a simplification may not occur in practice, it helps to develop and verify the method toward the solving of the nonminimum phase problem associated with the sideslip angle β.

Similarly, by choosing δ_a and δ_r as the inputs, β and ϕ as the outputs, and setting the third input δ_e to zero, system (3.24) becomes:

$$\begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \\ \Delta\dot{\alpha} \\ \dot{\beta} \\ \dot{\phi} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} l_\beta\beta + l_q q + l_r r + (l_{\beta\alpha}\beta + l_{r\alpha} r)\Delta\alpha + l_p p - i_1 qr \\ \overline{m}_\alpha \Delta\alpha + \overline{m}_q q + i_2 pr - m_{\dot{\alpha}} p\beta + m_{\dot{\alpha}}(g/V)(\cos\theta\cos\phi - \cos\theta_0) \\ n_\beta\beta + n_q q + n_r r + n_{p\alpha} p\Delta\alpha + n_p p - i_3 pq \\ q - p\beta + z_\alpha\Delta\alpha + (g/V)(\cos\theta\cos\phi - \cos\theta_0) \\ y_\beta\beta + p(\sin\alpha_0 + \Delta\alpha) - r\cos\alpha_0 + (g/V)\cos\theta\sin\phi \\ p + q\tan\theta\sin\phi + r\tan\theta\cos\phi \\ q\cos\phi - r\sin\phi \end{bmatrix} + \begin{bmatrix} \overline{l}_{\delta a} & l_{\delta r} \\ 0 & 0 \\ \overline{n}_{\delta a} & n_{\delta r} \\ 0 & 0 \\ y_{\delta a} & y_{\delta r} \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \delta_a \\ \delta_r \end{bmatrix}$$

$$y_1 = \beta; \quad y_2 = \phi \tag{3.26}$$

System (3.26) is a TITO system and represents the lateral channel of the UAV model.

The coefficients in systems (3.25) and (3.26) can be found in Table 3.1.

	Condition I (Nominal)	Condition II (perturbed)		Condition I (Nominal)	Condition II (perturbed)
l_β	-9.99	-20.91	n_q	0.223	0.222
l_q	0.107	0.108	n_r	-0.235	-0.377
l_r	0.126	0.221	n_p	0.002	0.013
l_p	-3.933	-5.786	$n_{\delta r}$	-6.51	-8.30
$l_{\delta r}$	-7.64	-10.05	$n_{\delta a}$	-0.921	-1.282
$l_{\delta a}$	-45.83	-60.27	$n_{p\alpha}$	-1.578	-1.583
$l_{r\alpha}$	8.39	13.16	$n_{\alpha\delta a}$	1.132	2.459
$l_{\beta\alpha}$	-684.40	-543.80	z_α	-1.329	-1.746
$l_{\alpha\delta a}$	63.50	64.60	$z_{\delta e}$	-0.168	-0.224
m_α	-23.18	-10.70	g/V	0.0345	0.0412
m_q	-0.814	-1.168	y_β	-0.196	-0.280
$m_{\dot{\alpha}}$	-0.173	-0.251	$y_{\delta r}$	0	0
$m_{\delta e}$	-28.37	-31.64	$y_{\delta a}$	0.0071	0.0119
n_β	5.67	8.88			

Note: α_0=1.5°, θ_0=0, i_1=0.727, i_2=0.949, i_3=0.716.

Table 3.1 Flight conditions (from reference [22])

In following chapters, our control algorithm will be developed and tested based on systems (3.25) and (3.26).

Chapter 4

Feedback Linearization and the Nonminimum Phase Problem

In this chapter, linear control schemes for aircraft control are reviewed, and a brief introduction to feedback linearization is also presented. However, the emphasis is given to the output redefinition method solving for nonminimum phase problem.

4.1 Review of Linear Control Laws for Flight Control Design

Linear control laws have been applied to design flight control systems for many years, among which the PID control is a basic one and has been extensively used. The block diagram of pitch angle control with PID controller is shown in Figure 3.1.

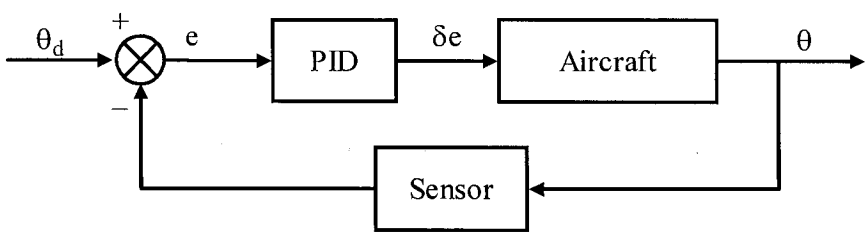

Figure 4.1 Block diagram of pitch angle control using PID controller

We can see from figure 4.1 that in order to obtain that the system output, the pitch angle θ, follows a desired pitch angle θ_d, the output θ is fed back from the sensor and compared with θ_d. The error e between θ_d and θ is then fed into the PID controller to form the elevator deflection command δ_e. The transfer function of the PID controller is

$$\delta_e(t) = K_P e(t) + K_I \int e(t)\ dt + K_D \frac{de(t)}{dt} \tag{4.1}$$

where K_P, K_I and K_D are respectively the proportional, integral and derivative gains of the PID controller. Similar control schemes are also applied to roll and yaw angle control

(See Figure 4.2 and 4.3).

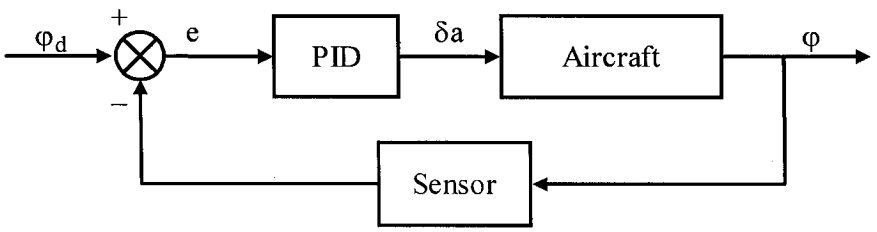

Figure 4.2 Block diagram of roll angle control using PID controller

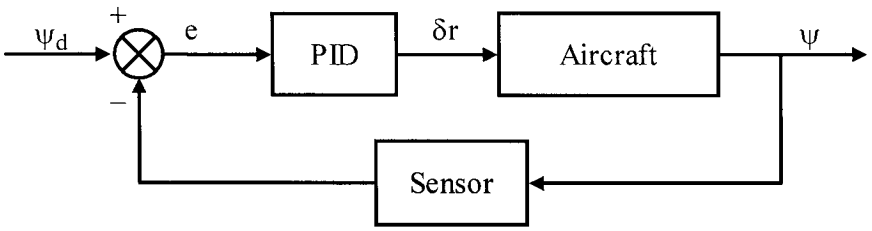

Figure 4.3 Block diagram of yaw angle control using PID controller

From above control frameworks, we can see that PID control is easy to implement and, like other linear control techniques, offers the advantage of easy tuning the transient response of the system. However, the gains of the controller are selected by trial and error method which should be avoided in aircraft control. Moreover, basic classic control techniques can only handle independent single input single output loops, i.e. SISO systems. To control an MIMO system using classical control techniques, one must neglect the coupling of state variables and approximate the system as a set of SISO systems. This approach is also known as decoupling control. However, since the aircraft is generally an MIMO system with heavily coupled state variables, simply approximating it as a set of SISO systems will definitely sacrifice its performance.

Modern control techniques, based on the state-space form of a system, are available to handle more than one input and/or output. Pole-placement is one of the modern control techniques. It allows us to assign the closed-loop poles to desired locations in one step by

solving equations for the feedback gains. A brief summary to this technique is as follows.

A state-space form of a linearized aircraft model may be written as

$$\begin{aligned} \dot{x} &= Ax + Bu \\ y &= Cx, \end{aligned} \tag{4.2}$$

where $x \in \Re^n$ is the state vector, $u \in \Re^m$ is the input vector, and $y \in \Re^p$ is the vector of measured output.

One may select a feedback control input as

$$u = -Kx + r, \tag{4.3}$$

where r is a reference input and K is an $m \times n$ gain matrix to be determined.

Substituting equation (4.3) into (4.2) yields the closed-loop system

$$\dot{x} = (A - BK)x + Br = \overline{A}x + Br \tag{4.4}$$

where $\overline{A}$ is the closed-loop plant matrix. The system (4.2) is controllable, if the controllability matrix $\begin{bmatrix} B & AB & A^2B & \cdots & A^{n-1}B \end{bmatrix}$ has full rank of n. As a result, it is easy to find the gain matrix K using computational software, such that the eigenvalues of matrix $\overline{A}$ will be placed precisely at specified locations.

Linear quadratic regulator (LQR) is another modern control technique which has been applied to flight control. In this approach, the gain matrix K is selected to minimize a quadratic cost function or performance index (PI) which is created according to certain performance criterion.

Modern control techniques discussed above offer us the ability to close all the feedback loops simultaneously by solving the matrix equations for the gain matrix K. They also guarantee the stability of the closed-loop system. These two fundamental properties make them very useful for aircraft control systems.

The aircraft model we used to design a linear controller is linearized from a

nonlinear model at certain flight condition. However, the dynamic pressure changes significantly at different heights and speeds, which causes large variations in the coefficients of the dynamic equations. Consequently, the control gains which have been selected to stabilize the aircraft at one flight condition may not valid for another. If the control gains are left fixed in the full flight envelope, then the closed-loop response will be different from what is expected.

To overcome this deficiency, gain scheduling is frequently used. In this approach, a set of linearized dynamic models are established for different flight conditions, then linear control tools discussed above can be used to design individual compensators to achieve closed-loop specifications. Next, these individual compensators are linked together with gain schedules to cover the full flight envelope. When a smoothly gain-scheduled controller is required, the linear controller designs are selected to have compatible structures which permit smooth interpolation between the designs.

The gain scheduling step is, however, time consuming, costly to iterate and not always successful for some flight missions when an aircraft is at considerable height and is rapidly climbing or diving, and when an aircraft is performing rapid maneuvers involving large angles of attack.

In summary, the performance achieved by linear design methodologies is not satisfactory for high-performance aircrafts due to the inherent nonlinearity, instability and large maneuverability of these vehicles. As the flight control applications become more and more complex, powerful nonlinear controllers are required.

4.2 Feedback Linearization

As mentioned above, in order to control highly maneuverable aerial vehicles, nonlinear flight control methodologies must be adopted. Over the last two decades, researchers have begun to apply feedback linearization or dynamic inversion to flight control design. The central idea of this approach is to transform nonlinear dynamics into a linear form by using state feedback and canceling out the nonlinearities. The feedback linearization controller takes into account the nonlinearities of the aircraft, and thus does not need gain-scheduling. It is suitable for a wide range of operating conditions, including high-angle-of-attack and hypervelocity design. A brief introduction about this approach is given as follows.

Consider an affine nonlinear time invariant system of the form

$$\begin{aligned} \dot{x} &= f(x) + g(x)u \\ y &= h(x) \end{aligned} \tag{4.5}$$

where state $x \in \Re^n$, control input $u \in \Re^m$, and output $y \in \Re^p$. $f(x)$, $g(x)$ and $h(x)$ are nonlinear functions and assumed to be smooth. The entire state $x(t)$ is observable for feedback purposes.

It is also assumed that the system is square, that is, the number of inputs m is equal to the number of outputs p so that vectors $u(t)$ and $y(t)$ have the same dimension.

To make the system follow a desired trajectory $y_d(t)$, the tracking error is defined as:

$$e(t) = y_d(t) - y(t) \tag{4.6}$$

To implement an input-output feedback linearization, one differentiates the output y until the control input $u(t)$ appears in the expression for the derivative. Taking the first derivative yields

$$\dot{y} = \frac{\partial h}{\partial x}\dot{x} = \frac{\partial h}{\partial x} f(x) + \frac{\partial h}{\partial x} g(x) \cdot u = F(x) + G(x) \cdot u \tag{4.7}$$

For aircraft, it is usually the case that $G(x)$ is nonzero, so we have the equation (4.7) clearly representing the relationship between $y(t)$ and $u(t)$.

We introduce a pseudo-control variable v as follows:

$$v = F(x) + G(x) \cdot u \tag{4.8}$$

Solving for u in equation (4.8), we obtain

$$u = G^{-1}(x) \cdot [v - F(x)] \tag{4.9}$$

Substituting equation (4.9) into (4.7) results a simple linear relationship between the output y and the new input v,

$$\dot{y} = v \tag{4.10}$$

The nonlinearities in (4.7) are thus canceled. The pseudo-control variable v can be obtained by using linear control techniques, including robust control, H-infinity, or LQR. For example, the variable v can be chosen as:

$$v = K(y_d - y) + \dot{y}_d \tag{4.11}$$

where y_d is the vector of desired outputs, K is the gain matrix to be determined.

Substituting equation (4.11) into (4.10) yields the closed-loop error dynamics

$$\dot{e} = -Ke \tag{4.12}$$

which is stable as long as the gain matrix K is positive definite.

The overall feedback linearization controller is given by

$$u = G^{-1}(x) \cdot [Ke + \dot{y}_d - F(x)] \tag{4.13}$$

The control scheme given by (4.13) is also shown in Figure 4.4. From Figure 4.4 we can see that there are two control loops, the inner feedback linearization loop and the outer tracking loop. The feedback linearization loop described by (4.9) makes the system from v to y appear like a linear system. The outer tracking loop described by (4.12) simply contains a feedback gain matrix K. There is also a feedforward term $\dot{y}_d$, which is

known as *velocity feedforward.* It greatly improves the tracking accuracy of the closed-loop system.

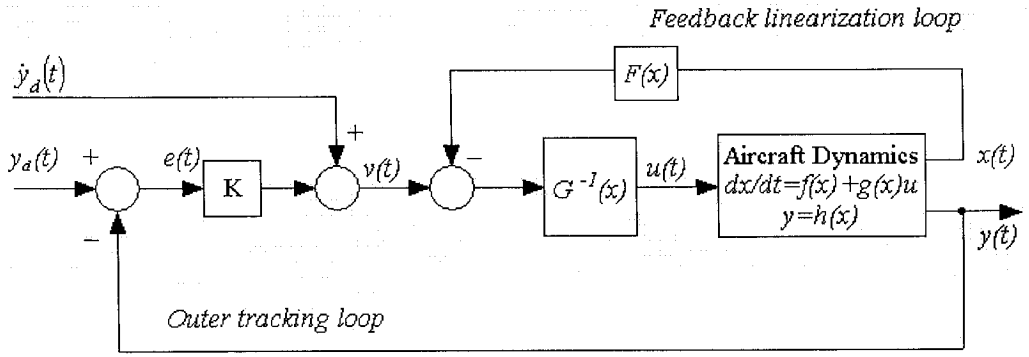

Figure 4.4 Feedback linearization control

It must be noted that the nonlinear functions *F(x)* and *G(x)* enter into the controller (4.13). Therefore, to implement the feedback linearization scheme, one must exactly know the nonlinear aircraft model described in form of (4.5), and all states must be available to feedback.

If the system outputs are selected to be exactly the system states, i.e. $y=x$, then the feedback linearization controller in (4.13) becomes

$$u = g^{-1}(x)\cdot[Ke + \dot{x}_d - f(x)] \tag{4.14}$$

where x_d is the vector of desired states, $e = x_d - x$ is the tracking error.

However, a nonlinear model of an aircraft normally has more states than inputs or outputs, which makes the inverse of *g(x)* nonsquare. One solution to this problem is to use state transformation, but the whole process and resulting equations are very complex, which is inconvenient for practical use. In practice, separate inner and outer loop inversions are employed based on two-time scale approximation normally inherent in aircraft dynamics. For aircraft attitude, the Euler angles ϕ, θ and sideslip angle β are

deemed as "slow" dynamics because the control effectiveness their dynamics are quite slow. While the angular rates p, q and r are considered "fast" dynamics because the control effectiveness on these body rates is high. The two-time scale method thus reformulates the original differential equation (4.5) into two sets of differential equations, one for the slow dynamics, and one for fast dynamics, which are shown below.

$$\dot{x}_s = f(x_s) + g(x_s)x_f \tag{4.15}$$

$$\dot{x}_f = h(x_s, x_f) + k(x_s, x_f)u \tag{4.16}$$

where x_s represents the slow states ϕ, θ and β, and x_f represents the fast states p, q and r.

Note that in equation (4.15), rate variables x_f form inputs for the slow dynamics, while the actual control surface commands u form inputs for the rate dynamics shown in equation (4.16). Since both the slow dynamics (4.15) and the fast dynamics (4.16) are square (the number of controls equals the number of degree of freedom), they can be inverted to generate the two feedback linearization control laws, the slow one (4.17), and the fast one (4.18).

$$x_{f\,cmd} = g^{-1}(x_s) \cdot [\dot{x}_{s\,cmd} - f(x_s)] \tag{4.17}$$

$$u = k^{-1}(x_s, x_f) \cdot [\dot{x}_{f\,cmd} - h(x_s, x_f)] \tag{4.18}$$

where x_{scmd} is the vector of the commanded slow states; x_{fcmd} is the vector of commanded fast states generated by the inversion of the slow dynamics.

A block diagram of this two-time scale approach is shown in Figure 4.5. From Figure 4.5 we can see that the slow inversion block produces the commanded rate variables using the slow inverse dynamics of (4.17). Then the commanded rate variables are fed to the fast inversion block. Using the fast inverse dynamics (4.18), the fast

inversion block produces the commanded control deflections which serve as the input to the aircraft dynamics.

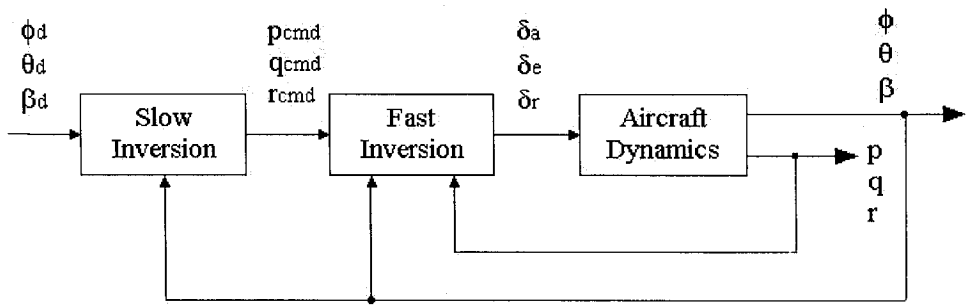

Figure 4.5 Block diagram of the two time scale approach

4.3 Zero -Dynamics and the Nonminimum Phase Problem

4.3.1 Relative Degree and Internal Dynamics

Consider the following SISO nonlinear system of the form

$$\begin{aligned} \dot{x} &= f(x) + g(x)u \\ y &= h(x) \end{aligned} \tag{4.19}$$

where state $x \in \Re^n$, control input $u \in \Re^1$, and output $y \in \Re^1$. *f(x)*, *g(x)* and *h(x)* are nonlinear functions and assumed to be smooth.

If it is needed to differentiate the system output r times to obtain a linear relationship between the output y and input u, i.e.

$$y^{(r)} = L_f^r h + L_g L_f^{r-1} h u \tag{4.20}$$

where $L_g L_f^{r-1} h \neq 0$, then the system is said to have **relative degree** r. This terminology is quite consistent with the linear case [36, p74-75]. For any controllable systems, it will take at most n differentiations on the output y to make the input u appear, i.e. $r \leq n$.

If the pseudo control v is defined as

$$v = L_f^r h + L_g L_f^{r-1} h u \quad , \tag{4.21}$$

then solving for the control input u, we have

$$u = \frac{1}{L_g L_f^{r-1} h}\left(v - L_f^r h\right) \tag{4.22}$$

Substituting (4.21) into (4.19) yields a linear differential relation between y and v

$$y^{(r)} = v \tag{4.23}$$

Equation (4.23) is of order r, while the plant model (4.19) has an order n. Therefore, the input-output feedback linearization only accounts for a part of the system dynamics of order r. The remaining part of the system dynamics (n-r order) has been rendered unobservable in the input-output feedback linearization. This part of the system dynamics is called the **internal dynamics** which can be derived using state transformation.

For the r components of the state vector of system (4.19), we can define: $\xi = [\xi_1, \xi_2, \ldots, \xi_r]^T = [y, \dot{y}, \ldots, y^{(r-1)}]^T$, and the rest n-r components of state vector can be chosen as $\eta = [\eta_1, \eta_2, \ldots, \eta_{n-r}]^T = [\eta, \dot{\eta}, \ldots, \eta^{(n-r-1)}]^T$, such that $\phi(x) = [\xi_1, \xi_2, \ldots, \xi_r, \eta_1, \eta_2, \ldots, \eta_{n-r}]^T$ is a local diffeomorphism. Then the dynamics of system (4.19) can be transformed into (ξ, η)-coordinates as follows:

$$\xi^{(r)} = a(\xi, \eta) + b(\xi, \eta)u(t) \tag{4.24}$$

$$\eta^{(n-r)} = p(\xi, \eta) + q(\xi, \eta)u(t) \tag{4.25}$$

with the output defined as $y=\xi_1$. By substituting (4.24) into (4.25), we obtain the internal dynamics

$$\eta^{(n-r)} = p(\xi, \eta) + \frac{q(\xi, \eta)\left[\xi^{(r)} - a(\xi, \eta)\right]}{b(\xi, \eta)}. \tag{4.26}$$

If the internal dynamics is unstable, the tracking controller will be unable to stabilize the system, because the control signal u will diverge exponentially, i.e. the inverse dynamics will be unbounded, which is practically not acceptable. Therefore, it is necessary to determine the stability of the internal dynamics of a system before applying

input-output feedback linearization.

4.3.2 Definition of Zero-Dynamics and Nonminimum Phase Systems

Since the internal dynamics is nonlinear, non-autonomous, and coupled with the external dynamics, it is very difficult to directly determine the stability of the internal dynamics. To simplify the problem, one usually studies the stability of the zero-dynamics instead of internal dynamics. The zero-dynamics of a nonlinear system is the internal dynamics of the system when input has been chosen to constrain the output to remain identically zero. This can be viewed as a special case of the internal dynamics of the system. For the nonlinear system (4.19) with its internal dynamics described by (4.26), the zero-dynamics is

$$\eta^{(n-r)} = p(0,\eta) - \frac{q(0,\eta)a(0,\eta)}{b(0,\eta)} \tag{4.27}$$

From equation (4.27) we can see that examining the stability of the zero-dynamics is much easier than examining the stability of the internal dynamics, because the zero-dynamics only involves the internal states.

It is interesting to compare the zeros of the transfer function of a linear system and the zero-dynamics of nonlinear system. J. E. Slotine and W. Li [7] have proved that the internal dynamics of a linear system is stable if the plant zeros are in the left-half of s plane (LHP). In other words, for a linear system, the stability of the internal dynamics is simply determined by the locations of the zeros. For nonlinear system, the stability of the zero-dynamics implies the local stability of the internal dynamics. No global stability of the internal dynamics is guaranteed from the zero-dynamics.

In a linear system, it has been proved that the eigenvalues of the zero-dynamics are exactly the zeros of the system [7]. The feedback control law only affects the closed-loop

poles of the system, and has no influence on zeros. In a nonlinear system, the eigenvalues of the linearized zero-dynamics coincide with the zeros of the Jacobian linearization of the system [36]. The zero-dynamics of the nonlinear system is an intrinsic feature of that system, which does not depend on the choice of the control law and the desired trajectory [7].

A linear system is said to be nonminimum phase if it has right-half of s plane (RHP) zeros. Similarly, a nonlinear system is defined nonminimum phase if its zero-dynamics is unstable. Nonminimum phase systems present "undershoot" to step response. Here the "undershoot" means the transient response to a step input starts off in a wrong direction. Examples 4.1 and 4.2 show this phenomenon in linear and nonlinear cases.

Example 4.1 The step response of a nonminimum phase linear system

Consider the linear system described by the transfer function

$$G(s) = -\frac{s-1}{s^2 + 2s + 2} \tag{4.28}$$

which has a real RHP zero (or positive zero) at $s=1$. The system response to a step input is shown in Figure 4.6. We can see that the system response starts in an opposite direction of the step input.

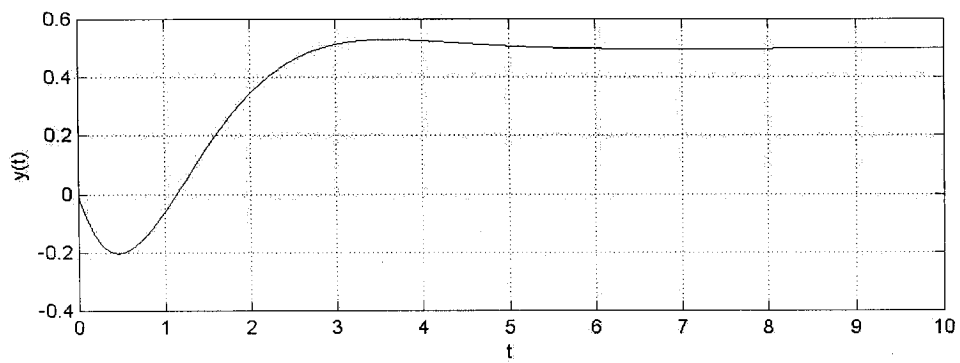

Figure 4.6 The step response of a nonminimum phase linear system

Example 4.2 The step response of a nonminimum phase nonlinear system

Consider the nonlinear system described by the state space form

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} -2x_1 - 2x_2 + u \\ x_1 + x_1 x_2 + u \end{bmatrix} \tag{4.29}$$

$$y = -x_1$$

To derive zero-dynamics of system (4.29), the first step is to differentiate its output until the input term appears, which gives

$$\dot{y} = -\dot{x}_1 = 2x_1 + 2x_2 - u \tag{4.30}$$

Let the system output be identically zero. We have

$$y = -x_1 = 0\,,\quad \dot{y} = -\dot{x}_1 = 0\,, \text{and}\quad u = 2x_1 + 2x_2 = 2x_2 \tag{4.31}$$

Substituing (4.31) into (4.29) yields the zero-dynamics $\dot{x}_2 - 2x_2 = 0$, which is unstable. Therefore, system (4.29) is a nonminimum phase system. Figure 4.7 shows its step response.

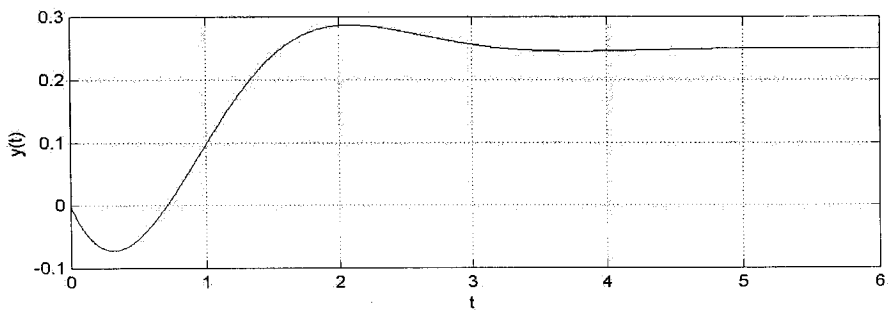

Figure 4.7 The step response of a nonminimum phase nonlinear system

Similar to the nonminimum phase linear system, the nonminimum phase nonlinear system also has an "undershoot" in its step response.

From above discussion we can see that the definition and the property of a nonminimum phase nonlinear system are much similar to their linear counterpart. Most importantly, perfect tracking of nonminimum phase linear or nonlinear systems always requires infinite control effort.

4.3.3 The Zero-Dynamics of UAV Models

As we have discussed above, the invertibility of a nonlinear system is locally determined by the stability of its zero-dynamics. Thereby, before we design a feedback control law for the UAV model (3.25) and (3.26), their zero-dynamics must be studied.

a) Zero-Dynamics of the UAV Model (3.25)

For the SISO nonlinear UAV model (3.25), its zero-dynamics can be derived as follows.

First, differentiating the output y of (3.25) yields the explicit relation between the output y and the input u.

$$\dot{y} = \dot{\beta} = y_{\beta} + p(\sin\alpha_0 + \Delta\alpha) - r\cos\alpha_0 + (g/V)\cos\theta\sin\phi + y_{\delta a}\delta_a \tag{4.32}$$

Secondly, let

$$y = \beta = 0 \tag{4.33}$$

and

$$\dot{y} = \dot{\beta} = y_{\beta} + p(\sin\alpha_0 + \Delta\alpha) - r\cos\alpha_0 + (g/V)\cos\theta\sin\phi + y_{\delta a}\delta_a = 0 \tag{4.34}$$

Solving for δ_a, we have

$$\delta_a = [-p(\sin\alpha_0 + \Delta\alpha) + r\cos\alpha_0 - (g/V)\cos\theta\sin\phi]/y_{\delta a} \quad . \tag{4.35}$$

Since the system (3.25) has a dimension of 7, and the output y is only differentiated once, there must be an internal dynamics of order 6. However it is unnecessary to use the state transformation to find the internal dynamics and then the zero-dynamics. Simply substituting equations (4.33)-(4.35) into (3.25) will give the nonlinear zero-dynamics.

$$\begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \\ \Delta\dot{\alpha} \\ \dot{\phi} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} l_q q + l_r r + l_{r\alpha} r\Delta\alpha + l_p p - i_1 qr + \bar{l}_{\delta a}(-p(\sin\alpha_0 + \Delta\alpha) + r\cos\alpha_0 - (g/V)\cos\theta\sin\phi)/y_{\delta a} \\ \bar{m}_{\alpha}\Delta\alpha + \bar{m}_q q + i_2 pr + m_{\dot{\alpha}}(g/V)(\cos\theta\cos\phi - \cos\theta_0) \\ n_q q + n_r r + n_{p\alpha} p\Delta\alpha + n_p p - i_3 pq + \bar{n}_{\delta a}(-p(\sin\alpha_0 + \Delta\alpha) + r\cos\alpha_0 - (g/V)\cos\theta\sin\phi)/y_{\delta a} \\ q + z_{\alpha}\Delta\alpha + (g/V)(\cos\theta\cos\phi - \cos\theta_0) \\ p + q\tan\theta\sin\phi + r\tan\theta\cos\phi \\ q\cos\phi - r\sin\phi \end{bmatrix} \tag{4.36}$$

The zero-dynamics (4.36) is nonlinear and of high order. Directly determining its stability is still quite difficult. A simpler way to do this is to check the eigenvalues of the linearization of the nonlinear zero-dynamics. If all its eigenvalues are negative, the zero-dynamics of the linearized system is stable. If some eigenvalues are positive, both the linearized and the nonlinearized zero-dynamics are unstable.

Calculating the Jacobian linearization of (4.36) about $x=0$ gives the linearized zero-dynamics

$$\begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \\ \Delta\dot{\alpha} \\ \dot{\phi} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} l_p - \sin\alpha_0 l_{\delta a}/y_{\delta a} & l_q & l_r + \cos\alpha_0 l_{\delta a}/y_{\delta a} & 0 & -(g/V)l_{\delta a}/y_{\delta a} & 0 \\ 0 & \overline{m}_q & 0 & \overline{m}_\alpha & 0 & 0 \\ n_p - \sin\alpha_0 n_{\delta a}/y_{\delta a} & n_q & n_r + \cos\alpha_0 n_{\delta a}/y_{\delta a} & 0 & -(g/V)n_{\delta a}/y_{\delta a} & 0 \\ 0 & 1 & 0 & z_\alpha & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} p \\ q \\ r \\ \Delta\alpha \\ \phi \\ \theta \end{bmatrix} \tag{4.37}$$

Substituting the aerodynamic parameters of flight condition I listed in Table 3.1 into (4.37), we obtain the eigenvalues of the linearized zero-dynamics in nominal condition:

$$[0 \quad -1.158+4.7876i \quad -1.158-4.7876i \quad 24.4583 \quad 10.3755 \quad 0.2085].$$

Since the linearized zero-dynamics has three positive eigenvalues, according to the Lyapunov's linearization method [7, page 55], the nonlinear zero-dynamics (4.36) is determined to be unstable.

By using the aerodynamic parameters of flight condition II in Table 3.1, another set of eigenvalues for the linearized zero-dynamics in perturbed condition is obtained:

$$[0 \quad -1.5825\pm 3.1993i \quad 9.3259\pm 18.4759i \quad 0.1859].$$

The zero-dynamics of system (3.25) in perturbed condition is also unstable due the three positive eignvalues. Therefore, the nonlinear UAV model (3.25) is recognized as a nonminimum phase system.

b) Zero-Dynamics of the UAV Model (3.26)

The UAV model (3.26) has two inputs and two outputs. In order to find its zero-dynamics, one must differentiate each output until the input appears.

First, differentiating the output $y_1=\beta$ of (3.26) yields the explicit relation between the first output β and the first input δ_a.

$$\dot{y}_1 = \dot{\beta} = y_\beta + p(\sin\alpha_0 + \Delta\alpha) - r\cos\alpha_0 + (g/V)\cos\theta\sin\phi + y_{\delta a}\delta_a \tag{4.38}$$

Then, differentiating the second output $y_2=\phi$ we have

$$\dot{y}_2 = \dot{\phi} = p + q\tan\theta\sin\phi + r\tan\theta\cos\phi \tag{4.39}$$

Since the input does not appear in (4.39), we have to differentiat (4.39) again.

$$\begin{aligned}\ddot{y}_2 = \ddot{\phi} &= \dot{p} + \dot{q}\tan\theta\sin\phi + \dot{\theta}\left(q\sin\phi/\cos^2\theta + r\cos\phi/\cos^2\theta\right) + \dot{r}\tan\theta\cos\phi \\ &\quad + \dot{\phi}(q\tan\theta\cos\phi - r\tan\theta\sin\phi) \\ &= l_\beta\beta + l_q q + l_r r + (l_{\beta\alpha}\beta + l_{r\alpha}r)\Delta\alpha + l_p p - i_1 qr + \bar{l}_{\delta a}\delta_a + l_{\delta r}\delta_r \\ &\quad + \dot{q}\tan\theta\sin\phi + \dot{\theta}\left(q\sin\phi/\cos^2\theta + r\cos\phi/\cos^2\theta\right) + \dot{r}\tan\theta\cos\phi \\ &\quad + \dot{\phi}(q\tan\theta\cos\phi - r\tan\theta\sin\phi)\end{aligned} \tag{4.40}$$

We stop differentiating here, because the control inputs δ_a and δ_r appear in the equation (4.40). The relative degree of an MIMO system is the sum of the relative degree associated with each output [7, page 264]. Therefore, the relative degree of system (3.26) is 3, and the zero-dynamics must be order 4.

Then let $y_1 = \dot{y}_1 = y_2 = \dot{y}_2 = \ddot{y}_2 = 0$, we have:

$$\delta_a = \left[-p(\sin\alpha_0 + \Delta\alpha) + r\cos\alpha_0\right]/y_{\delta a} \tag{4.41}$$

$$\delta_r = \left[-l_q q - l_r r - l_{r\alpha} r\Delta\alpha + i_1 qr - \bar{l}_{\delta a}\delta_a - qr/\cos^2\theta\right]/l_{\delta r} \tag{4.42}$$

Substituting equations (4.41)-(4.42) into (3.26) gives the nonlinear zero-dynamics of the system.

$$\begin{bmatrix} \dot{q} \\ \dot{r} \\ \Delta\dot{\alpha} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \overline{m}_\alpha \Delta\alpha + \overline{m}_q q + m_{\dot{\alpha}}(g/V)(\cos\theta - \cos\theta_0) \\ n_q q + n_r r + \dfrac{\overline{n}_{\delta a} r \cos\alpha_0}{y_{\delta a}} + n_{\delta r}\left[-l_q q - l_r r - l_{r\alpha} r\Delta\alpha + i_1 qr - \dfrac{\overline{l}_{\delta a} r\cos\alpha_0}{y_{\delta a}} - \dfrac{qr}{\cos^2\theta} \right] \Big/ l_{\delta r} \\ q + z_\alpha \Delta\alpha + (g/V)(\cos\theta - \cos\theta_0) \\ q \end{bmatrix} \tag{4.43}$$

Calculating the Jacobian linearization of (4.43) about $x=0$ gives the linearized zero-dynamics

$$\begin{bmatrix} \dot{q} \\ \dot{r} \\ \Delta\dot{\alpha} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \overline{m}_q & 0 & \overline{m}_\alpha & 0 \\ n_q - \dfrac{n_{\delta r} l_q}{l_{\delta r}} & n_r + \dfrac{n_{\delta a}\cos\alpha_0}{y_{\delta a}} - \left[l_r + \dfrac{l_{\delta a}\cos\alpha_0}{y_{\delta a}} \right] \dfrac{n_{\delta r}}{l_{\delta r}} & 0 & 0 \\ 1 & 0 & z_\alpha & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} q \\ r \\ \Delta\alpha \\ \theta \end{bmatrix} \tag{4.44}$$

Substituting the aerodynamic parameters of the flight condition I listed in Table 3.1 into (4.44), we obtain the eigenvalues of the linearized zero-dynamics in nominal condition

$$[0 \quad -1.158 + 4.7876i \quad -1.158 - 4.7876i \quad 5368.31]$$

Since the linearized zero-dynamics has a positive eigenvalues located at $s= 5368.31$, the nonlinear zero-dynamics (4.43) is determined to be unstable.

Using the aerodynamic parameters of flight condition II in Table 3.1, one can find another set of eigenvalues for the linearized zero-dynamics in perturbed condition

$$[0 \quad -1.5825 + 3.1993i \quad -1.5825 - 3.1993i \quad 4073.1]$$

The positive eigenvalue $s=4073.1$ make the system (3.26) also nonminimum phase in the perturbed condition. Therefore, the nonlinear UAV model (3.26) is also recognized as a nonminimum phase system.

We can see that for different flight conditions, the UAV system has different eigenvalues of the linearized zero-dynamics. However, how these eigenvalues change

with flight conditions is still unclear at this stage of our research.

We also observed that it is the term $y_{\delta a}$ in (3.25) and (3.26) that makes the lateral channel of the systems behave nonminimum phase. If we neglect this term from equations (3.25) and (3.26), then the zero-dynamics of these two systems will become stable in the neighbourhood of $x=0$. The term $y_{\delta a}$ represents the small force produced by the actuator—aileron. Generally the actuators, such as the aileron, elevator and rudder, are moment-producing devices, but they produce small forces acting on the aircraft body as well. It is this type of forces that make the fixed-wing aerial vehicles nonminimum phase systems. Many studies [20, 21, 37, 38] have shown that this is not only true for conventional take-off and landing (CTOL) aircrafts, but also common in vertical take-off and landing (VTOL) aircrafts and helicopters. Here, CTOL means the process of take-off and landing using the runways. VTOL describes aircrafts that can liftoff vertically, but helicopters are not included.

Example 4.3 is a case study on a nonminimum phase helicopter model.

Example 4.3 A Nonminimum Phase Helicopter Model

A nonlinear helicopter model in state space form is given by [37]:

$$\begin{bmatrix} \dot{P} \\ \dot{v}^p \\ \dot{\Theta} \\ \dot{\omega}^b \\ \dot{T}_M \\ \dot{T}_T \\ \dot{a}_{1s} \\ \dot{b}_{1s} \end{bmatrix} = \begin{bmatrix} v^p \\ \frac{1}{m} R f^b \\ \Psi \omega^b \\ I^{-1}\left(\tau^b - \omega^b \times I\omega^b\right) \\ w_1 \\ w_2 \\ w_3 \\ w_4 \end{bmatrix} \tag{4.45}$$

$$y = \left[p_x, p_y, p_z, \theta\right]$$

where $w = [w_1, w_2, w_3, w_4]^T$ are defined as auxiliary inputs to the system, $\boldsymbol{P}$ is the position

vector in earth-axis frame, v^p is the vector of velocity in earth-axis frame, $\Theta = [\phi, \theta, \psi]^T$ are Euler angles, $\omega^b = [p, q, r]^T$ are the body angular velocities, T_M and T_T are the thrust forces of the main rotor and the tail rotor, a_{1s} and b_{1s} in (4.45) are the longitudinal and lateral tilt of the main rotor used for torque control, f_b and τ_b are the body force and torque vectors, respectively, I is the inertia matrix, and finally, R and Ψ are the transform matrices. Here the states $x \in \Re^{16}$, the input $w \in \Re^4$, and the output $y \in \Re^4$. Therefore, the inpout-output system is square.

The body force f_b is a combination of T_M and T_T, as well as the gravity $G=mg$. Therefore, the force equation of the helicopter in body-axis frame can be written as

$$f^b = \begin{bmatrix} X_M \\ Y_M + T_T \\ Z_M \end{bmatrix} + R^T \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix} \tag{4.46}$$

where $[X_M, Y_M, Z_M]$ are the components of the T_M in body-axis frame. Since we have

$$\begin{aligned} X_M &= -T_M \sin a_{1s} \\ Y_M &= T_M \sin b_{1s} \\ Z_M &= -T_M \cos a_{1s} \cos b_{1s} \end{aligned} , \tag{4.47}$$

thus the force of the helicopter is affected by the tilt of the main rotor.

To derive the zero-dynamics of the system (4.45), one must differentiate each output three times to generate an explicit relation between y and w. By setting the output y to zero, we obtain a nonlinear zero-dynamics of order 4. Then a positive eigenvalue $s=21.107$ can be found by linearizing the zero-dynamics about $x=0$. Therefore the helicopter model is nonmunimum phase. For more details about the helicopter model (4.45) and its zero-dynamics, refer to [37].

To sum up, the UAV models (3.25) and (3.26) are both nonminimum phase due to their unstable zero-dynamics; therefore, the feedback linearization control law cannot be

directly applied to these two systems. Since the nonminimum phase problem is caused by the small force produced by the actuator, one may neglect the term $y_{\delta a}$ in system equations if it is small, but the stability of the new zero-dynamics is not guaranteed.

4.4 The Output Redefinition Method

4.4.1 Introduction

From the equations (4.24-4.27) we can see that the zero-dynamics or the internal dynamics is not only an intrinsic property of a nonlinear system, but also a function about the internal states and the chosen system output. Therefore, when we choose different outputs for the system, we will obtain different zero-dynamics. Inspired from this observation, many researches have been conducted to solve the nonminimum phase problem by defining a new output such that the resulting zero-dynamics is stable. Since this method only changes the output, a dynamic state variable compensator that asymptotically stabilize the modified system, will also asymptotically stabilize the original non-minimum phase one. This approach is called **output redefinition**.

In [24], tracking the normal acceleration a_n of a missile is substituted by tracking a new variable a_n^* defined as

$$a_n^* = a_n + b\dot{q} + c\dot{\alpha}$$

where the variables b and c are selected to ensure that the normal acceleration and the pitch moment remain bounded during the transient maneuvers.

Similarly in [8] and [13], the angular velocities of a fighter aircraft are defined as

Pitch axis:	$q + n_{zp}/V_{co} + kV_T$
Roll axis:	$p + \alpha$
Yaw axis:	$r - \alpha p - g\sin\phi\cos\theta/V + k\beta$

where k is a gain to be selected to stabilize the zero-dynamics.

These selections of the outputs are suitable for most conventional flight regimes and piloting tasks. However, they may need modifications for high-angle-of-attack or very-low-speed flight. Further more, the selection of control variables (CVs) still relies on trial and error to some extent.

Since for linear systems, the zeros of the transfer function are equivalent to the internal dynamics, J. E.Slotine and W. Li [7] proposed that the output redefinition can be implemented in a way such that the positive zeros are removed from the transfer function, and thus the system has stable internal dynamics and can be inverted. Compared with the approaches discussed above, this technique has many advantages. First, it does not involve any trial and error. Secondly, it offers us the freedom of manipulating the zeros of the system. Thirdly, since one only removes the positive zeros, and the remainder negative zeros are left unchanged, the new output is essentially the same as the original one in the frequency range of interest.

It should be noted that this technique is developed for the SISO linear systems, but it can be generalized for MIMO linear systems. For implementing this technique to nonlinear systems, Jacobian linearization about the operating point is needed. In this case, only the local stability of the internal dynamics is guaranteed.

4.4.2 Output Redefinition for SISO Linear Systems

Consider the following nonminimum phase SISO linear system [7]

$$y = \frac{(1 - s/b)B_0(s)}{A(s)}u \tag{4.48}$$

where $b>0$ is a positive zero; $B_0(s)$ is a polynomial with negative zeros; $A(s)$ is the denominator of the transfer function; y and u are the output and the input of the system respectively.

According to J. E.Slotine and W. Li [7] , the output redefinition method can be implemented by eliminating the positive zeros from the transfer function, while remaining the negative zeros in there original positions. Consequently, the output y in (4.48) can be redefined as [7]

$$y^* = \frac{B_0(s)}{A(s)}u \tag{4.49}$$

Since there is no positive zero in (4.49), the system has a asymptotically stable zero-dynamics, and the feedback control laws can be applied to this system to achieve a exponentially convergent tracking of y^* with proper initial conditions, i.e. in steady state, $y^* = y_d$, where y_d is the desired trajectory.

Substituting (4.49) into (4.48), we have

$$y = (1 - s/b)y^* = (1 - s/b)y_d \tag{4.50}$$

Then, the tracking error for the real output is

$$e(s) = y_d(s) - y(s) = \frac{sy_d(s)}{b}$$

$$e(t) = y_d(t) - y(t) = \frac{\dot{y}_d}{b} \tag{4.51}$$

From equation (4.51) we can see that when the frequency of the y_d is sufficiently small,

$$\frac{s}{b} \approx 0 \quad \text{and} \quad y \approx y_d,$$

which means the error between the output y and the desired output y_d is small, and a good tracking can be achieved [7].

However, when the frequency of the desired output is close to or larger than the value b, the tracking error could be very large. For example, when the desired output y_d is a step input, the initial tracking error of the system can go to infinity. This is the main limitation of this approach.

Inspired by the limitation discussed above, we may redefine the output by moving the positive zero to the left half of s plane and keep the magnitude unchanged. Thus, we have

$$y^* = \frac{(1 + s/b)B_0(s)}{A(s)}u \tag{4.52}$$

Since (4.52) only contains negative zeros, the system has a asymptotically stable zero-dynamics, and the feedback control laws can also be applied to this system to achieve a exponentially convergent tracking of y^* with proper initial conditions.

Substituting (4.52) into (4.48), we have

$$y(s) = \frac{(1 - s/b)}{(1 + s/b)}y^* = \frac{(1 - s/b)}{(1 + s/b)}y_d(s) \tag{4.53}$$

When the frequency of the desired output is very low,

$$1 + s/b \approx 1 \quad \text{and} \quad 1 - s/b \approx 1$$

such that

$$y \approx y_d,$$

which means a good approximated tracking can be achieved, similar to output redefinition (4.50) for low value of $\dot{y}_d$.

When the frequency of the desired output is much larger than the value b, equation (4.53) becomes

$$y(s) = \left(\frac{b/s - 1}{b/s + 1}\right)y_d(s) \approx -y_d, \tag{4.54}$$

i.e. 180° phase shift between y and y_d, such that the error is

$$e = y_d - y \approx 2y_d \tag{4.55}$$

From equations (4.54) and (4.55) we can see that however fast the desired output changes, the tracking error is always bounded. If y_d is a step input, the transient response will go to the opposite direction to the same value of y_d. If y_d is a sinusoidal input with a

high frequency much larger than b, the system output y will always has the same amplitude with y_d, but with a phase shift up to 180°.

In this thesis, both of these two types of output redefinition method are implemented to stabilize the zero-dynamics of the UAV model (3.25) and (3.26), the results of the trajectory tracking will be compared in Chapter 6.

4.4.3 Output Redefinition for MIMO Linear and Nonlinear Systems

As we mentioned earlier, the approach proposed by J. E. Slotine to redefine the system output is developed for the SISO linear systems, but it can be generalized for MIMO linear systems.

Consider a square MIMO linear system given by

$$\begin{aligned} \dot{x} &= Ax + Bu \\ y &= Cx, \end{aligned} \tag{4.56}$$

where $x \in \Re^n$ is the state vector, $u, y \in \Re^m$ are the input vector and the output vector respectively. Then the transfer matrix of system (4.56) is

$$H(s) = C(sI - A)^{-1}B \tag{4.57}$$

The problem is what the zeros of an MIMO linear system are, or how to find the zeros. In [39], the definition of the zeros of an MIMO linear system is given as "the roots of the (nonzero) numerator polynomials in the Smith-McMillan form of $H(s)$". Therefore, the zeros of system (4.56) are not the zeros of each transfer function of the transfer matrix (4.57). However, in this thesis, we will not discuss the Smith-McMillan form, because [39] gives us an important property of the zeros, that is the zeros are the poles of the inverse of $H(s)$. Therefore, we can simply find the zeros of system (4.56) by inverting the transfer matrix $H(s)$.

An alternative way to determine the zeros is to find the eigenvalues of the

zero-dynamics of the MIMO linear system. This approach is based on the fact that for linear systems, the eigenvalues of the zero-dynamics coincide with the zeros.

If a positive zero located at $s=b$ is identified using above techniques, then the next step is to define a new output vector $y^* = C^*x$ in a way such that the positive zero b is eliminated or moved to the LHP.

For eliminating the positive zero, one has the relation between the original output y and the new output y^* as follows [26]:

$$y(s) = \begin{bmatrix} y_1(s) \\ y_2(s) \\ \vdots \\ y_m(s) \end{bmatrix} = M \begin{bmatrix} y_1^*(s) - (s/b)y_1^*(s) \\ y_2^*(s) \\ \vdots \\ y_m^*(s) \end{bmatrix} \tag{4.58}$$

where M is a constant matrix satisfying the constraint

$$M^{-1}H(b) = \begin{bmatrix} 0 & 0 & \cdots & 0 \\ * & * & \cdots & * \\ \vdots & \vdots & \ddots & \vdots \\ * & * & \cdots & * \end{bmatrix} \tag{4.59}$$

Let the transfer matrix associated with the new output y^* be $H^*(s)$. Then equation (4.58) can be transformed to

$$H(s) = M \begin{bmatrix} 1 - \frac{s}{b} & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix} H^*(s) \tag{4.60}$$

Solving for $H^*(s)$, we have

$$H^*(s) = \begin{bmatrix} \frac{1}{1-(s/b)} & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix} M^{-1}H(s) \tag{4.61}$$

Finally, y^* can be defined as $y^* = C^*x$, where C^* satisfying the following constraint.

$$H^*(s) = C^*(sI - A)^{-1}B \tag{4.62}$$

For moving the positive zero to the LHP, there is a relation between the original output y and the new output y^* similar to (4.58):

$$y(s) = \begin{bmatrix} y_1(s) \\ y_2(s) \\ \vdots \\ y_m(s) \end{bmatrix} = M \begin{bmatrix} \frac{1 - s/b}{1 + s/b} y_1^*(s) \\ y_2^*(s) \\ \vdots \\ y_m^*(s) \end{bmatrix} \tag{4.63}$$

where M is a constant matrix satisfying the constraint (4.59).

Substituting $y = H(s)u$ and $y^* = H^*(s)u$ into (4.63), we have

$$H^*(s) = \begin{bmatrix} \frac{1 + s/b}{1 - s/b} & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix} M^{-1} H(s) \tag{4.64}$$

Then from equation (4.62) we obtain the new matrix C^* for the redefined output y^*.

After the output redefinition either by eliminating the positive zero or moving the positive zero to LHP, the resulting linear system becomes minimum phase and can be inverted.

However the output redefinition method by manipulating the zeros cannot be applied directly to nonlinear systems, because the nonlinear systems are impossible to be expressed in form of transfer functions. To solve this problem, one may first calculate the Jacobian linearization of the nonlinear system, and then apply above techniques to define a new output.

Consider a nonminimum phase nonlinear time invariant system of the form

$$\begin{aligned} \dot{x} &= f(x) + g(x)u \\ y &= h(x) \end{aligned} \tag{4.65}$$

where state $x \in \Re^n$, $u, y \in \Re^m$ are the control input and the output respectively. *f(x)*, *g(x)* and *h(x)* are nonlinear functions and assumed to be smooth.

The Jacobian linearization of system (4.65) about $x = 0$ is given by

$$\begin{aligned} \dot{x} &= Ax + Bu \\ y &= Cx, \end{aligned} \tag{4.66}$$

where $A = \left.\frac{\partial f}{\partial x}\right|_{x=0}$; $B = g(0)$; $C = \left.\frac{\partial h}{\partial x}\right|_{x=0}$

After finding the positive zero in (4.66), we can calculate C^* using above techniques. If we write the output in (4.65) as [26]

$$y = h(x) = Cx + \tilde{h}(x), \tag{4.67}$$

where $\tilde{h}(x)$ is of order 2 or higher in x, then the new output y^* is

$$y^* = C^* x + \tilde{h}(x) \tag{4.68}$$

Consequently, the nonlinear system (4.65) is approximated by a new system

$$\begin{aligned} \dot{x} &= f(x) + g(x)u \\ y^* &= C^* x + \tilde{h}(x) \end{aligned} \tag{4.69}$$

System (4.69) is a minimum phase system, and the feedback linearization control law can be applied to it.

4.4.4 Output Redefinition for Nonlinear UAV Models

We now redefine the output of systems (3.25) and (3.26) using above techniques to make the systems invertible.

a) Output Redefinition for UAV Model (3.25)

Applying the Jacobian linearization to the nonlinear UAV model (3.25) about $x=0$, we obtain a linearized SISO system described by [26]:

$$\begin{aligned} \dot{x} &= Ax + Bu \\ y &= Cx \end{aligned} \tag{4.70}$$

where

$$A=\begin{bmatrix} l_p & l_q & l_r & 0 & l_\beta & 0 & 0 \\ 0 & \overline{m}_q & 0 & \overline{m}_\alpha & 0 & 0 & 0 \\ n_p & n_q & n_r & 0 & n_\beta & 0 & 0 \\ 0 & 1 & 0 & z_\alpha & 0 & 0 & 0 \\ \sin\alpha_0 & 0 & -\cos\alpha_0 & 0 & y_\beta & g/V & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} ; \quad B=\begin{bmatrix} l_{\delta a} \\ 0 \\ n_{\delta a} \\ 0 \\ y_{\delta a} \\ 0 \\ 0 \end{bmatrix}$$

$$C=[0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0] ; \quad u=\delta_a$$

The transfer function of the linearized system (4.70) based on the nominal condition in table 3.1 is

$$H(s)=\frac{\beta(s)}{\delta_a(s)}=\frac{0.0071s^3-0.2488s^2+1.8533s-0.3756}{s^4+4.364s^3+7.6707s^2+22.8411s+0.0563} \tag{4.71}$$

which has positive zeros at *s=24.4583, s=10.3755* and *s=0.2085.*

Since (4.70) is an SISO linear system, we may directly redefine its transfer function by eliminating all its positive zeros, which gives

$$H^*(s)=\frac{-0.3756}{s^4+4.364s^3+7.6707s^2+22.8411s+0.0563} \tag{4.72}$$

Using MATLAB Symbolic Math Toolbox, it is easy to find the new output $y^*=C^*x$, where $C^*=[6.524\times10^{-4} \quad 0 \quad -3.2547\times10^{-2} \quad 0 \quad -1.1323\times10^{-2} \quad 2.5157\times10^{-3} \quad 0]$.

As we mentioned earlier, an alternative way to redefine the output is to move the positive zeros to the LHP and keep their magnitudes unchanged. This results a new transfer function as

$$H^*=\frac{-0.0071s^3-0.2488s^2-1.8533s-0.3756}{s^4+4.364s^3+7.6707s^2+22.8411s+0.0563} \tag{4.73}$$

According to the new transfer function (4.73), the new output is defined by $y^*=C^*x$, where $C^*=[-0.0036 \quad 0 \quad 0.1807 \quad 0 \quad -0.8237 \quad 0.0037 \quad 0]$.

Using the inverse Lapace transform, the outputs in equations (4.71)-(4.73) to unit

step inputs can be expressed in time domain by equations (4.74)-(4.76):

$$y(t) = \beta(t) = -6.6714 - 0.0423e^{-3.8984t} + 6.7601e^{-0.0025t} + (-0.0232 + 0.0297i)e^{(-0.2316+2.4085i)t} + (-0.0232 - 0.0297i)e^{(-0.2316-2.4085i)t} \quad (4.74)$$

$$y^*(t) = -6.6714 + 0.0020\ e^{-3.8984\ t} + 6.6769\ e^{-0.0025\ t} + (-0.002 - 0.0704\ i)e^{(-0.2316+2.4085\ i)t} + (-0.002 + 0.0704\ i)e^{(-0.2316-2.4085\ i)t} \quad (4.75)$$

$$y^*(t) = -6.6714 + 0.0113\ e^{-3.8984\ t} + 6.5956\ e^{-0.0025\ t} + (0.032 - 0.0148\ i)e^{(-0.2316+2.4085\ i)t} + (0.032 + 0.0148\ i)e^{(-0.2316-2.4085\ i)t} \quad (4.76)$$

The results are also shown in Figure 4.8-4.10.

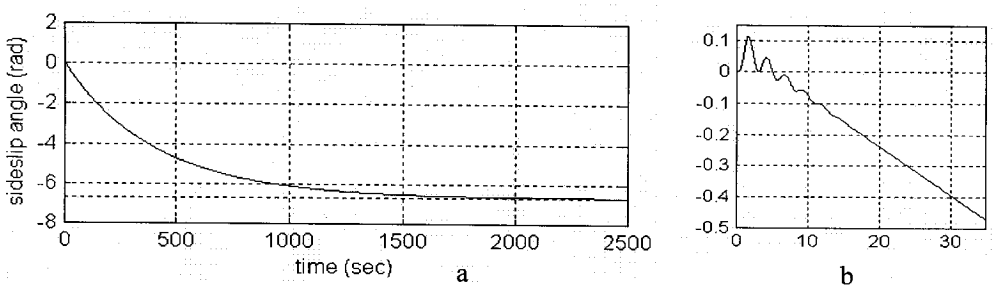

Figure 4.8 (a) The output (original) response to a unit step input
(b) Details of the transient response

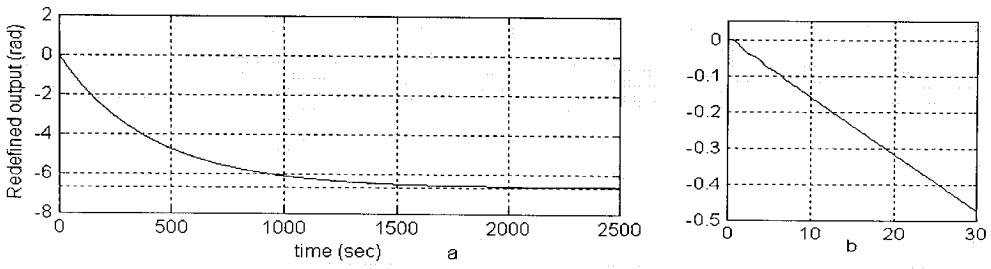

Figure 4.9 (a) The output (redefined) response to a unit step input
(b) Details of the transient response

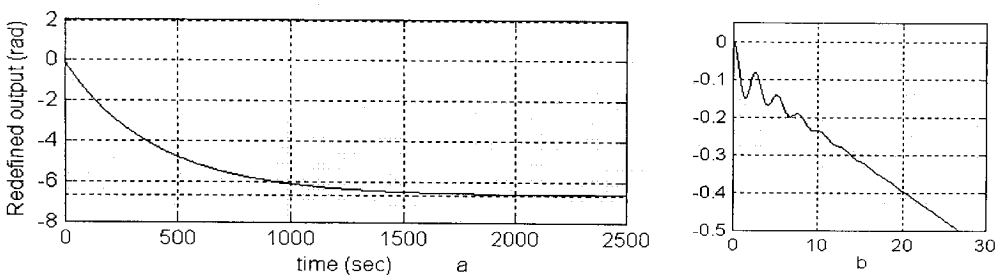

Figure 4.10 (a) The output (redefined) response to a unit step input
(b) Details of the transient response

From Figure 4.8 we can see that there is an "undershoot" at the transient stage. This is caused by the positive zeros in the open-loop transfer function (4.71). In Figure 4.9 and 4.10, the redefined outputs do not present an "undershoot" at the transient stage, because the positive zeros are removed from the redefined transfer functions (4.72) and (4.73).

b) Output Redefinition for UAV Model (3.26)

Similarly, the nonlinear UAV model (3.26) with two inputs and two outputs can also be linearized as

$$\begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \\ \Delta\dot{\alpha} \\ \dot{\beta} \\ \dot{\phi} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} l_p & l_q & l_r & 0 & l_\beta & 0 & 0 \\ 0 & \overline{m}_q & 0 & \overline{m}_\alpha & 0 & 0 & 0 \\ n_p & n_q & n_r & 0 & n_\beta & 0 & 0 \\ 0 & 1 & 0 & z_\alpha & 0 & 0 & 0 \\ \sin\alpha_0 & 0 & -\cos\alpha_0 & 0 & y_\beta & g/V & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} p \\ q \\ r \\ \Delta\alpha \\ \beta \\ \phi \\ \theta \end{bmatrix} + \begin{bmatrix} l_{\delta a} & l_{\delta r} \\ 0 & 0 \\ n_{\delta a} & n_{\delta r} \\ 0 & 0 \\ y_{\delta a} & y_{\delta r} \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \delta_a \\ \delta_r \end{bmatrix} \tag{4.77}$$

$$y = Cx = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} p & q & r & \Delta\alpha & \beta & \phi & \theta \end{bmatrix}^T$$

Then using equation (4.57), the transfer matrix of system (4.77) is found as [26]

$$H = \begin{bmatrix} H_{11} & H_{12} \\ H_{21} & H_{22} \end{bmatrix} \tag{4.78}$$

where

$$H_{11} = \frac{0.0071s^3 - 0.2488s^2 + 1.8533s - 0.3756}{s^4 + 4.364s^3 + 7.6707s^2 + 22.8411s + 0.0563}$$

$$H_{12} = \frac{6.3079s^2 + 25.2783s - 0.0902}{s^4 + 4.364s^3 + 7.6707s^2 + 22.8411s + 0.0563}$$

$$H_{21} = -\frac{45.83s^2 + 19.9397s + 271.1101}{s^4 + 4.364s^3 + 7.6707s^2 + 22.8411s + 0.0563}$$

$$H_{22} = -\frac{7.64s^2 + 4.1131s + 108.8293}{s^4 + 4.364s^3 + 7.6707s^2 + 22.8411s + 0.0563}$$

To find the zeros of the system (4.77), one may invert the transfer matrix (4.78) and consider its poles [39]. The inverse matrix of (4.78) is given by

$$H^{(inv)} = \begin{bmatrix} H_{11}^{(inv)} & H_{12}^{(inv)} \\ H_{21}^{(inv)} & H_{22}^{(inv)} \end{bmatrix} \tag{4.79}$$

where

$$H_{11}^{(inv)} = \frac{7.64s^2 + 4.1131s + 108.8293}{0.0542(s - 5368.31)}$$

$$H_{12}^{(inv)} = \frac{6.3079s^2 + 25.2783s - 0.0902}{0.0542(s - 5368.31)}$$

$$H_{21}^{(inv)} = -\frac{45.83s^2 + 19.9397s + 271.1101}{0.0542(s - 5368.31)}$$

$$H_{22}^{(inv)} = -\frac{0.0071s^3 - 0.2488s^2 + 1.8533s - 0.3756}{0.0542(s - 5368.31)}$$

Therefore the zero of the system (4.77) can easily be found at $s=5368.31$ which coincides with the positive eigenvalue of the zero-dynamics of the nonlinear system (3.21). The following steps will derive the new output for the system (3.21).

First, we should determine the constant matrix M^1 using equation (4.59). Since the value of the transfer matrix $H(s)$ at $s = 5368.31$ is

$$H(b) = \begin{bmatrix} 1.3129 & 0.2189 \\ -1.5891 & -0.2649 \end{bmatrix},$$

we may determine the constant matrix M^1 as [26]

$$M^{-1} = \begin{bmatrix} 1 & 0.8262 \\ 0 & 1 \end{bmatrix}$$

Then using equation (4.61), we can obtain a redefined transfer matrix without any positive zeros.

$$H^*(s) = \begin{bmatrix} H_{11}^* & H_{12}^* \\ H_{21}^* & H_{22}^* \end{bmatrix} = \begin{bmatrix} \frac{1}{1-(s/5368.31)} & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix} M^{-1} H(s)$$

where

$$H_{11}^* = -\frac{38.115s^2 + 14.6621s + 224.3589}{s^4 + 4.364s^3 + 7.6707s^2 + 22.8411s + 0.0563}$$

$$H_{12}^{*}=\frac{21.8634s-90.0019}{s^4+4.364s^3+7.6707s^2+22.8411s+0.0563}$$

$$H_{21}^{*}=-\frac{45.83s^2+19.9397s+271.1101}{s^4+4.364s^3+7.6707s^2+22.8411s+0.0563}$$

$$H_{22}^{*}=-\frac{7.64s^2+4.1131s+108.8293}{s^4+4.364s^3+7.6707s^2+22.8411s+0.0563}$$

Finally, the new output matrix C^* can be computed using equation (4.62) [26].

$$C^{*}=\begin{bmatrix}1.5866\times10^{-4} & 0 & -1.862\times10^{-4} & 0 & 1 & 0.82618 & 2.3455\times10^{-4}\\ 0 & 0 & 0 & 0 & 0 & 1 & 0\end{bmatrix} \tag{4.80}$$

Using equation (4.64), one may redefine the transfer matrix by moving the positive zero to the LHP.

$$H^{*}(s)=\begin{bmatrix}H_{11}^{*} & H_{12}^{*}\\ H_{21}^{*} & H_{22}^{*}\end{bmatrix}=\begin{bmatrix}\frac{1+(s/5368.31)}{1-(s/5368.31)} & 0 & \cdots & 0\\ 0 & 1 & \cdots & 0\\ \vdots & \vdots & \ddots & \vdots\\ 0 & 0 & \cdots & 1\end{bmatrix}M^{-1}H(s)$$

Thus, we have

$$H_{11}^{*}=-\frac{0.0071s^3+38.1177s^2+14.7039s+224.3589}{s^4+4.364s^3+7.6707s^2+22.8411s+0.0563}$$

$$H_{12}^{*}=\frac{0.0041s^2+21.8468s-90.0023}{s^4+4.364s^3+7.6707s^2+22.8411s+0.0563}$$

$$H_{21}^{*}=-\frac{45.83s^2+19.9397s+271.1101}{s^4+4.364s^3+7.6707s^2+22.8411s+0.0563}$$

$$H_{22}^{*}=-\frac{7.64s^2+4.1131s+108.8293}{s^4+4.364s^3+7.6707s^2+22.8411s+0.0563}$$

Then, using equation (4.62), we obtain another new output matrix C^*.

$$C^{*}=\begin{bmatrix}3.2053\times10^{-4} & 0 & -3.7617\times10^{-4} & 0 & 1 & 0.82619 & 0\\ 0 & 0 & 0 & 0 & 0 & 1 & 0\end{bmatrix} \tag{4.81}$$

Now, we have redefined the output for UAV models (3.25) and (3.26) either by neglecting the positive zeros or moving the positive zeros to the LHP, so that these two

systems with new outputs are now minimum phase, and can be inverted. In next chapter, the neural network based feedback linearization controller will be designed to obtain a good approximate tracking of desired trajectories.

Chapter 5

Neural Network Based Feedback Linearization Controller Design

Although the feedback linearization is a very effective nonlinear control law, the full-envelope nonlinear inversion of an aircraft dynamics is computationally intensive, because the nonlinear high order aircraft model should be inverted in real time. Neural networks offer the potential to overcome this difficulty through off-line learning.

5.1 An Introduction to Multi-layer Feedforward Neural Networks

Neural network, a facet of artificial intelligence, attempts to simulate the biological neural systems in fault-tolerance and self-learning capacity by modeling the low-level structure of the brain. It has been successfully applied across an extraordinary range of problem domains, such as engineering, finance, geology and medicine.

A typical diagram of a two layer feedforward neural network is shown in Figure 5.1.

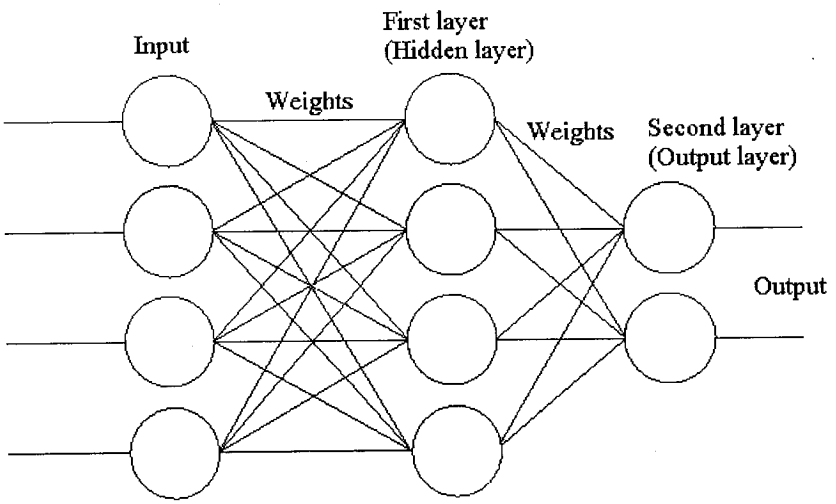

Figure 5.1 Block diagram of a two layer feedforward neural network

Each of circles in figure 5.1 is called a unit (or neuron) containing an input-output transfer function. Neurons are arranged in a distinct layered topology. The input is not really neural layer at all: these units simply accept the values of the input variables. The hidden and output layer neurons are each connected to all of the units in the preceding layer by a system of weights.

The mathematic model of a two layer neural network is shown in Figure 5.2.

The neural network in Figure 5.2 has R^1 inputs, S^1 neurons in the first layer, S^2 neurons in the second layer. Each layer has a weight matrix $\boldsymbol{W}$, a bias vector $\boldsymbol{b}$, and an output vector $\boldsymbol{a}$. The number of the layer is appended as a superscript to above variables. Therefore, the set of weighted sums that form the input to the first layer containing S^1 units is [40]

$$n^1 = IW^{1,1}p + b^1 \tag{5.1}$$

where $IW^{1,1}$ is an $S^1 \times R^1$ input weight matrix.

The output of the first layer or so called hidden layer will be a^1, an $S^1 \times 1$ vector determined by:

$$a^1 = f^1(n^1) = f^1(IW^{1,1}p + b^1) \tag{5.2}$$

where f^1 is the transfer function of the first layer, typically a step function or a sigmoid function, that takes the argument n^1 and produces the output a^1.

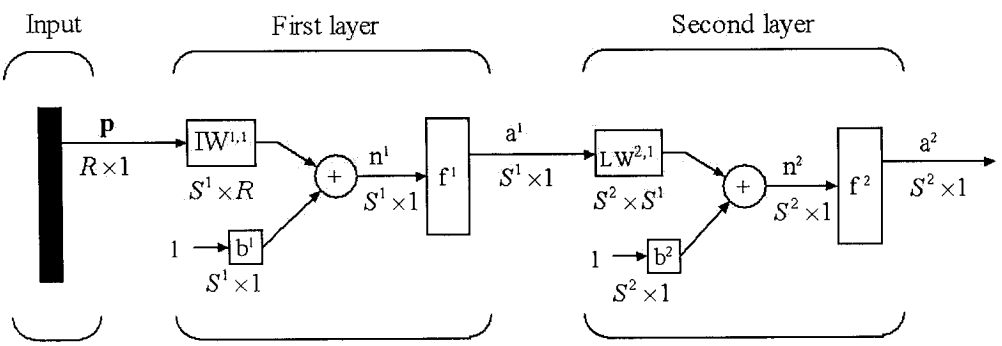

Figure 5.2 The mathematic model of a two layer feedforward neural network (from MATLAB Neural Network Toolbox User's Guide [40])

Since the output of the first layer is the input to the second layer, the layer 2 can be analyzed as a one layer network with S^1 inputs, S^2 neurons, and an $S^2 \times S^1$ weight matrix $LW^{2,1}$. Therefore, the output vector a^2 is found by

$$n^2 = LW^{2,1}a^1 + b^2 \tag{5.3}$$

$$a^2 = f^2(n^2) = f^2(LW^{2,1}a^1 + b^2) \tag{5.4}$$

The neural network can be trained by using backpropagation algorithm. The backpropagation algorithm for multilayer networks is a gradient descent optimization procedure in which we minimize a mean square error performance index. The algorithm is provided with a set of training data consisting of pairs:

$$\{\mathbf{p}_1, \mathbf{t}_1\}, \{\mathbf{p}_2, \mathbf{t}_2\}, \ldots, \{\mathbf{p}_Q, \mathbf{t}_Q\}$$

where $\mathbf{p}_q$ ($q=1, 2, \ldots Q$) is an input vector to the neural network, and $\mathbf{t}_q$ is the corresponding target output. As each input is applied to the network, the network output is compared to the target. The algorithm should adjust the network parameters in order to minimize the mean square error:

$$E = \sum_{q=1}^{Q} e_q^2 = \frac{1}{2}\sum_{q=1}^{Q}(t_q - a_q)^2 \tag{5.5}$$

where $\mathbf{a}_q$ is the output of the network for the qth input.

If the network has multiple outputs, (5.5) can be generalized to

$$E = \sum_{q=1}^{Q} e_q^T e_q = \frac{1}{2}\sum_{q=1}^{Q}(t_q - a_q)^T(t_q - a_q) \tag{5.6}$$

The steepest descent algorithm for the mean square error is

$$w_{i,j}^m(k+1) = w_{i,j}^m(k) - \alpha\frac{\partial E}{\partial w_{i,j}^m} \tag{5.7}$$

$$b_i^m(k+1) = b_i^m(k) - \alpha\frac{\partial E}{\partial b_i^m} \tag{5.8}$$

where $w_{i,j}^m$ represents the weight for the signal from the jth source to the ith neuron at layer m, k is the iteration number, and α is the learning rate.

Since the mean square error is an indirect function of the weights in the hidden layer, the chain rule of calculus is used to calculate the derivatives in equation (5.7) and equation (5.8).

$$\frac{\partial E}{\partial w_{i,j}^m} = \frac{\partial E}{\partial n_i^m} \times \frac{\partial n_i^m}{\partial w_{i,j}^m} \tag{5.9}$$

$$\frac{\partial E}{\partial b_i^m} = \frac{\partial E}{\partial n_i^m} \times \frac{\partial n_i^m}{\partial b_i^m} \tag{5.10}$$

The second term in each of these equations can be easily computed, since the net input to layer is an explicit function of the weights and bias in that layer:

$$n_i^m = \sum_{j=1}^{s^{m-1}} w_{i,j}^m a_j^{m-1} + b_i^m \tag{5.11}$$

Therefore,

$$\frac{\partial n_i^m}{\partial w_{i,j}^m} = a_j^{m-1}, \qquad \frac{\partial n_i^m}{\partial b_i^m} = 1. \tag{5.12}$$

If we define the error signal for the ith neuron at layer m as

$$\delta_i^m \equiv \frac{\partial E}{\partial n_i^m}, \tag{5.13}$$

then, equations (5.7) and (5.8) can be rewritten as

$$w_{i,j}^m(k+1) = w_{i,j}^m(k) - \alpha \delta_i^m a_j^{m-1} \tag{5.14}$$

$$b_i^m(k+1) = b_i^m(k) - \alpha \delta_i^m \tag{5.15}$$

For hidden units, we must propagate the error back from the output neurons. Again, using the chain rule, we can expand the error of a hidden unit in terms of its posterior nodes:

$$\delta_j^{m-1} = \sum_{i=1}^{s^m} \frac{\partial E}{\partial n_i^m} \frac{\partial n_i^m}{\partial a_j^{m-1}} \frac{\partial a_j^{m-1}}{\partial n_j^{m-1}} \tag{5.16}$$

Of the three factors inside the sum, the first is just the error of the ith neuron δ_i^m. The second is

$$\frac{\partial n_i^m}{\partial a_j^{m-1}} = \frac{\partial}{\partial a_j^{m-1}} \sum_{k=1}^{s^{m-1}} w_{i,k} a_k = w_{i,j}, \tag{5.17}$$

while the third is the derivative of the jth neuron's transfer function:

$$\frac{\partial a_j^{m-1}}{\partial n_j^{m-1}} = \frac{\partial f_j\left(n_j^{m-1}\right)}{\partial n_j^{m-1}} = f_j'\left(n_j^{m-1}\right) \tag{5.18}$$

Putting all the pieces together, we obtain

$$\delta_j^{m-1} = f_j'\left(n_j^{m-1}\right) \sum_{i=1}^{s^m} \delta_i^m w_{i,j} \tag{5.19}$$

Having derived the error backpropagation algorithm, the training therefore can be performed iteratively through a number of epochs. On each epoch, the targets and actual outputs are compared and the error is calculated. This error is used to adjust the weights, and then the process repeats. The initial network configuration is random and training stops when a given number of epochs elapse, or when the error reaches an acceptable level, or when the error stops improving. After the neural network is properly trained, it can be applied for practical use. This process is illustrated in Figure 5.3. After a proper training, the multi-layer neural network is able to approximate any nonlinear functions to desired accuracy.

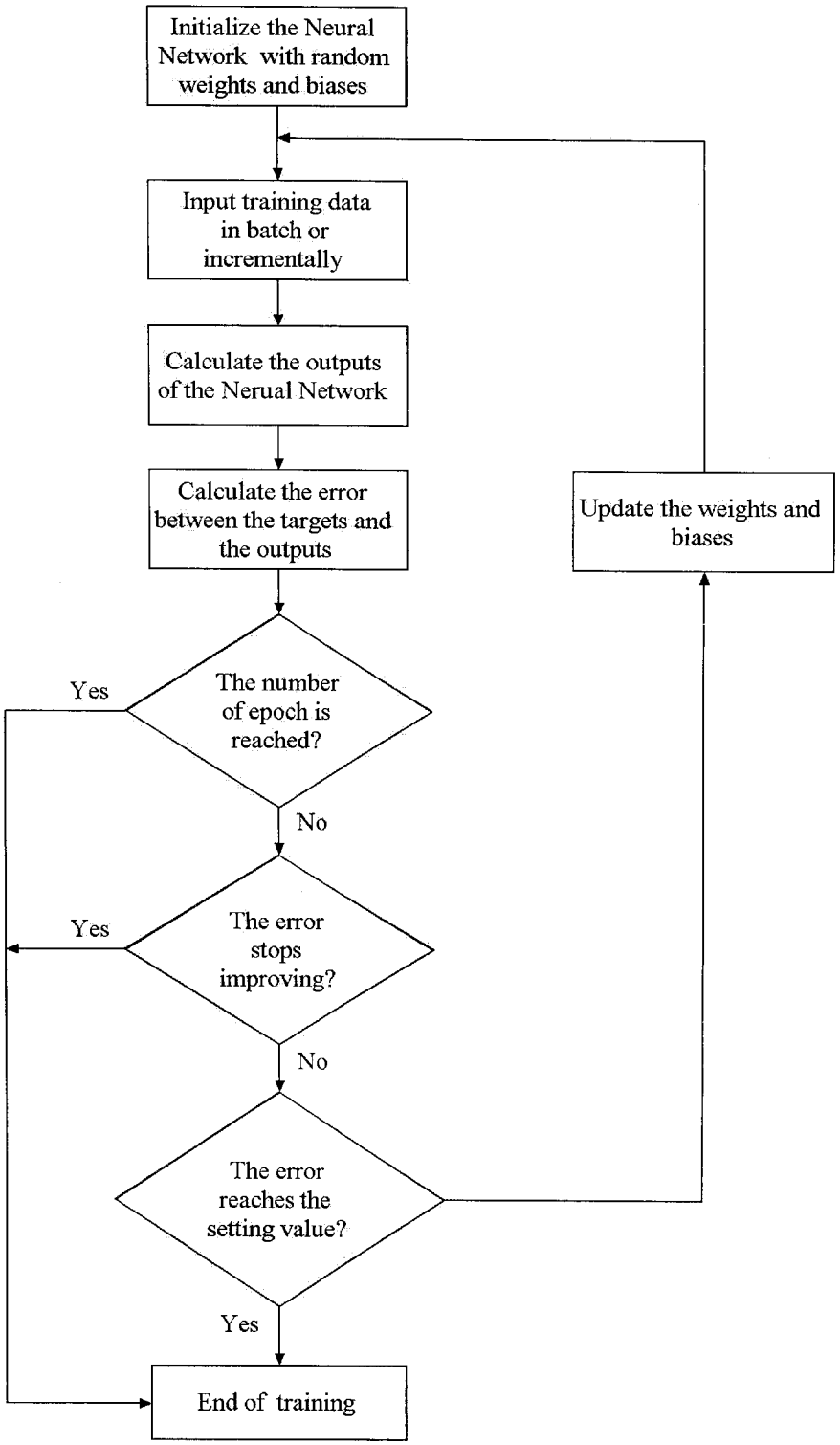

Figure 5.3 Flowchart of the neural network training process

5.2 Nonlinear Autoregressive Moving Average (NARMA) Representation of a Nonlinear System

In order to implement the neural network control algorithm, it is necessary to let the neural network model a discrete-time nonlinear system which is a discretization of a continuous-time nonlinear system to be controlled, so that a neural controller based on the neural network model can be derived.

A general form a discrete-time nonlinear SISO system can be represented by the state equations

$$\begin{aligned} x(k+1) &= f[x(k), u(k)] \\ y &= h[x(k)], \end{aligned} \tag{5.20}$$

where $\{u(k)\}$, $\{x(k)\}$ and $\{y(k)\}$ are discrete-time sequences with $x(k) \in \Re^n$, $u(k) \in \Re$, $y(k) \in \Re$, $f : \Re^n \times \Re \to \Re^n$, $h : \Re^n \to \Re$, and $f, h \in C^\infty$.

It is often the case that nonlinear functions f and h in (5.20) are unknown. Only a finite number of input-output pairs $(u(k), y(k))$ at instant time k generated from its continuous-time counterpart (4.5) are accessible. Therefore, the neural network identification and control have to be carried out using these input-output data.

One solution to this problem as proposed in MATLAB® [40] is to represent the system (5.20) with the nonlinear autoregressive moving average (NARMA) model:

$$y(k+d) = F[y(k), y(k-1),...,y(k-n+1), u(k), u(k-1),...u(k-n+1)] \tag{5.21}$$

where $u(k)$ and $y(k)$ are the input and output at instant k respectively, d is the time delay of the system, and $F : \Re^{2n} \to \Re$ and $F \in C^\infty$ is a nonlinear function. The derivation of the NARMA model (5.21) from nonlinear discrete-time system (5.20) can be found in [33].

If we want the system output $y(k+d)$ to follow a desired trajectory $y_d(k+d)$ at instant

k, it results in $y(k+d) = y_d(k+d)$. To achieve this, a nonlinear controller can be developed as:

$$u(k) = G[y(k), y(k-1),..., y(k-n+1), y_d(k+d), u(k-1),...u(k-n+1)] \qquad (5.22)$$

where $G: \Re^{2n} \to \Re$ and $G \in C^{\infty}$ exists. Consequently, the control problem is to determine the nonlinear map G from the measured values of inputs and outputs as well as the given desired output $y_d(k+d)$. A multiple-layer neural network (MNN) can be used to approximate the nonlinear map G, as has been implemented in reference [10]. Since the controller is in the feedback loop of a dynamical system, the dynamic backpropagation (online training) instead of static backpropagation (off-line training) must be used to train the neural controller. However, dynamic backpropagation is quite slow and computationally intensive compared with static backpropagation. To overcome this problem, K. S. Narendra and S.Mukhopadhyay [33] developed two approximate models, the NARMA-L1 and NARMA-L2 models, which permit the off-line training of a neural network controller, thus greatly reduces the difficulty of computation.

5.3 Approximation of the NARMA Model

K. S. Narendra and S.Mukhopadhyay [33] have proposed two approximate models, the NARMA-L1 and NARMA-L2 models, which are given below

NARMA-L1 Model:

$$\begin{aligned} y(k+d) = & f_0[y(k), y(k-1),..., y(k-n+1)] \\ & + \sum_{i=0}^{n-1} g_i[y(k), y(k-1),..., y(k-n+1)] \cdot u(k-i) \end{aligned} \qquad (5.23)$$

NARMA-L2 Model:

$$\begin{aligned} y(k+d) = & f[y(k), y(k-1),..., y(k-n+1), u(k-1),..., u(k-n+1)] \\ & + g[y(k),..., y(k-n+1), u(k-1),..., u(k-n+1)] \cdot u(k) \end{aligned} \qquad (5.24)$$

These two models are derived from different Taylor expansions of the right hand side of the NARMA model (5.21). It has been proved in [33] that these two models have bounded error with respect to the NARMA model.

We can see that in equation (5.23) of the NARMA-L1 model, the functions f_0 and g_i *(i=0,...,n-1)* are only functions of past values of the outputs, while in equation (5.24) of the NARMA-L2 model, f and g are both the functions of $y(k),...,y(k-n+1)$ and $u(k-1),...,u(k-n+1)$. But, the most important feature is the control input *u(k)* at time k (the instant of interest in the control problem) appears linearly in above equations, which permits the easy computation of the control input by rearranging the models.

Let the output exactly track a desired trajectory, i.e. $y(k+d)=y_d(k+d)$. Then, rearranging the model (5.23) and (5.24), we obtain [33]

NARMA-L1 Controller for SISO Systems:

$$u(k)=\frac{y_d(k+d)-f_0[y(k),...,y(k-n+1)]-\sum_{i=1}^{n-1}g_i[y(k),...,y(k-n+1)]u(k-i)}{g_0[y(k),...,y(k-n+1)]} \tag{5.25}$$

NARMA-L2 Controller for SISO Systems:

$$u(k)=\frac{y_d(k+d)-f[y(k),...,y(k-n+1),u(k-1),...,u(k-n+1)]}{g[y(k),...,y(k-n+1),u(k-1),...,u(k-n+1)]} \tag{5.26}$$

Since there is a realization problem associated with above controllers that the control input *u(k)* is determined based on the output at the same time *y(k)*, (5.25) and (5.26) are can be modified as [40]

$$u(k+1)=\frac{y_d(k+d)-f_0[y(k),...,y(k-n+1)]-\sum_{i=1}^{n-1}g_i[y(k),...,y(k-n+1)]u(k-i+1)}{g_0[y(k),...,y(k-n+1)]} \tag{5.27}$$

$$u(k+1)=\frac{y_d(k+d)-f[y(k),...,y(k-n+1),u(k),...,u(k-n+1)]}{g[y(k),...,y(k-n+1),u(k),...,u(k-n+1)]} \tag{5.28}$$

where $d \geq 2$.

From above equations, we can see that this control algorithm has the same effect with the feedback linearization control law that transforms the nonlinear system dynamics into linear dynamics by canceling the nonlinearities [33, 40]. And we also find that if neural networks are used to approximate the NARMA-L1 controller, then *(n+1)* networks must be created to model the functions f_0 and g_i $(i=0,...,n-1)$. While for NARMA-L2 controller, only two networks are needed to approximate the functions f and g. Therefore, it is evident that the NARMA-L2 neural network controller is easier to realize than a NARMA-L1 counterpart. In the following section, we will derive neural controllers based on the NARMA-L2 model.

5.4 NARMA-L2 Neural Network Based Feedback Linearization

To create a NARMA-L2 neural network controller, two separate neural networks are needed to model nonlinear functions f and g in (5.24). Therefore, the NARMA-L2 model, approximated by neural networks, can be defined as:

$$\begin{aligned}\hat{y}(k+d) = NN_f[y(k), y(k-1),..., y(k-n+1), u(k),..., u(k-n+1)] \\ + NN_g[y(k),..., y(k-n+1), u(k),..., u(k-n+1)] \cdot u(k+1)\end{aligned} \tag{5.29}$$

where $\hat{y}(k+d)$ is the approximate output at time $k+d$, NN_f and NN_g are the neural networks used to model the function f and g. Figure 5.4 shows the structure of a neural network representation.

Now let $\hat{y}(k+d) = y_d(k+d)$, and rearrange the equation (5.29) yielding the neural network controller for an SISO system:

$$u(k+1) = \frac{y_d(k+d) - NN_f[y(k),..., y(k-n+1), u(k),..., u(k-n+1)]}{NN_g[y(k),..., y(k-n+1), u(k),..., u(k-n+1)]} \tag{5.30}$$

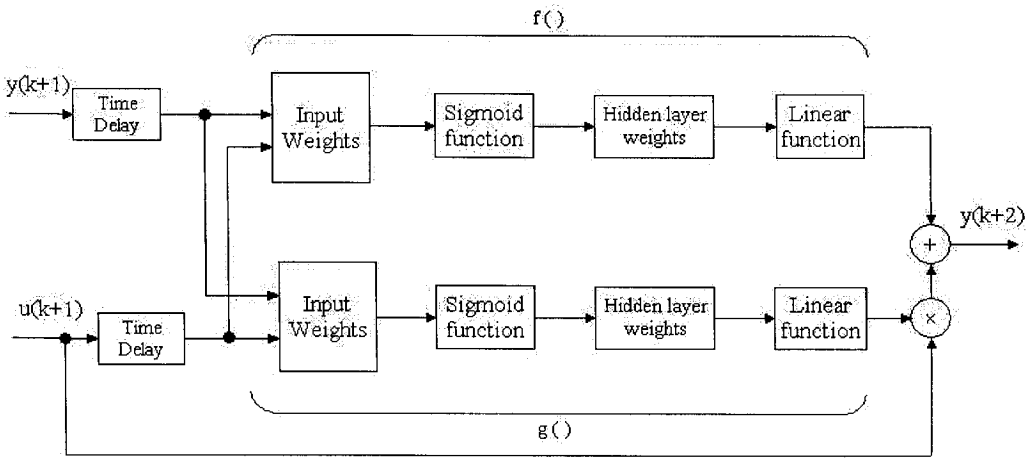

Figure 5.4 The structure of the neural network representation of the NARMA-L2 model
(Modified from MATLAB Neural Network Toolbox User's Guide [40])

The structure of the neural network controller (5.30) is shown in Figure 5.5. The source code of this controller can be found in MATLAB/Simulink® Neural Network Toolbox.

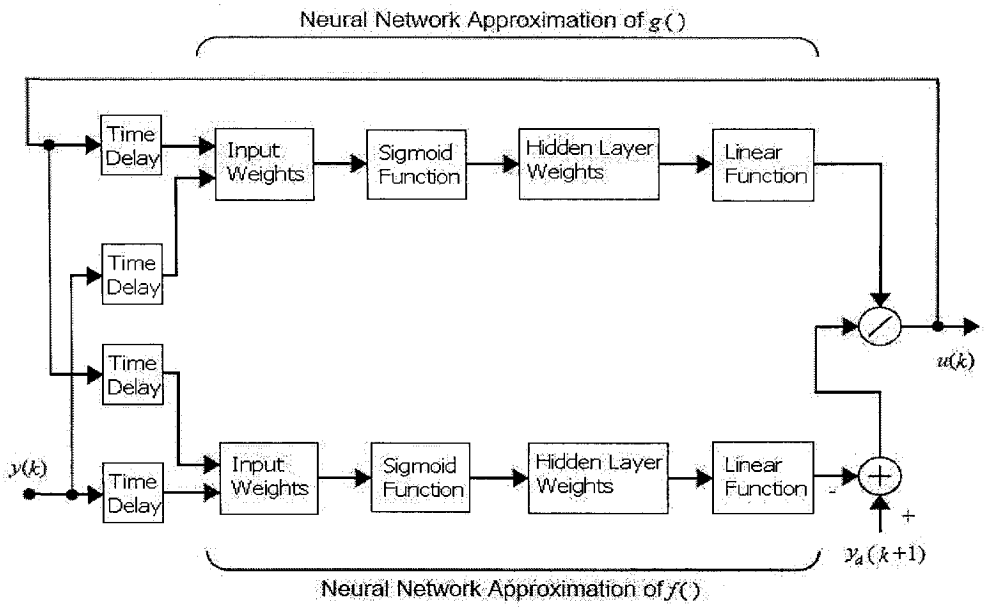

Figure 5.5 The structure of the NARMA-L2 neural network controller
(Modified from MATLAB Neural Network Toolbox User's Guide [40])

Since the NARMA-L2 model proposed by Narendra and Mukhopadhyay only handles one input and one output, it cannot be applied directly to MIMO systems. In

order to control the UAV model (3.26) which has two inputs and two outputs, the NARMA-L2 model (5.24) has to be modified in a way such that it can approximately model a TITO nonlinear system. The modified NARMA-L2 model for a TITO system is shown below:

$$\begin{aligned}
&\begin{bmatrix} y_1(k+d) \\ y_2(k+d) \end{bmatrix} \\
&= \begin{bmatrix} f_1[y_1(k),\ldots,y_1(k-n+1),y_2(k),\ldots,y_2(k-n+1),u_1(k),\ldots,u_1(k-n+1),u_2(k),\ldots,u_2(k-n+1)] \\ f_2[y_1(k),\ldots,y_1(k-n+1),y_2(k),\ldots,y_2(k-n+1),u_1(k),\ldots,u_1(k-n+1),u_2(k),\ldots,u_2(k-n+1)] \end{bmatrix} \\
&+ \begin{bmatrix} g_{11}[y_1(k),\ldots,y_1(k-n+1),y_2(k),\ldots,y_2(k-n+1),u_1(k),\ldots,u_1(k-n+1),u_2(k),\ldots,u_2(k-n+1)] \\ g_{21}[y_1(k),\ldots,y_1(k-n+1),y_2(k),\ldots,y_2(k-n+1),u_1(k),\ldots,u_1(k-n+1),u_2(k),\ldots,u_2(k-n+1)] \end{bmatrix} \cdot u_1(k+1) \\
&+ \begin{bmatrix} g_{12}[y_1(k),\ldots,y_1(k-n+1),y_2(k),\ldots,y_2(k-n+1),u_1(k),\ldots,u_1(k-n+1),u_2(k),\ldots,u_2(k-n+1)] \\ g_{22}[y_1(k),\ldots,y_1(k-n+1),y_2(k),\ldots,y_2(k-n+1),u_1(k),\ldots,u_1(k-n+1),u_2(k),\ldots,u_2(k-n+1)] \end{bmatrix} \cdot u_2(k+1)
\end{aligned} \tag{5.31}$$

To approximate this model using neural networks, six separate networks are needed for modeling functions of f_1, f_2, g_{11}, g_{21}, g_{12} and g_{22}. However, to simplify the network structure, we may use three networks, NN_{f1}, NN_{f2} and NN_g, with NN_g having four outputs. Hence the neural network approximation of (5.31) is defined as:

$$\begin{aligned}
&\begin{bmatrix} \hat{y}_1(k+d) \\ \hat{y}_2(k+d) \end{bmatrix} \\
&= \begin{bmatrix} NN_{f1}[y_1(k),\ldots,y_1(k-n+1),y_2(k),\ldots,y_2(k-n+1),u_1(k),\ldots,u_1(k-n+1),u_2(k),\ldots,u_2(k-n+1)] \\ NN_{f2}[y_1(k),\ldots,y_1(k-n+1),y_2(k),\ldots,y_2(k-n+1),u_1(k),\ldots,u_1(k-n+1),u_2(k),\ldots,u_2(k-n+1)] \end{bmatrix} \\
&+ \begin{bmatrix} NN_{g11}[y_1(k),\ldots,y_1(k-n+1),y_2(k),\ldots,y_2(k-n+1),u_1(k),\ldots,u_1(k-n+1),u_2(k),\ldots,u_2(k-n+1)] \\ NN_{g21}[y_1(k),\ldots,y_1(k-n+1),y_2(k),\ldots,y_2(k-n+1),u_1(k),\ldots,u_1(k-n+1),u_2(k),\ldots,u_2(k-n+1)] \end{bmatrix} \cdot u_1(k+1) \\
&+ \begin{bmatrix} NN_{g12}[y_1(k),\ldots,y_1(k-n+1),y_2(k),\ldots,y_2(k-n+1),u_1(k),\ldots,u_1(k-n+1),u_2(k),\ldots,u_2(k-n+1)] \\ NN_{g22}[y_1(k),\ldots,y_1(k-n+1),y_2(k),\ldots,y_2(k-n+1),u_1(k),\ldots,u_1(k-n+1),u_2(k),\ldots,u_2(k-n+1)] \end{bmatrix} \cdot u_2(k+1)
\end{aligned} \tag{5.32}$$

where NN_{g11}, NN_{g21}, NN_{g12} and NN_{g22} are four outputs of the network NN_g. Figure 5.6 shows the structure of this neural network model. And its source code for MATLAB/Simulink implementation can be found in Appendix A.

Again, let $\hat{y}_1(k+d) = y_{d1}(k+d)$ and $\hat{y}_2(k+d) = y_{d2}(k+d)$, and rearrange the terms in (5.32), yielding the neural controller for TITO systems.

$$\begin{bmatrix} u_1(k+1) \\ u_2(k+1) \end{bmatrix} = \begin{bmatrix} NN_{g11}(\,) & NN_{g12}(\,) \\ NN_{g21}(\,) & NN_{g22}(\,) \end{bmatrix}^{-1} \left(\begin{bmatrix} y_{d1}(k+d) \\ y_{d2}(k+d) \end{bmatrix} - \begin{bmatrix} NN_{f1}(\,) \\ NN_{f2}(\,) \end{bmatrix} \right) \tag{5.33}$$

where the four outputs of network NN_g are arranged to form a square matrix. Similar to the case in exact feedback linearization, for aircraft, this matrix is usually nonsingular and can be inverted.

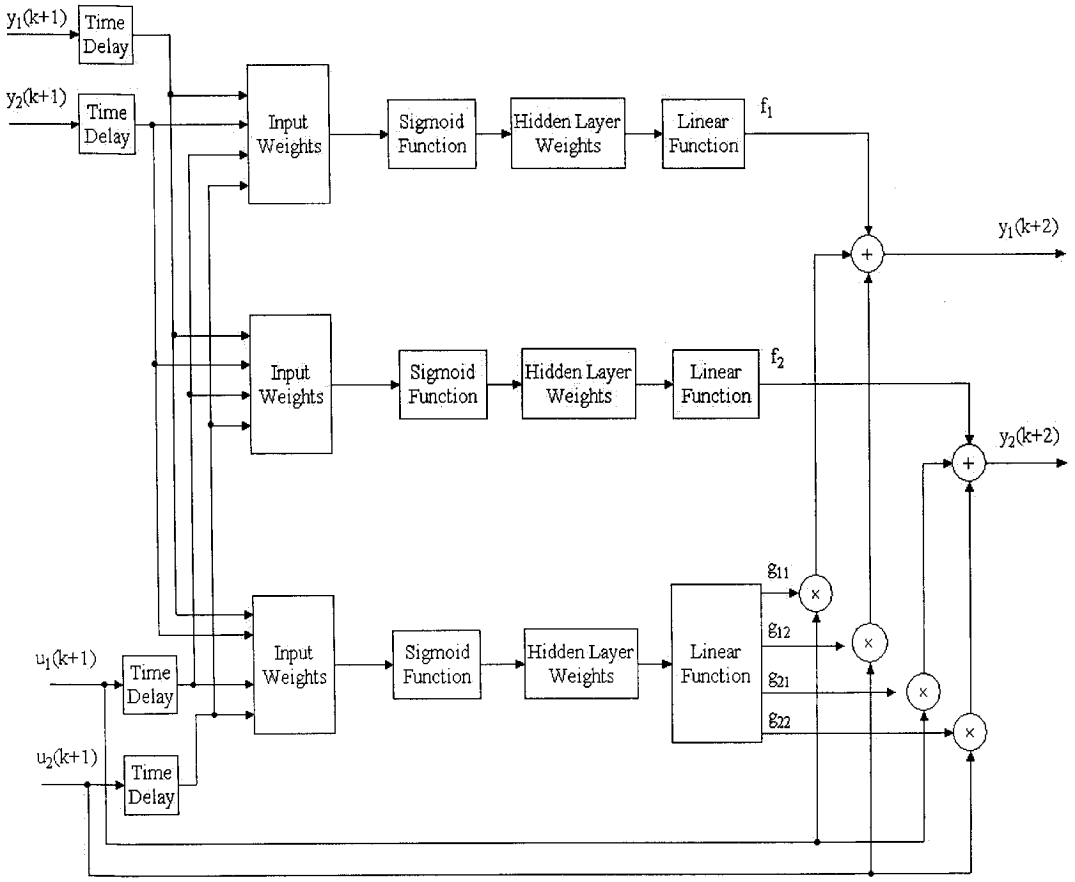

Figure 5.6 The structure of NARMA-L2 neural network for a TITO system

The block diagram of the neural controller for TITO systems (5.33) is shown in Figure 5.7.

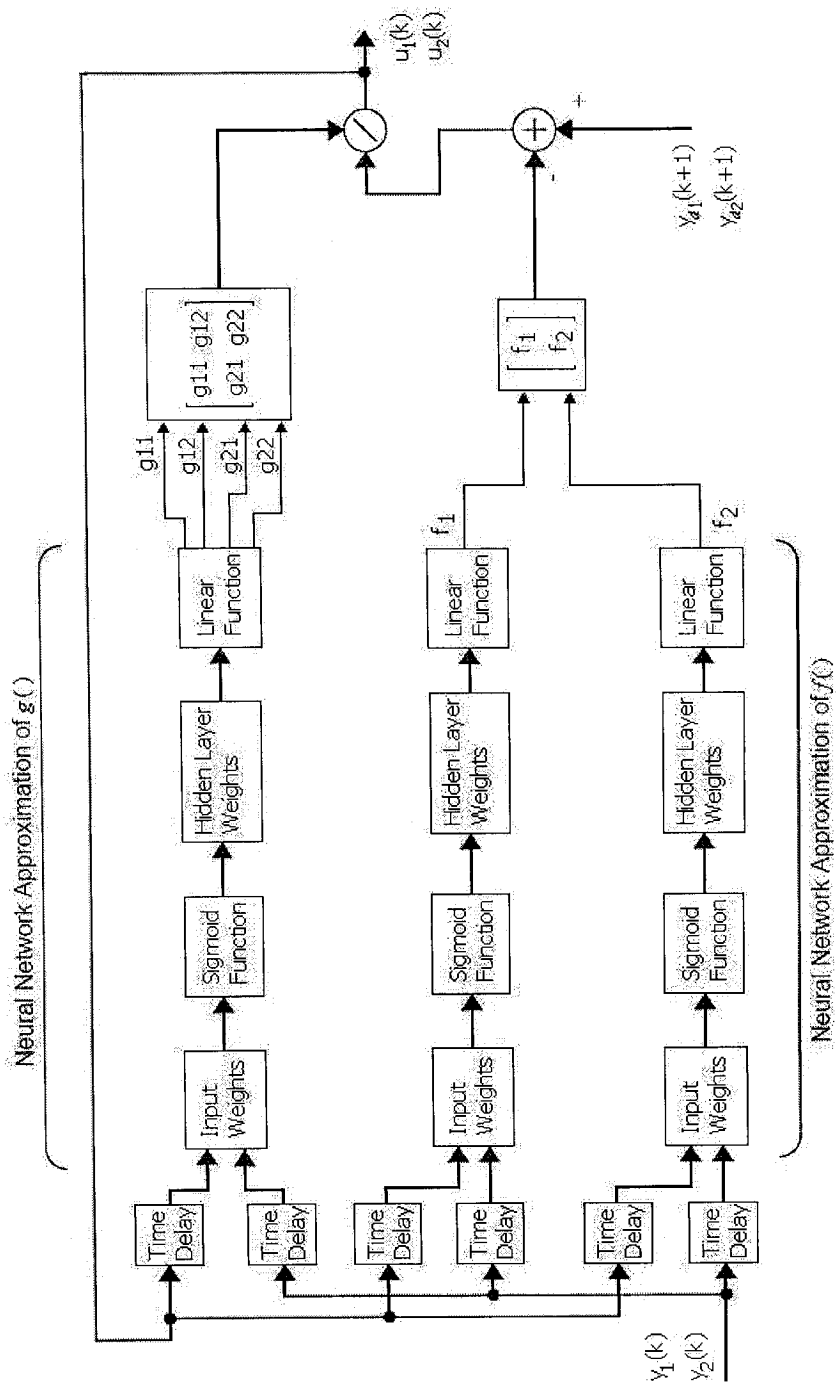

Figure 5.7 Block diagram of the neural controller for a TITO system

Chapter 6

Simulations and Results

In this chapter, the NARMA-L2 neural network based feedback linearization control and the output redefinition technique are applied to the lateral channel of the nonlinear nonminimum phase UAV systems developed and modified in chapter 3 and chapter 4.

6.1 Simulations on the SISO Assumption of the Nonlinear UAV Model

In order to verify if the control algorithm we discussed earlier is applicable to a nonminimum phase UAV system, we first do the simulation on an SISO assumption of the nonlinear UAV model described by

$$\begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \\ \Delta\dot{\alpha} \\ \dot{\beta} \\ \dot{\phi} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} l_\beta\beta + l_q q + l_r r + (l_{\beta\alpha}\beta + l_{r\alpha} r)\Delta\alpha + l_p p - i_1 qr \\ \overline{m}_\alpha \Delta\alpha + \overline{m}_q q + i_2 pr - m_{\dot{\alpha}} p\beta + m_{\dot{\alpha}}(g/V)(\cos\theta\cos\phi - \cos\theta_0) \\ n_\beta\beta + n_q q + n_r r + n_{p\alpha} p\Delta\alpha + n_p p - i_3 pq \\ q - p\beta + z_\alpha\Delta\alpha + (g/V)(\cos\theta\cos\phi - \cos\theta_0) \\ y_\beta\beta + p(\sin\alpha_0 + \Delta\alpha) - r\cos\alpha_0 + (g/V)\cos\theta\sin\phi \\ p + q\tan\theta\sin\phi + r\tan\theta\cos\phi \\ q\cos\phi - r\sin\phi \end{bmatrix} + \begin{bmatrix} \overline{l}_{\delta a} \\ 0 \\ \overline{n}_{\delta a} \\ 0 \\ y_{\delta a} \\ 0 \\ 0 \end{bmatrix} \delta_a$$

$$y = \beta \tag{6.1}$$

where the aileron deflection δ_a is the input, the sideslip angle β is chosen as the output, and the deflections of elevator δ_e and rudder δ_r in this assumption are set to zero. The block diagram of this model in MATLAB/Simulink® is shown in Figure 6.1.

From chapter 4, we know that this system is a nonminimum phase system which has three positive zeros (*s=24.4583, s=10.3755* and *s=0.2085*) in its Jacobian linearization. Therefore, before applying the feedback linearization control law to the system, the output must be redefined to yield a stable zero-dynamics.

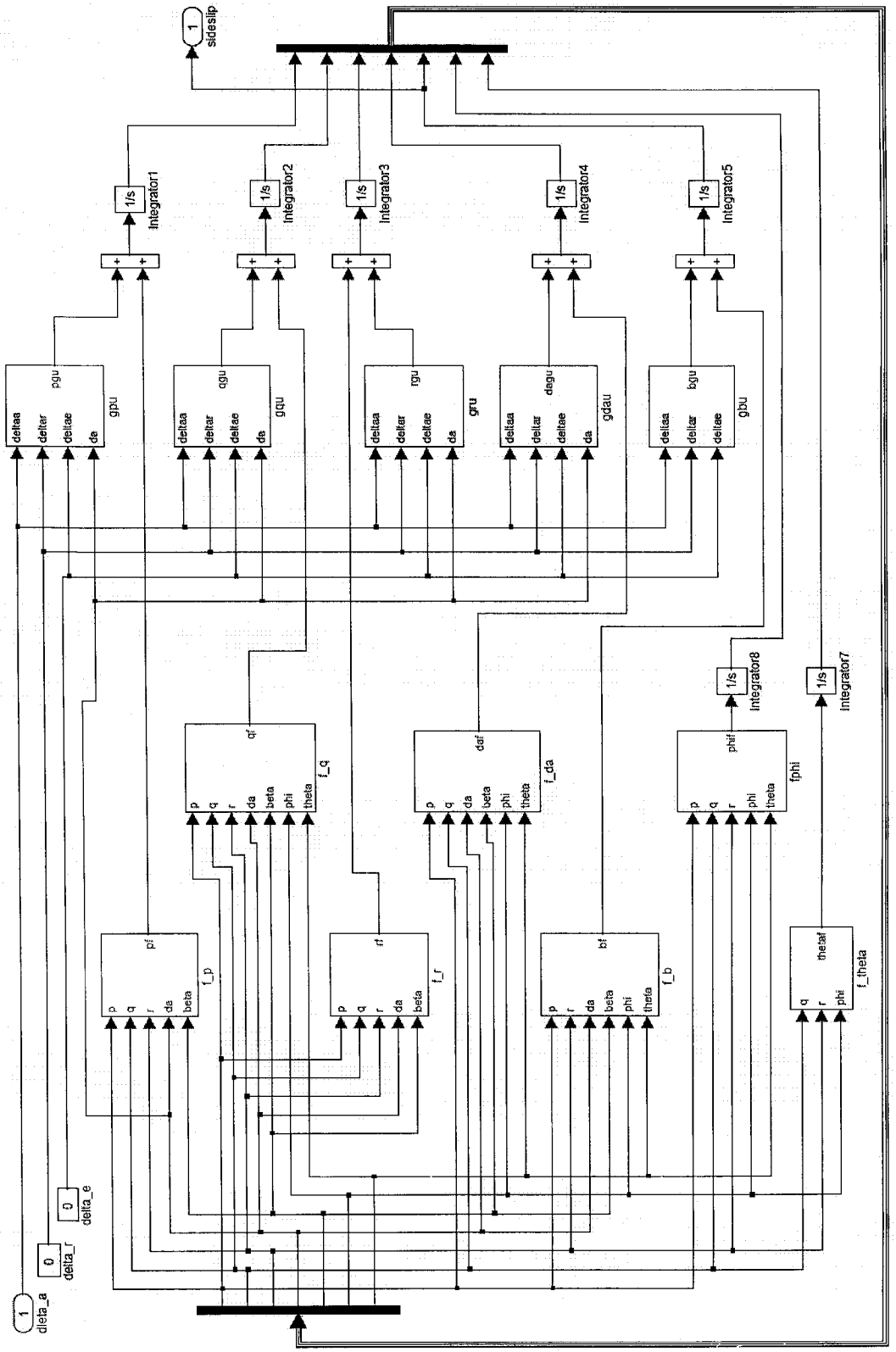

Figure 6.1 An SISO assumption of the nonlinear UAV model

6.1.1 Output Redefinition through Neglecting Positive Zeros

In chapter 4, the system output has been redefined as

$$y^* = 6.524\times10^{-4}p - 3.2547\times10^{-2}r - 1.1323\times10^{-2}\beta + 2.5157\times10^{-3}\phi, \qquad (6.2)$$

so that the positive zeros are eliminated from the transfer function of the Jacobian linearization of the nonlinear UAV model (6.1).

Then we generate two thousand input-output pairs in MATLAB/Simulink® from system (6.1) with the redefined output (6.2). These input-output pairs act as the training data for the NARMA-L2 neural network model. The NARMA-L2 neural network model has 10 neurons in its hidden layer, and all the weights and biases are initialized randomly. The training function we chose is Levenberg-Marquardt algorithm which is proven very efficient in MATLAB® implementation [40]. After several trials, the training stops when the mean square error between the neural output and target is less than 1×10^{-12}. Finally, we obtain the neural network controller by inverting the NARMA-L2 neural network model in Simulink®.

The Control scheme for the UAV model (6.1) is shown in figure 6.2.

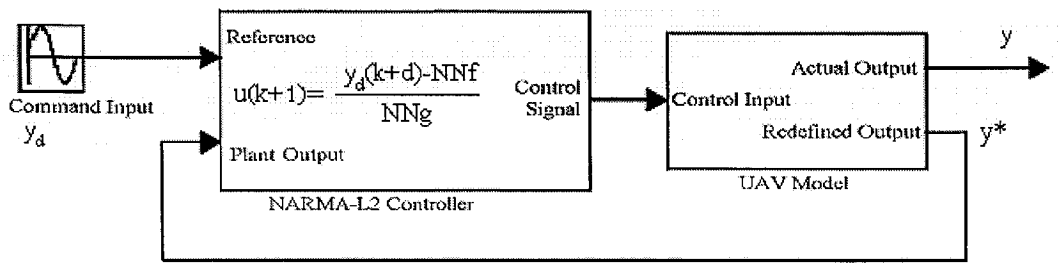

Figure 6.2 NARMA-L2 and output redefinition control scheme for SISO systems

From Figure 6.2 we can see that the delayed redefined outputs are fed back to the NARMA-L2 neural network controller. The controller, actually an approximate inverse model of the plant, cancels out the nonlinearities of the UAV model and forces the redefined output y^* to asymptotically track the desired trajectory (the command input in

Figure 6.2), which in turn yields an approximate tracking of the original output y.

In following simulations, the responses are based on the nominal flight condition I listed in Table 3.1. The initial conditions are $x(0)=0$, $\alpha_0=1.5^\circ$, $\theta_0=0$.

Figure 6.3 shows the response of the original output — sideslip angle β to a step command input.

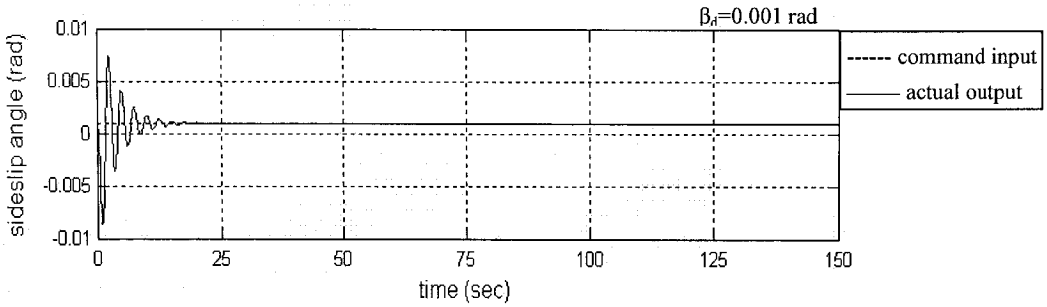

Figure 6.3 The sideslip angle response to a step command input

As we mentioned earlier, when the system output is redefined such that the positive zeros are eliminated, the tracking error will become large if the frequency of the desired output is high. For a step input, the frequency is very high at origin; therefore, we can see from Figure 6.3 that there is a significant tracking error at the transient stage of the step response.

The responses of the original output β to a desired sine wave trajectory are shown in figure 6.4. When the frequency of the desired trajectory is far less than the positive zero nearest to the imaginary axis, which is $s = 0.2085$, a good approximate tracking can be achieved. However, when the frequency of the command input is close to or larger than the positive zero nearest to the imaginary axis, the tracking performance becomes worse and cannot be accepted.

In Figure 6.5 and 6.6, the responses to the square wave and sawtooth wave of different frequencies are given. Similar to previous case, the tracking error becomes large

along with the increase of the frequency of the desired trajectory. Moreover, the tracking performance is always poor at points where the wave changes from the maximum value to the minimum value or vice versa.

From these simulations we can see that the output redefinition through eliminating the positive zeros only works well when the frequency of desired trajectory is much lower than the positive zero nearest to the imaginary axis. This is the main limitation of this technique.

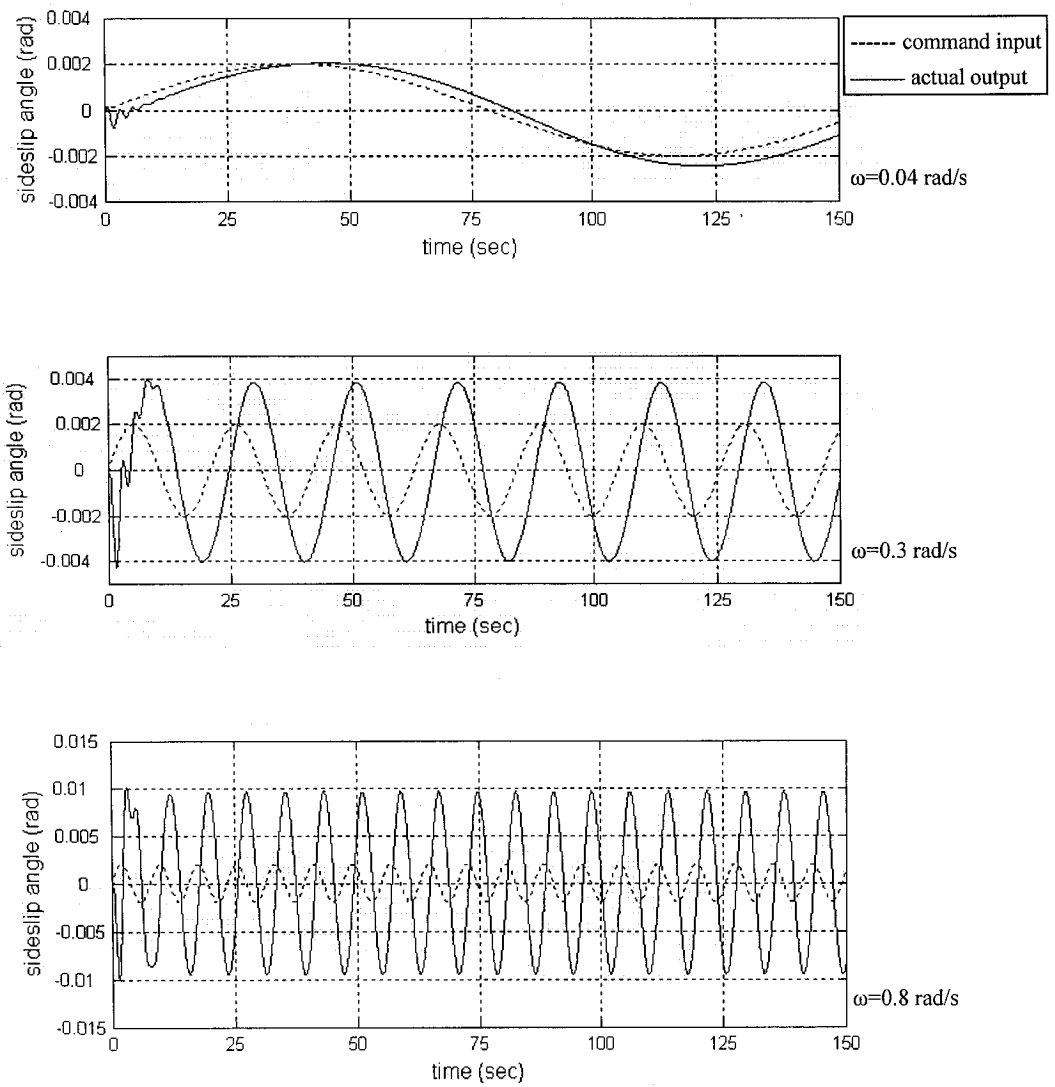

Figure 6.4 Sideslip angle responses to sine wave command inputs

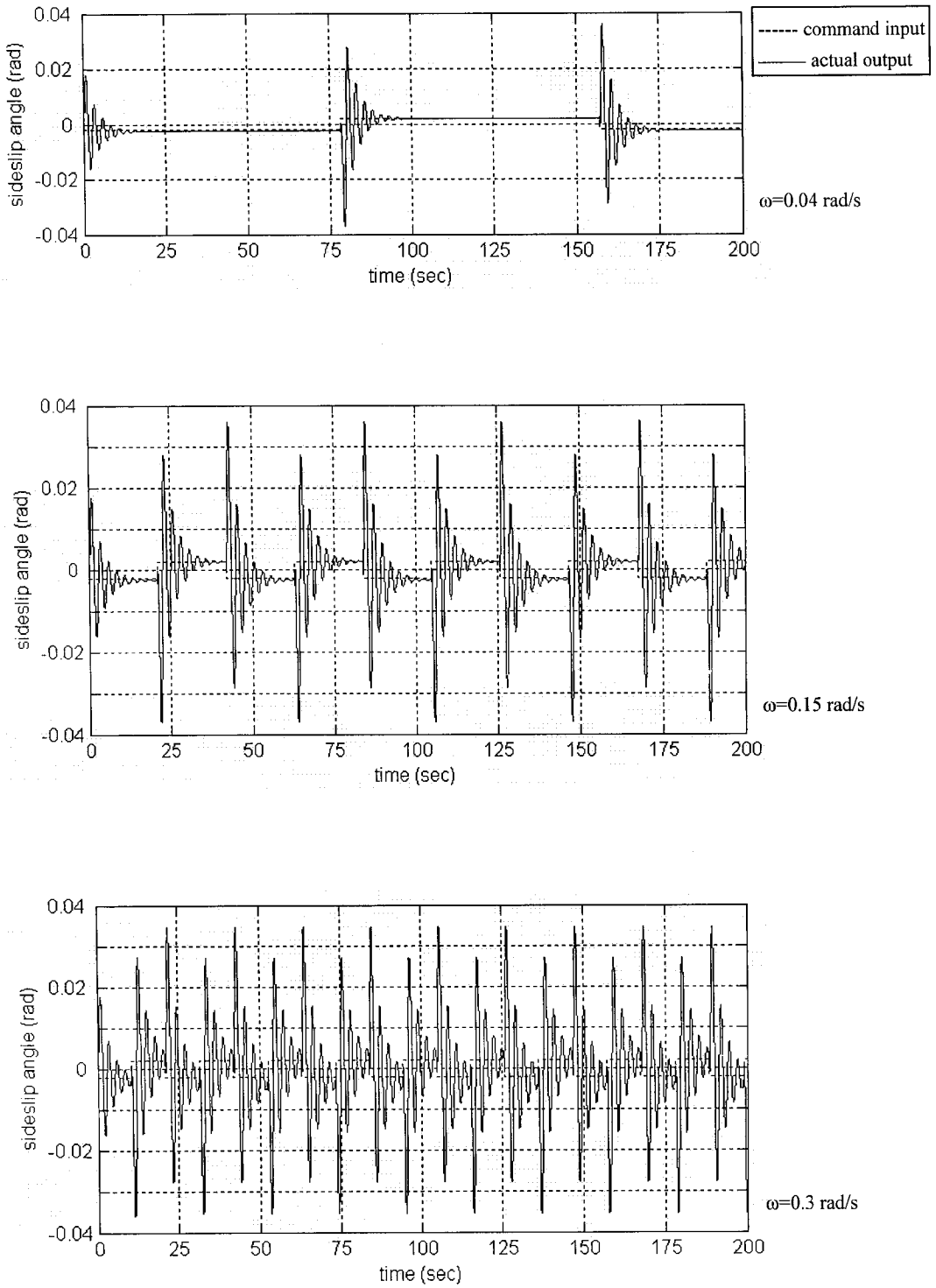

Figure 6.5 Sideslip angle responses to square wave command inputs

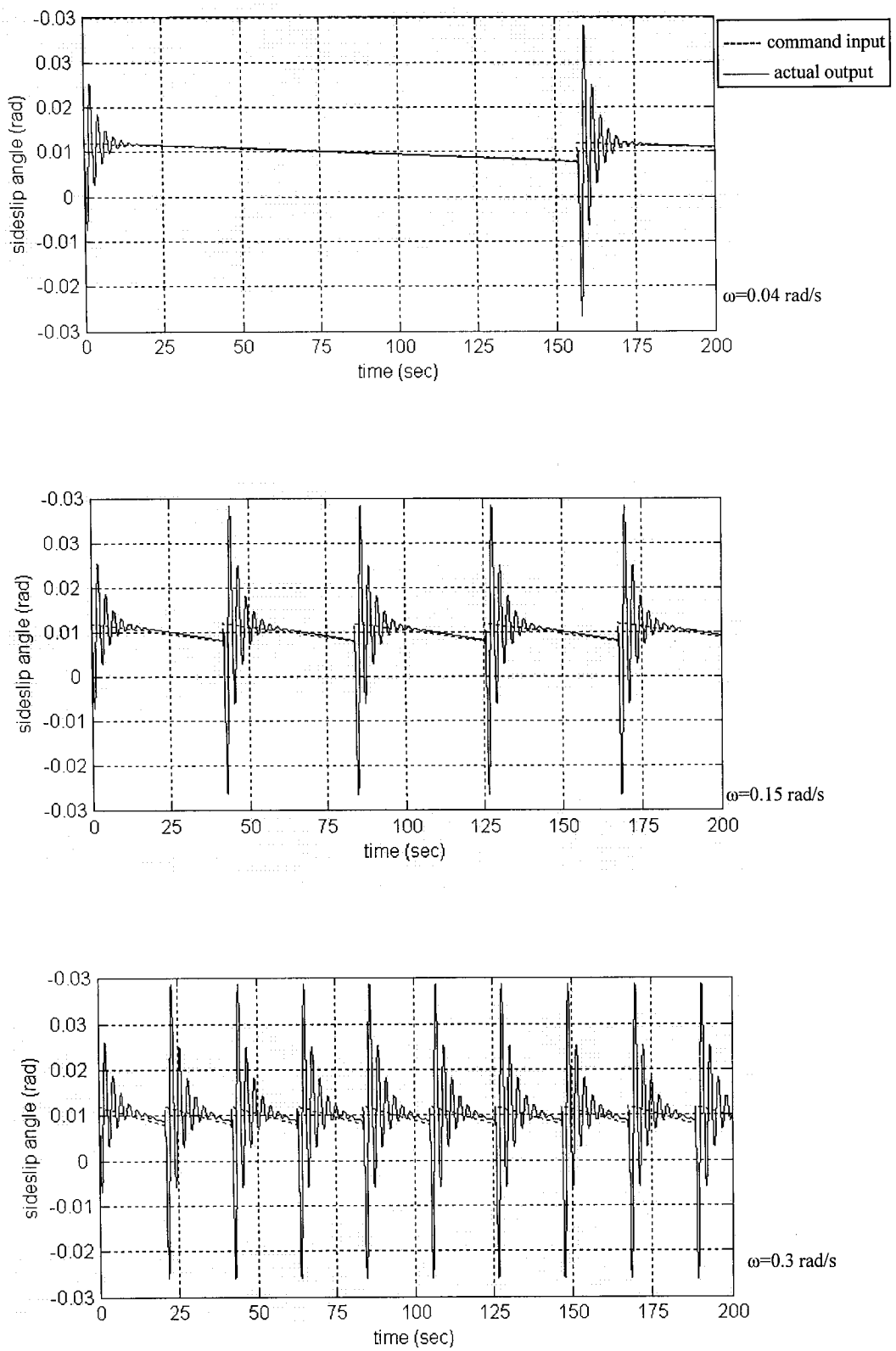

Figure 6.6 Sideslip angle responses to sawtooth wave command inputs

6.1.2 Output Redefinition through Moving Positive Zeros to LHP

Another method to redefine the output is to move the positive zeros to the left half of s plane and keep their magnitudes unchanged. By doing this in chapter 4, a new output of system (6.1) is defined as

$$y^* = -0.0036p + 0.1807r - 0.8237\beta + 0.0037\phi\,. \tag{6.3}$$

In this case, all the positive zeros (*s=24.4583, s=10.3755* and *s=0.2085*) become negative zeros, i.e. *s = -24.4583, s = -10.3755* and *s = -0.2085.*

Then, we generate another set of training data from system (6.1) with the output (6.3) for the NARMA-L2 network. All the settings to the network are the same as the previous case. When training is finished, we simply invert the neural model to obtain the neural network controller.

Using the same control scheme illustrated in Figure 6.2, the controller is able to cancel out the nonlinearities of the UAV model and forces the original output y to approximately track the desired trajectory.

Figure 6.7 shows the response of the original output — sideslip angle β to a step command input.

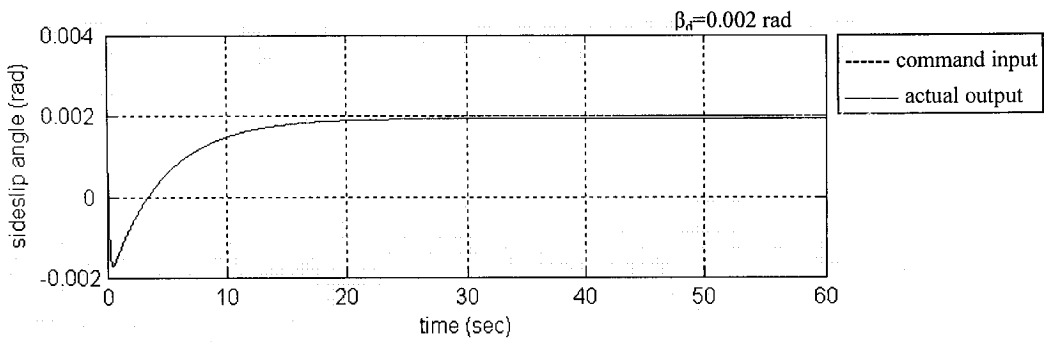

Figure 6.7 The sideslip angle response to a step command input

Compared with the step response in Figure 6.3, the transient response in Figure 6.7

does not contain significant oscillations. Also, in Figure 6.7 the transient tracking error is smaller than its counterpart in Figure 6.3. Therefore, from flight control point of view, the response shown in Figure 6.7 is superior to that in Figure 6.3.

The reason we have such a difference between responses shown in Figure 6.3 and 6.7 is that the output redefinition through moving the positive zeros to LHP always leads to a bounded tracking error however fast the frequency of the desired output is. This can further be verified by the following simulations.

The responses of the original output β to a desired sine wave trajectory are shown in Figure 6.8.

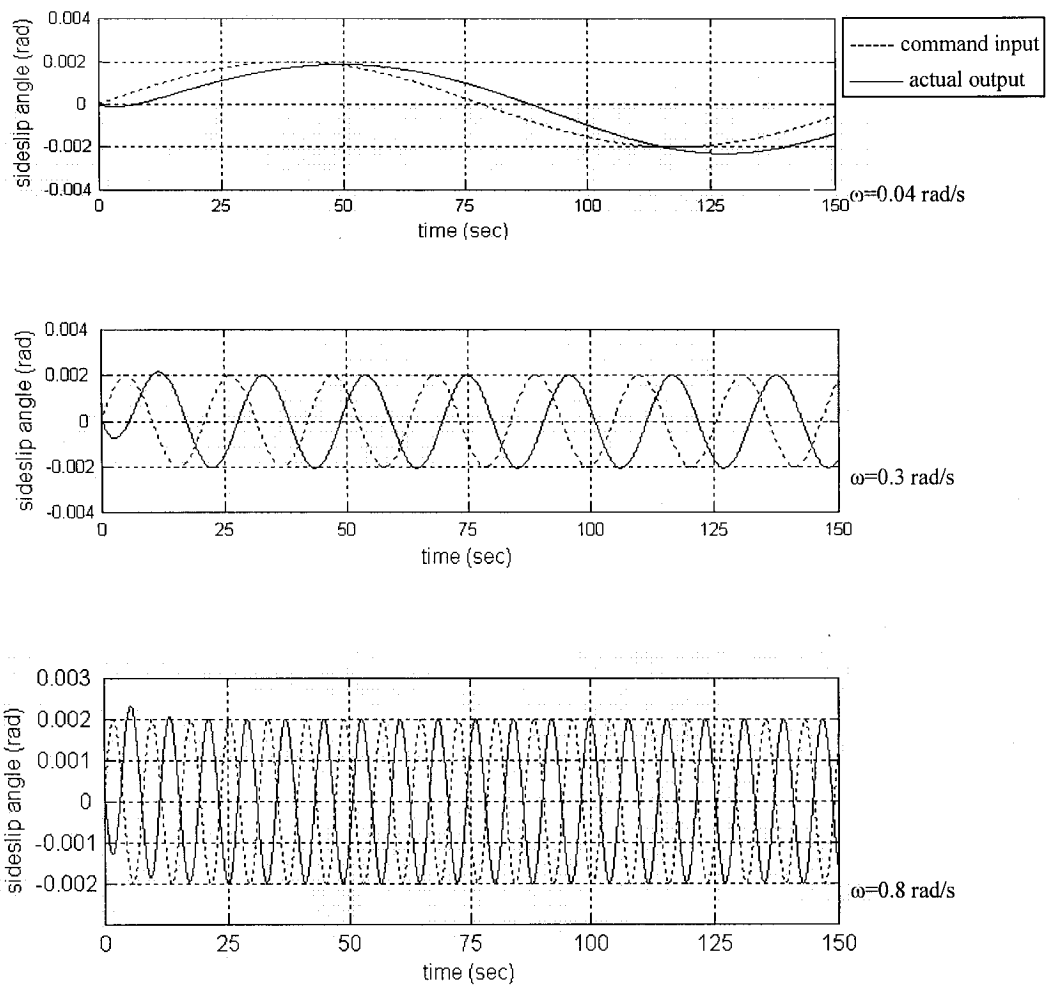

Figure 6.8 Sideslip angle responses to sine wave command inputs

In Figure 6.8, when the frequency of the desired trajectory is far less than the positive zero nearest to the imaginary axis, which is $s = 0.2085$, a good approximate tracking can be achieved, which is similar to the response shown in Figure 6.4. When the frequency of the command input is close to or larger than the positive zero nearest to the imaginary axis, the tracking performance becomes worse. But, different from the response shown in Figure 6.4, the tracking error in Figure 6.8 is always bounded. In extreme case when the frequency of the desired output goes to infinity, the system output y has the same amplitude with y_d, but with a phase shift of 180° (See equation 4.52-4.55).

In Figure 6.9 and 6.10, the responses to the square wave and sawtooth wave of different frequencies are given. Similar to previous case, the tracking error becomes large along with the increase of the frequency of the desired trajectory. We can also observe that the tracking error has the most significant value at points where the wave changes from the maximum value to the minimum value or vice versa, but compared with the response shown in Figure 6.5 and 6.6, the error is always bounded.

From these simulations we can see that the output redefinition through moving the positive zeros to LHP yields better tracking performance in that the tracking error is always bounded however fast the desired trajectory changes. The trade off is that this technique leads to a significant phase lag for high frequency sine wave input.

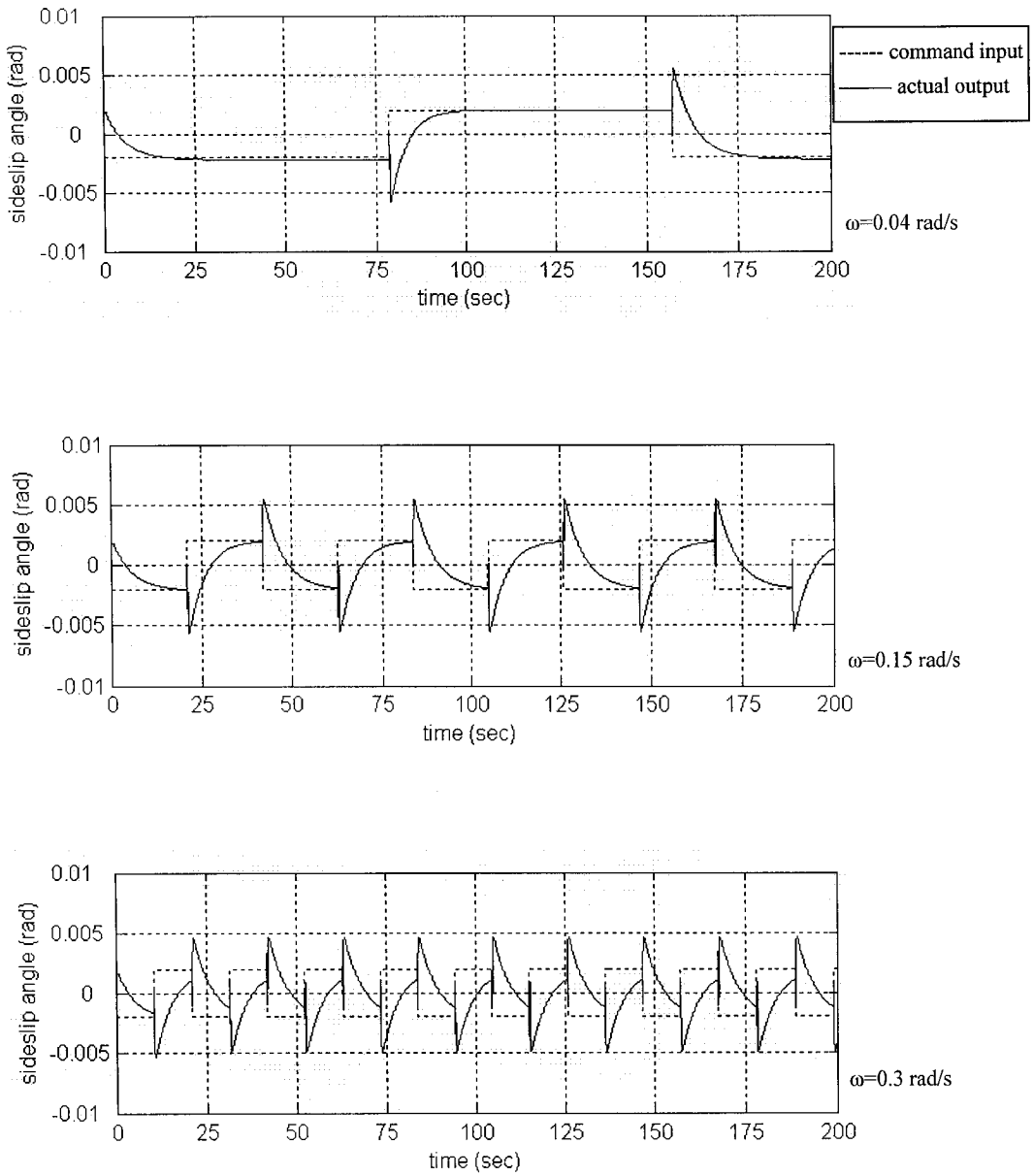

Figure 6.9 Sideslip angle responses to square wave command inputs

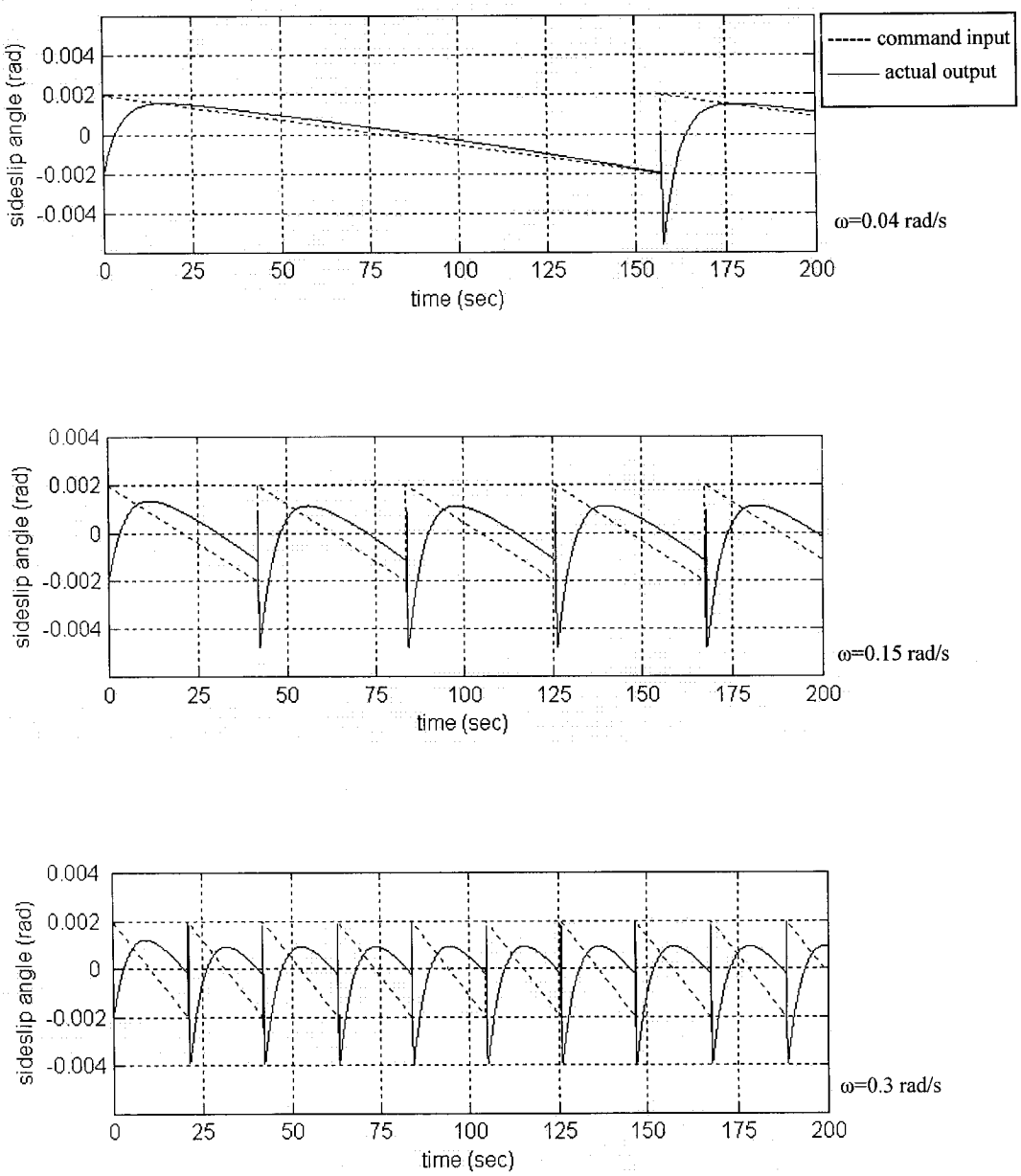

Figure 6.10 Sideslip angle responses to sawtooth wave command inputs

6.2 Simulations on the TITO Assumption of the Nonlinear UAV Model

In order to verify if the control algorithm we discussed earlier is applicable to an MIMO nonminimum phase UAV system, we will conduct the simulation on the TITO assumption of the nonlinear UAV model described by

$$\begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \\ \Delta\dot{\alpha} \\ \dot{\beta} \\ \dot{\phi} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} l_\beta\beta + l_q q + l_r r + (l_{\beta\alpha}\beta + l_{r\alpha} r)\Delta\alpha + l_p p - i_1 qr \\ \overline{m}_\alpha \Delta\alpha + \overline{m}_q q + i_2 pr - m_{\dot{\alpha}} p\beta + m_{\dot{\alpha}}(g/V)(\cos\theta\cos\phi - \cos\theta_0) \\ n_\beta\beta + n_q q + n_r r + n_{p\alpha} p\Delta\alpha + n_p p - i_3 pq \\ q - p\beta + z_\alpha\Delta\alpha + (g/V)(\cos\theta\cos\phi - \cos\theta_0) \\ y_\beta\beta + p(\sin\alpha_0 + \Delta\alpha) - r\cos\alpha_0 + (g/V)\cos\theta\sin\phi \\ p + q\tan\theta\sin\phi + r\tan\theta\cos\phi \\ q\cos\phi - r\sin\phi \end{bmatrix} + \begin{bmatrix} \bar{l}_{\delta a} & l_{\delta r} \\ 0 & 0 \\ \bar{n}_{\delta a} & n_{\delta r} \\ 0 & 0 \\ y_{\delta a} & y_{\delta r} \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \delta_a \\ \delta_r \end{bmatrix}$$

$$y_1 = \beta; \quad y_2 = \phi \tag{6.4}$$

where the aileron deflection δ_a and the rudder deflection δ_r are the inputs, the sideslip angle β and the roll angle ϕ are chosen as the outputs, and the elevator deflection δ_e in this experiment is set to zero. The block diagram of this model in MATLAB/Simulink® is shown in Figure 6.11.

From chapter 4, we know that this system is a nonminimum phase system, with a positive zero located at $s = 5368.31$ in its Jacobian linearization. The output redefinition through moving the positive zero to LHP has been used in chapter 4 to define new outputs which make the zero-dynamics asymptotically stable. The new outputs are given by

$$\begin{aligned} y_1^* &= 3.2053\times10^{-4} p - 3.7617\times10^{-4} r + \beta + 0.82619\phi \\ y_2^* &= \phi \end{aligned} \tag{6.5}$$

Then, we generate two thousand input-output pairs as training data in MATLAB/Simulink® from system (6.4) with the outputs (6.5). Different from the SISO case, each of the input-output pairs for the system (6.4) contains two inputs and two outputs. The NARMA-L2 neural network model has 15 neurons in its hidden layer. All other settings to the network are the same as the previous cases. Finally, we obtain the neural network controller by inverting the properly trained NARMA-L2 neural network model in Simulink®.

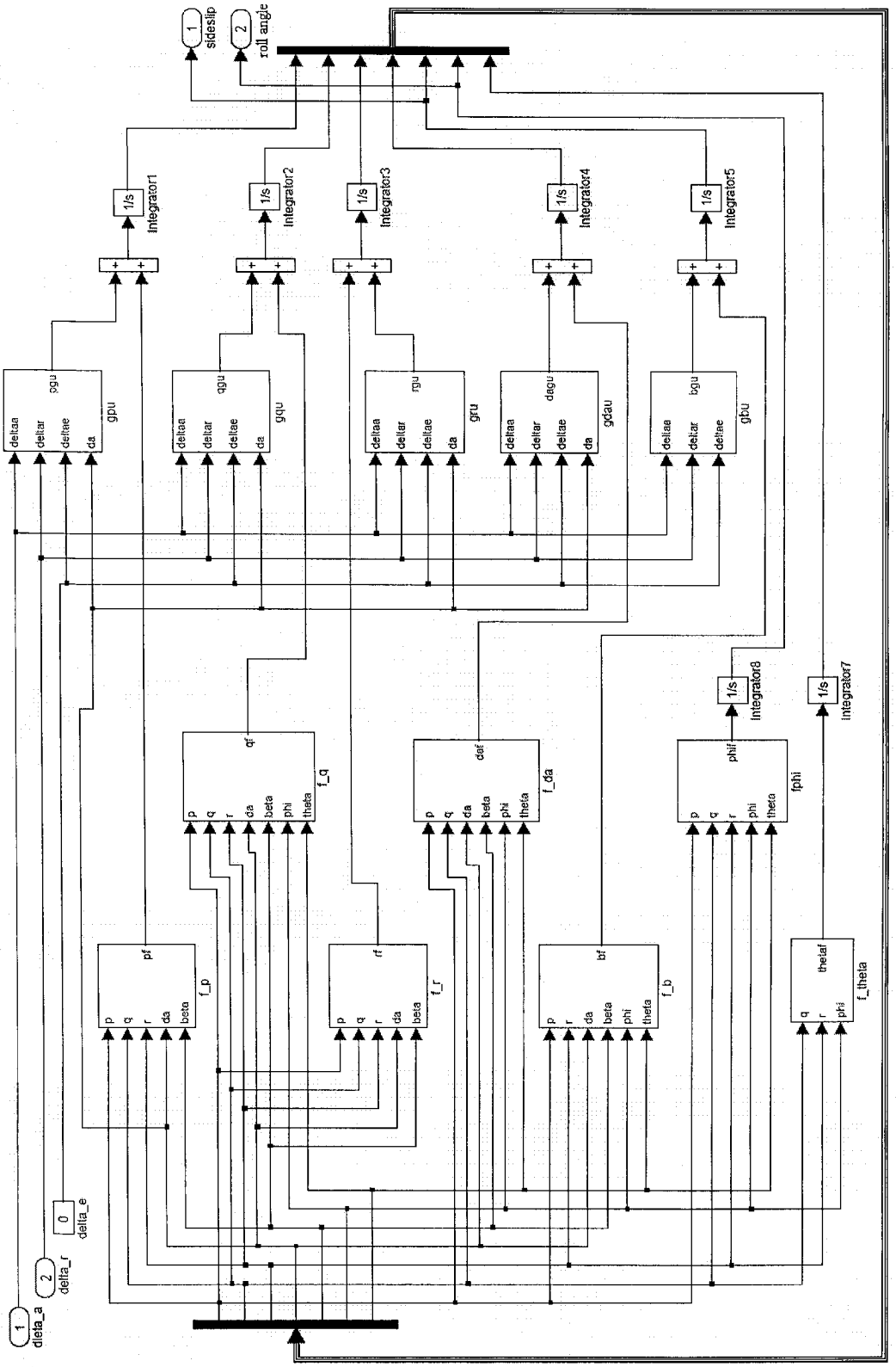

Figure 6.11 A TITO assumption of the nonlinear UAV model

The Control scheme for the UAV model (6.2) is shown in Figure 6.12.

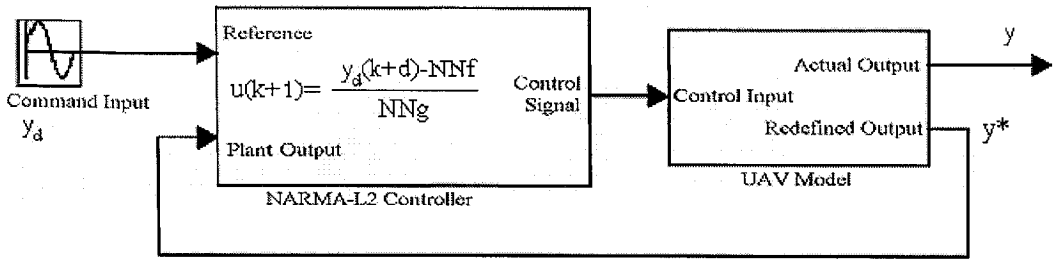

Figure 6.12 NARMA-L2 and output redefinition control scheme for TITO systems

From Figure 6.12 we can see that the whole system is able to handle two inputs and two outputs. The controller cancels out the nonlinearities of the UAV model and forces the redefined output vector y^* to asymptotically track the command input vector, which in turn yields an approximate tracking of the original output vector y.

In the following simulations, the desired sideslip angle β_d is set to zero, and the desired the roll angle ϕ_d is a step input in Figure 6.13 and a sine wave input in Figure 6.14. The responses are based on the nominal flight condition I listed in Table 3.1. The initial conditions are $x(0)=0$, $\alpha_0=1.5^\circ$, $\theta_0=0$.

From Figure 6.13 and 6.14 we can see that the outputs of the system have a good tracking performance that is because the frequency of the command inputs is far below the positive zero ($s=5368.31$).

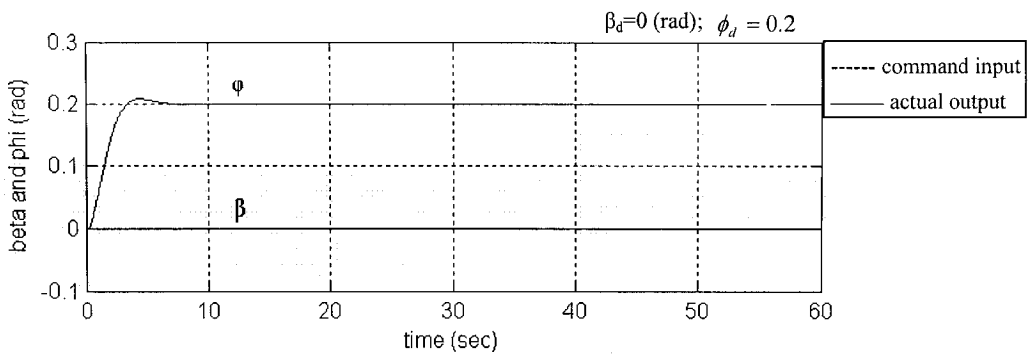

Figure 6.13 Sideslip angle and roll angle responses

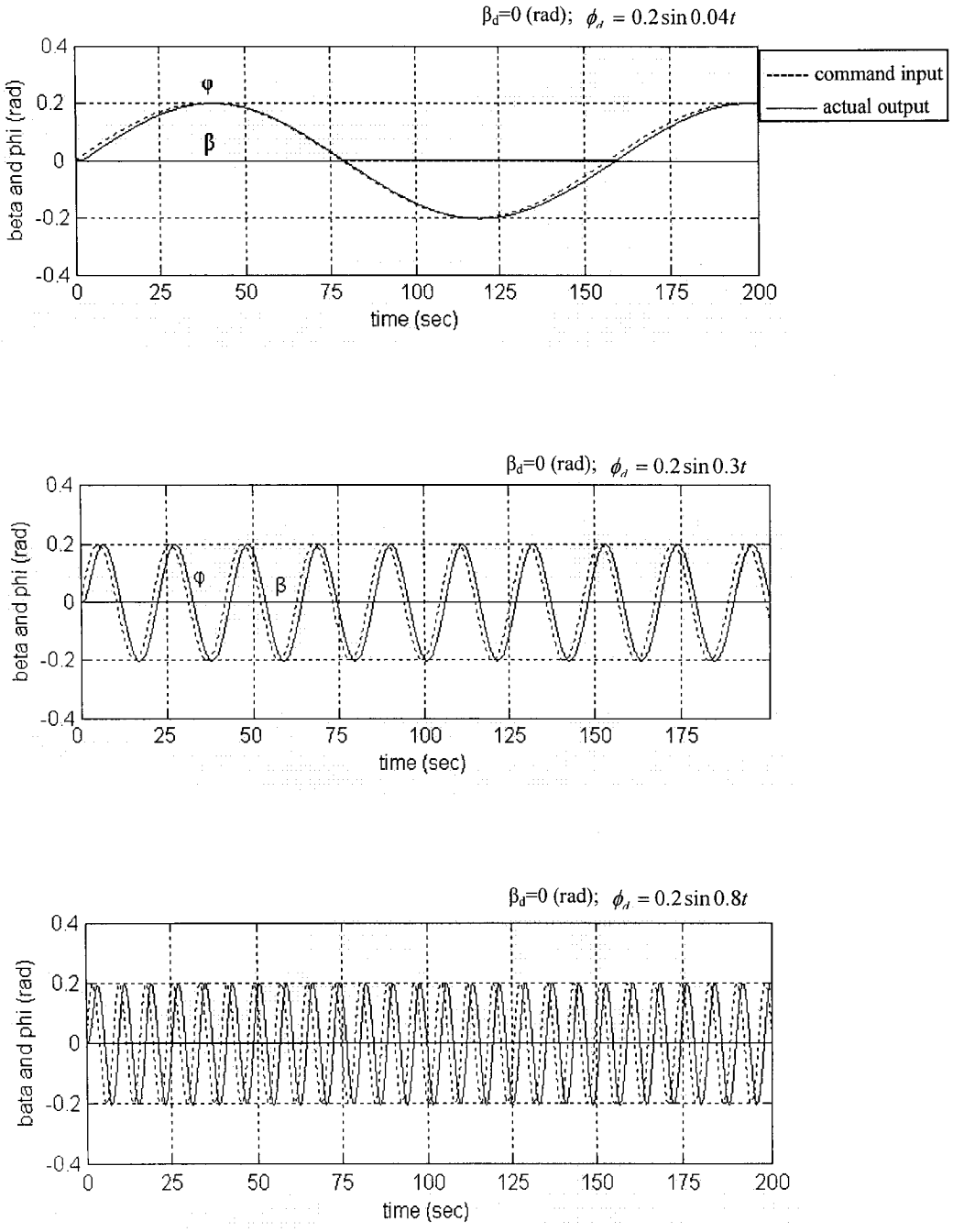

Figure 6.14 Sideslip angle and roll angle responses

Chapter 7

Conclusion and Future Work

7.1 Conclusion

In this dissertation we have presented a methodology for the nonlinear control of a nonminimum phase UAV system. The nonminimum phase UAV model is approximated with a minimum phase one using the output redefinition method which was carried out by either neglecting the positive zeros from the transfer function of the Jacobian linearization or our new method, i.e. moving the positive zeros to the left half of s plane. The simulation results shows that when the frequency of the desired trajectory is less than the smallest positive zero, a good approximate tracking can be achieved. However, when the frequency of the desired trajectory is close to or larger than the positive zeros, the tracking error becomes large. We also observed that our new method, using the output redefinition through moving the positive zeros to the left half of s plane, yields better tracking performance than the method of neglecting the positive zeros. This is also in good agreement with the theoretical prediction.

Another conclusion drawn from this research is that nonlinear autoregressive moving average (NARMA-L2) neural network based approximate feedback linearization is able to replace the exact feedback linearization for the control of the UAV. In this thesis, we expanded the nonlinear autoregressive moving average (NARMA-L2) neural network from one degree of freedom to two degree of freedom so that the neural network is able to model and control more complicated systems. Our simulation results show that the neural controller successfully modeled the inverse dynamics of the UAV model and forced the redefined output to asymptotically track the desired output. Therefore, it holds

the potential to simplify the problem of autopilot design for high performance UAV system. The limitation of this neural controller is that it is unable to offer the freedom to obtain the transient responses of the system with desired characteristics.

7.2 Contributions

As we mentioned earlier, the nonlinear inversion of a UAV model in real time is computationally intensive. To reduce the computation burden of the onboard computer of a UAV, an approximate feedback linearization methodology using offline trained nonlinear autoregressive moving average (NARMA-L2) neural networks are proposed. In this research, two neural network controllers respectively for a single-input-single-output (SISO) system and a two-input-two-output (TITO) system were established and simulated.

Another important issue that we focused on is the nonminimum phase problem. Since the feedback linearization cannot be directly applied to a UAV which is nonminimum phase, a new approach of the output redefinition method was suggested in this thesis to redefine the output of the system, such that the resulting new system has stable zero-dynamics and the inverse of the new system is asymptotically stable (In linear case, the positive zeros are moved to the left half of the s plane).

As a part of the our research on the nonminimum phase problem, we also tried to figure out which factor in the equations of a UAV model makes system behave nonminimum phase or contributes to positive zeros, and how the positive zeros (or the positive eigenvalues of the linearized zero-dynamics) change there values when the UAV is flying under different conditions. Since the positive zeros pose limitations on the frequency response of a UAV and hence affect its maneuverability, answers to these

questions may help to quantify the resulting path and corridor of a single UAV and multi-UAVs.

7.3 Recommendations for Future Work

Following the research described in this thesis, a number of projects arising from this work could be taken up.

- Study on the global stability of the internal dynamics of the UAV model.

  Since the stability of the zero-dynamics only guarantees the local stability of the internal dynamics, one may check the stability of the zero-dynamics for every operating point, or find a global stability design algorithm to stabilize the internal dynamics.

- Robust design of the neural controller.

  To apply the feedback linearization control law, the plant model must be exactly known. However, this is impossible in practice because of the uncertainties in parameters and the unmodeled dynamics. To solve this problem, one may adopt an online learning neural network to compensate the modeling error, thus improve the robustness of the controller.

- Implementation of this control algorithm to a helicopter model.

  A helicopter is inherently a nonminimum phase system. In chapter 3, we have identified the term in its motion equations which causes this problem. Therefore, the output redefinition method has the potential to overcome this difficulty. However since the helicopter is generally unstable, and heavily coupled in motions, it might be extremely difficult to force the helicopter to track a desired trajectory.

REFERENCES

[1] Herold, M. W. "The problems with the predator", Jan. 12, 2003. Retrieved Dec 12, 2004 on the World Wide Web: http://www.cursor.org/stories/dronesyndrome.htm.

[2] Quigley, M., Goodrich, M. A., Beard, R. W. "Semi-Autonomous Human-UAV Interfaces for Fixed-Wing Mini-UAVs". *Proceedings of IROS* 2004. Sept 28-Oct 2, 2004, Sendai, Japan.

[3] Mettler, B., Schouwenaars, T., How, J., Paunicka, J., Feron, E. "Autonomous UAV Guidance Build-Up: Flight-Test Demonstration and Evaluation Plan", *Proceeding of the AIAA Guidance, Navigation, and Control Conference*, Aug 2003. AIAA-2003-5744.

[4] Valenti, M., Schouwenaars, T., Kuwata, Y., Feron, E., How, J. "Implementation of a Manned Vehicle - UAV Mission System", *AIAA Guidance, Navigation, and Control Conference and Exhibit*, 16-19 Aug 2004, Providence, Rhode Island. AIAA-2004-5142.

[5] Christopher, M., Jones, A. "Unmanned Aerial Vehicles (UAVs): An Assessment of Historical Operations and Future Possibilities", *Air Command and Staff College, Air University. Report Number: AU/ACSC/0230D/97-03*. Available on the World Wide Web: https://research.au.af.mil/ papers/ay1997/acsc/97-0230D.pdf

[6] Isidori, A. *Nonlinear Control System*, Third Edition, Springer, 1995.

[7] Slotine, J. E., Li, W. *Applied Nonlinear Control*, Prentice-Hall, Englewood Cliffs, N.J., 1991.

[8] Enns, D., Bugajski, D.J., Hendrick, R.C., Stein, G. "Dynamic Inversion: An Evolving Methodology for Flight Control Design", *Int. J. Control*, 1994, Vol. 59, No.1, 71-91.

[9] Kim, B S., Calise, A. J. "Nonlinear Flight Control Using Neural Networks", *AIAA Journal of Guidance, Control and Dynamics*, Vol. 20, No. 1, January-February 1997.

[10] Yan, L., Li, C. J. "Robot learning control based on recurrent neural network inverse

model", *J. Robot. Syst.*, vol.14, no.3, pp.199–212, 1997.

[11] Plett, G. L. "Adaptive Inverse Control of Linear and Nonlinear Systems Using Dynamic Neural Networks", *IEEE Transactions on Neural Networks*, Vol. 14, No. 2, March 2003.

[12] McLean, D. *Automatic Flight Control System*, Prentice Hall 1990.

[13] Stevens, B. L., Lewis, F. L. *Aircraft Control and Simulation*, Second Edition, Wiley, 2003.

[14] Leith, D., Leithead, W. "Survey of gain-scheduling analysis and design", *Int. J. Control*, vol. 73, no. 11, pp. 1001–1025, 2000.

[15] Fujimori, A., Terui, F., Nikiforuk, P. N. "Flight Control Design of an Unmanned Space Vehicle Using Gain Scheduling", *AIAA Journal of Guidance Control and Dynamics*, v. 28 , n. 1 , p. 96 , 10 p, 2005.

[16] Ito, D., Georgie, J., Valasek, J., Ward, D. T. "Reentry Vehicle Flight Controls Design Guidelines: Dynamic Inversion", *NASA/TP*—2002—210771.

[17] Chen, Z. "Dynamic Inversion and Model Predictive Control for Unmanned Aerial Vehicles", *Master Thesis*, University of Ottawa, 2004.

[18] Johnson, E. N., Calise, A. J., El-Shirbiny, H. A., Rysdyk, R. T. "Feedback Linearization with Neural Network Augmentation applied to X-33 Attitude Control", in *AIAA Guidance, Navigation and Control Conference*, Denver, CO, August 2000.

[19] Steinberg, M. L. "Comparison of Intelligent, Adaptive, and Nonlinear Flight Control Laws", *AIAA Journal of Guidance, Control, and Dynamics*, Vol. 24, No. 4, July-August 2001.

[20] Hauser, J., Sastry, S., Meyer, G. "Nonlinear Control Design for Slightly Non-minimum Phase Systems: Application to V/STOL Aircraft", *Automatica*, Vol. 28, No. 4, pp. 665-679, 1992.

[21] Oishi, M., Tomlin, C. "Switching in Nonminimum Phase Systems: Applications to a VSTOL Aircraft", in *Proceedings of the American Control Conference*, 2000.

[22] Romano, J. J., Singh, S. N. "I-O Map Inversion, Zero Dynamics and Flight Control", *IEEE Transactions on Aerospace and Electronic Systems*, Vol. 26, No. 6, November 1990.

[23] Moallem, M., Khorasani, K., Patel, R. V. "An Inverse Dynamics Sliding Control Technique for Flexible Multi-Link Manipulators", *Proceedings of the American Control Conference*, Albuquerque, New Mexico, June 1997.

[24] Menon, P. K., Yousefpor, M. "Design of Nonlinear Autopilots for High Angle of Attack Missiles", *AIAA Guidance, Navigation and Control Conference*, August 11-13, San Diego, CA, 1996.

[25] Talebi, H. A., Khorasani, K., Patel, R.V. "Experimental Results on Tracking Control of a Flexible-Link Manipulator: A New Output Re-definition Approach", *Proceedings of the 1999 IEEE International Conference on Robotics and Automation*, Detroit, Michigan, May 1999.

[26] Benvenuti, L., Benedetto, M. D. DI. "Approximate Output Tracking for Nonlinear Non-minimum Phase Systems with An Application to Flight Control", *International Journal of Robust and Nonlinear Control*, Vol. 4, 397-414, 1994.

[27] Hornik, K., Stinchcombe, M., White, H. "Multilayer Feedforward Networks are Universal Approximators", *Neural Networks*, Vol.2 pp 356-368, 1989.

[28] Hunt, K. J., Sbarbaro, D., Zbikowski, R., Gawthrop, P. J. "Neural Networks for Control Systems — A Survey", *Automatica*, Vol.28, No.6, pp1083-1112, 1992.

[29] Calise, A. J., Sharma, M., Corban, J. E. "Adaptive Autopilot Design for Guided Munitions", *AIAA Journal of Guidance, Control, and Dynamics*, 23(5), 2000.

[30] Rysdyk, R. T., Calise, A. J. "Adaptive Model Inversion Flight Control for Tiltrotor Aircraft", *AIAA Journal of Guidance, Control, and Dynamics*, 22(3):402--407,

1999.

[31] Hussaina, M. A., Kittisupakorn, P., Daosud, W. "Implementation of Neural-Network-Based Inverse-Model Control Strategies on an Exothermic Reactor", *ScienceAisa* 27 (2001): 41-50.

[32] Su, Z., Khorasani, K. "A Neural-Network-Based Controller for a Single-Link Flexible Manipulator Using the Inverse Dynamics Approach", *IEEE Transactions on Industrial Electronics*, Vol. 48, No. 6, December 2001.

[33] Narendra, K. S., Mukhopadhyay, S. "Adaptive Control Using Neural Networks and Approximate Models", *IEEE Transactions on Neural Networks*, Vol. 8, No. 3, May 1997.

[34] He, S., Reif, K., Unbehauen, R. "A Neural Approach for Control of Nonlinear Systems with Feedback Linearization", *IEEE Transactions on Neural Networks*, Vol. 9, No. 6, November 1998.

[35] Hacker, T., Oprisiu, C. "A Discussion of the Roll-Coupling Problem", *Progress in Aerospace Science*, 15(1974), 151-180.

[36] Grizzle, J. W., Benedetto, M. D. DI. "Approximations by regular input-output maps", *IEEE Transactions on Automatic Control*, AC-37, 1052-1054, 1992.

[37] Koo, T. J., Sastry, S. "Output Tracking Control Design of a Helicopter Model Based on Approximate Linearization", *Proceedings of the 37th IEEE Conference on Decision & Control*, Tampa, Florida, USA, December 1998.

[38] Mahony, R., Hamel, T. "Robust Trajectory Tracking for a Scale Model Autonomous Helicopter", *International Journal of Robust and Nonlinear Control*, 2004; 14: 1035-1059.

[39] Kailath, T. *Linear Systems*, Prentice Hall, Englewood Cliffs, NJ, 1980.

[40] Demuth, H., Beale, M. *Neural Network Toolbox User's Guide*, Version 4, The MathWorks, 2002.

Appendix A

The Source Code of the TITO NARMA-L2 Neural Controller (modified from the code "nnident.m" in MATLAB/Simulink Neural Network Toolbox)

```
function nnident(cmd,arg1,arg2,arg3)

func_index =['trainbfg';'trainbr';'traincgb';'traincgf';'traincgp';'traingd';'traingdm';'traingda';'traingdx';...
'trainlm ';'trainoss';'trainrp';'trainscg'];                    %Define the names of training functions

if nargin == 0, cmd = ' '; else cmd = lower(cmd); end
fig = 0; fig=findall(0,'type','figure','tag','nnident');        % Find window if it exists
if (size(fig,1)==0), fig=0; end
if (length(get(fig,'children')) == 0), fig = 0; end
if fig                                                          % Get window data if it exists
  H = get(fig,'userdata');
  if strcmp(cmd,'')
    if get(H.gcbh_ptr,'userdata')~=arg1
      delete(fig);
      fig=0;
    end
  else
    if isfield(get(H.gcbh_ptr,'userdata'))
      if isfield(get_param(get_param(get(H.gcbh_ptr,'userdata'),'parent'),'objectparameters'),...
'SimulationStatus')
SimulationStatus=get_param(get_param(get(H.gcbh_ptr,'userdata'),'parent'),'simulationstatus');
      else  SimulationStatus='none';
      end
    else  SimulationStatus='none';
    end
    if (strcmp(SimulationStatus,'running')|strcmp(SimulationStatus,'paused')) & ~strcmp(cmd,'close')
      set(H.error_messages,'string','You must stop the simulation to change NN configuration...
parameters.');
      return;
    end
  end
end

if strcmp(cmd,'') | isempty(cmd)                                % Activate the window.
  if fig
    figure(fig)
    set(fig,'visible','on')
  else
    nncontrolutil('nnident','init',arg1,arg2,arg3)
  end
elseif strcmp(cmd,'close') & (fig)                              % Close the window.
```

```
    arg1=get(H.gcbh_ptr,'userdata');
    arg2=get(H.gcb_ptr,'userdata');
    if exist(cat(2,tempdir,'nnidentdata.mat'))
        delete(cat(2,tempdir,'nnidentdata.mat'));
    end
    parent_function=get(H.parent_function_ptr,'userdata');
    if ~strcmp(parent_function,'narma_l2')
        feval(parent_function,'',arg1,arg2,'nnident');
    end
    delete(fig);

elseif strcmp(cmd,'init') & (~fig)                    % Initialize the window. Create a user interface.
    sys_par=arg2;
    sys_par2=arg2;
    while ~isempty(sys_par2)
        sys_par=sys_par2;
        sys_par2=get_param(sys_par,'parent');
    end
    if strcmp('on',get_param(sys_par,'lock'))
        window_en='off';
    else   window_en='on';
    end
    H.StdColor = get(0,'DefaultUicontrolBackgroundColor');
    H.StdUnit='points';
    H.PointsToPixels = 72/get(0,'ScreenPixelsPerInch');
    if strcmp(arg3,'narma_l2')
       H.me='Plant Identification - NARMA-L2';
    end
    fig = figure('Units',H.StdUnit, ...
    'Color',[0.8 0.8 0.8], 'IntegerHandle',    'off', 'Interruptible','off', 'BusyAction','cancel', ...
     'HandleVis','Callback', 'MenuBar','none', 'Name',H.me, 'Numbertitle','off', ...
     'PaperUnits',H.StdUnit, 'Position',[45 30 350 358], 'Tag','nnident', 'ToolBar','none');
    frame4 = uicontrol('Parent',fig, 'Units',H.StdUnit, 'BackgroundColor',[0.8 0.8 0.8], ...
    'ListboxTop',0, 'Position',[5 2 340 22], 'Style','frame', 'Tag','Frame4');
    frame5 = uicontrol('Parent',fig, 'Units',H.StdUnit, 'BackgroundColor',[0.8 0.8 0.8], ...
     'ListboxTop',0, 'Position',[5 26 340 67], 'Style','frame', 'Tag','Frame5');
    h1 = uicontrol('Parent',fig, 'Units',H.StdUnit, 'BackgroundColor',[0.8 0.8 0.8], ...
       'Enable',window_en, 'ListboxTop',0, 'Position',[130 83.5 90.25 15], ...
     'String','Training Parameters', 'Style','text', 'Tag','StaticText1');
    frame1 = uicontrol('Parent',fig, 'Units',H.StdUnit, 'BackgroundColor',[0.8 0.8 0.8], ...
     'ListboxTop',0, 'Position',[5 103 340 137], 'Style','frame', 'Tag','Frame1');
    h1 = uicontrol('Parent',fig, 'Units',H.StdUnit, 'BackgroundColor',[0.8 0.8 0.8], ...
     'Enable',window_en, 'ListboxTop',0, 'Position',[140 229.75 70.25 15], ...
     'String','Training Data', 'Style','text', 'Tag','StaticText1');
    frame6 = uicontrol('Parent',fig, 'Units',H.StdUnit, 'BackgroundColor',[0.8 0.8 0.8], ...
     'ListboxTop',0, 'Position',[5 248 340 73], 'Style','frame', 'Tag','Frame6');
```

```
h1 = uicontrol('Parent',fig, 'Units',H.StdUnit, 'BackgroundColor',[0.8 0.8 0.8], ...
 'Enable',window_en, 'ListboxTop',0, 'Position',[130 310.75 90.25 15], ...
 'String','Network Architecture', 'Style','text', 'Tag','StaticText1');
H.Title_nnident = uicontrol('Parent',fig, 'Units',H.StdUnit, 'BackgroundColor',[0.8 0.8 0.8], ...
 'FontSize',14, 'ListboxTop',0, 'Position',[24.75 330 298.5 21.75], 'String',H.me, ...
 'Style','text', 'Tag','Title_nnident');
H.Hidden_layer_size = uicontrol('Parent',fig, 'Units',H.StdUnit, 'BackgroundColor',[1 1 1], ...
   'Enable',window_en, 'ListboxTop',0, 'Position',[125 296 45 15], 'Style','edit', ...
 'ToolTipStr','Defines the size of the second layer of the neural network plant model.',...
 'Tag','Hidden_layer');
H.Hidden_layer_text = uicontrol('Parent',fig, 'Units',H.StdUnit, ...
 'BackgroundColor',[0.8 0.8 0.8], 'Enable',window_en, 'HorizontalAlignment','right', ...
 'ListboxTop',0, 'Position',[10 296 110 15], 'String','Size of Hidden Layer', 'Style','text', ...
 'ToolTipStr','Defines the size of the second layer of the neural network plant model.',...
 'Tag','StaticText1');
H.simulink_file_text = uicontrol('Parent',fig, 'Units',H.StdUnit, 'BackgroundColor',[0.8 0.8 0.8], ...
 'Enable',window_en, 'ListboxTop',0, 'Position',[170 151 110 15], ...
 'String','Simulink Plant Model:', 'Style','text', ...
 'ToolTipStr','Simulink file containing the plant to be modeled.', 'Tag','StaticText1');
H.BrowseButton = uicontrol('Parent',fig, 'Unit',H.StdUnit, ...
 'Callback','nncontrolutil("nnident","browsesim",gcbf);', 'Enable',window_en, 'ListboxTop',0, ...
 'Position',[285 151 45 16], 'String','Browse',...
 'ToolTipStr','Allow the user to select a Simulink file.', 'Tag','BrowseButton');
H.simulink_file = uicontrol('Parent',fig, 'Units',H.StdUnit, 'BackgroundColor',[1 1 1], ...
   'Enable',window_en, 'Callback','nncontrolutil("nnident","clearpath",gcbf);', ...
   'HorizontalAlignment','left', 'ListboxTop',0, 'Position',[185 129 145 15], 'Style','edit', ...
 'ToolTipStr','Simulink file containing the plant to be modeled.', 'Tag','Plant_model');
H.Sampling_text = uicontrol('Parent',fig, 'Units',H.StdUnit, 'BackgroundColor',[0.8 0.8 0.8], ...
 'Enable',window_en, 'HorizontalAlignment','right', 'ListboxTop',0, 'Position',[10 275 110 15], ...
 'String','Sampling Interval (sec)', 'Style','text', ...
 'ToolTipStr','Sampling interval at which the data will be collected from the Simulink...
   plant model.', 'Tag','StaticText1');
H.Sampling_time = uicontrol('Parent',fig, 'Units',H.StdUnit, 'BackgroundColor',[1 1 1], ...
   'Enable',window_en, 'ListboxTop',0, 'Position',[125 275 45 15], 'Style','edit', ...
 'ToolTipStr','Sampling interval at which the data will be collected from the Simulink...
   plant model.', 'Tag','Sampling_time');
H.Delayed_input_text = uicontrol('Parent',fig, 'Units',H.StdUnit, ...
   'BackgroundColor',[0.8 0.8 0.8], 'Enable',window_en, 'HorizontalAlignment','right', ...
 'ListboxTop',0, 'Position',[170 296 110 15], 'String','No. Delayed Plant Inputs', 'Style','text', ...
 'ToolTipStr','Defines how many delays on the plant input will be used to feed the...
   neural network plant model.', 'Tag','StaticText1');
H.Delayed_input = uicontrol('Parent',fig, 'Units',H.StdUnit, 'BackgroundColor',[1 1 1], ...
   'Enable',window_en, 'ListboxTop',0, 'Max',500, 'Min',1, 'Position',[285 296 45 15], ...
 'Style','edit', 'Tag','Ni', ...
 'ToolTipStr','Defines how many delays on the plant input will be used to feed the...
   neural network plant model.', 'Value',1);
```

```
H.Delayed_output_text = uicontrol('Parent',fig, 'Units',H.StdUnit, ...
 'BackgroundColor',[0.8 0.8 0.8], 'Enable',window_en, 'HorizontalAlignment','right', ...
 'ListboxTop',0, 'Position',[170 274 110 15], 'String','No. Delayed Plant Outputs', 'Style','text', ...
 'ToolTipStr','Defines how many delays on the plant output will be used to feed the...
  neural network plant model.', 'Tag','StaticText1');
H.Delayed_output = uicontrol('Parent',fig, 'Units',H.StdUnit, 'BackgroundColor',[1 1 1], ...
  'Enable',window_en, 'ListboxTop',0, 'Max',500, 'Min',1, 'Position',[285 274 45 15], ...
 'Style','edit', 'Tag','Nj', ...
 'ToolTipStr','Defines how many delays on the plant output will be used to feed the...
  neural network plant model.', 'Value',1);
H.Limit_output_data = uicontrol('Parent',fig, 'Units',H.StdUnit, ...
 'BackgroundColor',[0.8 0.8 0.8], 'Callback','nncontrolutil("nnident","limit_output")', ...
  'Enable',window_en, 'ListboxTop',0, 'Position',[200 217 110 15], ...
 'String','Limit Output Data', 'Style','checkbox', 'Tag','checkbox3', ...
 'ToolTipStr','If selected, the plant output data will be bounded.', 'Value',1);
H.Normalize_data = uicontrol('Parent',fig, 'Units',H.StdUnit, 'BackgroundColor',[0.8 0.8 0.8], ...
  'Enable',window_en, 'ListboxTop',0, 'Position',[20 253 110 15], ...
 'String','Normalize Training Data', 'Style','checkbox', 'Tag','checkbox2', ...
 'ToolTipStr','If selected, the plant input-output data will be normalized.', 'Value',1);
H.Use_Previous_Weights_but = uicontrol('Parent',fig, 'Units',H.StdUnit, ...
 'BackgroundColor',[0.8 0.8 0.8], 'Enable',window_en, 'ListboxTop',0, 'Position',[20 48 90 15], ...
 'String','Use Current Weights', 'Style','checkbox', 'Tag','checkbox2', ...
 'ToolTipStr','If selected, the current weights are used as initial values for continued training.',...
 'Value',0);
H.Use_Validation_but = uicontrol('Parent',fig, 'Units',H.StdUnit, ...
 'BackgroundColor',[0.8 0.8 0.8], 'Enable',window_en, 'ListboxTop',0, 'Position',[130 48 90 15], ...
 'String','Use Validation Data', 'Style','checkbox', ...
 'ToolTipStr','A validation data set will be used to stop training.', 'Tag','checkbox1');
H.Use_Testing_but = uicontrol('Parent',fig, 'Units',H.StdUnit, 'BackgroundColor',[0.8 0.8 0.8], ...
  'Enable',window_en, 'ListboxTop',0, 'Position',[240 48 90 15], 'String','Use Testing Data', ...
 'Style','checkbox', 'ToolTipStr','A testing data set will be monitored during training.',...
 'Tag','checkbox1a');
H.Max_input_text = uicontrol('Parent',fig, 'Units',H.StdUnit, 'BackgroundColor',[0.8 0.8 0.8], ...
  'Enable',window_en, 'HorizontalAlignment','right', 'ListboxTop',0, 'Position',[10 195 110 15], ...
 'String','Maximum Plant Input', 'Style','text', ...
 'ToolTipStr','Defines an upper bound on the random plant input.',...
 'Tag','Maximum_input_Text');
H.Max_input = uicontrol('Parent',fig, 'Units',H.StdUnit, 'BackgroundColor',[1 1 1], ...
  'Enable',window_en, 'ListboxTop',0, 'Position',[125 195 45 15], 'Style','edit', ...
  'ToolTipStr','Defines an upper bound on the random plant input.', 'Tag','Maximum_input');
H.Min_input_text = uicontrol('Parent',fig, 'Units',H.StdUnit, 'BackgroundColor',[0.8 0.8 0.8], ...
  'Enable',window_en, 'HorizontalAlignment','right', 'ListboxTop',0, 'Position',[10 173 110 15], ...
 'String','Minimum Plant Input', 'Style','text', ...
 'ToolTipStr','Defines a lower bound on the random plant input.', 'Tag','Minimum_input_Text');
H.Min_input = uicontrol('Parent',fig, 'Units',H.StdUnit, 'BackgroundColor',[1 1 1], ...
  'Enable',window_en, 'ListboxTop',0, 'Position',[125 173 45 15], 'Style','edit', ...
```

```
  'ToolTipStr','Defines a lower bound on the random plant input.', 'Tag','Minimum_input');
H.max_int_text = uicontrol('Parent',fig, 'Units',H.StdUnit, 'BackgroundColor',[0.8 0.8 0.8], ...
  'Enable',window_en, 'HorizontalAlignment','right', 'ListboxTop',0, 'Position',[10 151 110 15], ...
 'String','Maximum Interval Value (sec)',   'Style','text', ...
 'ToolTipStr','Defines a maximum interval over which the random plant input will...
  remain constant.', 'Tag','StaticText2');
H.max_int_edit = uicontrol('Parent',fig, 'Units',H.StdUnit, 'BackgroundColor',[1 1 1], ...
  'Enable',window_en, 'ListboxTop',0, 'Position',[125 151 45 15], 'Style','edit', ...
 'ToolTipStr','Defines a maximum interval over which the random plant input...
  will remain constant.', 'Tag','max_r_edit');
H.min_int_text = uicontrol('Parent',fig, 'Units',H.StdUnit, 'BackgroundColor',[0.8 0.8 0.8], ...
  'Enable',window_en, 'HorizontalAlignment','right', 'ListboxTop',0, ...
 'Position',[10 129 110 15], 'String','Minimum Interval Value (sec)', 'Style','text', ...
 'ToolTipStr','Defines a minimum interval over which the random plant input...
  will remain constant.', 'Tag','StaticText2');
H.min_int_edit = uicontrol('Parent',fig, 'Units',H.StdUnit, 'BackgroundColor',[1 1 1], ...
  'Enable',window_en, 'ListboxTop',0, 'Position',[125 129 45 15], 'Style','edit', ...
 'ToolTipStr','Defines a minimum interval over which the random plant input...
  will remain constant.', 'Tag','EditText1');
H.Max_output = uicontrol('Parent',fig, 'Units',H.StdUnit, 'BackgroundColor',[1 1 1], ...
  'Enable',window_en, 'ListboxTop',0, 'Position',[285 195 45 15], 'Style','edit', ...
  'ToolTipStr','Defines an upper bound on the plant output.', 'Tag','Maximum_output');
H.Max_output_text = uicontrol('Parent',fig, 'Units',H.StdUnit, 'BackgroundColor',[0.8 0.8 0.8], ...
  'Enable',window_en, 'HorizontalAlignment','right', 'ListboxTop',0, ...
 'Position',[170 195 110 15], 'String','Maximum Plant Output', 'Style','text', ...
 'ToolTipStr','Defines an upper bound on the plant output.', 'Tag','Maximum_output_text');
H.Min_output_text = uicontrol('Parent',fig, 'Units',H.StdUnit, 'BackgroundColor',[0.8 0.8 0.8], ...
  'Enable',window_en, 'HorizontalAlignment','right', 'ListboxTop',0, ...
 'Position',[170 173 110 15], 'String','Minimum Plant Output', 'Style','text', ...
 'ToolTipStr','Defines a lower bound on the plant output.', 'Tag','Minimum_output_text');
H.Min_output = uicontrol('Parent',fig, 'Units',H.StdUnit, 'BackgroundColor',[1 1 1], ...
  'Enable',window_en, 'ListboxTop',0, 'Position',[285 173 45 15], 'Style','edit', ...
  'ToolTipStr','Defines a lower bound on the plant output.', 'Tag','Minimum_output');
H.Samples_text = uicontrol('Parent',fig, 'Units',H.StdUnit, 'BackgroundColor',[0.8 0.8 0.8], ...
  'Enable',window_en, 'HorizontalAlignment','right', 'ListboxTop',0, 'Position',[10 217 110 15], ...
  'String','Training Samples', 'Style','text', ...
  'ToolTipStr','Defines how many data points will be generated for training.', 'Tag','StaticText1');
H.Samples = uicontrol('Parent',fig, 'Units',H.StdUnit, 'BackgroundColor',[1 1 1], ...
  'Enable',window_en, 'ListboxTop',0, 'Position',[125 217 45 15], 'Style','edit', ...
  'ToolTipStr','Defines how many data points will be generated for training.',...
  'Tag','Training_examples');
H.trainfun_edit = uicontrol('Parent',fig, 'Units',H.StdUnit, 'BackgroundColor',[1 1 1], ...
  'Enable',window_en, 'ListboxTop',0, 'Position',[270 65 60 15], 'Max',17, ...
  'String',['trainbfg';'trainbr';'traincgb';'traincgf';'traincgp';'traingd';...
  'traingdm';'traingda';'traingdx';'trainlm ';'trainoss';'trainrp ';'trainscg'], ...
 'Style','popupmenu', 'Tag','PopupMenu1', 'Value',1);
```

```
H.trainfun_text = uicontrol('Parent',fig, 'Units',H.StdUnit, 'BackgroundColor',[0.8 0.8 0.8], ...
   'Enable',window_en, 'HorizontalAlignment','right', 'ListboxTop',0, 'Position',[180 65 85 15], ...
  'String','Training Function', 'Style','text', ...
  'ToolTipStr','Select a training function for neural network plant model training.',...
  'Tag','StaticText3');
H.epochs_text = uicontrol('Parent',fig, 'Units',H.StdUnit, 'BackgroundColor',[0.8 0.8 0.8], ...
   'Enable',window_en, 'HorizontalAlignment','right', 'ListboxTop',0, 'Position',[10 65 110 15], ...
  'String','Training Epochs', 'Style','text', ...
  'ToolTipStr','Defines number of iterations to train the neural network plant model.',...
  'Tag','StaticText1');
H.epochs_h = uicontrol('Parent',fig, 'Units',H.StdUnit, 'BackgroundColor',[1 1 1], ...
   'Enable',window_en, 'ListboxTop',0, 'Position',[125 65 45 15], 'Style','edit', ...
   'ToolTipStr','Defines number of iterations to train the neural network plant model.',...
   'Tag','Training_epochs');
H.Start_but = uicontrol('Parent',fig, 'Units',H.StdUnit, ...
  'BackgroundColor',[0.752941176470588 0.752941176470588 0.752941176470588], ...
  'Callback','nncontrolutil("nnident","continue_training")', 'Enable','off', 'ListboxTop',0, ...
  'Position',[20 30 85 16], 'String','Train Network', ...
  'ToolTipStr','Train the neural network plant using the parameters shown in this window.',...
  'Tag','Pushbutton1');
H.OK_but = uicontrol('Parent',fig, 'Units',H.StdUnit, ...
  'BackgroundColor',[0.752941176470588 0.752941176470588 0.752941176470588], ...
  'Callback','nncontrolutil("nnident","ok")', 'Enable','off', 'ListboxTop',0, ...
  'Position',[115 30 65 16], 'String','OK', ...
  'ToolTipStr','Save the neural network plant model into the controller block and...
   close this menu.', 'Tag','Pushbutton1');
H.Cancel_but = uicontrol('Parent',fig, 'Units',H.StdUnit, ...
  'BackgroundColor',[0.752941176470588 0.752941176470588 0.752941176470588], ...
  'Callback','nncontrolutil("nnident","close")', 'ListboxTop',0, 'Position',[190 30 65 16], ...
  'String','Cancel', 'ToolTipStr','Discard the neural network plant model and close this menu',...
  'Tag','Pushbutton1');
H.Apply_but = uicontrol('Parent',fig, 'Units',H.StdUnit, ...
  'BackgroundColor',[0.752941176470588 0.752941176470588 0.752941176470588], ...
  'Callback','nncontrolutil("nnident","apply")', 'Enable','off', 'ListboxTop',0, ...
  'Position',[265 30 65 16], 'String','Apply', ...
  'ToolTipStr','Save the neural network plant model into the controller block', ...
  'Tag','Pushbutton1');
H.Simulating_text = uicontrol('Parent',fig, 'Units',H.StdUnit, 'BackgroundColor',[0.8 0.8 0.8], ...
  'FontWeight','bold', 'ListboxTop',0, 'Position',[9 50 120 15], 'String','Simulating Plant', ...
  'Style','text', 'Tag','StaticText1', 'visible','off');
H.error_messages= uicontrol('Parent',fig, 'Units',H.StdUnit, ...
  'BackgroundColor',[0.752941176470588 0.752941176470588 0.752941176470588], ...
  'FontWeight','bold', 'ForegroundColor',[0 0 1], 'ListboxTop',0, 'Position',[6 3 338 20], ...
  'Style','text', 'ToolTipStr','Feedback line with important messages for the user.',...
  'Tag','StaticText1');
H.Gen_data_but = uicontrol('Parent',fig, 'Units',H.StdUnit, ...
```

```
    'BackgroundColor',[0.752941176470588 0.752941176470588 0.752941176470588], ...
    'Callback','nncontrolutil("nnident","gen_data")', 'Enable',window_en, 'ListboxTop',0, ...
    'Position',[20 106.75 100 16.5], 'String','Generate Training Data', 'Tag','Pushbutton1', ...
    'TooltipString','Generate data to be used in training the neural network plant model.');
if strcmp(arg3,'narma_l2')
    H.Get_data_file_but = uicontrol('Parent',fig, 'Units',H.StdUnit, ...
    'BackgroundColor',[0.752941176470588 0.752941176470588 0.752941176470588], ...
    'Callback','nncontrolutil("nndataimport","init",gcbf,"narma_l2");', 'Enable',window_en, ...
    'ListboxTop',0, 'Position',[130 106.75 95 16.5], 'String','Import Data', 'Tag','Pushbutton1', ...
    'TooltipString','Import training data from the workspace or from a file.');
    H.Save_to_file_but = uicontrol('Parent',fig, 'Units',H.StdUnit, ...
    'BackgroundColor',[0.752941176470588 0.752941176470588 0.752941176470588], ...
    'Callback','nncontrolutil("nndataexport","init",gcbf,"narma_l2");', 'Enable',window_en, ...
    'ListboxTop',0, 'Position',[235 106.75 95 16.5], 'String','Export Data', 'Tag','Pushbutton1', ...
    'TooltipString','Export training data to the workspace or to a file.');
end
H.Training_done=0;
H.Data_Generated=1;
H.Data_Available=0;
H.Data_Imported=0;

H.Handles.Menus.File.Top= uimenu('Parent',fig, 'Label','File');          % Create the menus for the block.
if strcmp(arg3,'narma_l2')
   H.Handles.Menus.File.ImportModel = uimenu('Parent', H.Handles.Menus.File.Top,...
      'Label','Import Network...', 'Accelerator','I',...
      'Callback','nncontrolutil("nnimport","init",gcbf,"narma_l2","nnident");',...
      'Enable',window_en, 'Tag','ImportModel');
   H.Handles.Menus.File.Export = uimenu('Parent',H.Handles.Menus.File.Top, ...
      'Label','Export Network...', 'Accelerator','E', ...
      'Callback','nncontrolutil("nnexport","init",gcbf,"narma_l2","nnident")', ...
      'Enable',window_en, 'Tag','ExportMenu');
end
H.Handles.Menus.File.Save_NN = uimenu('Parent', H.Handles.Menus.File.Top, 'Label','Save',...
    'Separator','on', 'Enable','off', 'Accelerator','S', 'Callback','nncontrolutil("nnident","apply");',...
    'Tag','ImportModel');
H.Handles.Menus.File.Save_Exit_NN = uimenu('Parent', H.Handles.Menus.File.Top,...
    'Label','Save and Exit', 'Enable','off', 'Accelerator','A',...
    'Callback','nncontrolutil("nnident","ok");', 'Tag','ImportModel');
H.Handles.Menus.File.Close = uimenu('Parent',H.Handles.Menus.File.Top, ...
    'Callback','nncontrolutil("nnident","close",gcbf);', 'Separator','on', ...
    'Label','Exit without saving', 'Accelerator','X', 'Tag','CloseMenu');
H.Handles.Menus.Window.Top = uimenu(fig, 'Label', 'Window', ...
    'Callback', winmenu('callback'), 'Tag', 'winmenu');
winmenu(fig);   % Initialize the submenu
H.Handles.Menus.Help.Top = uimenu('Parent',fig, 'Label','Help');
H.Handles.Menus.Help.Main = uimenu('Parent',H.Handles.Menus.Help.Top, ...
```

```
    'Label','Main Help', 'Callback','nncontrolutil("nnidenthelp","main");', 'Accelerator','H');
H.Handles.Menus.Help.PlantIdent = uimenu('Parent',H.Handles.Menus.Help.Top, ...
    'Label','Plant Identification...', 'Separator','on',...
    'CallBack','nncontrolutil("nnidenthelp","plant_ident");');

H.gcbh_ptr = uicontrol('Parent',fig,'visible','off');
set(H.gcbh_ptr,'userdata',arg1);
H.gcb_ptr = uicontrol('Parent',fig,'visible','off');
set(H.gcb_ptr,'userdata',arg2);

S1=get_param(arg1,'S1');                    % Get the number of the hidden layer
if isempty(S1)                              % If the field is empty we initialize default value.
    S1=num2str(0);
else
    set(H.Hidden_layer_size,'string',S1);
end
H.S1_ptr = uicontrol('Parent',fig,'visible','off');
set(H.S1_ptr,'userdata',str2num(S1));       % S1 is saved as number

sim_file=get_param(arg1,'sim_file');        % Get the name of the simulation file.
if isempty(sim_file)                        % If the field is empty we initialize default value.
    sim_file='';
end
H.sim_file_ptr = uicontrol('Parent',fig,'visible','off');
set(H.simulink_file,'string',sim_file,'UserData',struct('FileName',sim_file,'PathName',[]));
set(H.sim_file_ptr,'userdata',sim_file);

Ts=get_param(arg1,'Ts');                    % Get the sample time.
if isempty(Ts)
    Ts=num2str(0);
else
    set(H.Sampling_time,'string',Ts);
end
H.Ts_ptr = uicontrol('Parent',fig,'visible','off');
set(H.Ts_ptr,'userdata',str2num(Ts));

Ni=get_param(arg1,'Ni');                    % Get the number of delayed inputs.
if isempty(Ni)
    Ni=num2str(0);
else
    set(H.Delayed_input,'string',Ni);
end
H.Ni_ptr = uicontrol('Parent',fig,'visible','off');
set(H.Ni_ptr,'userdata',str2num(Ni));

Nj=get_param(arg1,'Nj');                    % Get the number of delayed outputs.
if isempty(Nj)
    Nj=num2str(0);
else
    set(H.Delayed_output,'string',Nj);
```

```
end
H.Nj_ptr = uicontrol('Parent',fig,'visible','off');
set(H.Nj_ptr,'userdata',str2num(Nj));
trainfun=get_param(arg1,'trainfun');
if size(trainfun,2)==6
    trainfun(7:8)=' ';
elseif size(trainfun,2)==7
    trainfun(8)=' ';
end
vv=strmatch(trainfun,func_index);
set(H.trainfun_edit,'value',vv);

Use_Validation=get_param(arg1,'Use_Validation');          % To see if validation data will be used.
if isempty(Use_Validation)
    Use_Validation=num2str(1);
end
set(H.Use_Validation_but,'value',str2num(Use_Validation));

Use_Testing=get_param(arg1,'Use_Testing');                % To see if testing data will be used.
if isempty(Use_Testing)
    Use_Testing=num2str(1);
end
set(H.Use_Testing_but,'value',str2num(Use_Testing))

max_i=get_param(arg1,'max_i');                            % Get the limit of the inputs.
if isempty(max_i)
    max_i=num2str(0);
else
    set(H.Max_input,'string',max_i);
end
H.max_i_ptr = uicontrol('Parent',fig,'visible','off');
set(H.max_i_ptr,'userdata',str2num(max_i));

min_i=get_param(arg1,'min_i');
if isempty(min_i)
    min_i=num2str(0);
else
    set(H.Min_input,'string',min_i);
end
H.min_i_ptr = uicontrol('Parent',fig,'visible','off');
set(H.min_i_ptr,'userdata',str2num(min_i));

max_i_int=get_param(arg1,'max_i_int');                    % Get the limit of the time interval.
if isempty(max_i_int)
    max_i_int=num2str(0);
else
    set(H.max_int_edit,'string',max_i_int);
end
H.max_i_int_ptr = uicontrol('Parent',fig,'visible','off');
```

```
set(H.max_i_int_ptr,'userdata',str2num(max_i_int));

min_i_int=get_param(arg1,'min_i_int');
if isempty(min_i_int)
    min_i_int=num2str(0);
else
    set(H.min_int_edit,'string',min_i_int);
end
H.min_i_int_ptr = uicontrol('Parent',fig,'visible','off');
set(H.min_i_int_ptr,'userdata',str2num(min_i_int));

Limit_output=get_param(arg1,'limit_output');                    % Get the limit of the outputs.
if isempty(Limit_output)
    Limit_output='0';
end
set(H.Limit_output_data,'value',str2num(Limit_output))

max_out=get_param(arg1,'max_output');
if isempty(max_out)
    max_out=num2str(0);
else
    set(H.Max_output,'string',max_out);
end
H.max_out_ptr = uicontrol('Parent',fig,'visible','off');
set(H.max_out_ptr,'userdata',str2num(max_out));

min_out=get_param(arg1,'min_output');
if isempty(min_out)
    min_out=num2str(0);
else
    set(H.Min_output,'string',min_out);
end
H.min_out_ptr = uicontrol('Parent',fig,'visible','off');
set(H.min_out_ptr,'userdata',str2num(min_out));

Use_Previous_Weights=get_param(arg1,'Use_Previous_Weights');   % If previous weights will be used.
if isempty(Use_Previous_Weights)
    Use_Previous_Weights=num2str(1);
end
set(H.Use_Previous_Weights_but,'value',str2num(Use_Previous_Weights));
H.Use_Previous_Weights_ptr = uicontrol('Parent',fig,'visible','off');
set(H.Use_Previous_Weights_ptr,'userdata',str2num(Use_Previous_Weights));

sam_training=get_param(arg1,'sam_training');                    % Get the number of training samples.
if isempty(sam_training)
    sam_training=num2str(0);
else
    set(H.Samples,'string',sam_training);
end
H.sam_training_ptr = uicontrol('Parent',fig,'visible','off');
```

```
set(H.sam_training_ptr,'userdata',str2num(sam_training));

epochs=get_param(arg1,'epochs');                    % Get the nmber of training epochs.
if isempty(epochs)
    epochs=num2str(0);
else
    set(H.epochs_h,'string',epochs);
end
H.epochs_ptr = uicontrol('Parent',fig,'visible','off');
set(H.epochs_ptr,'userdata',str2num(epochs));

if strcmp(arg3,'narma_l2')                          % Set the input layer weights.
  IW1_1=eval(strvcat(get_param(arg1,'IW1_1')),'0');
  H.IW1_1_ptr = uicontrol('Parent',fig,'visible','off');
  set(H.IW1_1_ptr,'userdata',IW1_1);

  IW3_1=eval(strvcat(get_param(arg1,'IW3_1')),'0');
  H.IW3_1_ptr = uicontrol('Parent',fig,'visible','off');
  set(H.IW3_1_ptr,'userdata',IW3_1);

  IW7_2=eval(strvcat(get_param(arg1,'IW7_2')),'0');
  H.IW7_2_ptr = uicontrol('Parent',fig,'visible','off');
  set(H.IW7_2_ptr,'userdata',IW7_2);

  IW8_3=eval(strvcat(get_param(arg1,'IW8_3')),'0');
  H.IW8_3_ptr = uicontrol('Parent',fig,'visible','off');
  set(H.IW8_3_ptr,'userdata',IW8_3);

  IW5_1=eval(strvcat(get_param(arg1,'IW5_1')),'0');
  H.IW5_1_ptr = uicontrol('Parent',fig,'visible','off');
  set(H.IW5_1_ptr,'userdata',IW5_1);

  IW10_2=eval(strvcat(get_param(arg1,'IW10_2')),'0');
  H.IW10_2_ptr = uicontrol('Parent',fig,'visible','off');
  set(H.IW10_2_ptr,'userdata',IW10_2);

  IW11_3=eval(strvcat(get_param(arg1,'IW11_3')),'0');
  H.IW11_3_ptr = uicontrol('Parent',fig,'visible','off');
  set(H.IW11_3_ptr,'userdata',IW11_3);


  LW2_1=eval(strvcat(get_param(arg1,'LW2_1')),'0');     % Set the hidden layer weights.
  H.LW2_1_ptr = uicontrol('Parent',fig,'visible','off');
  set(H.LW2_1_ptr,'userdata',LW2_1);

  LW4_3=eval(strvcat(get_param(arg1,'LW4_3')),'0');
  H.LW4_3_ptr = uicontrol('Parent',fig,'visible','off');
  set(H.LW4_3_ptr,'userdata',LW4_3);

  LW6_5=eval(strvcat(get_param(arg1,'LW6_5')),'0');
  H.LW6_5_ptr = uicontrol('Parent',fig,'visible','off');
  set(H.LW6_5_ptr,'userdata',LW6_5);
```

```
LW7_6=eval(strvcat(get_param(arg1,'LW7_6')),'0');
H.LW7_6_ptr = uicontrol('Parent',fig,'visible','off');
set(H.LW7_6_ptr,'userdata',LW7_6);

LW8_6=eval(strvcat(get_param(arg1,'LW8_6')),'0');
H.LW8_6_ptr = uicontrol('Parent',fig,'visible','off');
set(H.LW8_6_ptr,'userdata',LW8_6);

LW9_2=eval(strvcat(get_param(arg1,'LW9_2')),'0');
H.LW9_2_ptr = uicontrol('Parent',fig,'visible','off');
set(H.LW9_2_ptr,'userdata',LW9_2);

LW9_7=eval(strvcat(get_param(arg1,'LW9_7')),'0');
H.LW9_7_ptr = uicontrol('Parent',fig,'visible','off');
set(H.LW9_7_ptr,'userdata',LW9_7);

LW9_8=eval(strvcat(get_param(arg1,'LW9_8')),'0');
H.LW9_8_ptr = uicontrol('Parent',fig,'visible','off');
set(H.LW9_8_ptr,'userdata',LW9_8);

LW10_6=eval(strvcat(get_param(arg1,'LW10_6')),'0');
H.LW10_6_ptr = uicontrol('Parent',fig,'visible','off');
set(H.LW10_6_ptr,'userdata',LW10_6);

LW11_6=eval(strvcat(get_param(arg1,'LW11_6')),'0');
H.LW11_6_ptr = uicontrol('Parent',fig,'visible','off');
set(H.LW11_6_ptr,'userdata',LW11_6);

LW12_4=eval(strvcat(get_param(arg1,'LW12_4')),'0');
H.LW12_4_ptr = uicontrol('Parent',fig,'visible','off');
set(H.LW12_4_ptr,'userdata',LW12_4);

LW12_10=eval(strvcat(get_param(arg1,'LW12_10')),'0');
H.LW12_10_ptr = uicontrol('Parent',fig,'visible','off');
set(H.LW12_10_ptr,'userdata',LW12_10);

LW12_11=eval(strvcat(get_param(arg1,'LW12_11')),'0');
H.LW12_11_ptr = uicontrol('Parent',fig,'visible','off');
set(H.LW12_11_ptr,'userdata',LW12_11);


B1=eval(strvcat(get_param(arg1,'B1')),'0');          % Set the bias weights.
H.B1_ptr = uicontrol('Parent',fig,'visible','off');
set(H.B1_ptr,'userdata',B1);

B2=eval(strvcat(get_param(arg1,'B2')),'0');
H.B2_ptr = uicontrol('Parent',fig,'visible','off');
set(H.B2_ptr,'userdata',B2);

B3=eval(strvcat(get_param(arg1,'B3')),'0');
H.B3_ptr = uicontrol('Parent',fig,'visible','off');
set(H.B3_ptr,'userdata',B3);

B4=eval(strvcat(get_param(arg1,'B4')),'0');
```

```
    H.B4_ptr = uicontrol('Parent',fig,'visible','off');
    set(H.B4_ptr,'userdata',B4);

    B5=eval(strvcat(get_param(arg1,'B5')),'0');
    H.B5_ptr = uicontrol('Parent',fig,'visible','off');
    set(H.B5_ptr,'userdata',B5);

    B6=eval(strvcat(get_param(arg1,'B6')),'0');
    H.B6_ptr = uicontrol('Parent',fig,'visible','off');
    set(H.B6_ptr,'userdata',B6);
  end

  Normalize=str2num(get_param(arg1,'Normalize'));          % If normalization is needed.
  if isempty(Normalize)
     Normalize=0;
  end
  H.Normalize_ptr = uicontrol('Parent',fig,'visible','off');
  set(H.Normalize_ptr,'userdata',Normalize);
  set(H.Normalize_data,'value',Normalize);

  H.In_training_ptr = uicontrol('Parent',fig,'visible','off');
  set(H.In_training_ptr,'userdata',0);

  H.parent_function_ptr = uicontrol('Parent',fig,'visible','off');
  set(H.parent_function_ptr,'userdata',arg3);
  set(fig,'userdata',H);
  set(H.error_messages,'string',sprintf('Generate or import data before training the...
  neural network plant.'));
nncontrolutil('nnident','limit_output');                   % To see id the outputs shall be limited.
elseif strcmp(cmd,'limit_output') & (fig)
    limit_output=get(H.Limit_output_data,'Value');
    if limit_output==1
      set(H.Max_output,'enable','on')
      set(H.Max_output_text,'enable','on')
      set(H.Min_output,'enable','on')
      set(H.Min_output_text,'enable','on')
    else
      set(H.Max_output,'enable','off')
      set(H.Max_output_text,'enable','off')
      set(H.Min_output,'enable','off')
      set(H.Min_output_text,'enable','off')
    end

elseif strcmp(cmd,'data_no_ok') & (fig)                    % If the training is OK or not.
%   set(H.Start_but,'enable','on')
   set(H.Cancel_but,'enable','on')
   if H.Training_done
      set(H.OK_but,'enable','on')
      set(H.Apply_but,'enable','on')
      set(H.Handles.Menus.File.Save_NN,'enable','on')
```

```
    set(H.Handles.Menus.File.Save_Exit_NN,'enable','on')
  end
  if H.Data_Available
     load(cat(2,tempdir,'nnidentdata.mat'),'N2');
     set(H.Start_but,'enable','on')
     st=sprintf('Your training data set has %d samples.\nYou can now train the network.',N2-1);
     set(H.error_messages,'string',st);
  else                          % If the training data is not available, then prepare for generating the data.
    set(H.error_messages,'string',sprintf('Generate or import data before training the neural network...
plant.'));
     set(H.Max_input,'enable','on')
     set(H.Max_input_text,'enable','on')
     set(H.Min_input,'enable','on')
     set(H.Min_input_text,'enable','on')
     set(H.max_int_edit,'enable','on')
     set(H.max_int_text,'enable','on')
     set(H.min_int_edit,'enable','on')
     set(H.min_int_text,'enable','on')
     set(H.Samples_text,'enable','on')
     set(H.Samples,'enable','on')
     set(H.Sampling_text,'enable','on')
     set(H.Sampling_time,'enable','on')
     set(H.Limit_output_data,'enable','on');
     limit_output=get(H.Limit_output_data,'Value');          % Set the output limit.
     if limit_output==1
        set(H.Max_output,'enable','on')
        set(H.Max_output_text,'enable','on')
        set(H.Min_output,'enable','on')
        set(H.Min_output_text,'enable','on')
     end
     set(H.BrowseButton,'enable','on');
     set(H.simulink_file,'enable','on');
     set(H.simulink_file_text,'enable','on');
  end
  set(H.Simulating_text,'visible','off'); drawnow;          % pause needed to refresh the message
  fig2=findall(0,'type','figure','tag','nnidentdata');
  delete(fig2);

  if exist(cat(2,tempdir,'nnidentdata2.mat'))
     delete(cat(2,tempdir,'nnidentdata2.mat'));
  end

                                                            % Refresh the menu.
  arg1=get(H.gcbh_ptr,'userdata');
  arg2=get(H.gcb_ptr,'userdata');
  nncontrolutil('nnident',",arg1,arg2,");

elseif strcmp(cmd,'browsesim')                              % Locate the simulation file.
```

```
    filterspec = '*.mdl';
    udFileEdit = get(H.simulink_file,'UserData');
    LastPath = udFileEdit.PathName;
    CurrentPath=pwd;
    if ~isempty(LastPath),
        cd(LastPath);
    end
    [filename,pathname] = uigetfile(filterspec,'Simulink Plant Model:');
    if ~isempty(LastPath),
        cd(CurrentPath);
    end
    if filename,
        if ~strcmpi(pathname(1:end-1),CurrentPath)
            ImportStr = [pathname,filename(1:end-4)];
        else
            ImportStr = filename(1:end-4);
        end
        udFileEdit.PathName=pathname;
        udFileEdit.FileName=filename;
        set(H.simulink_file,'String',filename(1:end-4),'UserData',udFileEdit);
    end
elseif strcmp(cmd,'clearpath') & (fig)
    NewName = get(gcbo,'String');              %Whenever a new name is entered, update the Userdata
    indDot = findstr(NewName,'.');
    if ~isempty(indDot),
        NewName=NewName(1:indDot(end)-1);
        set(H.simulink_file,'String',NewName)
    end
elseif strcmp(cmd,'erase_data') & (fig)    % If training data is erased, prepare for generating new data.
    set(H.Max_input,'enable','on')
    set(H.Max_input_text,'enable','on')
    set(H.Min_input,'enable','on')
    set(H.Min_input_text,'enable','on')
    set(H.max_int_edit,'enable','on')
    set(H.max_int_text,'enable','on')
    set(H.min_int_edit,'enable','on')
    set(H.min_int_text,'enable','on')
    set(H.Samples_text,'enable','on')
    set(H.Samples,'enable','on')
    set(H.Sampling_text,'enable','on')
    set(H.Sampling_time,'enable','on')
    set(H.Limit_output_data,'enable','on');
    limit_output=get(H.Limit_output_data,'Value');
    if limit_output==1
        set(H.Max_output,'enable','on')
        set(H.Max_output_text,'enable','on')
```

```
        set(H.Min_output,'enable','on')
        set(H.Min_output_text,'enable','on')
    end
    set(H.BrowseButton,'enable','on');
    set(H.simulink_file,'enable','on');
    set(H.simulink_file_text,'enable','on');
    H.Data_Generated=0;
    H.Data_Imported=0;
    H.Data_Available=0;
    set(H.Start_but,'enable','off')
    if exist(cat(2,tempdir,'nnidentdata2.mat'))
        delete(cat(2,tempdir,'nnidentdata2.mat'));
    end
    if exist(cat(2,tempdir,'nnidentdata.mat'))
        delete(cat(2,tempdir,'nnidentdata.mat'));
    end
    set(fig,'UserData',H);
    set(H.Gen_data_but,'String','Generate Training Data', ...
                'Callback','nncontrolutil("nnident","gen_data")', ...
               'TooltipString','Generate data to be used in training the neural network plant model.');
    set(H.error_messages,'string',sprintf('Generate or import data before training the...
        neural network plant.'));
elseif (strcmp(cmd,'start_training') | strcmp(cmd,'continue_training') | strcmp(cmd,'data_ok') | ...
          strcmp(cmd,'gen_data') | strcmp(cmd,'have_file')) & (fig)
    if strcmp(cmd,'gen_data') & (fig)                    % Generate the training data.
      H.Data_Imported=0;
      set(fig,'UserData',H);
    elseif strcmp(cmd,'have_file') & (fig)               % Get the training data from a file.
      ImportStr=arg1; H.Data_Imported=1;
      if nargin==3
          Data_Name=arg2;
      else
          U_Name=arg2; Y_Name=arg3;
      end
    end

    set(H.Start_but,'enable','off')
    set(H.Cancel_but,'enable','off')
    set(H.OK_but,'enable','off')
    set(H.Apply_but,'enable','off')
    set(H.Handles.Menus.File.Save_NN,'enable','off')
    set(H.Handles.Menus.File.Save_Exit_NN,'enable','off')
    if (strcmp(cmd,'gen_data') | strcmp(cmd,'have_file'))%strcmp(cmd,'start_training')
      arg1=get(H.gcbh_ptr,'userdata');
      a1 = str2num(get(H.Sampling_time,'string'));
      Ts=get_param(arg1,'Ts');
      if length(a1) == 0,
```

```
    present_error(fig,H,H.Sampling_time,Ts,1, ...
        'You must initialize the sampling interval of your plant before training the...
         neural network');
    return
elseif a1<=0
    present_error(fig,H,H.Sampling_time,Ts,1, ...
        'You must set a positive sampling interval of your plant before training the...
         neural network');
    return
else Ts=a1; set(H.Ts_ptr,'userdata',Ts);   end

fig2=findall(0,'type','figure','tag','nnidentdata');
if size(fig2,1)==0, fig2=0; end

if strcmp(cmd,'have_file')                          % Locate the training data file.
  if nargin==3              % Structure
    if isempty(ImportStr)    % Workspace
      tr_dat=evalin('base',Data_Name);
      if ~isfield(tr_dat,'.flag')
          tr_dat.flag=ones(size(tr_dat.Y));
      end
      if ~isfield(tr_dat,'.Ts')
          tr_dat.Ts=Ts;
      end
    else
      a1 = ImportStr; a2 = which(cat(2,a1,'.mat'));
      if (length(a1) == 0 | length(a2) == 0),
          present_error(fig,H,0,0,0, ...
              'You must enter a valid filename for your training data,...
               or the file directory must be defined in the MATLAB Path.');
          return
      else file_data=a1; end
      temp=load (file_data,Data_Name);
      tr_dat.U=getfield(temp,Data_Name,'U');      % Get the input-ouput pairs from a file.
      tr_dat.Y=getfield(temp,Data_Name,'Y');
      if isfield(eval(cat(2,'temp.',Data_Name)),'flag')
          tr_dat.flag=getfield(temp,Data_Name,'flag');
      else
          tr_dat.flag=ones(size(tr_dat.Y));
      end
      if isfield(eval(cat(2,'temp.',Data_Name)),'Ts')
          tr_dat.Ts=getfield(temp,Data_Name,'Ts');
      else
          tr_dat.Ts=Ts;
      end
    end
  else          % Arrays.
```

```
 if isempty(ImportStr)     % Workspace
    tr_dat=struct('U',evalin('base',U_Name),'Y',evalin('base',Y_Name));
    tr_dat.flag=ones(size(tr_dat.Y));
    tr_dat.Ts=Ts;
 else
    a1 = ImportStr; a2 = which(cat(2,a1,'.mat'));
    if (length(a1) == 0 | length(a2) == 0),
        present_error(fig,H,0,0,0, ...
            'You must enter a valid filename for your training data,...
              or the file directory must be defined in the MATLAB Path.');
        return
    else file_data=a1; end
    temp=load (file_data,U_Name,Y_Name);
    tr_dat.U=getfield(temp,U_Name); tr_dat.Y=getfield(temp,Y_Name);
    tr_dat.flag=ones(size(tr_dat.Y));    tr_dat.Ts=Ts;
  end
end
                                          % Verify direction of the input vectors.
if size(tr_dat.U,1)<=2
    tr_dat.U=tr_dat.U';
end
if size(tr_dat.Y,1)<=2
    tr_dat.Y=tr_dat.Y';
end
if size(tr_dat.flag,1)<=2
    tr_dat.flag=tr_dat.flag';
end
sam_training=size(tr_dat.Y,1)-1;
if fig2==0                                % Set a new window for generating training data.
   pos_fig2=get(fig,'Position');
   fig2 = figure('Units', H.StdUnit,...
              'CloseRequestFcn','nncontrolutil("nnident","data_NO_ok");', ...
              'Interruptible','off', 'BusyAction','cancel', 'HandleVis','Callback', ...
              'Name', 'Plant Input-Output Data', 'Tag', 'nnidentdata', 'NumberTitle', 'off',...
              'Position', [45 30 500 358], 'IntegerHandle', 'off', 'Toolbar', 'none', ...
              'WindowStyle','modal');
   f2.h1=axes('Position',[0.10 0.60 0.37 0.32],'Parent',fig2);
   f2.h2=axes('Position',[0.10 0.15 0.37 0.32],'Parent',fig2);
   f2.h3=axes('Position',[0.57 0.60 0.37 0.32],'Parent',fig2);
   f2.h4=axes('Position',[0.57 0.15 0.37 0.32],'Parent',fig2);
   f2.message= uicontrol('Parent',fig2, 'Units',H.StdUnit, ...
               'BackgroundColor',[0.752941176470588   0.752941176470588...
               0.752941176470588], 'FontWeight','bold', 'ForegroundColor',[0 0 1], ...
               'ListboxTop',0, 'Position',[156 3 188 20], 'Style','text', 'Tag','StaticText1');
 else
```

```
    f2=get(fig2,'userdata'); figure(fig2);
  end

  f2.accept_but = uicontrol('Parent',fig2, 'Units',H.StdUnit, ...
   'BackgroundColor',[0.752941176470588 0.752941176470588 0.752941176470588], ...
   'Callback','nncontrolutil("nnident","data_ok");', 'ListboxTop',0, 'Position',[2 2 68.75 15], ...
   'String','Accept Data', 'Tag','Pushbutton1');
  st=sprintf('The imported data has %d samples.\nPlease Accept or Reject...
        Data to continue.',sam_training);
  set(H.error_messages,'string',st); set(f2.message,'string',st);
else    %strcmp(cmd,'gen_data')
  a1 = get(H.simulink_file,'string');
  udFileEdit = get(H.simulink_file,'UserData');
  LastPath = udFileEdit.PathName;
  if isempty(LastPath),
     a2 = which(cat(2,a1,'.mdl'));
  else
     a2 = which(cat(2,LastPath,cat(2,a1,'.mdl')));
  end
  if (length(a1) == 0 | length(a2) == 0),
    present_error(fig,H,H.simulink_file,a1,0, ...
        'You must enter a valid filename for your Simulink plant model');
    return
  else
    sim_file=a1; OpenFlag=1;
    ErrorFlag=isempty(find_system(0,'flat','Name',sim_file));
    if ErrorFlag,
       ErrorFlag=~(exist(sim_file)==4);
       if ~ErrorFlag,
          OpenFlag=0;
          load_system(sim_file);
       end
    end
    if ErrorFlag,
       ErrMsg=[sim_file ' must be the name of a Simulink model.'];
       present_error(fig,H,H.simulink_file,a1,0,ErrMsg);
       return
    end

    blk=get_param(sim_file,'blocks'); iblk=0;oblk=0;
    for k=1:size(blk,1)
       if strcmp(get_param(cat(2,cat(2,sim_file,'/'),blk{k}),'blocktype'),'Inport')
          iblk=iblk+1;
       end
       if strcmp(get_param(cat(2,cat(2,sim_file,'/'),blk{k}),'blocktype'),'Outport')
          oblk=oblk+1;
       end
```

```
    end
    if ~OpenFlag,close_system(sim_file,0);end

    if iblk~=2 | oblk~=2
        present_error(fig,H,H.simulink_file,a1,0, ...
            'The Simulink plant model must have two Inports and two Outports');
        return
    end
    sim_path=a2(1:findstr(a2,a1)-1); set(H.sim_file_ptr,'userdata',sim_file);
end

 a1 = str2num(get(H.Max_input,'string'));
 max_i=get_param(arg1,'max_i');
 if length(a1) == 0,
     present_error(fig,H,H.Max_input,max_i,1, ...
         'You must enter a valid number for the maximum plant input.');
     return
 else max_i=a1; set(H.max_i_ptr,'userdata',max_i); end

 a1 = str2num(get(H.Min_input,'string'));
 min_i=get_param(arg1,'min_i');
 if length(a1) == 0,
     present_error(fig,H,H.Min_input,min_i,1, ...
         'You must enter a valid number for the minimum plant input.');
     return
 elseif a1>=max_i
     present_error(fig,H,H.Min_input,min_i,1, ...
         'You must enter valid numbers for the maximum and minimum plant inputs.');
     return
 else min_i=a1; set(H.min_i_ptr,'userdata',min_i); end

 a1 = str2num(get(H.max_int_edit,'string'));
 max_i_int=get_param(arg1,'max_i_int');
 if (length(a1) == 0) | a1<=0,
     present_error(fig,H,H.max_int_edit,max_i_int,1, ...
         'You must enter a valid number for the maximum interval value over...
          which the random input is constant.');
     return
 else max_i_int=a1; set(H.max_i_int_ptr,'userdata',max_i_int); end
 a1 = str2num(get(H.min_int_edit,'string'));
 min_i_int=get_param(arg1,'min_i_int');
 if (length(a1) == 0) | a1<=0,
     present_error(fig,H,H.min_int_edit,min_i_int,1, ...
         'You must enter a valid number for the minimum interval value over...
          which the random input is constant.');
     return
 elseif a1>=max_i_int
     present_error(fig,H,H.min_int_edit,min_i_int,1, ...
```

```
        'You must enter valid maximum and minimum interval values for constant input...
          to the plant.');
    return
else min_i_int=a1; set(H.min_i_int_ptr,'userdata',min_i_int); end

a1 = str2num(get(H.Samples,'string'));
sam_training=get_param(arg1,'sam_training');
if (length(a1) == 0) | (a1 < 1) | (floor(a1)~=a1),
    present_error(fig,H,H.Samples,sam_training,1, ...
        'You must enter a valid number of samples for training');
    return
else sam_training=a1; set(H.sam_training_ptr,'userdata',sam_training); end

Limit_output=get(H.Limit_output_data,'value');
if Limit_output
    a1 = str2num(get(H.Max_output,'string'));
    max_out=get_param(arg1,'max_output');
    if length(a1) == 0,
        present_error(fig,H,H.Max_output,max_out,1, ...
            'You must enter a valid maximum plant output');
        return
    else max_out=a1; set(H.max_out_ptr,'userdata',max_out); end

    a1 = str2num(get(H.Min_output,'string'));
    min_out=get_param(arg1,'min_output');
    if length(a1) == 0,
        present_error(fig,H,H.Min_output,min_out,1, ...
            'You must enter a valid minimum plant output');
        return
    elseif a1>=max_out
        present_error(fig,H,H.Min_output,min_out,1, ...
            'You must enter valid maximum and minimum plant outputs');
        return
    else min_out=a1; set(H.min_out_ptr,'userdata',min_out); end
else
    max_out=Inf; set(H.max_out_ptr,'userdata',max_out);
    min_out=-Inf; set(H.min_out_ptr,'userdata',min_out);
end

if fig2==0
   pos_fig2=get(fig,'Position');
   fig2 = figure('Units', H.StdUnit, 'Interruptible','off', 'BusyAction','cancel', ...
            'HandleVis','Callback', ...
            'CloseRequestFcn','nncontrolutil("nnident","data_NO_ok");', ...
            'Name', 'Plant Input-Output Data', 'Tag', 'nnidentdata', 'NumberTitle', 'off',...
            'Position', [45 30 500 358], 'IntegerHandle', 'off', 'Toolbar', 'none', ...
            'WindowStyle','modal');
   f2.h1=axes('Position',[0.10 0.60 0.37 0.32],'Parent',fig2);
```

```
  f2.h2=axes('Position',[0.10 0.15 0.37 0.32],'Parent',fig2);
  f2.h3=axes('Position',[0.57 0.60 0.37 0.32],'Parent',fig2);
  f2.h4=axes('Position',[0.57 0.15 0.37 0.32],'Parent',fig2);
  f2.message= uicontrol('Parent',fig2, 'Units',H.StdUnit, ...
                'BackgroundColor',[0.752941176470588   0.752941176470588 ...
                 0.752941176470588], 'FontWeight','bold', 'ForegroundColor',[0 0 1], ...
                 'ListboxTop',0, 'Position',[156 3 188 20], 'Style','text', 'Tag','StaticText1');
else
   f2=get(fig2,'userdata');
   figure(fig2);
end

f2.accept_but = uicontrol('Parent',fig2, 'Units',H.StdUnit, ...
 'BackgroundColor',[0.752941176470588 0.752941176470588 0.752941176470588], ...
 'Callback','nncontrolutil("nnident","stop_sim");','ListboxTop',0, 'Position',[2 2 68.75 15], ...
 'String','Stop Simulation', 'Tag','Pushbutton1');
f2.stop=0;
set(fig2,'UserData',f2);
set(H.error_messages,'string','Simulating plant. Wait until sample data points...
    are generated');
drawnow;                                          % pause needed to refresh the message
options_ini=simset('OutputPoints','all'); options=simset('OutputPoints','all');
step_size1=5; step_size2=5;                  % Set initial step size for generating training data.

k=1; k11=1; k12=1;
set(fig,'pointer','watch');
Actual_path=pwd;
if isempty(sim_path)
    sim_path=Actual_path;   end
cd(sim_path);
tr_dat.Ts=Ts;
min_k=ceil(min_i_int/Ts);
while k<=sam_training %for k=1:sam_training                  % Get the random input set.
   if ceil((k11-1)/step_size1)==(k11-1)/step_size1
       newsample1=rand*(max_i-min_i)+min_i;
       k11=1;
       step_size1=ceil(max([min([(rand*(max_i_int-min_i_int)+min_i_int) max_i_int]) ...
                   min_i_int])/Ts);
   end
   if ceil((k12-1)/step_size2)==(k12-1)/step_size2
       newsample2=rand*(max_i-min_i)+min_i;
       k12=1;
       step_size2=ceil(max([min([(rand*(max_i_int-min_i_int)+min_i_int)max_i_int]) ...
       min_i_int])/Ts);
   end
   k11=k11+1; k12=k12+1;
   tr_dat.U(k,1)=newsample1; tr_dat.U(k,2)=newsample2;
```

```
[time,xx0,yy] = sim(sim_file,[(k-1)*Ts k*Ts],options,[[(k-1)*Ts k*Ts]' [tr_dat.U(k,:);...
                tr_dat.U(k,:)]]);                                    % Get the output set.
options.InitialState=xx0(size(xx0,1),:);
if k==1
   tr_dat.Y(1,:)=yy(1,:); end
tr_dat.Y(k+1,:)=yy(size(yy,1),:);
tr_dat.flag(k,1)=1;

if k>(min_k*2+1)                                                     % verify the training data.
   if (abs(mean(tr_dat.Y(k-min_k+1:k+1,1))-mean(tr_dat.Y(k-min_k*2:k-min_k,1)))) < ...
    1e-10 & tr_dat.U(k,1) == tr_dat.U(k-min_k*2,1)
      newsample1=rand*(max_i-min_i)+min_i;
      k11=2;
      step_size1=ceil(max([min([(rand*(max_i_int-min_i_int)+min_i_int) max_i_int]) ...
                min_i_int])/Ts);
   end
   if (abs(mean(tr_dat.Y(k-min_k+1:k+1,2))-mean(tr_dat.Y(k-min_k*2:k-min_k,2)))) < ...
      1e-10 & tr_dat.U(k,2) == tr_dat.U(k-min_k*2,2)
      newsample2=rand*(max_i-min_i)+min_i;
      k12=2;
      step_size2=ceil(max([min([(rand*(max_i_int-min_i_int)+min_i_int) max_i_int]) ...
                min_i_int])/Ts);
   end
end

if ceil(k/100)==k/100                                                % Stop generating training data.
  f2=get(fig2,'userdata');
  if f2.stop~=0
   st=sprintf('Simulation stopped by the user.\nPlease Accept or Reject Data to continue.');
   set(H.error_messages,'string',st);
   H.Data_Available=0;
    set(fig,'UserData',H);
    sam_training=k;
    k=k+1;
    break
  end

  st=sprintf('Processing sample # %d of %d total samples.',k,sam_training);
  set(H.error_messages,'string',st); set(f2.message,'string',st);

  plot((0:k-1)*Ts,tr_dat.U(1:k,1),'Parent',f2.h1);                   % plot the training data.
  plot((0:k-1)*Ts,tr_dat.Y(2:k+1,1),'Parent',f2.h2);
  plot((0:k-1)*Ts,tr_dat.U(1:k,2),'Parent',f2.h3);
  plot((0:k-1)*Ts,tr_dat.Y(2:k+1,2),'Parent',f2.h4);
  set(get(f2.h1,'Title'),'string','Plant Input 1','fontweight','bold');
  set(get(f2.h2,'Title'),'string','Plant Output 1','fontweight','bold');
  set(get(f2.h3,'Title'),'string','Plant Input 2','fontweight','bold');
  set(get(f2.h4,'Title'),'string','Plant Output 2','fontweight','bold');
```

```
        set(get(f2.h1,'XLabel'),'string','time (s)'); set(get(f2.h2,'XLabel'),'string','time (s)');
        set(get(f2.h3,'XLabel'),'string','time (s)'); set(get(f2.h4,'XLabel'),'string','time (s)');
        set(fig2,'UserData',f2);
        drawnow;
      end
      k=k+1;
    end
    if ~f2.stop
        st=sprintf('Simulation concluded.\nPlease Accept or Reject Data to continue.');
        set(H.error_messages,'string',st); set(f2.message,'string',st);
    end
    set(fig,'pointer','arrow');
    cd(Actual_path);
    tr_dat.U(k,1)=newsample1; tr_dat.U(k,2)=newsample2;          % U and Y have the same size.
    set(f2.message,'string',st);
    set(f2.accept_but,'Callback','nncontrolutil("nnident","data_ok");', ...
        'String','Accept Data');
end

set(H.Max_input,'enable','off')
set(H.Max_input_text,'enable','off')
set(H.Min_input,'enable','off')
set(H.Min_input_text,'enable','off')
set(H.max_int_edit,'enable','off')
set(H.max_int_text,'enable','off')
set(H.min_int_edit,'enable','off')
set(H.min_int_text,'enable','off')
set(H.Samples_text,'enable','off')
set(H.Samples,'enable','off')
set(H.Sampling_text,'enable','off')
set(H.Sampling_time,'enable','off')
set(H.Max_output,'enable','off')
set(H.Max_output_text,'enable','off')
set(H.Min_output,'enable','off')
set(H.Min_output_text,'enable','off')
set(H.Limit_output_data,'enable','off');
set(H.BrowseButton,'enable','off');
set(H.simulink_file,'enable','off');
set(H.simulink_file_text,'enable','off');

f2.Reject_but = uicontrol('Parent',fig2, 'Units',H.StdUnit, ...          % Reject the training data.
    'BackgroundColor',[0.752941176470588 0.752941176470588 0.752941176470588], ...
  'Callback','nncontrolutil("nnident","data_NO_ok");', 'ListboxTop',0, ...
  'Position',[75 2 68.75 15], 'String','Reject Data', 'Tag','Pushbutton1');

plot((0:sam_training-1)*Ts,tr_dat.U(1:sam_training,1),'Parent',f2.h1);
plot((0:sam_training-1)*Ts,tr_dat.Y(2:sam_training+1,1),'Parent',f2.h2);
plot((0:sam_training-1)*Ts,tr_dat.U(1:sam_training,2),'Parent',f2.h3);
```

```
  plot((0:sam_training-1)*Ts,tr_dat.Y(2:sam_training+1,2),'Parent',f2.h4);
  set(f2.h1,'xlim',[0 (sam_training-1)*Ts]); set(f2.h2,'xlim',[0 (sam_training-1)*Ts]);
  set(f2.h3,'xlim',[0 (sam_training-1)*Ts]); set(f2.h4,'xlim',[0 (sam_training-1)*Ts]);

  set(get(f2.h1,'Title'),'string','Plant Input 1','fontweight','bold');
  set(get(f2.h2,'Title'),'string','Plant Output 1','fontweight','bold');
  set(get(f2.h3,'Title'),'string','Plant Input 2','fontweight','bold');
  set(get(f2.h4,'Title'),'string','Plant Output 2','fontweight','bold');
  set(get(f2.h1,'XLabel'),'string','time (s)'); set(get(f2.h2,'XLabel'),'string','time (s)');
  set(get(f2.h3,'XLabel'),'string','time (s)'); set(get(f2.h4,'XLabel'),'string','time (s)');
  set(fig,'userdata',H)
  set(fig2,'UserData',f2);
  save(cat(2,tempdir,'nnidentdata2.mat'));
  return;
elseif strcmp(cmd,'data_ok')                                    % Accept the training data.
  load(cat(2,tempdir,'nnidentdata2.mat'));
  delete(cat(2,tempdir,'nnidentdata2.mat'));
  delete(fig2);

  N2=length(tr_dat.U);                                          % Get the length of the training data.
  st=sprintf('Your training data set has %d samples.\nYou can now train the network.',N2-1);
  set(H.error_messages,'string',st);

  if H.Training_done==1
     set(H.Apply_but,'enable','on'); set(H.OK_but,'enable','on');
     set(H.Handles.Menus.File.Save_NN,'enable','on')
     set(H.Handles.Menus.File.Save_Exit_NN,'enable','on')
  end
  set(H.Start_but,'enable','on'); set(H.Cancel_but,'enable','on');

  H.Data_Available=1;
  if H.Data_Imported
     H.Data_Generated=0;
     set(H.Gen_data_but,'String','Erase Imported Data', ...
       'Callback','nncontrolutil("nnident","erase_data")', ...
       'TooltipString','The imported data will be erased and the Training Data...
        section be enabled.');
  else
     H.Data_Generated=1;
     set(H.Gen_data_but,'String','Erase Generated Data', ...
       'Callback','nncontrolutil("nnident","erase_data")', ...
       'TooltipString','The generated data will be erased and the Training Data...
        section will be enabled.');
  end
  set(fig,'userdata',H)
  save(cat(2,tempdir,'nnidentdata.mat'));
  return
elseif strcmp(cmd,'continue_training')
```

```
  load(cat(2,tempdir,'nnidentdata.mat'));
  HH=msgbox({['The Neural Network is being configured.']...
             ['Training will start shortly.'] },H.me,'warn');
  delete(findobj(HH,'style','pushbutton'));
pause(1);

  a1 = str2num(get(H.Hidden_layer_size,'string'));
  S1=get_param(arg1,'S1');
  if length(a1) == 0,
     present_error(fig,H,H.Hidden_layer_size,S1,1, ...
        'You must initialize the size of the hidden layer before training the neural network.');
     delete(HH);
     return
  elseif a1<=0 | fix(a1)~=a1
     present_error(fig,H,H.Hidden_layer_size,S1,1, ...
        'You must set the size of the hidden layer to a positive integer before generating data...
         or training the neural network');
     delete(HH);
     return
  else S1=a1; set(H.S1_ptr,'userdata',S1);   end

  a1 = str2num(get(H.Delayed_input,'string'));
  if (length(a1) == 0) | (a1 < 1) | (floor(a1)~=a1),
     Ni=get_param(arg1,'Ni');
     present_error(fig,H,H.Delayed_input,Ni,1, ...
        'You must enter a valid number of delayed plant inputs');
     delete(HH);
     return
  else Ni=a1; set(H.Ni_ptr,'userdata',Ni);   end

  a1 = str2num(get(H.Delayed_output,'string'));
  if (length(a1) == 0) | (a1 < 0) | (floor(a1)~=a1),
     Nj=get_param(arg1,'Nj');
     present_error(fig,H,H.Delayed_output,Nj,1, ...
        'You must enter a valid number of delayed plant outputs');
     delete(HH);
     return
  else Nj=a1; set(H.Nj_ptr,'userdata',Nj);   end

  a1 = get(H.trainfun_edit,'value');
  if (a1 < 1) | (a1 > 13),
     trainfun=get_param(arg1,'trainfun');
     a1=strmatch(trainfun,func_index);
     present_error(fig,H,H.trainfun_edit,a1,0, ...
          'Please, correct the training function');
     delete(HH);
     set(H.trainfun_edit,'value',a1);
     return
```

```
else
    trainfun=func_index(a1,:);
    set_param(arg1,'trainfun',trainfun);
    for k1=8:-1:1
        if trainfun(k1)==' '
            trainfun=trainfun(1:k1-1);
        else
            break
        end
    end
end

a1 = str2num(get(H.epochs_h,'string'));
epochs=get_param(arg1,'epochs');
if (length(a1) == 0) | (a1 < 1) | (floor(a1)~=a1),
    present_error(fig,H,H.epochs_h,epochs,1, ...
        'You must enter a valid number of epochs before training the neural network');
    delete(HH);
    return
else epochs=a1; set(H.epochs_ptr,'userdata',epochs); end

Use_Previous_Weights=get(H.Use_Previous_Weights_but,'value');
set(H.Use_Previous_Weights_ptr,'userdata',Use_Previous_Weights);

Normalize=get(H.Normalize_data,'Value');
set(H.Normalize_ptr,'userdata',Normalize);
set(H.In_training_ptr,'userdata',1);

if Normalize ~= 1
  Y=tr_dat.Y;
  U=tr_dat.U;
end
Use_Validation=get(H.Use_Validation_but,'value');
Use_Testing=get(H.Use_Testing_but,'value');

if Use_Validation & Use_Testing
  N1=floor(N2/2); N3=floor(N2*3/4);
elseif Use_Testing
  N1=floor(N2*3/4); N3=floor(N2*3/4);
elseif Use_Validation
  N1=floor(N2*3/4); N3=N2;
else
  N1=N2; N3=N2;
end
max_Ni_Nj=max([Ni Nj]);
no_valid_data0=find(tr_dat.flag==0);
no_valid_data=no_valid_data0;
for k=1:max_Ni_Nj-1
    no_valid_data=[no_valid_data;no_valid_data0+k];
```

```
end
size_no_valid_data=size(no_valid_data,1);
train_points=max_Ni_Nj:N1-1; valid_points=N1:N3-1; test_points=N3:N2-1;
for k=1:size_no_valid_data                    % Set the training points, validation points and test points.
    train_points=train_points(find(train_points(:)~=no_valid_data(k)));
    valid_points=valid_points(find(valid_points(:)~=no_valid_data(k)));
    test_points=test_points(find(test_points(:)~=no_valid_data(k)));
end

S2=1; S4=1; S6=9; S3=S1; S5=S1;          % Set the number of neurons for all layers.
f1 = 'tansig'; f2 = 'purelin';                    % Set the transfer functions.
parent_function=get(H.parent_function_ptr,'userdata');
if strcmp(parent_function,'narma_l2')     % Generate training data, validation data and test data.
   ptr=cell(3,1); ttr=cell(2,1);
   vv.P=cell(3,1); vv.T=cell(2,1);
   tt.P=cell(3,1); tt.T=cell(2,1);
   ptr{1,1}=[Y(train_points,:)'];
   vv.P{1,1}=[Y(valid_points,:)'];
   tt.P{1,1}=[Y(test_points,:)'];
   for k=1:Nj-1
      ptr{1,1}=[ptr{1,1};[Y(train_points-k,:)']];
      vv.P{1,1}=[vv.P{1,1};Y(valid_points-k,:)'];
      tt.P{1,1}=[tt.P{1,1};Y(test_points-k,:)'];
   end
   ptr{2,1}=U(train_points,1)';
   vv.P{2,1}=U(valid_points,1)';
   tt.P{2,1}=U(test_points,1)';
   ptr{3,1}=U(train_points,2)';
   vv.P{3,1}=U(valid_points,2)';
   tt.P{3,1}=U(test_points,2)';
   for k=1:Ni-1
      ptr{1,1}=[ptr{1,1};[U(train_points-k,:)']];
      vv.P{1,1}=[vv.P{1,1};U(valid_points-k,:)'];
      tt.P{1,1}=[tt.P{1,1};U(test_points-k,:)'];
   end
   ttr{1,1}=Y(train_points+1,1)';
   vv.T{1,1}=Y(valid_points+1,1)';
   tt.T{1,1}=Y(test_points+1,1)';
   ttr{2,1}=Y(train_points+1,2)';
   vv.T{2,1}=Y(valid_points+1,2)';
   tt.T{2,1}=Y(test_points+1,2)';

   netn = newff(minmax([ptr{1,1} vv.P{1,1} tt.P{1,1}]),[S1 S2 S3 S4 S5 S6 1 1 1 1 1 1],...
          {f1,f2,f1,f2,f1,f2,f2,f2,f2,f2,f2,f2});          % Create the neural network.
   netn.numInputs=3;
   netn.numLayers=12;
   netn.inputs{2}.range=minmax(ptr{2,1});
```

```
netn.inputs{3}.range=minmax(ptr{3,1});
netn.biasConnect=[ones(1,6) zeros(1,6)]';
netn.layers{7}.netInputFcn='netprod';
netn.layers{8}.netInputFcn='netprod';
netn.layers{10}.netInputFcn='netprod';
netn.layers{11}.netInputFcn='netprod';
netn.inputConnect=[1 0 0;0 0 0;1 0 0;0 0 0;1 0 0;0 0 0;0 1 0;0 0 1;0 0 0;0 1 0;0 0 1;0 0 0];
netn.layerConnect=[zeros(1,12);1 zeros(1,11);zeros(1,12);0 0 1 zeros(1,9);zeros(1,12);...
    0 0 0 0 1 zeros(1,7);zeros(1,5) 1 zeros(1,6);zeros(1,5) 1 zeros(1,6);...
    0 1 zeros(1,4) 1 1 zeros(1,4);zeros(1,5) 1 zeros(1,6);zeros(1,5) 1 zeros(1,6);...
    0 0 0 1 zeros(1,5) 1 1 0];
netn.outputConnect=[0 0 0 0 0 0 0 0 1 0 0 1];
netn.targetConnect=[0 0 0 0 0 0 0 0 1 0 0 1];
netn=initlay(netn);

IW1_1=netn.IW{1,1}; IW3_1=netn.IW{3,1}; IW5_1=netn.IW{5,1};      % Initialize the weights.
IW7_2=netn.IW{7,2}; IW8_3=netn.IW{8,3};
IW10_2=netn.IW{10,2}; IW11_3=netn.IW{11,3};
LW2_1=netn.LW{2,1}; LW4_3=netn.LW{4,3}; LW6_5=netn.LW{6,5};
LW7_6=netn.LW{7,6}; LW8_6=netn.LW{8,6};
LW10_6=netn.LW{10,6}; LW11_6=netn.LW{11,6};
LW9_2=netn.LW{9,2}; LW9_7=netn.LW{9,7}; LW9_8=netn.LW{9,8};
LW12_4=netn.LW{12,4}; LW12_10=netn.LW{12,10}; LW12_11=netn.LW{12,11};
B1=netn.b{1}; B2=netn.b{2}; B3=netn.b{3}; B4=netn.b{4}; B5=netn.b{5}; B6=netn.b{6};

if Use_Previous_Weights & ~isempty(strvcat(get_param(arg1,'IW1_1')))
   IW1_1b=get(H.IW1_1_ptr,'userdata'); IW3_1b=get(H.IW3_1_ptr,'userdata');
   IW5_1b=get(H.IW5_1_ptr,'userdata');
   IW7_2b=get(H.IW7_2_ptr,'userdata'); IW8_3b=get(H.IW8_3_ptr,'userdata');
   IW10_2b=get(H.IW10_2_ptr,'userdata'); IW11_3b=get(H.IW11_3_ptr,'userdata');
   LW2_1b=get(H.LW2_1_ptr,'userdata');LW4_3b=get(H.LW4_3_ptr,'userdata');
   LW6_5b=get(H.LW6_5_ptr,'userdata');
   LW7_6b=get(H.LW7_6_ptr,'userdata'); LW8_6b=get(H.LW8_6_ptr,'userdata');
   LW10_6b=get(H.LW10_6_ptr,'userdata'); LW11_6b=get(H.LW11_6_ptr,'userdata');
   LW9_2b=get(H.LW9_2_ptr,'userdata'); LW9_7b=get(H.LW9_7_ptr,'userdata');
   LW9_8b=get(H.LW9_8_ptr,'userdata'); LW12_4b=get(H.LW12_4_ptr,'userdata');
   LW12_10b=get(H.LW12_10_ptr,'userdata'); LW12_11b=get(H.LW12_11_ptr,'userdata');
   B1b=get(H.B1_ptr,'userdata'); B2b=get(H.B2_ptr,'userdata');
   B3b=get(H.B3_ptr,'userdata'); B4b=get(H.B4_ptr,'userdata');
   B5b=get(H.B5_ptr,'userdata'); B6b=get(H.B6_ptr,'userdata');

   IW1_1a=eval(strvcat(get_param(arg1,'IW1_1')));
   IW3_1a=eval(strvcat(get_param(arg1,'IW3_1')));
   IW5_1a=eval(strvcat(get_param(arg1,'IW5_1')));
   IW7_2a=eval(strvcat(get_param(arg1,'IW7_2')));
   IW8_3a=eval(strvcat(get_param(arg1,'IW8_3')));
   IW10_2a=eval(strvcat(get_param(arg1,'IW10_2')));
   IW11_3a=eval(strvcat(get_param(arg1,'IW11_3')));
```

```
LW2_1a=eval(strvcat(get_param(arg1,'LW2_1')));
LW4_3a=eval(strvcat(get_param(arg1,'LW4_3')));
LW6_5a=eval(strvcat(get_param(arg1,'LW6_5')));
LW7_6a=eval(strvcat(get_param(arg1,'LW7_6')));
LW8_6a=eval(strvcat(get_param(arg1,'LW8_6')));
LW10_6a=eval(strvcat(get_param(arg1,'LW10_6')));
LW11_6a=eval(strvcat(get_param(arg1,'LW11_6')));
LW9_2a=eval(strvcat(get_param(arg1,'LW9_2')));
LW9_7a=eval(strvcat(get_param(arg1,'LW9_7')));
LW9_8a=eval(strvcat(get_param(arg1,'LW9_8')));
LW12_4a=eval(strvcat(get_param(arg1,'LW12_4')));
LW12_10a=eval(strvcat(get_param(arg1,'LW12_10')));
LW12_11a=eval(strvcat(get_param(arg1,'LW12_11')));
B1a=eval(strvcat(get_param(arg1,'B1'))); B2a=eval(strvcat(get_param(arg1,'B2')));
B3a=eval(strvcat(get_param(arg1,'B3'))); B4a=eval(strvcat(get_param(arg1,'B4')));
B5a=eval(strvcat(get_param(arg1,'B5'))); B6a=eval(strvcat(get_param(arg1,'B6')));
if (size(IW1_1)==size(IW1_1a)) & (size(IW3_1)==size(IW3_1a)) & ...
        (size(IW5_1)==size(IW5_1a)) ...
        & (size(IW7_2)==size(IW7_2a)) & (size(IW8_3)==size(IW8_3a)) ...
        & (size(IW10_2)==size(IW10_2a)) & (size(IW11_3)==size(IW11_3a)) ...
        & (size(LW2_1)==size(LW2_1a)) & (size(LW4_3)==size(LW4_3a)) &...
            (size(LW6_5)==size(LW6_5a)) ...
        & (size(LW7_6)==size(LW7_6a))   & (size(LW8_6)==size(LW8_6a)) ...
        & (size(LW10_6)==size(LW10_6a)) & (size(LW11_6)==size(LW11_6a)) & ...
            (size(LW9_2)==size(LW9_2a)) ...
        & (size(LW9_7)==size(LW9_7a)) & (size(LW9_8)==size(LW9_8a)) ...
        & (size(LW12_4)==size(LW12_4a)) & (size(LW12_10)==size(LW12_10a)) &...
            (size(LW12_11)==size(LW12_11a)) ...
        & (size(B1)==size(B1a)) & (size(B2)==size(B2a)) & (size(B3)==size(B3a)) & ...
            (size(B4)==size(B4a)) & (size(B5)==size(B5a)) & (size(B6)==size(B6a))
    % If Weights different from last generated, we use Simulink weights.
    if (size(IW1_1)==size(IW1_1b)) & (size(IW3_1)==size(IW3_1b)) &...
              (size(IW5_1)==size(IW5_1b)) ...
            & (size(IW7_2)==size(IW7_2b)) & (size(IW8_3)==size(IW8_3b)) ...
            & (size(IW10_2)==size(IW10_2b)) & (size(IW11_3)==size(IW11_3b)) ...
            & (size(LW2_1)==size(LW2_1b)) & (size(LW4_3)==size(LW4_3b)) & ...
              (size(LW6_5)==size(LW6_5b)) ...
            & (size(LW7_6)==size(LW7_6b))   & (size(LW8_6)==size(LW8_6b)) ...
            & (size(LW10_6)==size(LW10_6b)) & (size(LW11_6)==size(LW11_6b)) &...
              (size(LW9_2)==size(LW9_2b)) ...
            & (size(LW9_7)==size(LW9_7b)) & (size(LW9_8)==size(LW9_8b)) ...
            & (size(LW12_4)==size(LW12_4b)) & (size(LW12_10)==size(LW12_10b)) &...
                (size(LW12_11)==size(LW12_11b)) ...
            & (size(B1)==size(B1b)) & (size(B2)==size(B2b)) & (size(B3)==size(B3b)) & ...
                (size(B4)==size(B4b)) ...
            & (size(B5)==size(B5b)) & (size(B6)==size(B6b))
```

```
    cx=IW1_1b==IW1_1a;
    if sum(cx(:))~=size(IW1_1(:),1)
        if ishandle(HH)
            delete(HH);
        end
        switch questdlg(...
            {'You have a set of weights in the Simulink model and...
             another set of weights generated in the current training process.'
             ' ';
             'Select which set of weights you want to use. If you select Simulink...
             model weights, the generated weights are discarded.'
             ' '},...
             'Weight Selection','Simulink Model Weights','Generated Weights',...
             'Generated Weights');
        case 'Simulink Model Weights'
            overwriteOK = 1;
        case 'Generated Weights'
            overwriteOK = 0;
        end % switch questdlg
    else
        overwriteOK = 0;
    end
else
    overwriteOK = 0;
end
if overwriteOK
    netn.IW{1,1}=IW1_1a; netn.IW{3,1}=IW3_1a; netn.IW{5,1}=IW5_1a;
    netn.IW{7,2}=IW7_2a; netn.IW{8,3}=IW8_3a;
    netn.IW{10,2}=IW10_2a; netn.IW{11,3}=IW11_3a;
    netn.LW{2,1}=LW2_1a; netn.LW{4,3}=LW4_3a; netn.LW{6,5}=LW6_5a;
    netn.LW{7,6}=LW7_6a; netn.LW{8,6}=LW8_6a;
    netn.LW{10,6}=LW10_6a; netn.LW{11,6}=LW11_6a;
    netn.LW{9,2}=LW9_2a; netn.LW{9,7}=LW9_7a; netn.LW{9,8}=LW9_8a;
    netn.LW{12,4}=LW12_4a; netn.LW{12,10}=LW12_10a;
    netn.LW{12,11}=LW12_11a;
    netn.b{1}=B1a; netn.b{2}=B2a; netn.b{3}=B3a; netn.b{4}=B4a;
    netn.b{5}=B5a; netn.b{6}=B6a;
else
    netn.IW{1,1}=IW1_1b; netn.IW{3,1}=IW3_1b; netn.IW{5,1}=IW5_1b;
    netn.IW{7,2}=IW7_2b; netn.IW{8,3}=IW8_3b;
    netn.IW{10,2}=IW10_2b; netn.IW{11,3}=IW11_3b;
    netn.LW{2,1}=LW2_1b; netn.LW{4,3}=LW4_3b; netn.LW{6,5}=LW6_5b;
    netn.LW{7,6}=LW7_6b; netn.LW{8,6}=LW8_6b;
    netn.LW{10,6}=LW10_6b; netn.LW{11,6}=LW11_6b;
    netn.LW{9,2}=LW9_2b; netn.LW{9,7}=LW9_7b; netn.LW{9,8}=LW9_8b;
    netn.LW{12,4}=LW12_4b; netn.LW{12,10}=LW12_10b;
```

```
                netn.LW{12,11}=LW12_11b;
                netn.b{1}=B1b; netn.b{2}=B2b; netn.b{3}=B3b; netn.b{4}=B4b;
                netn.b{5}=B5b; netn.b{6}=B6b;
            end
        end
    end
end
                                        % Training function could be changed in continue training.
a1 = get(H.trainfun_edit,'value');
if (a1 < 1) | (a1 > 13),
    trainfun=get_param(arg1,'trainfun');
    a1=strmatch(trainfun,func_index);
    present_error(fig,H,H.trainfun_edit,a1,0, 'Please, correct the training function');
    if ishandle(HH)
        delete(HH);
    end
    return
else
    trainfun=func_index(a1,:);
    set_param(arg1,'trainfun',trainfun);
    for k1=8:-1:1
        if trainfun(k1)==' '
            trainfun=trainfun(1:k1-1);
        else
            break
        end
    end
end
netn.trainFcn=trainfun;
end
netn.trainParam.epochs = epochs; netn.trainParam.show = 1;
netn.trainParam.min_grad=1e-10;

set(H.error_messages,'string','Training Neural Network');
set(fig,'pointer','watch');
if ishandle(HH)
    delete(HH);
end
pause(1);
if ~Use_Testing & ~Use_Validation                % Train the neural network.
    [netn,tr] = train(netn,ptr,ttr);
elseif ~Use_Testing
    [netn,tr] = train(netn,ptr,ttr,[ ],[ ],vv);
elseif ~Use_Validation
    [netn,tr] = train(netn,ptr,ttr,[ ],[ ],[ ],tt);
else
    [netn,tr] = train(netn,ptr,ttr,[ ],[ ],vv,tt);
```

```
end
set(H.Simulating_text,'visible','off');
save(cat(2,tempdir,'nnidentdata.mat'));

parent_function=get(H.parent_function_ptr,'userdata');
if strcmp(parent_function,'narma_l2')
    title_fig2='NN NARMA L2';
end
Ysim = sim(netn,ptr);                                   % Get the simulation results.

if strcmp(parent_function,'narma_l2')                   % Update the weights.
   IW1_1=netn.IW{1,1}; IW3_1=netn.IW{3,1}; IW5_1=netn.IW{5,1};
   IW7_2=netn.IW{7,2}; IW8_3=netn.IW{8,3}; IW10_2=netn.IW{10,2}; IW11_3=netn.IW{11,3};
   LW2_1=netn.LW{2,1}; LW4_3=netn.LW{4,3}; LW6_5=netn.LW{6,5};
   LW7_6=netn.LW{7,6}; LW8_6=netn.LW{8,6}; LW10_6=netn.LW{10,6};
   LW11_6=netn.LW{11,6};
   LW9_2=netn.LW{9,2}; LW9_7=netn.LW{9,7}; LW9_8=netn.LW{9,8};
   LW12_4=netn.LW{12,4}; LW12_10=netn.LW{12,10}; LW12_11=netn.LW{12,11};
   B1=netn.b{1}; B2=netn.b{2}; B3=netn.b{3}; B4=netn.b{4}; B5=netn.b{5}; B6=netn.b{6};

   set(H.IW1_1_ptr,'userdata',IW1_1); set(H.IW3_1_ptr,'userdata',IW3_1);
   set(H.IW5_1_ptr,'userdata',IW5_1);
   set(H.IW7_2_ptr,'userdata',IW7_2); set(H.IW8_3_ptr,'userdata',IW8_3);
   set(H.IW10_2_ptr,'userdata',IW10_2); set(H.IW11_3_ptr,'userdata',IW11_3);
   set(H.LW2_1_ptr,'userdata',LW2_1); set(H.LW4_3_ptr,'userdata',LW4_3);
   set(H.LW6_5_ptr,'userdata',LW6_5);
   set(H.LW7_6_ptr,'userdata',LW7_6); set(H.LW8_6_ptr,'userdata',LW8_6);
   set(H.LW10_6_ptr,'userdata',LW10_6); set(H.LW11_6_ptr,'userdata',LW11_6);
   set(H.LW9_2_ptr,'userdata',LW9_2); set(H.LW9_7_ptr,'userdata',LW9_7);
   set(H.LW9_8_ptr,'userdata',LW9_8);
   set(H.LW12_4_ptr,'userdata',LW12_4); set(H.LW12_10_ptr,'userdata',LW12_10);
   set(H.LW12_11_ptr,'userdata',LW12_11);
   set(H.B1_ptr,'userdata',B1); set(H.B2_ptr,'userdata',B2); set(H.B3_ptr,'userdata',B3);
   set(H.B4_ptr,'userdata',B4);
   set(H.B5_ptr,'userdata',B5); set(H.B6_ptr,'userdata',B6);
end

set(H.error_messages,'string','Training complete. You can generate or import new data,...
     continue training or save results by selecting OK or Apply.');

H.Training_done=1;
set(H.Apply_but,'enable','on'); set(H.OK_but,'enable','on');
set(H.Handles.Menus.File.Save_NN,'enable','on')
set(H.Handles.Menus.File.Save_Exit_NN,'enable','on')
set(H.Start_but,'enable','on'); set(H.Cancel_but,'enable','on');
set(H.Use_Previous_Weights_ptr,'userdata',1);
set(H.Use_Previous_Weights_but,'value',1);
set(H.In_training_ptr,'userdata',0);
set(fig,'userdata',H,'pointer','arrow');
```

```
    figure(fig);
elseif strcmp(cmd,'stop_sim')
    fig2=findall(0,'type','figure','tag','nnidentdata');
    if size(fig2,1)==0, fig2=0; end
    f2=get(fig2,'userdata'); f2.stop=1;
    set(fig2,'UserData',f2);
    return;
elseif (strcmp(cmd,'apply') | strcmp(cmd,'ok')) & (fig)
    if get(H.In_training_ptr,'userdata')~=0
        return
    end
    arg1=get(H.gcbh_ptr,'userdata');

    S1=get(H.S1_ptr,'userdata');
    sim_file=get(H.sim_file_ptr,'userdata');
    Ts=get(H.Ts_ptr,'userdata');
    Ni=get(H.Ni_ptr,'userdata');
    Nj=get(H.Nj_ptr,'userdata');
    Use_Previous_Weights = get(H.Use_Previous_Weights_ptr,'userdata');
    Use_Validation=get(H.Use_Validation_but,'value');
    Use_Testing=get(H.Use_Testing_but,'value');
    max_i=get(H.max_i_ptr,'userdata');
    min_i=get(H.min_i_ptr,'userdata');
    max_i_int=get(H.max_i_int_ptr,'userdata');
    min_i_int=get(H.min_i_int_ptr,'userdata');
    sam_training=get(H.sam_training_ptr,'userdata');
    epochs=get(H.epochs_ptr,'userdata');

    parent_function=get(H.parent_function_ptr,'userdata');
    if strcmp(parent_function,'narma_l2')
      IW1_1=get(H.IW1_1_ptr,'userdata'); IW3_1=get(H.IW3_1_ptr,'userdata');
      IW5_1=get(H.IW5_1_ptr,'userdata');
      IW7_2=get(H.IW7_2_ptr,'userdata'); IW8_3=get(H.IW8_3_ptr,'userdata');
      IW10_2=get(H.IW10_2_ptr,'userdata'); IW11_3=get(H.IW11_3_ptr,'userdata');
      LW2_1=get(H.LW2_1_ptr,'userdata'); LW4_3=get(H.LW4_3_ptr,'userdata');
      LW6_5=get(H.LW6_5_ptr,'userdata');
      LW7_6=get(H.LW7_6_ptr,'userdata'); LW8_6=get(H.LW8_6_ptr,'userdata');
      LW10_6=get(H.LW10_6_ptr,'userdata'); LW11_6=get(H.LW11_6_ptr,'userdata');
      LW9_2=get(H.LW9_2_ptr,'userdata');LW9_7=get(H.LW9_7_ptr,'userdata');
      LW9_8=get(H.LW9_8_ptr,'userdata');
      LW12_4=get(H.LW12_4_ptr,'userdata'); LW12_10=get(H.LW12_10_ptr,'userdata');
      LW12_11=get(H.LW12_11_ptr,'userdata');
      B1=get(H.B1_ptr,'userdata'); B2=get(H.B2_ptr,'userdata'); B3=get(H.B3_ptr,'userdata');
      B4=get(H.B4_ptr,'userdata'); B5=get(H.B5_ptr,'userdata'); B6=get(H.B6_ptr,'userdata');

      set_param(arg1,'IW1_1',mat2str(IW1_1,20)); set_param(arg1,'IW3_1',mat2str(IW3_1,20));
      set_param(arg1,'IW5_1',mat2str(IW5_1,20));
      set_param(arg1,'IW7_2',mat2str(IW7_2,20)); set_param(arg1,'IW8_3',mat2str(IW8_3,20));
```

```
  set_param(arg1,'IW10_2',mat2str(IW10_2,20));
  set_param(arg1,'IW11_3',mat2str(IW11_3,20));
  set_param(arg1,'LW2_1',mat2str(LW2_1,20)); set_param(arg1,'LW4_3',mat2str(LW4_3,20));
  set_param(arg1,'LW6_5',mat2str(LW6_5,20));
  set_param(arg1,'LW7_6',mat2str(LW7_6,20)); set_param(arg1,'LW8_6',mat2str(LW8_6,20));
  set_param(arg1,'LW10_6',mat2str(LW10_6,20));
  set_param(arg1,'LW11_6',mat2str(LW11_6,20));
  set_param(arg1,'LW9_2',mat2str(LW9_2,20)); set_param(arg1,'LW9_7',mat2str(LW9_7,20));
  set_param(arg1,'LW9_8',mat2str(LW9_8,20));
  set_param(arg1,'LW12_4',mat2str(LW12_4,20));
  set_param(arg1,'LW12_10',mat2str(LW12_10,20));
  set_param(arg1,'LW12_11',mat2str(LW12_11,20));
  set_param(arg1,'B1',mat2str(B1,20)); set_param(arg1,'B2',mat2str(B2,20));
  set_param(arg1,'B3',mat2str(B3,20));
  set_param(arg1,'B4',mat2str(B4,20)); set_param(arg1,'B5',mat2str(B5,20));
  set_param(arg1,'B6',mat2str(B6,20));
end

Normalize=get(H.Normalize_ptr,'userdata');
Limit_output=get(H.Limit_output_data,'value');
max_out=get(H.max_out_ptr,'userdata');
min_out=get(H.min_out_ptr,'userdata');
set_param(arg1,'S1',num2str(S1)); set_param(arg1,'sim_file',sim_file);
set_param(arg1,'Ts',num2str(Ts)); set_param(arg1,'Ni',num2str(Ni));
set_param(arg1,'Nj',num2str(Nj));
set_param(arg1,'Use_Previous_Weights',num2str(Use_Previous_Weights));
set_param(arg1,'Use_Validation',num2str(Use_Validation));
set_param(arg1,'Use_Testing',num2str(Use_Testing));
set_param(arg1,'max_i',num2str(max_i));
set_param(arg1,'min_i',num2str(min_i));
set_param(arg1,'max_i_int',num2str(max_i_int));
set_param(arg1,'min_i_int',num2str(min_i_int));
set_param(arg1,'sam_training',num2str(sam_training));
set_param(arg1,'epochs',num2str(epochs));
set_param(arg1,'Normalize',num2str(Normalize));
set_param(arg1,'limit_output',num2str(Limit_output));
set_param(arg1,'max_output',num2str(max_out));
set_param(arg1,'min_output',num2str(min_out));

if strcmp(cmd,'ok')
   if ~strcmp(parent_function,'narma_l2')
      arg2=get(H.gcb_ptr,'userdata');
      feval(parent_function,'',arg1,arg2,'nnident');
   end
   delete(fig)
   if exist(cat(2,tempdir,'nnidentdata.mat'))
       delete(cat(2,tempdir,'nnidentdata.mat'));
```

```
    end
  end
end

function present_error(fig,H,text_field,field_value,field_type,message)

if H.Data_Available
   set(H.Start_but,'enable','on')
end
if H.Training_done
   set(H.OK_but,'enable','on')
   set(H.Apply_but,'enable','on')
   set(H.Handles.Menus.File.Save_NN,'enable','on')
   set(H.Handles.Menus.File.Save_Exit_NN,'enable','on')
end
set(H.Cancel_but,'enable','on');
if text_field~=0
   if field_type          % Number
      set(text_field,'string',num2str(field_value));
   else                   % ASCII or No change.
      set(text_field,'string',field_value);
   end
else
   text_field=0;
end
set(H.error_messages,'string',message);
errordlg(message,'Plant Identification Warning','modal');
set(fig,'pointer','arrow');
```