Algebraic and Adaptive Learning in Neural Control Systems

Silvia Ferrari

A DISSERTATION

PRESENTED TO THE FACULTY OF PRINCETON UNIVERSITY

IN CANDIDACY FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

RECOMMENDED FOR ACCEPTANCE BY THE

DEPARTMENT OF MECHANICAL AND AEROSPACE ENGINEERING

NOVEMBER 2002

Abstract

A systematic approach is developed for designing adaptive and reconfigurable nonlinear control systems that are applicable to plants modeled by ordinary differential equations. The nonlinear controller comprising a network of neural networks is taught using a two-phase learning procedure realized through novel techniques for initialization, on-line training, and adaptive critic design. A critical observation is that the gradients of the functions defined by the neural networks must equal corresponding linear gain matrices at chosen operating points. On-line training is based on a dual heuristic adaptive critic architecture that improves control for large, coupled motions by accounting for actual plant dynamics and nonlinear effects. An action network computes the optimal control law; a critic network predicts the derivative of the cost-to-go with respect to the state. Both networks are algebraically initialized based on prior knowledge of satisfactory pointwise linear controllers and continue to adapt on line during full-scale simulations of the plant.

On-line training takes place sequentially over discrete periods of time and involves several numerical procedures. A backpropagating algorithm called Resilient Backpropagation is modified and successfully implemented to meet these objectives, without excessive computational expense. This adaptive controller is as conservative as the linear designs and as effective as a global nonlinear controller. The method is successfully implemented for the full-envelope control of a six-degree-of-freedom aircraft simulation. The results show that the on-line adaptation brings about improved performance with respect to the initialization phase during aircraft maneuvers that involve large-angle and coupled dynamics, and parameter variations.

Acknowledgments

I am especially grateful to my advisor, Professor Robert F. Stengel, for honoring me as his protégéé during these past five years. He is the role model I wish to follow in my future professional and personal life, because of his distinct creativity, wisdom, and integrity. Thanks to him, my graduate experience has surpassed any of the dreams and expectations I had as an incoming student. I thank Prof. Philip Holmes and Prof. Jeremy Kasdin for serving as readers and for providing valuable insight and advice. The completion of this dissertation would not have been possible without the literature contributions cited herein. In particular, I wish to thank Dr. Paul Werbos, Prof. Andrew Barto, Prof. Bernard Widrow, and Prof. Robert Vanderbei for inspiring me through their work and conversations.

I am indebted to those institutions which contributed financially to my graduate education: Princeton University provided support through its Wallace Memorial Honorific Fellowship in Engineering; the Zonta Foundation supported me through the Zonta Amelia Earhart Fellowship; the American Society of Mechanical Engineers awarded me a Graduate Teaching Fellowship; the American Astronautical Society contributed with a Donald K. “Deke” Slayton Memorial Fellowship; and the American Institute of Aeronautics and Astronautics supported me by means of the Guidance, Navigation, and Control Graduate Award. The funding for this research was provided by the Federal Aviation Administration and the National Aeronautics and Space Administration under FAA Grant No. 95-G-0011. It has been an honor and a privilege to take part in the FAA/NASA Joint University Program.

I owe a debt of gratitude to many members of the Princeton University community: Etta Recke, Arla Dittrick, Sharon Matarese, Maureen Hickey, Barbara Myers, Jessica Buchanan, Anna Marie Peloso, and many other members of our staff helped me in countless ways, countless times. I wish to thank everyone in the LCA Laboratory, Dr. Sai Gopisety, Prof. Qian Wang, Nilesh Kulkarni, and Russ Arrell for their help and companionship. I am especially grateful to my friends Samaya Nissanke and Arron Melvin, whose paths I have crossed here at Princeton and with whom I share a common mind and soul. Together, with Giorgia Seghedoni, Fabio Raimondi, Fabio Bonvicini, Samantha Rossi, Sharon Santos, and Daria Biancardi, they have nurtured my happiness and peace of mind, all along.

I wish to dedicate this thesis to those who make any of my accomplishments possible: my family. Kervin Johnson, my other half, took daily care of my health and spirit. He always has been by my side, sharing every moment and emotion, and making them evermore meaningful. Carlo Ferrari and Tina Serino, my parents, provided me with all that in life is precious: love, empathy, liberty, and adventure. My love and appreciation for them are simply endless.

This dissertation carries the number 3106-T in the records of the Department of Mechanical and Aerospace Engineering of Princeton University.

Silvia Ferrari

Princeton, New Jersey

August, 2002

Tables of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Recent advances in a variety of technologies and applications call for improved performance and reliability, while exacerbating the complexity and uncertainty of systems and their surroundings. In many instances, the operation of systems and devices can be modified and, possibly, optimized by the intervention of a *control system*, that is, an additional mechanism comprised of several components, such as sensors, computers, and actuators, that act upon an available input. The dynamic characteristics and physical properties of the system to be controlled (the plant) can be exploited to design automatic control systems. Some of the main difficulties to be overcome by the designer are the nonlinear plant dynamics and the uncertainties caused by differences between actual and assumed dynamic models. A fixed control design whose performance remains satisfactory in the presence of uncertainty is said to be *robust*. A controller whose parameters vary *on line* during operation is considered to be *adaptive* and can be expected to accommodate for a higher degree of uncertainty than a fixed control structure. If it is capable of adapting to system failures that are reflected by the state of the plant, then the controller also is *reconfigurable*.

The objective of this thesis is to develop a novel approach for the design of adaptive control systems that are both robust and reconfigurable, and that apply to plants modeled by nonlinear ordinary differential equations. The potential brought about by using post-modern computational paradigms, such as neural networks and fuzzy logic, in conjunction with conventional control techniques has been recognized in the past by

several authors [1-4]. In some cases [5-7], a global controller was obtained by training a neural network to approximate the linear gains provided by linear multivariable control. In other applications [8-10], a conventional control system architecture was augmented by a neural or fuzzy computational structure that provided for on-line adaptation. In this thesis, classical and neural control systems are synthesized to combine the most effective elements of old and new design concepts with the promise of producing better control systems. The novel approach to nonlinear control design retains the characteristics of stability and robustness of classical, linear control laws, while capitalizing on the broader capabilities of a so-called *adaptive critic* neural network. First, the neural control system's architecture and parameters are determined from the initial specification of the control law by solving algebraic linear systems of equations during a so-called *pre-training phase*. Secondly, the neural parameters are modified during an *on-line training phase* to account for uncertainties that were not captured in the linearizations, such as nonlinear effects, control failures, and parameter variations.

1.1 Background and Motivation

The foundations of the nonlinear system design lie in the field of dynamic programming [11], which is perhaps the most general approach for solving optimal control problems. Its globally-convergent properties can be exploited in conjunction with an approximating, parametric structure to solve for a near-optimal solution on line, accounting for information about the state of the system as it arises. Neural networks are the parametric structure of choice, because of their ability to approximate unknown nonlinear mappings over high-dimensional compact spaces, and because of their potential for on-line learning. *A-priori* knowledge of the system to be controlled is incorporated in

the neural controller in the form of *gain-scheduled* linear designs. Gain scheduling is a conventional approach to global control design that correlates the gains of a linear multivariable control structure with a set of dynamically significant variables in the system. It is widely used in the aerospace industry because it affords a straightforward and effective procedure for applying linear control theory to the nonlinear aircraft dynamics.

1.1.1 Approximate Dynamic Programming and Reinforcement Learning

Dynamic programming (DP) methods use the *principle of optimality* [11] to find a strategy of action that optimizes a desired performance metric or *cost* subject to nonlinear dynamical constraints. The cost, J, to be optimized typically is defined as a function that measures performance with respect to quantities that are to be either minimized or maximized by a given process. The principle of optimality states that given the initial path a-b, and the optimal cost J_{abc}^{*} associated with going from a to c through b, then the cost J_{bc} also is optimal for the path that goes from b to c. Figure 1 illustrates this principle for a two-stage process, with $(\bullet)^{*}$ denoting optimality. *Backward* or *discrete* dynamic programming discretizes the state space and makes a direct comparison of the cost associated with all feasible trajectories, guaranteeing a solution of the global optimal control problem [12]. The space of admissible solutions is reduced by examining a multi-stage decision process as a sequence of one-stage processes.

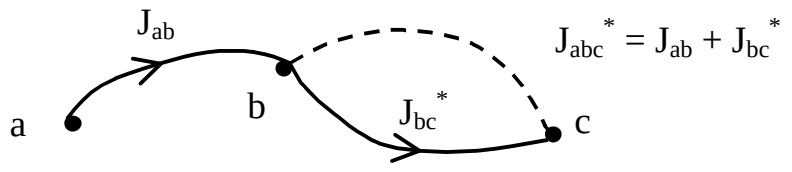

Figure 1. The principle of optimality applied to a two-stage process.

As an example, the discrete dynamic programming approach is applied to the process shown in Fig. 2. The last stage's optimal paths are computed for all possible intermediate state values c, d, and e, thereby producing the optimal costs J_{cf}^*, J_{df}^*, and J_{ef}^*, respectively. Then, by the principle of optimality, these paths also must be optimal for the last stage of the optimal trajectories that go from b to f through the respective state values, e.g., $J_{bcf}^* = J_{bc} + J_{cf}^*$, and so on. Comparing the optimal costs over the last stage reveals which is the optimal (smallest or largest) among these costs, but does not necessarily reveal which cost is associated with the globally-optimal path from b to f. Instead, if the last-stage's costs, J_{cf}^*, J_{df}^*, and J_{ef}^*, are stored, then the total costs J_{bcf}^*, J_{bdf}^*, and J_{bef}^* can be computed and compared to find the globally-optimal path from b to f. For a process with more than two stages, b is only one of many possible state values. Hence, the same computation needs to be carried out for all other possible values at this second-to-last stage (such as g, in Fig. 2), in order to determine all possible optimal paths for the last two stages (e.g., J_{bf}^*, J_{gf}^*, and so on).

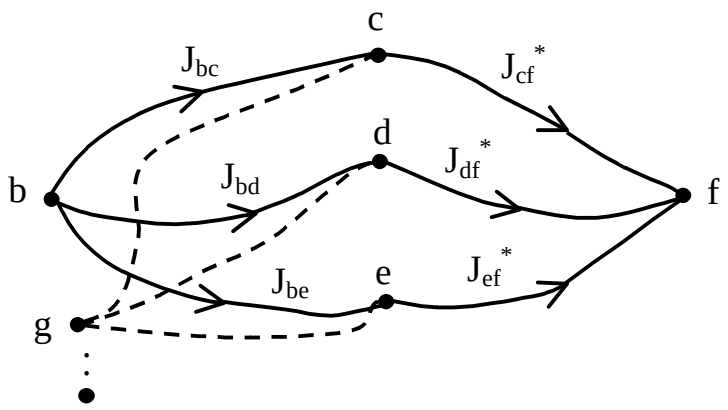

Figure 2. Discrete or backward dynamic programming approach.

Although the backward DP approach reduces the space of admissible solutions, it remains computationally too expensive for higher dimensional systems, with a large

number of stages. The required multiple generation and expansion of the state and the storage of all optimal costs lead to a number of computations that grows exponentially with the number of state variables, commonly referred to as the "curse of dimensionality" or "expanding grid" [12]. Approximate dynamic programming (ADP) and temporal difference methods use incremental optimization combined with a parametric structure to reduce the computational complexity associated with evaluating the cost [13-15]. Unlike discrete DP, ADP algorithms progress forward in time, and approximate both the optimal policy and the cost in real time by considering only the present value of the state. Suppose a is the initial state of the process shown in Fig. 3, then the cost for the first stage, J_{ab}, can be accurately computed based on present information. The optimal cost over all future stages or *cost-to-go*, J_{bf}^{*}, is predicted, or estimated $(\hat{\bullet})$ in the presence of uncertainty, by a function approximator or parametric structure. A parametric structure consists of any functional relationship whose adjustable parameters allow it to approximate different mappings. At the next stage, b-c, this procedure is repeated, only now the policy and cost approximations have had a chance to improve based on the information gathered during the first stage. Therefore the next path, from c to f, is closer to the optimal trajectory.

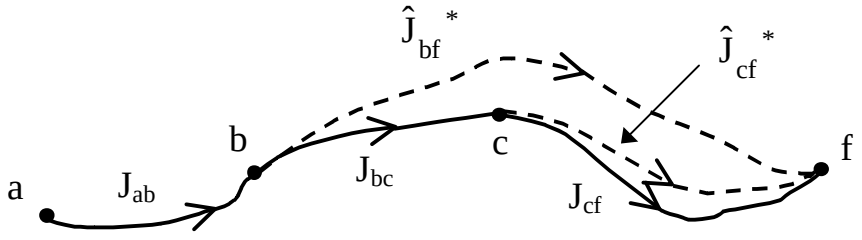

Figure 3. Approximate or forward dynamic programming approach.

Adaptive critic designs (ACD) reproduce the most general solution of ADP by deriving recurrence relations for the optimal policy, the cost, and, possibly, their derivatives. The goal is to overcome the curse of dimensionality, while ensuring convergence to a near-optimal solution over time [4, 16]. Although they can be viewed as "complex" or "intimidating" [17], adaptive critics offer a unified approach to dealing with the controller's nonlinearity, robustness, and reconfiguration for a system whose dynamics can be modeled by a general ordinary differential equation. Perhaps the most critical aspects of ACD are found in the implementation. The simplest form of adaptive critic design, Heuristic Dynamic Programming (HDP), uses a parametric structure called an action network to approximate the control policy and another parametric structure called a critic network to approximate the future cost or cost-to-go. In practice, since the parameters of this architecture adapt only by means of the scalar cost, HDP has been shown to converge very slowly [18].

An alternative approach referred to as Dual Heuristic Programming (DHP) has been proposed [18, 19]. Here, the critic network approximates the derivatives of the future cost with respect to the state, thereby correlating the adjustable parameters in the architecture to a larger number of dependent variables. Although the advantages of DHP over HDP have been discussed extensively in the literature from a theoretical point of view, few successful implementations have been reported. Due to the use of derivative information, the recurrence relations that can be obtained for DHP are more involved and require an accurate model of the system to be controlled. The critic network approximates a nonlinear mapping characterized by a much larger-dimensional output

space. Therefore, practical aspects such as function approximation are more challenging in DHP than they are in HDP.

Many other methodologies have been proposed over the years to alleviate some of the difficulties mentioned above, producing more advanced designs. Globalized Dual Heuristic Programming (GDHP), for example, has been developed with the purpose of combining the advantages of both HDP and DHP architectures [14, 20, 21]. In this case, the critic network approximates both the cost-to-go and its derivatives. Action-dependent (AD) versions of all these approaches are obtained by designing a critic network that has direct knowledge of the control policy (produced by the action network) through its inputs, as opposed to only having knowledge of its derivatives through its adaptation (as in the action-independent ACD designs). The motivation behind this [22] and other [23] methodologies is to achieve faster learning by the parametric structures and faster convergence of the overall scheme.

Aside from having common roots in ADP, these methods are related through the idea of "linking backpropagation with reinforcement learning via the critic network" [24]. The near-optimal trajectory is found and learned without having explicit knowledge of the near-optimal control action, as normally would be required by supervised learning. Reinforcement learning is based on the principle that knowledge of the ideal cost-to-go suffices to modify the course of action accordingly, in real time. In other words the control system is expected to learn from its own mistakes, or by anticipating them through an indirect measure of performance, that is, the cost-to-go function. This is reminiscent of the task faced by an animal in a Pavlovian or classical conditioning experiment [3]. The "payoff" that is delivered to the animal as a result of its actions is

referred to as *primary reinforcement*. Anticipation of events that would eventually provide primary reinforcement is referred to as *secondary reinforcement* and constitutes a framework similar to that of adaptive critic designs.

1.1.2 Neural Networks as Universal Function Approximators

Function approximation is a principal element of ADP methods. The recurrence relations that satisfy existing convergence proofs [4, 16] imply global convergence of the approximating schemes at every stage or time interval. The behavior of the adaptive controller is closely related to the effectiveness of the function approximator of choice, as it determines future performance. The adaptive critic approach also requires the approximating tool to be flexible. In most practical problems, some *a-priori* knowledge of the system is available from analytical studies, modeling, or experiments. Regardless of how this information is obtained, it always can be gathered into one or more sets of data and exploited to initialize the system. Additional information becomes available over time, during the adaptation, and also needs to be incorporated in the control system. Hence, the function approximators must be able to learn both in batch and incremental mode and to deal with the multidimensional, nonlinear action and critic functionals.

Neural networks are the only class of universal function approximators that possess all of these properties. Other parametric structures, such as splines and wavelets, have become standard tools in regression and signal analysis involving input spaces that are up to three dimensional [25-28]. However, much of univariate approximation theory does not generalize well to multi-dimensional input and output spaces [29]. For example, the majority of spline-based solutions for multivariate approximation problems involve tensor product spaces that are highly dependent on the coordinate system of choice [30-

32]. Artificial neural networks can easily deal with multivariate inputs and outputs because of their inherent parallel architecture. Furthermore, given a sufficient number of nonlinear units, they can approximate any continuous nonlinear function on a compact space with an arbitrary error [33-35], and match any input/output or gradient data set exactly [36, 37]. Because of their unique capabilities, neural networks likely are the best candidate for ACD implementations. The mathematical investigation of their approximation properties should be considered as an integral part of ACD.

An artificial neural architecture consists of one or more layers of nodes or transfer functions. Each node is characterized by one scalar-input scalar-output function that can be either linear or nonlinear and can take virtually any shape. Sigmoidal neural networks are characterized by one or more layers of sigmoidal transfer functions with a fixed shape dictated by an exponential or hyperbolic-tangent relationship. They are characterized by global support, and are appropriate for modeling functionals that are expected to be nonlinear but smooth. Another popular class of neural networks implements radial-basis transfer functions whose shape (i.e., width and center) is customized, by varying the parameters of each basis function. These networks have many desirable properties, such as fast learning, but they do not generalize local information as well as sigmoidal networks [8]. Therefore, they perform best when large data sets are available [38]. This thesis deals exclusively with the class of single-hidden-layer sigmoidal neural networks. Although, in some cases, multiple-nonlinear-layer architectures require a lesser number of adjustable parameters to approximate the same function, it is not known if these results are generally applicable.

The typical search criterion, used in virtually all supervised learning algorithms, consists of minimizing some measure of the error between the desired output (and/or derivative) and the network's actual performance. In particular, many backpropagation-based techniques have been devised for this purpose [39]. Although the problem of minimizing some form of sigmoidal-network error with respect to its adjustable parameters appears computationally tractable, it is characterized by many local minima whose number grows with the dimensions of the surface being approximated. This in turns leads to problems such as *overfitting*, where the training algorithm converges to a solution that minimizes the network error, but does not produce proper generalization properties. In order to smooth the interpolating surface, the number of hidden nodes typically is decreased until a satisfactory tradeoff between close matching of the training set and good generalization properties is found. Another difficulty associated with this approach is that optimization-based training algorithms that are robust and fast converging, such as that due to Levenberg and Marquardt [40, 41], are too computationally expensive for large, non-sparse problems (e.g., with large data sets, and many adjustable parameters).

The philosophy advocated in the following chapters does not rely solely on a parsimonious use of the adjustable parameters. In fact, a clear implication of all parametric structures is that the class of functions they can approximate increases with the number of parameters. Rather, it reveals that allowing for a certain degree of redundancy in the approximating structures may be fruitful for on-line applications, where the class of functions to be approximated should be minimally restricted *a priori*. The limitations that arise with large architectures are associated with the training

algorithms rather that with the neural networks themselves. Therefore, one aim of this thesis is to investigate alternative neural network training techniques.

1.1.3 Adaptive Flight Control Systems

The control design approach developed in this thesis is applied to the adaptive control of aircraft. This section provides an overview of other existing control designs that have been proposed over the years to address uncertainties and nonlinearities in the aircraft dynamics. Gain scheduling is by far the most commonly used design in actual applications, such as digital 'fly-by-wire' control systems. It consists of designing locally optimal linear control laws for a selected number of nominal flight conditions or equilibria. In between the chosen equilibria, the linear controllers are interpolated through auxiliary variables that capture the most relevant system dynamics, also referred to as *scheduling variables*, in order to produce a global design [42-44]. A disadvantage of gain-scheduled controllers is that their performance may deteriorate when extreme maneuvers and flight conditions that were not accounted for by the linear designs arise. Also based on the principle of scheduling system linearizations are the *extended linearization* [45, 46] and the *nonlinear tracking* approaches [47]. Both methods are computationally intensive for multivariate aircraft models and prolonged time horizons, and remain based on linearized dynamics.

In an effort to better account for nonlinearities, the broad class of techniques commonly known as *feedback linearization* has been extensively investigated for advanced flight control systems [48-50]. With this approach, accurate knowledge of the aircraft nonlinear dynamics is required in order to represent it as an equivalent linear system by means of a coordinate transformation. Robust nonlinear schemes that exploit

nonlinear dynamic inversion for feedback linearization also have been developed [52-56] to address uncertainties, such as unmodeled dynamics, as well as parametric and nonlinear uncertainties.

Adaptive control systems that account for system dynamics as they take place aim at improving performance while retaining a certain degree of robustness to unmodeled dynamics and uncertainties. *Sliding mode* controllers [57], for example, switch control laws to track a desired trajectory, combining both high precision and robustness. In many recent designs, function-approximation tools are used in combination with one of the conventional approaches mentioned above. For example, an adaptive controller can be obtained by using B-splines to approximate aerodynamic data for use in a nonlinear inverse dynamic architecture [58]. Another successful approach consists of using neural networks for scheduling real-time switching controllers [59, 60]. Neural networks also have been used extensively in conjunction with feedback linearization, to compensate for unknown or unmodeled nonlinear dynamics, as well as control failures [61-63]. In principle, adaptive critics allow for a more general approach to formulating the control law and to incorporating prior control knowledge.

1.2 Research Objectives

The primary goal of this dissertation is to develop a control system that is as conservative as the classical designs and as effective as a global nonlinear controller. The nonlinear control system must retain the same characteristics of stability and robustness of an equivalent gain-scheduled controller. Furthermore, it must adapt on line to provide near-optimal performance for all operating conditions, as well as for possible control failures and parameter variations, without that they are necessarily accounted for

a priori. The controller can be assumed to operate in continuous time and to be based on full-state feedback, with perfect measurements. Although the proposed design is expected to offer dynamic compensation in the presence of constant and slowly-varying disturbances, unmodeled inputs and stochastic effects are not investigated in this thesis. An accurate model of the plant is obtainable from a full-scale simulation of the aircraft that is built from mathematical models, full-scale wind tunnel data, and actual physical and performance characteristics of an idealized twin-jet business aircraft [64]. However, the model is not perfect; for example, it does not predict the control system failures and the parameter variations simulated in this thesis.

An important objective is the development of a novel procedure for incorporating the classical designs, i.e., gain-scheduled linear controllers, in an existing adaptive critic architecture. Here, these linear designs are obtained for a subset of operating conditions under assumptions of decoupled longitudinal and lateral-directional dynamics, small perturbations, and small time-varying dynamic effects. The adaptation is expected to improve performance with respect to the gain-scheduled designs for maneuvers and conditions that do not meet the above assumptions, the first time that they are encountered. Therefore, the simulation, which plays the role of the actual aircraft, is allowed to explore the entire operational domain. Another objective of this dissertation is to develop training techniques that allow adaptive controllers to retain prior global information, while improving upon them locally. This often is recognized as a major conundrum in adaptive control and constitutes a challenge to be overcome by the learning algorithm.

The most general approach for incorporating prior domain-specific knowledge consists of initializing the neural parameters. This allows the designer to take into consideration the known dynamics, without restricting the overall approach to their particular structure. This thesis aims not only at investigating new and effective ways to initialize the neural parameters, but also at devising methodologies to preserve *a-priori* knowledge during adaptation. Finally, since the class of functions that can be approximated by the action and the critic strictly depends on the number of nonlinear units in these neural networks, a systematic procedure for determining their size also is developed. Special consideration is dedicated to reducing the computational complexity of the numerical solution.

1.3 Thesis Organization

The main body of the thesis is organized in four chapters. Chapter 2 lays the foundations of the nonlinear adaptive control design. The proposed philosophy is formalized by reviewing the Linear Quadratic Regulator (LQR) and by linking this classical design to the adaptive critic architecture of choice, i.e., Dual Heuristic Programming Adaptive Critics. This chapter provides a theoretical framework and background, and suggests a general solution procedure that can be applied to a wide range of nonlinear optimal control problems. The classical LQR solution is combined with the neural network-based design in a novel idea referred to as Classical/Neural Synthesis of Nonlinear Control Systems. Chapter 3 supplies a general introduction to the new learning techniques that were specifically developed with the control design objectives in mind. A sample architecture of a single-layer scalar-output sigmoidal

network is introduced in Section 3.1, and used in the remainder of the chapter to illustrate the algebraic training and modified resilient backpropagation algorithms.

Chapters 4 and 5 describe the control design procedure in a sequential fashion. Each chapter begins with the most general formulation of the design objectives, and progressively focuses on the specific application treated in this thesis, i.e., Proportional-Integral Neural Network Control of Aircraft. Chapter 4 explains the pre-training phase, where a well-established linear-design procedure, referred to as proportional-integral control with implicit model following, is used to produce a set of linear controllers. The novel algebraic training approach is used to incorporate the linear controllers in the nonlinear neural networks by initializing their parameters, simultaneously determining their architecture. The neural networks are pre-trained and tested in Sections 4.3-4.5, based on the control knowledge obtained in Section 4.1. Section 4.2 describes the neural network control structure motivated by the linear design and by the adaptive critic architecture.

Chapter 5 shows how the neural controller that was pre-trained in Chapter 4 is adapted on line. Section 5.1 deals with newly proposed adaptive critic implementation details and algorithms that may prove useful for other ACD applications, as well. Section 5.2 presents the adaptive control results, and compares its performance to that of the pre-trained control structure obtained in Chapter 4. The end of Chapter 5 (Sections 5.3-5.4) is dedicated to preliminary results regarding the stability of the on-line training algorithm (presented in Section 5.1.3) and the computation time required by the proposed numerical schemes. Both sections are meant to consolidate the results of Section 5.2, as well as to provide a stepping-stone for future work. Finally, the Appendices contain the

nomenclature (Appendix A) and the key algorithms coded in MATLAB (Appendix B), the proofs (Appendix C), the data sets (Appendix D), and the software architectures (Appendix E) that were omitted from the main body of the thesis for sake of continuity.

1.3.1 Summary of Results

The adaptive controller improves performance with respect to the classical designs during large angle and extreme maneuvers, when nonlinear and coupling dynamic effects become significant. In the case of a large-bank-angle maneuver, it even is capable of preventing loss of stability of the closed-loop system. The control design is applicable to the general form of the governing ordinary differential equation, as the adaptation accounts for those dynamics that were ignored or neglected by the conventional design. The approach also exploits the reconfigurable nature of neural networks, and is able to deal with unanticipated failures of the controls. The adaptation is sufficiently fast to learn from unforeseen conditions soon after they arise, relative to the time scale of the aircraft dynamics.

The results also show that convergence always is achieved by the adaptive critic architecture, provided that the neural network inputs are bounded. The two-phase approach to design and the novel training techniques together achieve the learning objectives described in Section 1.2. It is demonstrated through both simulations and analytical results that the adaptive neural networks improve their performance locally on line, while preserving prior knowledge over the unexplored state space. They also are capable of building upon performance that was assimilated or "learned" on line, allowing the control system to enhance its capabilities as it revisits the same maneuvers again and again, over time.

A systematic approach for designing adaptive control systems is developed and demonstrated on a reasonable-sized problem. The adaptive critic methodology allows for extensions that can address many aspects of the control design, such as robustness, reconfiguration, system identification, and inequality constraints. The supervised and reinforcement learning techniques developed are fairly general in nature and, thus, can be used in many other neural network and adaptive critic applications. In particular, the algebraic training technique shows great potential for investigating neural approximation properties and for guaranteeing performance baselines both before and during on-line training.

Chapter 2

Foundations of the Neural Control Design

The problem of determining a functional that optimizes a desired metric over time is one of comprehensive relevance, as it lies at the basis of many control and identification schemes. Optimal control laws and satisfactory stability and robustness results can be derived for linear systems, in particular when these systems also are time invariant. As a consequence, a wide range of linear control and identification designs have been developed and are commonly implemented in the industry. In actuality, all plants are characterized by nonlinear dynamics and are subject to change due to both internal and external effects. Furthermore, advances in a variety of technologies and applications demand better performance, while exacerbating the complexity and uncertainty of systems and their surroundings.

There is considerable precedent for applying gain-scheduled linear controllers to nonlinear systems, especially those that can be locally approximated as linear-parameter-varying systems. Gain-scheduled designs adapt to changing operating conditions, but their performance typically deteriorates when rapid changes occur and when highly nonlinear or unforeseen regimes are encountered. Artificial neural networks potentially can compensate for these shortcomings, because of their ability to approximate unknown nonlinear mappings with high-dimensional input spaces and their promise for real-time learning. Extensive numerical studies [5, 33, 65] have shown that they are capable of dealing with those difficulties typically associated with complex control applications, such as nonlinearity and uncertainty. However, practical applications also call for a

better understanding of the theoretical principles involved [65]. In particular, there is no simple way to apply the insights afforded by classical control methods to the specification and preliminary design of neural network controllers.

In this chapter, a novel approach for designing adaptive control systems is introduced. The method, referred to as classical/neural synthesis of control systems, takes advantage of prior knowledge and experience gained from scheduled linear controllers, while capitalizing on the broader capabilities of adaptive, nonlinear control theory and computational neural networks. The nonlinear control system, comprising a network of networks, is motivated by a corresponding linear structure and specified using a two-phase learning procedure. A key, novel observation is that the gradients of the functions defined by the neural networks must equal corresponding linear gain matrices at chosen operating points. On-line learning is based on a DHP adaptive-critic approach [18] that improves control response by accounting for differences between actual and assumed dynamic models and for nonlinear effects not captured in the linearizations. Control theory provides a unifying framework for both design phases. The initial specification of the control law is based on the linear quadratic regulator; the DHP approach is based on approximate dynamic programming.

2.1 The Nonlinear Optimal Control Problem

An initial assumption is that a nonlinear differential equation that models the plant dynamics is available in the form

$$\dot{\mathbf{x}} = \mathbf{f}\left[\mathbf{x}(t), \mathbf{p}_{\mathrm{m}}(t), \mathbf{u}(t)\right] \tag{1}$$

where $\mathbf{x}$ is the $n \times 1$ plant state, $\mathbf{p}_m$ is a $l \times 1$ vector of plant and observation parameters, and $\mathbf{u}$ is the $m \times 1$ control vector. The equation may represent a "lumped-parameter" system, or it may be the approximation to an unsteady partial differential equation. Plant motions, controls, and disturbances typically are sensed in the $e_s \times 1$ output vector, $\mathbf{y}_s$:

$$\mathbf{y}_s(t) = \mathbf{h}_s[\mathbf{x}(t), \mathbf{p}_m(t), \mathbf{u}(t)] \tag{2}$$

Here, it is assumed that perfect output measurements are available and that the output views all elements of the state, i.e., $\mathbf{y}_s(t) = \mathbf{x}(t)$. The design objective is to specify a control law of the general form

$$\mathbf{u}(t) = \mathbf{c}[\mathbf{y}_s(t), \mathbf{p}_m(t), \mathbf{y}_c(t)] \tag{3}$$

that has two properties: it achieves mission goals, as expressed by the $e_c \times 1$ command input, $\mathbf{y}_c$, and it furnishes adequate stability and transient response, assuring that excursions from $\mathbf{y}_c$ caused by disturbance or measurement error are acceptably small and do not require excessive control use.

The command input, $\mathbf{y}_c$, can be viewed as some desirable combination of state and control elements, and its dimension, e_c, is less than or equal to the number of independent controls, m:

$$\mathbf{y}_c(t) = \mathbf{h}_c[\mathbf{x}_c(t), \mathbf{u}_c(t)] \tag{4}$$

Equation 4 could be the result of external trajectory planning -- for example, a prescribed path -- or it may be due to a loosely defined, subjective process such as the expression of a human operator's intent through command inputs. Hence, the control law can be formulated as

$$\mathbf{u}(t) = \mathbf{c}[\mathbf{x}(t), \mathbf{p}_m(t), \mathbf{y}_c(t)] \tag{5}$$

in terms of a functional, $\mathbf{c}[\bullet]$, that may contain functions of its arguments such as integrals and derivatives. For simplicity, the vector of parameters, $\mathbf{p}_m$, is assumed to be known without error.

When the control law depends on parameters or command inputs explicitly [66], an augmented state can be defined to include these additional elements, as will be shown in Section 4.1.1. Therefore, the control can be viewed as a function solely of the state, without loss of generality. Furthermore, eq. 5 always can be written as the sum of nominal and perturbed effects:

$$\mathbf{u}(t) = \mathbf{c}_0[\mathbf{x}_0(t)] + \Delta\mathbf{c}[\mathbf{x}(t)] = \mathbf{u}_0(t) + \Delta\mathbf{u}(t) \tag{6}$$

The anticipated nominal value of the state is $\mathbf{x}_0$, so the actual value can be written by adding the respective perturbation, $\Delta\mathbf{x}$:

$$\mathbf{x}(t) = \mathbf{x}_0(t) + \Delta\mathbf{x}(t) \tag{7}$$

The control law can be expressed conveniently in these terms,

$$\mathbf{u}(t) = \mathbf{c}_0[\mathbf{x}_0(t)] + \Delta\mathbf{c}[\mathbf{x}_0(t), \Delta\mathbf{x}(t)] \tag{8}$$

where, for sufficiently small state perturbations, the perturbed effect is linear in $\Delta\mathbf{x}$:

$$\Delta\mathbf{u}(t) = \Delta\mathbf{c}[\bullet] = \frac{\partial\mathbf{c}}{\partial\mathbf{x}}[\mathbf{x}_0(t)]\Delta\mathbf{x} = -\mathbf{C}\Delta\mathbf{x} \tag{9}$$

$\mathbf{C}$ contains the *m* gradients, or gains, of the control law evaluated at $\mathbf{x}_0(t)$, as explained in Section 2.2, and the minus sign is introduced for convention.

The design objectives are expressed by a scalar integral function of the state and control and by a scalar terminal cost,

$$\mathrm{J} = \varphi[\mathbf{x}(t_f)] + \int_{t_0}^{t_f} \mathrm{L}[\mathbf{x}(\tau), \mathbf{u}(\tau)]d\tau \tag{10}$$

The cost function, J, is to be minimized with respect to the control, **u**, subject to the dynamic constraint imposed by the model of the plant, eq. 1. In the nonlinear control system explored here, the minimizing control law is modeled by a neural network that is referred to as an *action network*. At any moment in time, $t_0 \le t \le t_f$, an optimal cost-to-go or value function, $V^*(t)$, corresponding to eq. 10 can be defined,

$$V^*\left[\mathbf{x}^*(t)\right] = \min_{\mathbf{u}(t)} \left\{ \varphi\left[\mathbf{x}^*(t_f)\right] - \int_{t_f}^{t} L\left[\mathbf{x}^*(\tau), \mathbf{u}(\tau)\right] d\tau \right\} \tag{11}$$

where $(\bullet)^*$ denotes the optimal solution. A *critic network* evaluates the action network performance by approximating the derivative of the corresponding cost-to-go with respect to the state:

$$\boldsymbol{\lambda}^*\left[\mathbf{x}^*(t)\right] \equiv \frac{\partial V^*\left[\mathbf{x}^*(t)\right]}{\partial \mathbf{x}^*(t)} \tag{12}$$

$\boldsymbol{\lambda}^*[\mathbf{x}^*(t)]$, or simply $\boldsymbol{\lambda}^*(t)$, provides an indirect measure of performance that is used to formulate an optimality criterion explicitly with respect to **u**, as will be explained in Section 2.4.

2.2 The Linear Quadratic Regulator

The goal of the first learning phase is to incorporate gain-scheduled designs into nonlinear neural networks. Gain scheduling is a design procedure that enjoys widespread usage in a variety of industrial applications. While it requires considerable ad-hoc practice, it also exploits linear control theory. Both the theory and heuristics of gain-scheduled designs will be incorporated into the neural network controllers, by means of the approach referred to as Classical/Neural Control Synthesis of Nonlinear Control

Systems (Section 2.3). This section reviews the LQR theory that is relevant to gain-scheduled controllers and, in particular, to the designs to be implemented in Section 4.1.

The basic assumption in gain scheduling is that the nonlinear system in eq. 1 has a parametrized family of equilibrium points,

$$\mathbf{0} = \mathbf{f}\left[\mathbf{x}_0(\mathbf{a}), \mathbf{p}_m(\mathbf{a}), \mathbf{u}_0(\mathbf{a})\right] \tag{13}$$

where, **a** is a *scheduling vector* of dynamically significant variables in the system. The set of equilibria, also referred to as operating points (or conditions), is denoted by *OP* and indexed by $\kappa = 1, 2, \ldots, p$. Linearized models of the plant can be obtained from the nonlinear dynamic equation (eq. 1) by holding **a** fixed, assuming small perturbations about corresponding equilibria, and ignoring time-varying effects:

$$\Delta\dot{\mathbf{x}}(t) = \mathbf{F}\Delta\mathbf{x}(t) + \mathbf{G}\Delta\mathbf{u}(t), \quad \Delta\mathbf{x}(t_0) \text{ given} \tag{14}$$

The optimization goals are expressed as a quadratic function of the state and control,

$$\mathrm{J} = \frac{1}{2}\int_0^{t_f}\left[\Delta\mathbf{x}^T(\tau)\mathbf{Q}\Delta\mathbf{x}(\tau) + 2\Delta\mathbf{x}^T(\tau)\mathbf{M}\Delta\mathbf{u}(\tau) + \Delta\mathbf{u}^T(\tau)\mathbf{R}\Delta\mathbf{u}(\tau)\right]d\tau \tag{15}$$

When the plant is subject to continuing disturbance inputs and t_f becomes infinite in the limit, the value of J may still be bounded by defining an average cost,

$$\mathrm{J}_A = \lim_{t_f \to \infty} \frac{\mathrm{J}}{t_f} \tag{16}$$

that has the same optimality conditions as J. As t_f approaches infinity, it is reasonable to let the terminal cost, $\varphi[\mathbf{x}(t_f)]$, equal zero [66].

When the system dynamic is linear (eq. 14) and the cost function is quadratic (eq. 15), an optimal closed-form solution can be obtained for the control perturbation $\Delta\mathbf{u}$. One approach to this Linear Quadratic (LQ) problem derives from the Calculus of Variations

[12]. In the general case, the Hamiltonian can be defined by adjoining the dynamic constraint (eq. 1) to the Lagrangian, L[•], by the adjoint vector $\boldsymbol{\lambda}$:

$$\mathrm{H}[\mathbf{x}(t),\mathbf{u}(t),\boldsymbol{\lambda}(t)] \equiv \mathrm{L}[\mathbf{x}(t),\mathbf{u}(t)] + \boldsymbol{\lambda}^T(t)\mathbf{f}[\mathbf{x}(t),\mathbf{p}_m(t),\mathbf{u}(t)] \tag{17}$$

Differentiating eq. 11 with respect to *t*, the following is found to hold on the optimal trajectory:

$$\begin{aligned} \frac{\partial \mathrm{V}^*}{\partial t}[\mathbf{x}^*(t)] &= -\mathrm{L}[\mathbf{x}^*(t),\mathbf{u}^*(t)] - \frac{\partial \mathrm{V}^*}{\partial \mathbf{x}^*}[\mathbf{x}^*(t)]\mathbf{f}[\mathbf{x}^*(t),\mathbf{p}_m(t),\mathbf{u}^*(t)] \\ &= -\min_{\mathbf{u}(t)} \mathrm{H}\left\{\mathbf{x}^*(t),\mathbf{u}(t),\frac{\partial \mathrm{V}^*}{\partial \mathbf{x}^*}[\mathbf{x}^*(t)]\right\} \end{aligned} \tag{18}$$

The optimal adjoint vector, $\boldsymbol{\lambda}^*$, is equal to the derivative of the optimal value function with respect to the state, $\partial \mathrm{V}^*/\partial \mathbf{x}^*$, which is approximated by the critic network (eq. 12) in the nonlinear control system. This partial differential equation is known as the Hamilton-Jacobi-Bellman (HJB) equation and is a sufficient condition for optimality that is used here to derive the LQ control law.

It can be shown [12] that the following constitutes an optimal value function, for the LQ problem of minimizing eq. 15 subject to the linear dynamic constraint in eq. 14:

$$\mathrm{V}^*[\Delta\mathbf{x}^*(t)] = \frac{1}{2}\Delta\mathbf{x}^{*T}(t)\mathbf{P}(t)\Delta\mathbf{x}^*(t) \tag{19}$$

$\mathbf{P}(t)$ is a positive definite symmetric matrix which, for a linear time-invariant (LTI) system (eq. 14), is guaranteed to approach a steady-state value, $\mathbf{P}$, in the limit $t_f \to \infty$ [6]. The HJB equation is expressed in terms of the perturbations $\Delta\mathbf{x}$ and $\Delta\mathbf{u}$. Its right side is found by differentiating the corresponding Hamiltonian with respect to $\Delta\mathbf{u}$ and setting it equal to zero to solve for $\Delta\mathbf{u}^*$. The remaining terms are found by differentiating eq. 19 with respect to $\Delta\mathbf{x}^*$ and *t*. Then, the HJB equation simplifies to,

$$\Delta \mathbf{x}^{*T}(t)\mathbf{M} + \Delta \mathbf{u}^{*T}(t)\mathbf{R} + \Delta \mathbf{x}^{*T}(t)\mathbf{PG} = \mathbf{0} \tag{20}$$

producing the LQ optimal control law:

$$\Delta \mathbf{u}^{*}(t) = -\mathbf{R}^{-1}\left[\mathbf{G}^{\mathbf{T}}\mathbf{P} + \mathbf{M}^{\mathbf{T}}\right]\Delta \mathbf{x}^{*}(t) = -\mathbf{C}\Delta \mathbf{x}^{*}(t) \tag{21}$$

Substituting in eq. 18 and canceling $\Delta \mathbf{x}^{*}$ from both sides leads to the Riccati equation,

$$\begin{aligned}\dot{\mathbf{P}}(t) = &-\left(\mathbf{F} - \mathbf{G}\mathbf{R}^{-1}\mathbf{M}^{T}\right)\mathbf{P}(t) - \mathbf{P}(t)\left(\mathbf{F} - \mathbf{G}\mathbf{R}^{-1}\mathbf{M}^{T}\right) \\ &+ \mathbf{P}(t)\mathbf{G}\mathbf{R}^{-1}\mathbf{G}^{T}\mathbf{P}(t) - \mathbf{Q} + \mathbf{M}\mathbf{R}^{-1}\mathbf{M}^{T}\end{aligned}, \quad \mathbf{P}(t_f) = \mathbf{0} \tag{22}$$

LTI control laws that satisfy desired engineering criteria [66-70] are designed for the family of linear systems $\{\mathbf{F}, \mathbf{G}\}_{\kappa = 1, \ldots, p}$, in order to provide a corresponding set of locally optimal gains and Riccati matrices $\{\mathbf{C}, \mathbf{P}\}_{\kappa = 1, \ldots, p}$. Typically, the Riccati equation (eq. 22) corresponding to the κ^{th} linear system is solved for the steady-state value, $\mathbf{P}^{\kappa}$, by setting its right-hand side equal to zero; then, $\mathbf{P}^{\kappa}$ is used in eq. 21 to solve for $\mathbf{C}^{\kappa}$. The family $\{\mathbf{C}, \mathbf{P}\}_{\kappa = 1, \ldots, p}$ is obtained by repeating the design at all equilibria in *OP*, i.e., for all linear systems indexed by κ. In gain scheduling, a global controller for the nonlinear system (eq. 1) is obtained by interpolating the local designs to intermediate operating regions through the scheduling vector **a**. For this reason, in past applications, the number of interpolating variables has been kept small. The novel approach introduced in the next section, incorporates the family of linear gains $\{\mathbf{C}, \mathbf{P}\}_{\kappa = 1, \ldots, p}$, or simply $\{\mathbf{C}, \mathbf{P}\}_{\kappa}$, into nonlinear neural networks that automatically interpolate the designs for any dimension of **a**. This affords an improvement with respect to earlier gain-scheduled controllers. More importantly, it provides an excellent initialization point for the on-line learning phase, whose foundations are presented in Section 2.4.

2.3 Classical/Neural Synthesis of Nonlinear Control Systems

The nonlinear control system is comprised of a critic network and an action network that approximates the global control based on the nonlinear plant and its model, as shown in Fig. 4. A key, novel observation is that the gradients of these networks must equal corresponding locally optimal gains obtained from the LQ solutions (Section 2.2). The linear controllers establish appropriate performance targets that, later, are used to define the nonlinear system architecture and initial parameters.

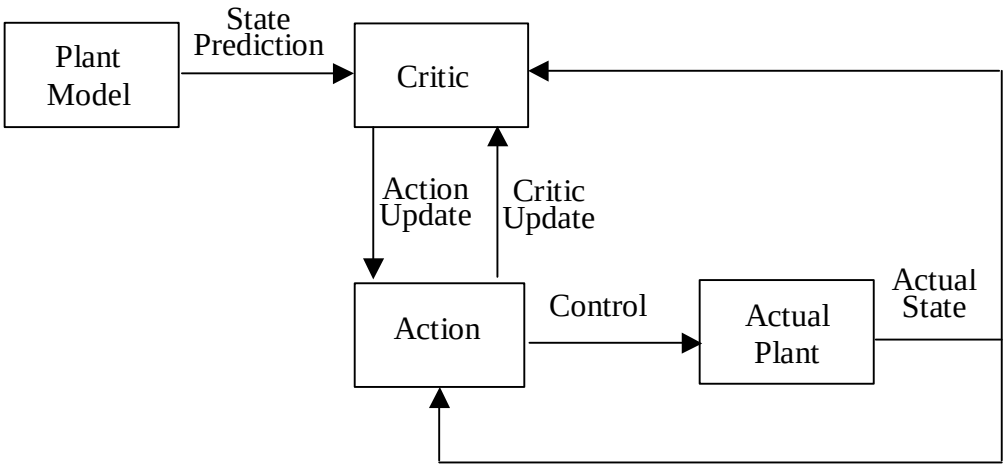

Figure 4. Dual heuristic programming adaptive critic control design.

An artificial neural network consists of a nonlinear mapping, denoted by **NN**, that performs a nonlinear transformation of a *q*-dimensional input, **p**, into an *r*-dimensional output, **z**:

$$\mathbf{z} = \mathbf{NN}(\mathbf{p}) \tag{23}$$

The network architecture and parameters characterize the nature of this transformation and can be determined based on input, output, and derivative information pertaining to the function to be approximated. The common denominator among all neural

architectures is the highly parallel and distributed computations they perform. As anticipated in Section 2.1, the action network approximates the optimal control law (eq. 8). The critic network evaluates the action network performance by approximating the derivative of the optimal value function with respect to the state (eq. 12):

$$\begin{aligned} \mathbf{u}(t) &= \mathbf{NN}_A[\mathbf{p}(t)] \equiv \mathbf{z}_A(t) \\ \boldsymbol{\lambda}(t) &= \mathbf{NN}_C[\mathbf{p}(t)] \equiv \mathbf{z}_C(t) \end{aligned} \tag{24}$$

The input to both networks includes the dynamically significant auxiliary inputs, **a**, as they may or may not be contained explicitly by **x**, i.e., $\mathbf{p}(t) = [\mathbf{x}(t)^T\ \mathbf{a}(t)^T]^T$.

The objective of the first learning phase, or pre-training, is to incorporate the set of locally optimal gains $\{\mathbf{C}, \mathbf{P}\}_{\kappa = 1, \ldots, p}$, typically used for gain scheduling, into the nonlinear networks. The first step towards accomplishing this objective consists of identifying appropriate performance requirements to be satisfied by the network parameters. For every point in *OP*, the gradient of the action network can be found by differentiating eq. 6 and eq. 9 with respect to **x** and $\Delta\mathbf{x}$, respectively:

$$\left.\frac{\partial \mathbf{z}_A(t)}{\partial \mathbf{x}(t)}\right|_{\mathbf{p}(t)=\left[\mathbf{x}_0^*(t)^T\ \ \mathbf{a}_\kappa^T\right]^T} = \left.\frac{\partial \mathbf{u}(t)}{\partial \mathbf{x}(t)}\right|_{\mathbf{x}(t)=\mathbf{x}_0^*(t),\ \mathbf{a}(t)=\mathbf{a}_\kappa} = \left.\frac{\partial \Delta\mathbf{u}^*(t)}{\partial \Delta\mathbf{x}^*(t)}\right|_{\Delta\mathbf{x}^*(t)=\mathbf{0},\ \mathbf{a}(t)=\mathbf{a}_\kappa} \tag{25}$$

Hence, at the κ^{th} operating point, the action network gradient must equal the κ^{th} LQ gain,

$$\left.\frac{\partial \mathbf{z}_A(t)}{\partial \mathbf{x}(t)}\right|_\kappa = -\mathbf{C}^\kappa \tag{26}$$

where $\mathbf{C}^\kappa$ is known from eq. 21, and the subscript indicates at which operating conditions the derivative is being evaluated.

In infinite-horizon problems, the structure of the value function is independent of time; therefore, a single time-invariant critic network can be used to approximate $\boldsymbol{\lambda}^*(t)$ (eq. 12).

The LQ optimal value function, eq. 19, can be differentiated twice with respect to the state to seek the derivative of the critic output with respect to the input **x**,

$$\left.\frac{\partial \mathbf{z}_C(t)}{\partial \mathbf{x}(t)}\right|_{\mathbf{p}(t)=\left[\mathbf{x}_0^*(t)^T\ \ \mathbf{a}_\kappa^T\right]^T} = \left.\frac{\partial \boldsymbol{\lambda}(t)}{\partial \mathbf{x}(t)}\right|_{\mathbf{x}(t)=\mathbf{x}_0^*(t),\ \mathbf{a}(t)=\mathbf{a}_\kappa} = \left.\frac{\partial^2 \mathrm{V}^*\left[\Delta\mathbf{x}^*(t)\right]}{\partial \Delta\mathbf{x}^*(t)^2}\right|_{\Delta\mathbf{x}^*(t)=\mathbf{0},\ \mathbf{a}(t)=\mathbf{a}_\kappa} \tag{27}$$

revealing that, at the κ^{th} operating condition, the critic network gradient equals the κ^{th} LQ Riccati matrix:

$$\left.\frac{\partial \mathbf{z}_C(t)}{\partial \mathbf{x}(t)}\right|_{\kappa} = \mathbf{P}^{\kappa} \tag{28}$$

$\mathbf{P}^{\kappa}$ is known at all conditions considered in the linear design and, thus, is used to pre-train the critic network.

Because the gains $\mathbf{C}^{\kappa}$ and $\mathbf{P}^{\kappa}$ can be designed for a family of equilibria *OP*, the gradient, $\partial\mathbf{z}[\mathbf{p}(t)]/\partial\mathbf{x}(t)$, of the action and critic networks is known at all points $\kappa \in OP$. In addition, the following condition applies to the networks' input/output relationship:

$$\mathbf{z}\left[\mathbf{x}(t),\mathbf{a}(t)\right]_{\mathbf{x}(t)=\mathbf{x}_0^*(t),\ \mathbf{a}(t)=\mathbf{a}_\kappa} \equiv \mathbf{z}\left[\mathbf{x}(t),\mathbf{a}(t)\right]_{\kappa} = \mathbf{0} \tag{29}$$

The requirements in eq. 26, 28, and 29 state that the network control system must be characterized by the same local performance as the LQ controllers at all conditions in *OP*. The neural parameters that satisfy these performance targets will be obtained using the novel algebraic initialization technique described in Section 3.1.1. Chapter 4 will demonstrate the pre-training procedure for a representative control structure, the proportional-integral controller. Thanks to their generalization abilities, the neural networks interpolate the sampled data learned over *OP* to cover the intermediate regions not considered by the linear designs. This new pre-training approach constitutes an

excellent starting point for the on-line learning phase, while providing the stability, performance, and robustness characteristics of the linear designs for small perturbations.

2.4 Dual Heuristic Programming Adaptive Critics

During the on-line learning phase, the pre-trained action and critic networks are updated over time to more closely approximate the globally optimal control law, with the critic evaluating the action network performance. The adaptation improves control response for those conditions that were missed or unaccounted for by the linear gain-scheduled designs, such as large, coupled motions, full-envelope maneuvers, and unforeseen conditions. The adaptation (outlined in Fig. 4) takes place while the plant is operating over the entire range of state and command-input elements, $\{\mathbf{x}(\mathbf{y}_c), \mathbf{y}_c\}$, or some suitably dense set in the space denoted by *OR*. The actual plant state, $\mathbf{x}$, and the command input, $\mathbf{y}_c$, are fed to the controller on-line and are unknown prior to operation. The on-line logic is implemented in discrete time through an incremental optimization scheme, dual heuristic programming [18-20], which is based on the recurrence relation of dynamic programming reviewed in this section.

During each time interval $\Delta t = t_{k+1} - t_k$, the action and critic networks are adapted to more closely approximate the optimal control law and value function derivatives, respectively. The recurrence relation provides for adaptation criteria that, over time, guarantee convergence to the optimal solution. Because the on-line adaptation utilizes the actual state of the plant (eq. 1), the control system performance is improved with respect to the initialized neural network controller. Prior knowledge, incorporated during the pre-training phase, is retained during on-line learning thanks to the incremental training algorithms that will be introduced in Section 3.2 and 3.3.

The dual heuristic programming adaptation criteria are derived from the recurrence relation by discretizing the infinite-horizon optimal control problem. With restriction to piecewise-constant inputs and constant time intervals, the discrete or sampled-data model equivalent of eq. 1 can be expressed as:

$$\mathbf{x}(t_{k+1}) = \mathbf{f}_{SD}\left[\mathbf{x}(t_k), \mathbf{p}_m(t_k), \mathbf{u}(t_k)\right], \;\; \mathbf{x}(t_0) \text{ given} \tag{30}$$

The same metric optimized during the initialization phase, eq. 15, is optimized in the on-line phase, affording a systematic approach to the control design. The corresponding cost function can be written as the sum of incremental costs accrued during the time intervals:

$$\mathrm{J}_{0,N} = \sum_{k=0}^{N-1} \mathrm{L}_{SD}\left[\mathbf{x}(t_k), \mathbf{u}(t_k)\right], \;\; N\Delta t \to \infty \tag{31}$$

The cost of operation from the k^{th}-instant, t_k, to the final time, t_N, i.e., $\mathrm{J}_{k,N}$, corresponds to the *value function* $\mathrm{V}(t_k)$ or cost-to-go at t_k and can be written as,

$$\mathrm{V}(t_k) = \mathrm{L}_{SD}\left[\mathbf{x}(t_k), \mathbf{u}(t_k)\right] + \mathrm{V}(t_{k+1})\left[\mathbf{x}(t_{k+1}), \mathbf{u}(t_{k+1}), \ldots, \mathbf{u}(t_{N-1})\right] \tag{32}$$

since, from eq. 30, all future values of the state, $\mathbf{x}(t_{k+2})$, …, $\mathbf{x}(t_N)$, depend on $\mathbf{x}(t_{k+1})$ and on the future control history, $\mathbf{u}(t_k)$, …, $\mathbf{u}(t_{N-1})$. Similarly, $\mathbf{x}(t_{k+1})$ depends on $\mathbf{x}(t_k)$ and $\mathbf{u}(t_k)$. Therefore, the optimal cost for the $(N-k)$-stage policy is found by minimizing the following functional with respect to the control history,

$$\mathrm{V}^*\left[\mathbf{x}^*(t_k)\right] \equiv \min_{\mathbf{u}(t_k), \ldots, \mathbf{u}(t_{N-1})} \left\{\mathrm{V}\left[\mathbf{x}^*(t_k), \mathbf{u}(t_k), \ldots, \mathbf{u}(t_{N-1})\right]\right\} \tag{33}$$

By the Principle of Optimality [11], if a policy is optimal over $(N-k)$ stages, whatever the initial (k^{th}) state and decision are, the remaining decisions also must constitute an optimal policy with regard to the state resulting from the first decision, i.e., $\mathbf{x}^*(t_{k+1})$:

$$V^*\left[\mathbf{x}^*(t_k)\right] = \min_{\mathbf{u}(t_k)} \left\{ L_{SD}\left[\mathbf{x}^*(t_k), \mathbf{u}(t_k)\right] + V^*\left[\mathbf{x}^*(t_{k+1})\right] \right\} \tag{34}$$

This equation constitutes the recurrence relation of dynamic programming.

This recurrence relation can be used backwards in time, starting from the final time t_N, to obtain an approximate solution to the exact optimal control history [12]. This approach is computationally intensive and not suitable for on-line solution [11]. In approximate dynamic programming methods, Howard's form of the recurrence relation [16] is used to approximate the minimum value function on-line,

$$V[\mathbf{x}(t_k)] = L_{SD}[\mathbf{x}(t_k), \mathbf{u}(t_k)] + V[\mathbf{x}(t_{k+1})] \tag{35}$$

where $V[\mathbf{x}(t_{k+1})]$ is necessarily a predicted value. The control $\mathbf{u}(t_k)$ is defined as the function of $\mathbf{x}(t_k)$ that minimizes the right-hand side of eq. 35 for any $\mathbf{x}(t_k)$. Howard shows [16] that when the function $V[\mathbf{x}(t_k)]$ is calculated from eq. 35 based on the current control, and the control is adjusted to minimize this approximation to the optimal value function, the method iteratively converges to an optimal strategy. The same is true in the presence of random disturbances as long as expected values of the cost-to-go are considered in eq. 35. For simplicity, the asterisks are omitted, and convergence to optimality is implied in the remainder of the thesis.

At time t_k, the control strategy for which the value function (eq. 35) is stationary satisfies the following optimality condition:

$$\frac{\partial V[\mathbf{x}(t_k)]}{\partial \mathbf{u}(t_k)} = \frac{\partial L_{SD}[\mathbf{x}(t_k), \mathbf{u}(t_k)]}{\partial \mathbf{u}(t_k)} + \boldsymbol{\lambda}(t_{k+1}) \frac{\partial \mathbf{x}(t_{k+1})}{\partial \mathbf{u}(t_k)} = 0 \tag{36}$$

Equation 35 is differentiated with respect to the state to obtain a recurrence relation for the DHP critic, which approximates the functional $\boldsymbol{\lambda}(t)$ in eq. 12:

$$\boldsymbol{\lambda}(t_k) \equiv \frac{\partial \mathrm{V}[\mathbf{x}(t_k)]}{\partial \mathbf{x}(t_k)} = \frac{\partial \mathrm{L}_{SD}[\mathbf{x}(t_k),\mathbf{u}(t_k)]}{\partial \mathbf{x}(t_k)} + \frac{\partial \mathrm{L}_{SD}[\mathbf{x}(t_k),\mathbf{u}(t_k)]}{\partial \mathbf{u}(t_k)} \frac{\partial \mathbf{u}[\mathbf{x}(t_k)]}{\partial \mathbf{x}(t_k)} + \boldsymbol{\lambda}(t_{k+1}) \frac{\partial \mathbf{x}(t_{k+1})}{\partial \mathbf{x}(t_k)} + \boldsymbol{\lambda}(t_{k+1}) \frac{\partial \mathbf{x}(t_{k+1})}{\partial \mathbf{u}(t_k)} \frac{\partial \mathbf{u}[\mathbf{x}(t_k)]}{\partial \mathbf{x}(t_k)} \tag{37}$$

Once the prediction of the state, $\mathbf{x}(t_{k+1})$, is known from the model of the plant (eq. 30), the critic can be used to compute $\boldsymbol{\lambda}(t_{k+1})$ in eq. 36. Using the critic to approximate the value function derivatives instead of the value function alone improves speed by giving an indication of how individual control elements influence the overall cost. Equations 36 and 37 can be viewed as criteria for the action and critic on-line adaptation. Together they lead the action network to converge to the optimal control functional. The numerical schemes and the overall implementation that ensure the consecutive realization of these requirements over time are discussed in the remaining chapters.

2.5 Chapter Summary

A two-phase approach is proposed for approximating the nonlinear optimal control law of an infinite-horizon problem on line, subject to the actual dynamics of the plant. During the first phase, the initial specification of the control law is determined from classical linear control theory, and is realized in the form of a nonlinear neural controller. The nonlinear controller is motivated by a corresponding linear structure, and is defined by the architecture and parameters of a network of neural networks. A key, novel observation is that the gradient of these networks must equal corresponding gain matrices at selected operating points. During the second phase, the neural parameters are updated on-line, through a dual heuristic programming adaptive critic architecture. The recurrence relation of dynamic programming can be used to derive adaptation criteria that guarantee convergence to the optimal solution, over time.

Chapter 3

Advancements in Neural Network Learning Theory: Algebraic Training and Modified Resilient Backpropagation Techniques

Computational neural networks are massively parallel computational paradigms inspired by biological neural formations. They are used in a variety of applications because they can learn by example and provide excellent universal function approximation for multivariate input/output spaces. Particularly, they afford a general approach for modeling, identification, and control of nonlinear systems that shows great promise, as neural networks can potentially adjust to complex situations on line thanks to their generalizing and adaptive capabilities.

Considerable effort has gone into the mathematical investigation of neural networks' approximation properties [33-35, 39, 71]. Whereas these results appear attractive, they provide little insight into practical, key questions such as, "What architecture should be used", and "How many nodes are required in each layer"? This chapter describes a novel algebraic training approach that provides a general framework for answering these questions as well as learning theory advancements that improve upon a number of approximation characteristics. This method is used to derive training algorithms that can achieve either exact or approximate matching of noise-free data with considerable computational savings and better interpolation properties than classical, backpropagation-based algorithms. Some of the results developed here were announced in [36, 37].

3.1 Algebraic Training

The typical search criterion, used in virtually all supervised learning algorithms, consists of minimizing some measure of the error between the desired input/output (and/or derivative) information and the actual network's performance. The approach taken here consists of formulating training as a root-finding problem whose solution achieves exact fitting of the training set. Although related in principle, the problems of minimization and multidimensional root finding are substantially different in practice. The problem of minimizing some form of neural network error appears computationally more tractable, but it may not solve the problem of exact fitting. On the other hand, solving the corresponding nonlinear equations appears virtually impossible for any decent-sized network. So what is the reason behind attempting to solve a harder version of the same problem? As it happens, these nonlinear equations can easily be transformed into linear equations, bringing about much simplified training methods and affording deep insight into neural approximators.

A set of nonlinear equations to be solved for the neural network adjustable parameters is obtained by imposing the requirements derived from the training set on the neural network input/output and gradient equations. The goal is to approximate a smooth scalar function of q inputs, denoted by $h : \Re^q \rightarrow \Re$, using a simply connected sigmoidal network of the type shown in Fig. 5. The approach also can be extended to include vector-output functions, as will be demonstrated in later chapters. Typically, the function is not known analytically, but a set of input/output samples $\{\mathbf{y}^k, u^k\}_{k = 1, \ldots, p}$ can be generated such that $u^k = h(\mathbf{y}^k)$, for all values of k. Using derivative information during

training can improve upon the network's generalization properties [72]. Therefore, when the partial derivatives of the function $h(\bullet)$ are known with respect to e of its inputs,

$$\mathbf{c}^k \equiv \left[\left.\frac{\partial u}{\partial y_1}\right|_{\mathbf{y}^k} \cdots \left.\frac{\partial u}{\partial y_e}\right|_{\mathbf{y}^k} \right]^T, \quad e \le q \tag{38}$$

they also are incorporated in the training set: $\{\mathbf{y}^k, u^k, \mathbf{c}^k\}_{k = 1, \ldots, p}$.

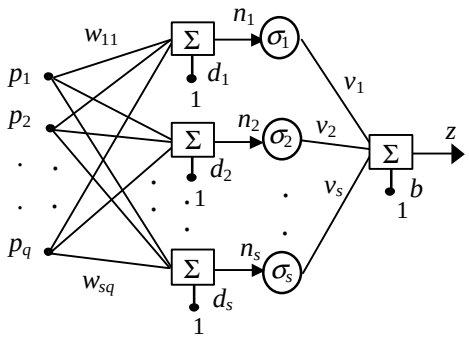

Figure 5. Sample scalar-output network with q inputs and s nodes in the hidden layer.

The output of the network is computed as the nonlinear transformation of the weighted sum of the input, **p**, and a bias **d**, plus an output bias, b:

$$z = \mathbf{v}^T \boldsymbol{\sigma}[\mathbf{W}\mathbf{p} + \mathbf{d}] + b \tag{39}$$

$\boldsymbol{\sigma}[\bullet]$ is composed of sigmoidal functions, such as $\sigma(n) = (e^n - 1)/(e^n + 1)$, evaluated at all input-to-node variables, n_i, with $i = 1, \ldots, s$,

$$\boldsymbol{\sigma}[\mathbf{n}] \equiv [\sigma(n_1) \quad \cdots \quad \sigma(n_s)]^T \tag{40}$$

where:

$$\mathbf{n} = \mathbf{W}\mathbf{p} + \mathbf{d} \tag{41}$$

W and **v** contain the input and output weights, respectively. Together with **d** and b they constitute the adjustable parameters of the network. The order of differentiability of eq.

39 is the same as that of the activation function, $\sigma(\bullet)$. Given that sigmoid functions are infinitely differentiable, the derivative of the network output with respect to its inputs is:

$$\frac{\partial z}{\partial p_j} = \sum_{i=1}^{s} \frac{\partial z}{\partial n_i} \frac{\partial n_i}{\partial p_j} = \sum_{i=1}^{s} v_i \sigma'(n_i) w_{ij} \ , \ j = 1, \ldots, q \tag{42}$$

$\sigma'(\bullet)$ denotes the derivative of the sigmoidal function with respect to its scalar input. w_{ij} denotes the element in the i^{th}-row and the j^{th}-column of the matrix **W**, and it represents the connection weight between the j^{th}-input and the i^{th}-node of the network.

The computational neural network achieves exact fitting of the input/output training set, $\{\mathbf{y}^k, u^k\}_{k = 1, \ldots, p}$, when given the input $\mathbf{y}^k$ it produces u^k as the output, for all k:

$$z(\mathbf{y}^k) = u^k \tag{43}$$

This is equivalent to stating that the neural adjustable parameters must satisfy the following nonlinear equations,

$$u^k = \mathbf{v}^T \boldsymbol{\sigma}[\mathbf{W}\mathbf{y}^k + \mathbf{d}] + b \, , \ k = 1, \ldots, p \tag{44}$$

that are referred to as *output weight equations*. When all the known output elements from the training set are grouped in a vector,

$$\mathbf{u} = [u^1 \ \cdots \ u^p]^T \tag{45}$$

eq. 44 can be written using matrix notation:

$$\mathbf{u} = \mathbf{S}\mathbf{v} + \mathbf{b} \tag{46}$$

b is a s-dimensional vector composed of the scalar output bias, b. **S** is a matrix of sigmoidal functions evaluated at input-to-node values, n_i^k, each representing the magnitude of the input-to-node variable n_i to the i^{th} node, for the training pair k:

$$\mathbf{S} \equiv \begin{bmatrix} \sigma\left(n_1^1\right) & \sigma\left(n_2^1\right) & \dots & \sigma\left(n_s^1\right) \\ \sigma\left(n_1^2\right) & \sigma\left(n_2^2\right) & \dots & \sigma\left(n_s^2\right) \\ \vdots & \vdots & \ddots & \vdots \\ \sigma\left(n_1^p\right) & \sigma\left(n_2^p\right) & \dots & \sigma\left(n_s^p\right) \end{bmatrix} \tag{47}$$

The nonlinearity of these equations arises purely from the implicit dependence of the nonlinear function's argument on the input weights, **W**, and bias, **d**.

The known gradients, $\mathbf{c}^k$, correspond to the partial derivatives of the neural network's output with its inputs evaluated at the training pair *k*. Exact matching of the gradient training set, $\{\mathbf{y}^k, \mathbf{c}^k\}_{k = 1, \dots, p}$, is achieved when the neural network gradient corresponding to the input $\mathbf{y}^k$ equals $\mathbf{c}^k$, for all *k*:

$$\left.\frac{\partial z}{\partial p_j}\right|_{\mathbf{y}^k} = c_j^k, \quad j = 1, \dots, e \tag{48}$$

Therefore, the neural network adjustable parameters must satisfy the following *gradient weight equations*,

$$\mathbf{c}^k = \mathbf{W}(\bullet,\ 1 \div e)^T \left\{\mathbf{v} \otimes \boldsymbol{\sigma}'\left[\mathbf{n}^k\right]\right\}, \quad k = 1, \dots, p \tag{49}$$

where the symbol "$\otimes$" denotes element-wise vector multiplication, and $\mathbf{W}(\bullet, 1 \div e)$ represents the first *e* columns of the input-weight matrix. *Input-to-node weight equations* are obtained from the arguments of the nonlinear functions in eq. 46 and 49:

$$\mathbf{n}^k = \mathbf{W}\mathbf{y}^k + \mathbf{d}, \quad k = 1, \dots, p \tag{50}$$

$\boldsymbol{\sigma}'[\bullet]$ is a vector-valued function whose elements consist of the function $\sigma'(\bullet)$ evaluated component-wise at each element of its vector argument:

$$\boldsymbol{\sigma}'[\mathbf{n}] \equiv \left[\sigma'(n_1) \ \dots \ \sigma'(n_s)\right]^T \tag{51}$$

Equation 49 can be written as,

$$\mathbf{c}^k = \left[\mathbf{B}^k \mathbf{W}(\bullet,\ 1 \div e)\right]^T, \tag{52}$$

with the matrix,

$$\mathbf{B}^k \equiv [v_1 \sigma'(n_1^k)\ \ v_2 \sigma'(n_2^k)\ \ \ldots\ \ v_s \sigma'(n_s^k)] \tag{53}$$

explicitly containing only sigmoidal functions and output weights.

The full training set $\{\mathbf{y}^k, u^k, \mathbf{c}^k\}_{k=1,\ldots,p}$ is matched exactly when the output and the gradient weight equations are solved simultaneously for the neural network's adjustable parameters. If the derivative information, $\mathbf{c}^k$, is not available, the output equations are solved and eq. 52 is ignored, conversely if the output information, $\mathbf{u}^k$, is not available eq. 46 is ignored. Algebraic training is based on the key observation that if all input-to-node values, n_i^k, are known, then the nonlinear transcendental weight equations, eq. 46 and 52, become algebraic and linear. Based on this assumption, **S** is a known matrix and eq. 46 can be solved for **v**; then, all of the $\mathbf{B}^k$ matrices also are known and eq. 52 can be solved for $\mathbf{W}(\bullet, 1 \div e)$. The following sections show four techniques based on this approach whose effectiveness is demonstrated throughout the remaining chapters.

3.1.1 Exact Gradient-based Solution

A case that is particularly relevant for control applications is that in which some of the neural network gradients are known. The neural network inputs can be divided into e inputs for which the gradients are known, **x**, and into $(q - e)$ inputs for which the gradients are not known, **a**. Each training input $\mathbf{y}^k$ can be partitioned into $\mathbf{y}^k = [\mathbf{x}^{k^T} \mid \mathbf{a}^{k^T}]^T$. The input-weight matrix also is partitioned into weights corresponding to **x**, $\mathbf{W_x}$, and weights corresponding to **a**, $\mathbf{W_a}$, as follows:

$$\mathbf{W} = [\mathbf{W_x} \mid \mathbf{W_a}] \tag{54}$$

In this case, the gradients, $\mathbf{c}^k$, are known when $\mathbf{x}^k = \mathbf{0}$, i.e., $\mathbf{y}^k = [\mathbf{0} \mid \mathbf{a}^{k\,T}]^T$, and $u^k = 0$, for all k. The resulting training set $\{[\mathbf{0} \mid \mathbf{a}^{k\,T}]^T, 0, \mathbf{c}^k\}_{k = 1, \ldots, p}$ is a special case for which the weight equations always exhibit an exact solution.

The output weight equations, eq. 44, take the form,

$$0 = \mathbf{v}^T \boldsymbol{\sigma}[\mathbf{W_a} \mathbf{y}^k + \mathbf{d}] + b, \;\; k = 1, \ldots, p \tag{55}$$

and are independent of the $\mathbf{W_x}$ input weights, because $\mathbf{x}^k$ equals zero in all p training triads. The gradient weight equations, eq. 49, depend on the $\mathbf{W_a}$ input weights only implicitly,

$$\mathbf{c}^k = \mathbf{W_x}^T \left\{ \mathbf{v} \otimes \boldsymbol{\sigma}' \left[\mathbf{W_a} \mathbf{y}^k + \mathbf{d} \right] \right\}, \;\; k = 1, \ldots, p \tag{56}$$

where eq. 50 simplifies to:

$$\mathbf{n}^k = \mathbf{W_a} \mathbf{a}^k + \mathbf{d}, \;\; k = 1, \ldots, p \tag{57}$$

Therefore, if all sigmoidal arguments (or input-to-node values) in eq. 57 are known, the system in eq. 55 becomes linear,

$$\mathbf{b} = -\mathbf{S}\mathbf{v} \tag{58}$$

and can be solved for the output weights $\mathbf{v}$. $\mathbf{b}$ and $\mathbf{S}$ are defined as in eq. 47. When the number of nodes is chosen equal to the number of training pairs, $s = p$, the matrix $\mathbf{S}$ is square; provided it also is non-singular, eq. 55 admits a unique solution for $\mathbf{v}$.

Once the output weights are known, $\mathbf{v}$ can be treated as a constant, and eq. 56 also can be treated as linear:

$$\boldsymbol{\varsigma} = \mathbf{X}\mathbf{w_x} \tag{59}$$

In this system of equations, the unknowns consist of the input weights associated with the state deviations that, for convenience, have been reorganized in the vector $\mathbf{w_x}$,

$\mathbf{w_x} \equiv \mathrm{Vec}(\mathbf{W_x})$. "Vec" indicates *Vec Operation*, which consists of columnwise reorganization of matrix elements into a vector [73]. The vector $\boldsymbol{\varsigma}$ is obtained from the known gradients in the training set,

$$\boldsymbol{\varsigma} \equiv \left[\mathbf{c}^{1^T} \mid \ldots \mid \mathbf{c}^{p^T} \right]^T \tag{60}$$

$\mathbf{X}$ denotes an $ep \times es$ sparse matrix composed of p block-diagonal sub-matrices of dimensions $e \times es$:

$$\mathbf{X} \equiv \left[\begin{array}{cccc} \mathbf{B}^1 & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{B}^1 & \mathbf{0} & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{B}^1 \\ \hline & & \vdots & \\ \hline \mathbf{B}^p & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{B}^p & \mathbf{0} & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{B}^p \end{array}\right] \begin{array}{l} \left.\begin{array}{l} \\ \\ \\ \\ \end{array}\right\} e-\text{rows} \\ \vdots \\ \left.\begin{array}{l} \\ \\ \\ \\ \end{array}\right\} e-\text{rows} \end{array} \tag{61}$$

Every block $\mathbf{B}^k$, defined in eq. 53, is known when $\mathbf{v}$ and all input-to-node values, n_i^k, are known. Furthermore, when $s = p$, $\mathbf{X}$ is a square matrix, and the system in eq. 59 can be solved uniquely for $\mathbf{w_x}$, provided $\mathbf{X}$ also is non-singular.

Finally, the third set of linear equations is obtained from the assumption that the input-to-node values are known. For convenience, all n_i^k values are reorganized into the following array:

$$\boldsymbol{\eta} \equiv \left[\mathbf{n}^{1^T} \mid \ldots \mid \mathbf{n}^{p^T} \right]^T \tag{62}$$

With $s = p$, $\boldsymbol{\eta}$ becomes a known p^2-dimensional column vector. The p^2 linear equations in eq. 57 can be written as:

$$\boldsymbol{\eta} = \mathbf{A}\mathbf{w_a} \tag{63}$$

$\mathbf{A}$ is a $ps \times (q - e + 1)s$ matrix that is computed from all $\mathbf{a}^k$-input vectors in the training set. Each of these vectors contains $(q - e)$ elements and the superscript indicates at which training pair each element has been evaluated:

$$\mathbf{A} \equiv \left[\begin{array}{c|c|c} a_1^1\mathbf{I}_s & \cdots & a_{(q-e)}^1\mathbf{I}_s \quad \mathbf{I}_s \\ a_1^2\mathbf{I}_s & & a_{(q-e)}^2\mathbf{I}_s \quad \mathbf{I}_s \\ \vdots & & \vdots \quad \vdots \\ a_1^p\mathbf{I}_s & & a_{(q-e)}^p\mathbf{I}_s \quad \mathbf{I}_s \end{array}\right] \tag{64}$$

The only parameters that are left unknown in eq. 63 are the input weights associated with the scheduling variable, $\mathbf{W_a}$, and the input bias, $\mathbf{d}$. These are conveniently contained in the vector $\mathbf{w_a}$ that corresponds to the following rearrangement: $\mathbf{w_a} \equiv \text{Vec}[\mathbf{W_a} \mid \mathbf{d}]$.

The third linear system, eq. 63, is the first to be solved, since the input-to-node values are needed in order to obtain eq. 58 and eq. 59 from the nonlinear weight equations. In principle, one could assign arbitrary values to the elements of $\boldsymbol{\eta}$ and then invert $\mathbf{A}$ to obtain $\mathbf{w_a}$. However, this system usually is *overdetermined* and only a least-squares estimate of $\mathbf{w_a}$ can be obtained through the *left-pseudoinverse* of $\mathbf{A}$, $\mathbf{A}^{PI}$ [74]. If the redundant information contained in eq. 63 is consistent, $\mathbf{A}^{PI}$ produces the exact value of $\mathbf{w_a}$. Otherwise, the left-pseudoinverse solution provides the estimate that minimizes the *mean-squared error* of $\mathbf{w_a}$. A second alternative consists of choosing $\mathbf{w_a}$ and solving for $\boldsymbol{\eta}$. This is equivalent to setting up consistent equations, such that an exact solution always can be obtained for $\mathbf{w_a}$. It can be shown that the input-to-node values determine the nature of $\mathbf{S}$ and $\mathbf{X}$, for repetitive values in $\boldsymbol{\eta}$ will render their determinants zero. The following algorithm determines an effective distribution for the elements in $\boldsymbol{\eta}$ so that the weight equations can be solved for the neural parameters in one step.

The solution order of the above linear equations is key. Using all the training set data and choosing a number of nodes, *s*, equal to the number of training pairs, *p*, $\mathbf{A}$ and $\boldsymbol{\varsigma}$ are determined from eq. 64 and 60, respectively. Next, the vector $\boldsymbol{\eta}$ is determined such that the matrix $\mathbf{S}$ is well-conditioned, i.e., with condition number less than $\varepsilon^{-1/2}$, where ε is the smallest positive number such that $\varepsilon + 1 > 1$ on the computing machine. A strategy that produces a well-conditioned $\mathbf{S}$, with probability one, consists of generating $\boldsymbol{\eta}$ according to the following rule,

$$n_i^k = \begin{cases} r_i^k, & if \quad i \neq k \\ 0, & if \quad i = k \end{cases} \tag{65}$$

where r_i^k is chosen from a normal distribution with mean zero and variance one obtained using a random number generator with a single seed. When $i = k$, n_i^k is assigned a zero value so that each sigmoid is centered at one of the training pairs. This is equivalent to distributing the sigmoids across the input space as accomplished by the Nguyen-Widrow initialization algorithm [75].

Equation 63 is solved for $\mathbf{w_a}$ using the *left pseudoinverse* $\mathbf{A}^{PI}$:

$$\hat{\mathbf{w}}_\mathbf{a} = \mathbf{A}^{PI}\boldsymbol{\eta} \tag{66}$$

$\hat{\mathbf{w}}_\mathbf{a}$ is the best approximation to the solution, as this overdetermined system is not likely to have a solution. When this value for $\mathbf{w_a}$ is substituted back in eq. 63, only an estimate to the chosen values (eq. 65) is obtained for $\boldsymbol{\eta}$:

$$\hat{\boldsymbol{\eta}} = \mathbf{A}\hat{\mathbf{w}}_\mathbf{a} \tag{67}$$

However, nothing prevents us from using this value for $\boldsymbol{\eta}$. Although overdetermined, this system has a unique solution because the equations are now consistent.

$\hat{\boldsymbol{\eta}}$ is computed on the basis of eq. 40; therefore, the sigmoids are very nearly centered. While it is desirable for each sigmoid to be centered for a given input, $\mathbf{y}^k$, the same sigmoid should be close to saturation for any other known input in order to prevent ill-conditioning of **S**. Considering that the sigmoids come close to being saturated for an input whose absolute value is greater than 5, it is found desirable for the input-to-node values in $\boldsymbol{\eta}$ to have variance of about 10. A factor *f* can be obtained for this purpose from the absolute value of the largest element in $\hat{\boldsymbol{\eta}}$; then the final values for $\boldsymbol{\eta}$ and $\mathbf{w_a}$ can be obtained by multiplying both sides of eq. 67 by *f*:

$$\begin{aligned} \boldsymbol{\eta} &= f\hat{\boldsymbol{\eta}} \\ \mathbf{w_a} &= f\hat{\mathbf{w}}_\mathbf{a} \end{aligned} \tag{68}$$

Subsequently, the matrix **S** can be computed from $\boldsymbol{\eta}$, and the system in eq. 58 can be solved for **v**. With the knowledge of **v** and $\boldsymbol{\eta}$, the matrix **X** can be formed as stated in eq. 61, and the system eq. 59 can be solved for $\mathbf{w}_{\tilde{\mathbf{x}}}$. The matrices **S** and **X** in eq. 58 and 59 are consistently well-conditioned, rendering the solution of these linear systems by matrix inversion straight-forward as well as highly accurate. Thus, both output and gradient weight equations, originally in the form of eq. 55 and 56, are solved exactly for the network's parameters in a non-iterative fashion.

A simple two-node neural network makes the point. The training set for a scalar function with inputs *x* and *a* contains two training pairs, $\{[0 \mid a^k]^T, 0, c^k\}_{k=1,2}$, and is shown in the legend of Fig. 6. The number of nodes in the network approximating this function is chosen equal to the number of training pairs. For the known inputs, the network output must equal zero and the gradient of the output surface must match the corresponding values of *c* (Fig. 6). The method computes the weights of the network

producing the minimal-order smooth interpolating surface shown in Fig. 6, and solves problems such as network sizing and data over-fitting at their origin. The same approach is used to initialize the neural network control system in Chapter 4.

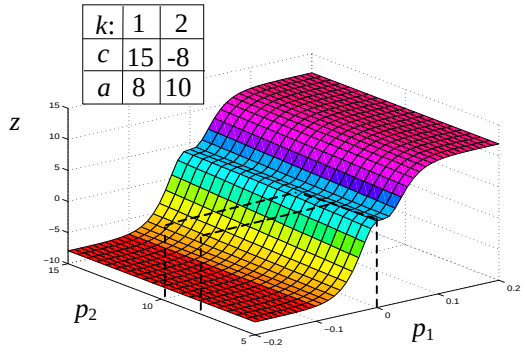

Figure 6. Output surface of a two-node sigmoidal neural network corresponding to the algebraic solution matching the training data provided in the legend for two points.

3.1.2 Exact Input/Output-based Solution

In neural network applications, it often is the case that the only knowledge available about the function to be approximated consists of sampled input/output information. In this case, the training set takes the form $\{\mathbf{y}^k, u^k\}_{k=1,\ldots,p}$ and the output weight equations alone (eq. 44) are to be solved for the neural parameters. Under the assumption of known input-to-node values, $\mathbf{S}$ is again a $p \times s$ known matrix. The output system of weight equations, eq. 46, can be written as,

$$\mathbf{u} = \mathbf{S}\mathbf{v} \tag{69}$$

letting the output bias, b, equal zero without loss of generality, and is to be solved for $\mathbf{v}$. This linear system admits a unique solution if and only if rank($\mathbf{S}|\mathbf{u}$) = rank($\mathbf{S}$) = s, where rank(•) represents the rank of the matrix [e.g., see 74], and it admits an [s – rank($\mathbf{S}$)]-parameter family of solutions if and only if rank($\mathbf{S}|\mathbf{u}$) = rank($\mathbf{S}$) < s.

When the number of nodes, s, is chosen equal to the number of training pairs, p, $\mathbf{S}$ is square. If it also is non-singular and the training data is consistent (e.g., different outputs do not correspond to the same input), eq. 69 is a full-rank linear system for which a unique solution always exists. The input parameters affect the solution of the output initialization equations only through the input-to-node values and determine the nature of $\mathbf{S}$. Another strategy that produces a well-conditioned $\mathbf{S}$ consists of generating the input weights according to the following rule,

$$w_{ij} = fr_{ij} \tag{70}$$

where r_{ij} is chosen from the same distribution as r_i^k, in eq. 65. Here, the scalar f is arbitrary and of order $O(10)$; it can be slightly varied based on how closely spaced the training pairs are. The input bias, $\mathbf{d}$, is computed to center each sigmoid at one of the training pairs, $\{\mathbf{y}^k, u^k\}$, from eq. 50, setting $n_i^k = 0$ when $i = k$:

$$\mathbf{d} = -\operatorname{diag}\left(\mathbf{Y}\mathbf{W}^T\right) \tag{71}$$

Here, the "diag" operator extracts the diagonal of its argument (a square matrix) and reshapes it into a column vector. If the argument is a vector, then "diag" places it on the diagonal of a zero square matrix of appropriate dimensions. $\mathbf{Y}$ is a matrix composed of all the input vectors in the training set:

$$\mathbf{Y} \equiv \left[\mathbf{y}^1 \ \cdots \ \mathbf{y}^p\right]^T \tag{72}$$

The input elements, $\mathbf{y}^k$, from the training set are normalized, and $\mathbf{d}$ is computed based on the input weights according to eq. 71. Thus, the scaling factor f scales the distribution of the input-to-node values, establishing their order of magnitude. While p sigmoids are centered, the remaining sigmoids come close to being saturated for inputs whose absolute value is greater than 5. A variance of order $O(10)$ allows a good fraction of the sigmoids

to be highly saturated, contributing to a smooth approximating function and producing a non-singular **S**. The approach is implemented to train a sigmoidal neural network that approximates a nonlinear function having two inputs and one output, based on 45 input/output samples. Figure 7 shows the actual function being approximated; the intersections of the solid lines on the surface delineate the input space grid being plotted (the software interpolates between these points). The training samples, symbolized by asterisks, are superimposed on the surface.

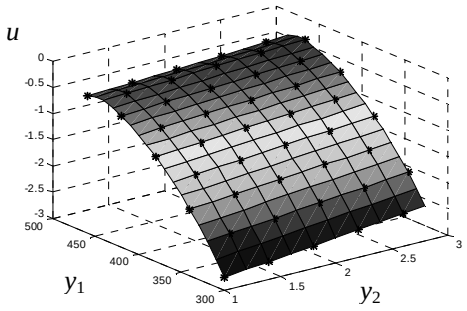

Figure 7. Actual surface being approximated and corresponding training samples, represented by the asterisks.

The neural network is chosen to have 45 nodes and its parameters are determined in one step using the algebraic procedure described above. The neural output surface is plotted over a fine-grid input space in Fig. 8, to demonstrate the network's interpolation abilities. The training time is remarkable (a *MATLAB* code trained a 45-node network in less than 0.16 sec on a 650 MHz computer). For comparison, the 45-node neural network is trained to approximate the data in Fig. 7 using both the *MATLAB 5.3* Levenberg-Marquardt and Resilient Backpropagation functions. Table 1 shows that the performance of the algebraic algorithm is superior to that of the two conventional algorithms in all respects.

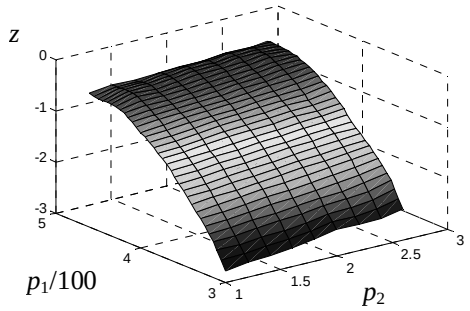

Figure 8. Final function approximation obtained with a neural network algebraically trained using output weight equations.

Algorithm:	**Time (Scaled):**	**Flops:**	**Lines of code (MATLAB®):**	**Epochs:**	**Final error:**
Algebraic	1	2×10^5	8	1	0
Levenberg-Marquardt	50	5×10^7	150	6	10^{-26}
Resilient Backprop.	150	1×10^7	100	150	0.006

Table 1. Performance comparison of algebraic training with two optimization-based algorithm, for the approximation of a scalar function by a 45-node neural network.

The results also show that the algebraic approach manages data over-fitting even when the network size is large, because it addresses the input-to-node values, and hence the level of saturation of the sigmoids, directly. This is found to be particularly challenging when many nodes are used to approximate a relatively flat surface, such as one in Fig. 7. But, choosing the number of nodes equal to the number of training pairs guarantees the existence of a solution that matches the training set exactly. In some cases, it is possible to achieve exact matching using $s < p$, provided the rank condition is satisfied. For instance, suppose s is chosen equal to p and the rank($\mathbf{S}$) is found to be less than s. Then, the number of solutions for a given set of input-to-node values can be made

unique simply by reducing the number of nodes in the network, eliminating the linearly-dependent columns of **S**, until s = rank(**S**).

3.1.3 Approximate Input/Output-based Solution

The algebraic approach outlined in previous sections also can be used to seek approximate solutions of the weight equations, and to obtain a parsimonious network when the number of training pairs, p, is large. Section 3.1.2 showed how exact matching of an input/output training set, $\{\mathbf{y}^k, u^k\}_{k=1,\ldots,p}$, can be achieved by choosing a number of nodes, s, that equals p. An exact solution also could be obtained using less nodes than there are training pairs, i.e., $s < p$, provided the rank condition rank(**S**|**u**) = rank(**S**) = s is satisfied. When the linear system in eq. 69 is not square ($s \neq p$), an inverse relationship between **u** and **v** can be defined using the *generalized inverse* or *pseudoinverse matrix*, denoted by $\mathbf{S}^{PI}$ [74]. Typically, eq. 69 will be overdetermined, with more equations than there are unknowns, therefore its solution will be given by,

$$\mathbf{v} = \left(\mathbf{S}^T\mathbf{S}\right)^{-1}\mathbf{S}^T\mathbf{u} = \mathbf{S}^{PI}\mathbf{u} \tag{73}$$

where $\mathbf{S}^{PI}$ constitutes the left pseudoinverse. If the equations all are consistent, eq. 73 provides the exact value for **v**. If they are not consistent, rank(**S**|**u**) ≠ rank(**S**), and the system in eq. 69 has no solution. In the latter case, eq. 73 provides the estimate that minimizes the mean-square error in the estimate of **v** and can be used to obtain an approximate solution for the output weight equations.

Whenever a neural network is trained by a conventional algorithm (such as backpropagation [76]) that does not achieve exact matching, the corresponding output weight equations fall into the approximate case above. This is because, given a training set, corresponding weight equations can be written for any network, whose parameters

constitute either an exact or an approximate solution of these equations. Letting $\hat{\mathbf{u}}$ denote the best approximation to $\mathbf{u}$ obtained from the final neural parameters, the following holds:

$$\hat{\mathbf{u}} = \mathbf{S}\mathbf{v} \tag{74}$$

Regardless of how the actual network output weight vector $\mathbf{v}$ has been determined, it will satisfy eq. 74, along with the actual value of $\mathbf{S}$. Incidentally, eq. 74 minimizes the error $(\mathbf{u} - \hat{\mathbf{u}})$, which is the same error minimized by conventional optimization-based training algorithms [76]. This observation completes the big picture by showing how the algebraic approach deals with the case of $s < p$, typically found in the neural network literature. More importantly, it can be exploited to develop approximate techniques of solution that are computationally more efficient than the conventional iterative methods, such as the one outlined below and implemented in Section 4.4.

Based on the ideas above, an algebraic technique that combines many networks into one is developed. Suppose a neural network is needed to approximate a large training set (i.e., $p \sim O(10^5)$) using a parsimonious number of nodes, s. Conventional methods, such as Levenberg-Marquardt and Resilient Backpropagation [41, 77], can successfully train networks with $s < p$, minimizing the error $(\mathbf{u} - \hat{\mathbf{u}})$, but they quickly run out of memory if a large set is used at once in what is referred to as *batch training*. If the training set is divided into smaller subsets, training becomes particularly challenging as the neural network is likely to forget previously learned subsets while it is being trained with new ones. Furthermore, these difficulties are exacerbated by the problem of finding the appropriate number of nodes. On the other hand, when a small subset is used, batch training can be very effective. Many of the conventional algorithms converge rapidly and

the network generalization abilities can be optimized by finding the "best" number of nodes through a trial and error procedure.

The technique described here combines networks that individually map the scalar function $h : \Re^q \to \Re$ over portions of its input space into one network that models h over its entire input space. The full training set $\{\mathbf{y}^k, u^k\}_{k = 1, \ldots, p}$, covering the full range of the h input space, is divided into m subsets: $\{\mathbf{y}^k, u^k\}_{k=1, \ldots, p_1}$, $\{\mathbf{y}^k, u^k\}_{k=p_1+1, \ldots, p_2}$, …, $\{\mathbf{y}^k, u^k\}_{k=p_{m-1}+1, \ldots, p_m}$, where $p_m = p$. Each subset is used to train a sigmoidal neural network of the type shown in Fig. 5 whose parameters are indexed by g, where $g = 1, \ldots, m$. That is, each s_g-node network, or *subnetwork*, models the g^{th} subset $\{\mathbf{y}^k, u^k\}_{k=p_{g-1}+1, \ldots, p_g}$, using the weights $\mathbf{W}_g$, $\mathbf{d}_g$, and $\mathbf{v}_g$, and $s_g < p_g$. As suggested by the schematic in Fig. 9, the m networks are *superimposed* to form a s-node network that models the full training set using the weights $\mathbf{W}$, $\mathbf{d}$, and $\mathbf{v}$, and $s = s_1 + \ldots + s_m$.

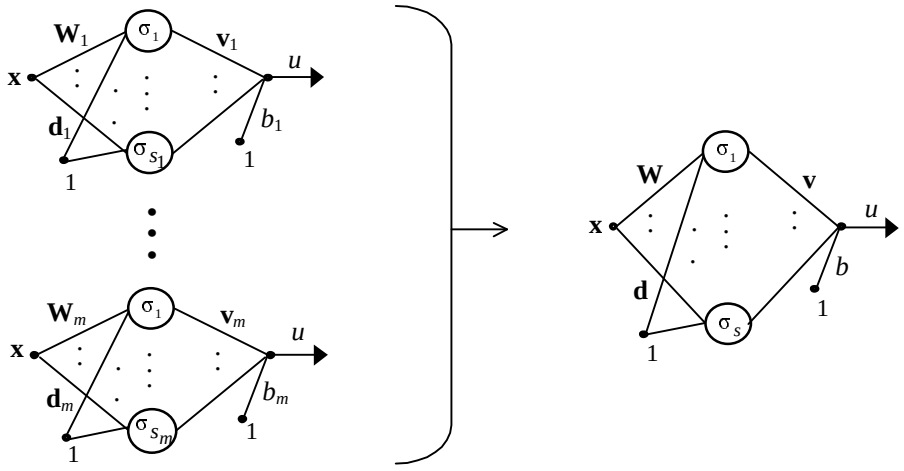

Figure 9. Superposition of m s_g-nodes neural networks into one equivalent s-nodes neural network with same input, $\mathbf{x}$, and the same output, u.

The output weight equations of each subnetwork fall into the approximate case described above. Therefore, the g^{th} neural network approximates the vector $\mathbf{u}_g \equiv [u^{p_{g-1}+1} \ \ldots \ u^{p_g}]^T$ by the estimate,

$$\hat{\mathbf{u}}_g = \mathbf{S}_g \mathbf{v}_g \tag{75}$$

where $\mathbf{v}_g$ is the actual output weight vector and $\text{rank}(\mathbf{S}_g|\hat{\mathbf{u}}_g) = \text{rank}(\mathbf{S}_g) = s_g$. The input weights of the m networks are preserved in the full input weight matrix,

$$\mathbf{W} = \begin{bmatrix} \mathbf{W}_1 \\ \vdots \\ \mathbf{W}_m \end{bmatrix} \tag{76}$$

and input bias vector:

$$\mathbf{d} = \begin{bmatrix} \mathbf{d}_1 \\ \vdots \\ \mathbf{d}_m \end{bmatrix} \tag{77}$$

As a consequence, the matrix of input-to-node values for the full network,

$$\mathbf{N} \equiv \begin{bmatrix} n_1^1 & n_2^1 & \ldots & n_s^1 \\ n_1^2 & n_2^2 & \ldots & n_s^2 \\ \vdots & \vdots & \ddots & \vdots \\ n_1^p & n_2^p & \ldots & n_s^p \end{bmatrix} \tag{78}$$

contains the input-to-node value matrices of the m networks along its main diagonal:

$$\mathbf{N} = \begin{bmatrix} \mathbf{N}_1 & \ldots & \mathbf{N}_{1m} \\ \vdots & \ddots & \vdots \\ \mathbf{N}_{m1} & \ldots & \mathbf{N}_m \end{bmatrix} \tag{79}$$

From eq. 50, it can be shown that the off-diagonal terms, such as $\mathbf{N}_{1m}$ and $\mathbf{N}_{m1}$, are columnwise linearly dependent on the elements in $\mathbf{N}_1$, $\mathbf{N}_2$, …, and $\mathbf{N}_m$, so $r(\mathbf{N}) = r(\mathbf{N}_1) + \ldots + r(\mathbf{N}_m) = s_1 + \ldots + s_m = s$. Also, it is found that in virtually all cases examined

rank(**S**) = rank(**N**). Although a rigorous proof cannot be provided because of the nonlinearity of the sigmoidal function, typically it follows that rank(**S**) = *s*.

Finally, the output weight equations are solved for the output weight vector that approximates the full training set:

$$\mathbf{v} = \mathbf{S}^{PI} \left[\hat{\mathbf{u}}_1^T \ \ldots \ \hat{\mathbf{u}}_m^T\right]^T \tag{80}$$

Because **S** was constructed to be of rank *s*, the rank of (**S**|[$\hat{\mathbf{u}}_1^T$ … $\hat{\mathbf{u}}_m^T$]) is *s* or, at most, *s* + 1, bringing about a zero or small error during the superposition. More importantly, because the error does not increase with *m*, several subnetworks can be algebraically superimposed to model one large training set using a parsimonious number of nodes. In practice, the vector [$\hat{\mathbf{u}}_1^T$ … $\hat{\mathbf{u}}_m^T$] in eq. 80 can be substituted by the vector [$\mathbf{u}_1^T$ … $\mathbf{u}_m^T$], that is directly obtained from the training set and, effectively, contains the output values to be approximated. The method will be demonstrated in Section 4.4, where a neural network is trained based on a large training set simply by superimposing several vector-output subnetworks. Even in this case, the development is identical to the above because the same **S** matrix appears in all neural outputs' weight equations. Generally speaking, the key to developing algebraic training techniques is to construct an **S**, through **N**, that will display the desired characteristics. In the case of approximate input/output-based solutions, **S** must be of rank *s* whereas *s*, the number of nodes, should be as small as possible to produce a parsimonious neural approximator.

3.1.4 Approximate General Solution

Exact matching of both input/output and gradient information, i.e., $\{\mathbf{y}^k, u^k, \mathbf{c}^k\}_{k = 1, \ldots, p}$, is achieved by solving the output and gradient weight equations, eq. 46 and 52,

simultaneously for the neural parameters **W**, **d**, and **v**. Without loss of generality, b can be set equal to zero so that eq. 46 simplifies to eq. 69. It is possible to solve both equations exactly when the dimension of the inputs for which the gradient is unspecified, $(q - e)$, equals p or in the special case described in Section 3.1.1. In general, it is found that a suitable way to incorporate the gradient equations in the training process is to use eq. 52 to obtain a more stringent criterion of formation for the input weights. The approach of Section 3.1.2 has proven that there exist many p-node networks capable of fitting input/output information exactly. Using derivative information during training helps to choose the solution that has the best generalization properties among these networks.

A first estimate of the output weights, **v**, and input-to-node values, n_i^k, to be used in eq. 52 can be obtained from the solution of eq. 69 based on the randomized **W**, as outlined in Section 3.1.2. This solution already fits the input/output training data. The input weights and the remaining parameters can be refined to more closely match the known gradients using a p-step node-by-node update algorithm. The underlying concept is that the input bias, d_i, and the input-to-node values associated with the i^{th} node,

$$\mathbf{n}_i \equiv \left[n_i^1 \; \cdots \; n_i^k \right]^T \tag{81}$$

can be computed solely from the input weights associated with it:

$$\mathbf{w}_i \equiv \left[w_{i1} \; \cdots \; w_{iq} \right]^T \tag{82}$$

At each step, the i^{th} sigmoid is centered at the k^{th} training pair through the input bias d_i, i.e., $n_i^k = 0$, when $i = k$. The k^{th} gradient equations are solved for the input weights associated with the i^{th} node, i.e., from eq. 52:

$$v_i \sigma'\left(n_i^k\right)\mathbf{w}_i = \mathbf{c}^k - \begin{bmatrix} \sum_l v_l \sigma'\left(n_l^k\right) w_{l1} \\ \vdots \\ \sum_l v_l \sigma'\left(n_l^k\right) w_{lq} \end{bmatrix}, \; l = 1,\ldots,(i-1),(i+1),\ldots,p \text{ and } l \neq i \quad (83)$$

The remaining variables are obtained from the initial estimate of the weights. The i^{th} input bias is computed individually,

$$d_i = -\mathbf{y}^k \mathbf{w}_i \quad (84)$$

and p of the input-to-node values are updated:

$$\mathbf{n}_i = d_i + \mathbf{Y}\mathbf{w}_i \quad (85)$$

Y is a matrix composed of all the input vectors in the training set, as defined in eq. 72. At the end of each step, eq. 69 is solved for a new value of **v**, based on the latest input-to-node values.

The gradient equations are solved within a user-specified tolerance. At each iteration, the error enters through **v** and through the input weights to be adjusted in later steps, w_{lj} with $l = (i+1),\ldots, p$. The basic assumption is that the i^{th} node input weights mainly contribute to the k^{th} partial derivatives, $\mathbf{w}_i$, because the i^{th} sigmoid is centered and **v** can be kept bounded for a well-conditioned **S**. As other sigmoids approach saturation their slopes approach zero, decreasing the error associated with w_{lj}. If the gradient with respect to some inputs is unknown, the corresponding input weights can be treated similarly to the input bias. In the limit of p "free" inputs, all initialization equations can be solved exactly for the network's parameters.

Similarly to Section 3.1.2, the approach is demonstrated by training a sigmoidal neural network to approximate a nonlinear function having two inputs and one output,

based on 45 input/output and gradient samples. Figure 10 shows the function being approximated along with the training samples used (symbolized by asterisks).

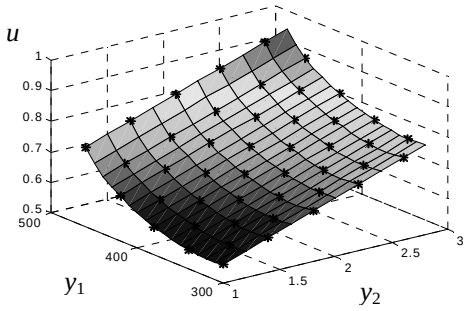

Figure 10. Actual surface being approximated and corresponding training samples, superimposed as asterisks on the graph.

The neural network is chosen to have 45 nodes, and gradient tolerances are 0.05 ($\partial h/\partial y_1$) and 0.5 ($\partial h/\partial y_1$). The set of parameters initially obtained from the output equations produces a lumpy surface (Fig. 11), and the gradient tolerances are not satisfied.

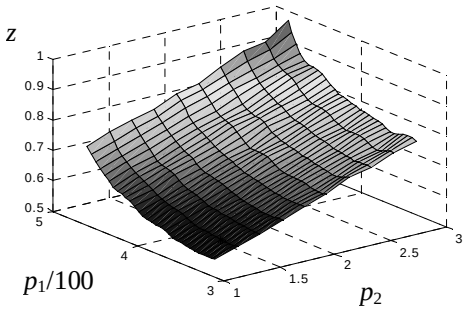

Figure 11. Neural network approximation obtained from output weight equations.

Therefore, the weights are further refined using the *p*-step gradient algorithm, finally producing the output surface in Fig. 12. This approximating function could be improved by running the *p*-step algorithm again with smaller gradient tolerances. If gradient information were not available, the function could be improved by increasing the number

of training pairs, *p*, or by comparing the interpolation properties of different solutions to pick the best one.

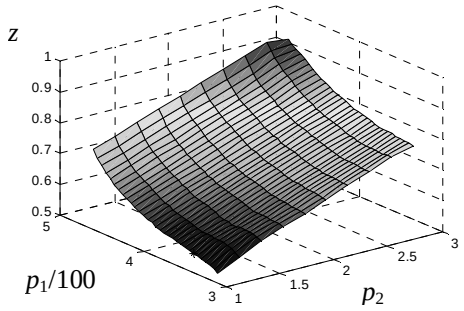

Figure 12. Final neural network approximation obtained from the output and gradient equations combined.

3.2 Modified Resilient Backpropagation

The algebraic techniques are designed for off-line neural network training, where an entire set of data is available at once and can be used in a batch mode. They are most useful when *a-priori* knowledge is first incorporated into neural networks; thus, they also are referred to as *initialization* techniques. In on-line training, new information is obtained over time, and the neural parameters are incrementally updated without waiting for all of the data to be available at once. This kind of incremental training improves upon the network approximation properties using solely new training data, in order to lower computational cost and complexity. Therefore, while the weights are continuously modified, previously learned information must be preserved by the neural network.

Another important distinction to be made between batch training and incremental training is that while the former can be performed globally, the latter is intrinsically local. Global optimization techniques search the entire parameter space and, as a consequence, they only are meaningful if they utilize information about the entire function's input

space. Instead, local optimization algorithms search only the neighborhood surrounding the current parameter value and make good use of partial input/output information. In order to be successful, they must begin searching within the vicinity of the optimizing solution. When on-line training remains local, it also is likely to preserve knowledge by not altering previously learned information.

In supervised learning, the approximating network performance is judged by comparing the actual network output, z, to a desired output or target, u_D, for the corresponding input, $\mathbf{y}_D$. The training algorithm adjusts the network parameters in order to produce a network output that is closer to that target. Algebraic training also is a form of supervised learning, because it is based on p-input/output and, possibly, gradient targets to be met. Because in incremental supervised learning there is only one input/output training pair available, the training set takes the form $\{\mathbf{y}_D, u_D\}$ and indexing of the pairs is not needed. For the architecture shown in Fig. 5, the training objective can be formulated as the minimization of a performance function,

$$E(\mathbf{w}) \equiv \frac{1}{2}[u_D - z(\mathbf{w})]^2 \tag{86}$$

with respect to a vector of ordered weights indexed by ℓ:

$$\mathbf{w} \equiv \left[\text{vec}(\mathbf{W})^T \;\; \mathbf{d}^T \;\; \text{vec}(\mathbf{V})^T \;\; b\right]^T = \{w_\ell\}_{\ell=1,\,2,\,\ldots} \tag{87}$$

Given an initial set of neural parameters, $\mathbf{w}^{(0)}$, at each epoch, i, the value of each weight, $w_\ell^{(i)}$, is modified by a small increment, $\Delta w_\ell^{(i)}$, based on corresponding derivative information, $\partial E(\mathbf{w})/\partial w_\ell$, such that:

$$w_\ell^{(i+1)} = w_\ell^{(i)} + \Delta w_\ell^{(i)} \tag{88}$$

The issues commonly associated with optimization-based training techniques include the computational burden involved in computing and storing derivative information, and the scaling effects associated with dissimilar parameter sizes. Normally, the search for the optimal value of a parameter ceases when the corresponding derivative approaches zero. In the case of sigmoidal neural networks, saturated processing functions also exhibit small gradients, independently of how close to their optimal values the parameters really are. Highly dissimilar parameter sizes may worsen these effects by causing the derivatives to have very different orders of magnitude, blurring the role of the individual parameters in the optimization process.

The resilient backpropagation (RPROP) training algorithm [77] eliminates the harmful effects caused by the magnitudes of the partial derivatives and displays excellent computation and memory requirements. It is, therefore, particularly suited for on-line training, where efficiency and reliability are of special concern. In RPROP training, only the temporal behavior of the sign of the gradients is used to determine the direction and size of the weight increments. The magnitude of the derivative $\partial E(\mathbf{w})/\partial w_\ell$ has no effect on the w_ℓ weight update. The individual size of each increment, denoted by Δ_ℓ, is adjusted at each epoch according to the following rule,

$$\Delta_\ell^{(i)} = \begin{cases} \eta^+ \Delta_\ell^{(i-1)}, & \text{if} \quad \dfrac{\partial E(\mathbf{w})}{\partial w_\ell}^{(i-1)} \dfrac{\partial E(\mathbf{w})}{\partial w_\ell}^{(i)} > 0 \\ \eta^- \Delta_\ell^{(i-1)}, & \text{if} \quad \dfrac{\partial E(\mathbf{w})}{\partial w_\ell}^{(i-1)} \dfrac{\partial E(\mathbf{w})}{\partial w_\ell}^{(i)} < 0 \\ \Delta_\ell^{(i-1)}, & \text{if} \quad \dfrac{\partial E(\mathbf{w})}{\partial w_\ell}^{(i-1)} \dfrac{\partial E(\mathbf{w})}{\partial w_\ell}^{(i)} = 0 \end{cases} \tag{89}$$

where $0 < \eta^- < 1 < \eta^+$. The increment size is increased by the factor η^+ when the algorithm is converging to a minimum and the derivative is not changing sign, while it is

decreased by the factor η^- when the algorithm is jumping over a local minimum and the derivative is changing sign. This process accelerates convergence in shallow regions and slows the search down when local minima are missed.

Once all Δ_ℓ are adjusted, each weight is modified in the direction of gradient descent,

$$\Delta w_\ell^{(i)} = \begin{cases} \Delta_\ell^{(i)} \operatorname{sgn}\left[\dfrac{\partial E(\mathbf{w})^{(i)}}{\partial w_\ell} \right], & \text{if} \quad \dfrac{\partial E(\mathbf{w})^{(i-1)}}{\partial w_\ell} \dfrac{\partial E(\mathbf{w})^{(i)}}{\partial w_\ell} \geq 0 \\ -\Delta w_\ell^{(i-1)}, & \text{if} \quad \dfrac{\partial E(\mathbf{w})^{(i-1)}}{\partial w_\ell} \dfrac{\partial E(\mathbf{w})^{(i)}}{\partial w_\ell} < 0 \end{cases} \tag{90}$$

where sgn[•] represents the signum function. When the error derivative changes sign indicating that a minimum was missed, the weight $w_\ell^{(i+1)}$ is brought back to its previous value $w_\ell^{(i-1)}$ by a *backtracking* epoch [77]. In this case, the increment size does not need adjustment in the next epoch; therefore $\partial E(\mathbf{w})^{(i-1)}/\partial w_\ell$ is set equal to zero until eq. 89 has found the appropriate Δ_ℓ.

When the RPROP algorithm is used for on-line training, the initial increment value $\Delta_\ell^{(0)}$ is a crucial ingredient. Setting all initial increments equal to the same constant value (e.g., 0.1) for weights of dissimilar sizes [77] is equivalent to disregarding prior network weights. Instead, initial increments are chosen commensurate with a fraction, f_w, of the corresponding prior weights and perturbed by f_0 to account for zero weights:

$$\Delta_\ell^{(0)} = f_w |w_\ell| + f_0 \tag{91}$$

Backtracking also is an algorithmic feature that is key to on-line training. It allows the search to remain local even when, due to the progress made by eq. 89, the increment size becomes large enough to bring the search to further minima. The effectiveness of the

modified RPROP algorithm will be demonstrated in Section 5.1.3, where the same neural network is trained on line with and without the proposed modifications.

3.3 Algebraically Constrained Supervised Training

The local nature of on-line training implies that its effectiveness is largely dependent upon the initial values of the parameters. In most neural network applications, some knowledge about the function being approximated is available prior to their implementation. On the other hand, assimilating information on line, while the neural network is being implemented and the system is operating, translates into updating and improving upon the network performance virtually in real time. To exploit both aspects of learning, algebraic training is used to incorporate *a-priori* system knowledge into a neural network, and RPROP training is used to continue adapting the same network on line. The algebraic training procedure, in this case, is referred to as *initialization* and provides an excellent starting point for the on-line training routine.

Initializing the neural network off line improves reliability and convergence during the on-line phase, but, in order to maintain the characteristics acquired during initialization, on-line learning must improve upon performance without unlearning prior knowledge. Redundancy in the network parameters contributes to these objectives, as some parameters may be used for preserving information and others for improving performance. The RPROP-algorithm modifications of Section 3.2 also are useful in this regard, but they do not guarantee continued matching of the initialization requirements, eq. 43 and 48. Incorporating the weight equations in the on-line adaptation brings about a constrained supervised training algorithm that minimizes E (eq. 86), subject to the initialization requirements.

In the following paragraphs, the algebraically constrained training technique is explained for a simple scalar case. In Section 5.3, the same approach will be explained for the on-line training of an adaptive critic control system. In this simple example, a scalar-output neural network is initialized through the gradient-based approach of Section 3.1.1 and, subsequently, adapted on line. Suppose the training set for the function to be modeled, $u = h(\mathbf{y})$, is gradient based, $\{\mathbf{y}^k, 0, \mathbf{c}^k\}_{k = 1, \ldots, p}$ with $\mathbf{y}^k = [\mathbf{x}^{k^T} \,|\, \mathbf{a}^{k^T}]^T$ $= [\mathbf{0} \,|\, \mathbf{a}^{k^T}]^T$. When the $\mathbf{x}^k$ training inputs are partitioned as in,

$$\mathbf{y}^k = \begin{bmatrix} \mathbf{x}_1^k \\ \mathbf{x}_2^k \\ \mathbf{a}^k \end{bmatrix} \tag{92}$$

the known gradients, $\mathbf{c}^k$, can be written as,

$$\mathbf{c}^k = \begin{bmatrix} \left.\dfrac{\partial u}{\partial \mathbf{x}_1}\right|_{\mathbf{y}^k} \\ \left.\dfrac{\partial u}{\partial \mathbf{x}_2}\right|_{\mathbf{y}^k} \end{bmatrix} \tag{93}$$

where $\partial u/\partial \mathbf{x}_1$ and $\partial u/\partial \mathbf{x}_2$ are defined as column vectors. $\mathbf{a}$ contains all inputs for which no derivative information is available.

The training set can be partitioned into two independent sets with equivalent information: $\{[\mathbf{x}_1^{k^T} \,|\, \mathbf{a}^{k^T}]^T, 0, \partial u/\partial \mathbf{x}_1|^k\}_{k = 1, \ldots, p}$ and $\{[\mathbf{x}_2^{k^T} \,|\, \mathbf{a}^{k^T}]^T, 0, \partial u/\partial \mathbf{x}_2|^k\}_{k = 1, \ldots, p}$, where $\mathbf{x}_1^k = \mathbf{x}_2^k = \mathbf{0}$ for $\forall\, k$. Because these training sets also display the gradient-based form described in Section 3.1.1, they can be used independently to initialize two p-nodes scalar networks of the type shown in Fig. 5. The corresponding weight equations are solved exactly using the gradient-based algebraic procedure. The parameters $\mathbf{W}_{\mathbf{x}_1}$, $\mathbf{W}_{\mathbf{a}_1}$,

$\mathbf{d}_1$, $\mathbf{v}_1$, and b_1 are obtained for the first network, and the parameters $\mathbf{W}_{\mathbf{x}_2}$, $\mathbf{W}_{\mathbf{a}_2}$, $\mathbf{d}_2$, $\mathbf{v}_2$, and b_2 are obtained for the second network. This notation is consistent with the one used in Section 3.1.1, with the addition of the subscript that indicates a parameter's network.

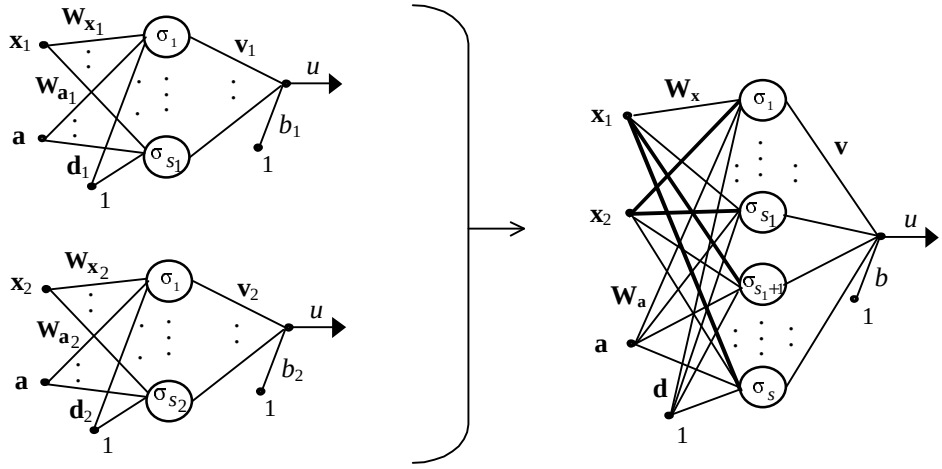

Figure 13. Two s_g-node neural networks are combined into one s-node neural network with the same output u and input $\mathbf{a}$, and both inputs $\mathbf{x}_1$ and $\mathbf{x}_2$; the dark lines represent the new connections being introduced.

A single scalar-output network that models all of the original data, is obtained by *combining* these two networks, as suggested by Fig. 13. According to the procedure in Section 3.1.1, each initialized network contains p nodes; therefore the final network contains $2p$ nodes. Its input weight matrix, $\mathbf{W}$, also is composed of weights associated with the input $\mathbf{x}$ and of weights associated with the input $\mathbf{a}$, i.e., $\mathbf{W} = [\mathbf{W}_\mathbf{x} \mid \mathbf{W}_\mathbf{a}]$. In order to preserve the exact matching of the training set, the new connection weights (shown in Fig. 13) are initially set equal to zero:

$$\mathbf{W}_\mathbf{x} = \begin{bmatrix} \mathbf{W}_{\mathbf{x}_1} & \mathbf{0} \\ \mathbf{0} & \mathbf{W}_{\mathbf{x}_2} \end{bmatrix} \tag{94}$$

Since $\mathbf{a}$ is an input to both p-node networks, there are no zero weights in $\mathbf{W}_\mathbf{a}$:

$$\mathbf{W_a} = \begin{bmatrix} \mathbf{W_{a_1}} \\ \mathbf{W_{a_2}} \end{bmatrix} \tag{95}$$

The input bias and the output weight vector are obtained in a similar fashion: $\mathbf{d} = [\mathbf{d}_1^T \ \ \mathbf{d}_2^T]^T$ and $\mathbf{v} = [\mathbf{v}_1^T \ \ \mathbf{v}_2^T]^T$. The output bias is computed from the initialized biases, $b = b_1 + b_2$, to satisfy the output requirement $z(\mathbf{y}^k) = 0$. It is easily shown that these parameters satisfy the weight equations of the 2p-node network exactly.

The 2p-node network obtained thus far approximates the function $u = h(\mathbf{y})$ based on the off-line training set, $\{[\mathbf{0} \mid \mathbf{a}^{k\,T}]^T, 0, \mathbf{c}^k\}_{k = 1, \ldots, p}$. Assuming the network is implemented in a scenario where further information about the function becomes available over time, it can be periodically adapted on line to improve performance. When the neural network is trained on line in a supervised fashion, it learns target input/output data, $\{\mathbf{y}_D, u_D\}$, as described in Section 3.2.

The weights that were initially set equal to zero now provide for the desired redundancy and can be used to optimize the performance, E, in eq. 86. The remaining parameters are used to satisfy the on-line constraints imposed by the weight equations, in eq. 43 and 48. To emphasize this, $\mathbf{W_x}$ is partitioned into four matrices,

$$\mathbf{W_x} = \begin{bmatrix} \mathbf{W}_{11} & \mathbf{W}_{12} \\ \mathbf{W}_{21} & \mathbf{W}_{22} \end{bmatrix} \tag{96}$$

where $\mathbf{W}_{ij}$ contains the input weights connecting the $\mathbf{x}_j$-portion of the network input $\mathbf{p}$ to the nodes indexed by $[(i - 1)p + 1]$ through ip.

The weights $\mathbf{W}_{12}$ and $\mathbf{W}_{21}$ are used specifically to minimize E. Given the on-line training set $\{\mathbf{y}_D, u_D\}$, at each epoch, i, the on-line performance (eq. 86) is minimized with respect to the vector,

$$\mathbf{w} \equiv \left[\text{vec}(\mathbf{W}_{12})^T \;\; \text{vec}(\mathbf{W}_{21})^T\right]^T = \{w_\ell\}_{\ell=1,\,2,\,\ldots} \tag{97}$$

by adding an increment $\Delta w_\ell^{(i)}$ to each weight value $w_\ell^{(i)}$ in $\mathbf{w}$, according to eq. 88-89. While initially $\mathbf{w}^{(0)} = \mathbf{0}$, at any other epoch $\mathbf{w}^{(i)} \neq \mathbf{0}$, as determined by the RPROP algorithm. With new values for the parameters $\mathbf{W}_{12}$ and $\mathbf{W}_{21}$, the full weight equations, eq. 55, 56, and 57 (with $s = 2p$), are no longer satisfied by the initialized values of $\mathbf{W}_{11}$, $\mathbf{W}_{22}$, $\mathbf{W_a}$, $\mathbf{d}$, $\mathbf{v}$, and b. Therefore, these parameters also must be determined at every epoch, as described below.

The full weight equations represent the algebraic constraints that are imposed on the minimization of E, in order to preserve knowledge of the off-line training set. They must be satisfied at each epoch, i, without becoming a computational burden. The important result is that new parameter values that satisfy the algebraic constraint when $\mathbf{w}^{(i)} \neq \mathbf{0}$ can be computed at each epoch, without solving eq. 55-57 in the manner used for initialization. If $\mathbf{W_a}$ and $\mathbf{d}$ are kept constant throughout the on-line update, i.e., $\mathbf{W_a}^{(i)} = \mathbf{W_a}^{(0)}$ and $\mathbf{d}^{(i)} = \mathbf{d}^{(0)}$ for all i, then the input-to-node values, n_i^k, also remain constant as shown by eq. 57. In particular, it follows that $\mathbf{S}^{(i)} = \mathbf{S}^{(0)}$, where $\mathbf{S}^{(0)}$ is equal to the $p \times 2p$ initialization matrix,

$$\mathbf{S} = [\mathbf{S}^1 \;\; \mathbf{S}^2] \tag{98}$$

composed of the p-node network sigmoidal matrices $\mathbf{S}^1$ and $\mathbf{S}^2$. Since $\mathbf{S}$ is unchanged, the initialized values of the output weights and bias continue to satisfy the output equations (eq. 55), hence $\mathbf{v}^{(i)} = \mathbf{v}^{(0)}$ and $b^{(i)} = b^{(0)}$.

The gradient weight equations (eq. 56) change at every epoch, because $\mathbf{W_x}^{(i)} \neq \mathbf{W_x}^{(0)}$. For the $2p$-node network, eq. 56 can be reformulated as two equations,

$$\begin{aligned} \boldsymbol{\varsigma}^1 &= \mathbf{X}_1 \,\mathrm{Vec}(\mathbf{W}_{11}) + \mathbf{X}_2 \,\mathrm{Vec}(\mathbf{W}_{21}) \\ \boldsymbol{\varsigma}^2 &= \mathbf{X}_1 \,\mathrm{Vec}(\mathbf{W}_{12}) + \mathbf{X}_2 \,\mathrm{Vec}(\mathbf{W}_{22}) \end{aligned} \tag{99}$$

where $\boldsymbol{\varsigma}^i \equiv [(\partial u/\partial \mathbf{x}_i|^1)^T \ldots (\partial u/\partial \mathbf{x}_i|^k)^T]^T$. The $\mathbf{X}_1$ and $\mathbf{X}_2$ matrices are defined similarly to $\mathbf{X}$ in eq. 61. Except that for $\mathbf{X}_1$, $\mathbf{B}^k = [v_1\sigma'(n_1{}^k) \ldots v_p\sigma'(n_p{}^k)]$ and e equals the dimension of $\mathbf{x}_1$; for $\mathbf{X}_2$, $\mathbf{B}^k = [v_{p+1}\sigma'(n_{p+1}{}^k) \ldots v_{2p}\sigma'(n_{2p}{}^k)]$ and e equals the dimension of $\mathbf{x}_2$. At each epoch, $\mathbf{W}_{12}$ and $\mathbf{W}_{21}$ are known from $\mathbf{w}$, so eq. 99 can be solved for the $\mathbf{W}_{11}$ and $\mathbf{W}_{22}$:

$$\begin{aligned} \mathrm{Vec}(\mathbf{W}_{11}) &= (\mathbf{X}_1)^{-1}[\boldsymbol{\varsigma}^1 - \mathbf{X}_2 \,\mathrm{Vec}(\mathbf{W}_{21})] \\ \mathrm{Vec}(\mathbf{W}_{22}) &= (\mathbf{X}_2)^{-1}[\boldsymbol{\varsigma}^2 - \mathbf{X}_1 \,\mathrm{Vec}(\mathbf{W}_{12})] \end{aligned} \tag{100}$$

Both $\mathbf{v}$ and all input-node-values, $n_i{}^k$, remain constant during the on-line update. Thus, the $\mathbf{X}_1$ and $\mathbf{X}_2$ matrices also remain unchanged and their inverse can be computed a priori. In fact, all four matrices $\mathbf{K}_{11} \equiv (\mathbf{X}_1)^{-1}\boldsymbol{\varsigma}^1$, $\mathbf{K}_{12} \equiv (\mathbf{X}_1)^{-1}\mathbf{X}_2$, $\mathbf{K}_{22} \equiv (\mathbf{X}_2)^{-1}\boldsymbol{\varsigma}^2$, and $\mathbf{K}_{21} \equiv (\mathbf{X}_2)^{-1}\mathbf{X}_1$ can be determined off-line and stored for on-line usage. In summary, at each epoch the network parameters are updated according to the following rule,

$$\begin{cases} w_\ell^{(i+1)} = w_\ell^{(i)} + \Delta w_\ell^{(i)} \;\rightarrow\; \mathbf{W}_{21}^{(i+1)},\ \mathbf{W}_{12}^{(i+1)} \\ \mathbf{W}_{\mathbf{a}}^{(i+1)} = \mathbf{W}_{\mathbf{a}}^{(i)},\ \mathbf{d}^{(i+1)} = \mathbf{d}^{(i)} \\ \mathbf{v}^{(i+1)} = \mathbf{v}^{(i)},\ b^{(i+1)} = b^{(i)} \\ \mathrm{Vec}\left(\mathbf{W}_{11}^{(i+1)}\right) = \mathbf{K}_{11} - \mathbf{K}_{12}\,\mathrm{Vec}\left(\mathbf{W}_{21}^{(i+1)}\right) \\ \mathrm{Vec}\left(\mathbf{W}_{22}^{(i+1)}\right) = \mathbf{K}_{22} - \mathbf{K}_{21}\,\mathrm{Vec}\left(\mathbf{W}_{12}^{(i+1)}\right) \end{cases} \tag{101}$$

where w_ℓ is defined in eq. 97 and Δw_ℓ is given by eq. 89 and 90. The values $\mathbf{W}_{11}{}^{(0)}$, $\mathbf{W}_{12}{}^{(0)}$, $\mathbf{W}_{21}{}^{(0)}$, $\mathbf{W}_{22}{}^{(0)}$, $\mathbf{W}_{\mathbf{a}}{}^{(0)}$, $\mathbf{d}^{(0)}$, $\mathbf{v}^{(0)}$, and $b^{(0)}$ all correspond to the initialization weights described in the previous paragraphs. The algorithm locally searches for the optimal set of weights and, simultaneously, satisfies the algebraic constraints expressed by eq. 55-57. Convergence is achieved when $\partial E(\mathbf{w})/\partial w_\ell \rightarrow 0$ for $\forall\ \ell$.

Similar results are achieved when redundant output weights are present, as for the case in which vector-output neural networks are obtained by joining initialized scalar networks. The procedure is very similar to the one described above for the 2*p*-node scalar-output neural network. The main difference is that part of the output weights also are modified by the RPROP rule. Therefore, the output weight equations must be satisfied adjusting the remaining output weights. The computation involved can still be kept to a minimum by means similar to the above. The vector-output extension of the algebraically constrained learning technique is demonstrated in Section 5.3.

3.4 Chapter Summary

Neural networks are massively parallel computational paradigms that are used for function approximation or identification in a variety of applications. They are considered to be more powerful than other universal function approximators, because they are intrinsically capable of dealing with nonlinearities and multi-dimensional input and output spaces. A novel algebraic training approach is developed that consists of formulating the training set as requirements to be imposed on the network equations. It can produce exact or approximate solutions, depending on the number of nodes in the nonlinear layer. In particular, it is found that an exact input/output or gradient-based solution always can be obtained by using as many nodes as there are training pairs. Also, the resilient backpropagation approach is modified to obtain an algorithm that has faster convergence and better preserves the initial parameters. When this backpropagation-based algorithm is combined with the algebraic training approach, a scheme that allows for incremental learning and guarantees preservation of *a-priori* information is obtained.

Chapter 4

Initial Specification of the Neural Network Control System by an Algebraic Training Approach

In this chapter, the neural network controller structure and initial parameters are specified in what is referred to as the pre-training phase. In this stage of the design, the system architecture and appropriate performance baselines are identified based on classical/modern feedback, inner/outer loop, proportional-integral-derivative control formulations. Appropriate system requirements are estimated by considering the performance of an equivalent linear controller at a set of operating points *OP*, and are imposed on the neural network control structure. The neural networks' size and parameters that meet these requirements are determined solely by solving linear systems of equations, using the algebraic training techniques introduced in Section 3.1, in what is also referred to as initialization.

The results show that the pre-training phase alone defines a global neural network controller that is capable of performing at least as well as an equivalent gain-scheduled controller. In fact, the algebraic training techniques aim not only at meeting the desired performance targets at the chosen design set *OP*, but also at warranting satisfactory approximation properties over the entire input space corresponding to the convex hull [78] of *OP*, i.e., the interpolating region *IR*, illustrated in Fig. 14. The on-line training phase, that will be described in Chapter 5, improves control response for large-angle amplitude and coupled motions, fast transitions between equilibria, and unforeseen conditions, throughout the operating region *OR*. The full operating region *OR* includes

IR, as well as a set *ER* of extrapolating conditions excluded by *IR*, $ER \cap IR = \varnothing$, such that $ER \cup IR = OR$ (Fig. 14).

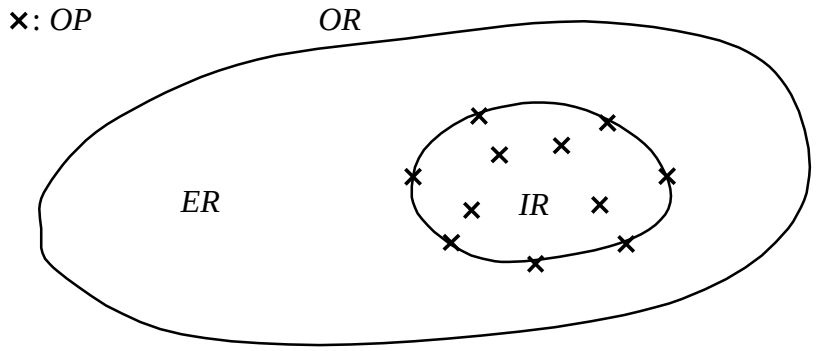

Figure 14. Abstract representation of the full operating region *OR* and the relevant operating subsets: the set *OP* of design operating points (designated by crosses), its convex hull or interpolating region *IR*, and the set *ER* of extrapolation points.

The two-phase learning approach is demonstrated by designing a neural network controller for a business jet aircraft model. The nonlinear control system always is motivated by a multivariable linear control structure; the Proportional-Integral (PI) controller is chosen for illustration. The corresponding nonlinear controller is obtained by replacing the linear gains of a PI controller with nonlinear neural networks: a *forward neural network*, a *feedback neural network*, and a *command-integral neural network* replace the respective gains. In addition to these control networks, a critic network is introduced in order to evaluate their performance, as anticipated in Section 2.1.

A full six-degree-of-freedom simulation of the business jet aircraft is available in the form of a nonlinear differential equation:

$$\dot{\mathbf{x}} = \mathbf{f}\left[\mathbf{x}(t), \mathbf{p}_{\mathrm{m}}(t), \mathbf{u}(t)\right] \tag{1}$$

It is based on mathematical models, full-scale wind tunnel data, and actual physical and performance characteristics of an early twin-jet configuration [64]. The control design

takes into account the full state vector, $\mathbf{x} = [V\ \gamma q\ \theta r\ \beta p\ \mu]^T$, comprising airspeed V (m/s), path angle γ(rad), pitch rate q (rad), pitch angle θ(rad), yaw rate r (rad/s), sideslip angle β(rad), roll rate p (rad/s), and bank angle μ (rad). The altitude and velocity also are specified through the *scheduling vector* $\mathbf{a} = [V\ H]^T$. The independent controls being generated are throttle δT (%), stabilator δS (rad), aileron δA (rad), and rudder δR (rad); i.e., $\mathbf{u} = [\delta T\ \delta S\ \delta A\ \delta R]^T$. A description of the simulated equations of motion will be provided in Section 4.4 and in Appendix F.

During the on-line phase, the simulation is allowed to explore the entire operational domain, *OR*, defined as the envelope for which there exist control settings $\mathbf{u}_c$ capable of trimming the aircraft at corresponding values of state and command input ($\mathbf{x}_c$, $\mathbf{y}_c$). The command input, $\mathbf{y}_c = [V_c\ \gamma_c\ \mu_c\ \beta_c]^T$, contains the state elements that, given the altitude H_c (m), uniquely specify a longitudinal-lateral-directional steady maneuver (e.g., a coordinated turn), postulating $\dot{\phi}_c = \dot{\theta}_c = 0$ with ϕ as the Euler roll angle. For simplicity, the commanded altitude, H_c, is approximated by the aircraft altitude, H. Trim control settings, $\mathbf{u}_c$, that realize the commanded maneuver can be defined solely in terms of $\mathbf{y}_c$:

$$\mathbf{0} = \mathbf{f}\left[\mathbf{x}_c, \mathbf{p}_m, \mathbf{u}_c(\mathbf{y}_c)\right] \tag{102}$$

In fact, the commanded state, $\mathbf{x}_c$, and corresponding flight conditions, $\mathbf{p}_m$, also are defined exclusively by $\mathbf{y}_c$, as their elements either correspond to elements in $\mathbf{y}_c$ (i.e., V_c, γ_c, μ_c, β_c) or can be computed from $\mathbf{y}_c$ (i.e., q_c, θ_c, r_c, p_c) such that they do not oppose the commanded maneuver, as shown in Section 4.4.

4.1 Linear Design

Linear controllers that satisfy established engineering criteria [67-66] are designed for a family of linearized models obtained at the set *OP* of equilibria, providing for the desired performance targets to be matched by the neural network controller. The nonlinear aircraft model is approximated as a linear-parameter-varying system over the two-dimensional flight envelope, shown in Fig. 15, assuming steady-level flight, i.e., $\gamma_0 = \mu_0 = \beta_0 = 0$.

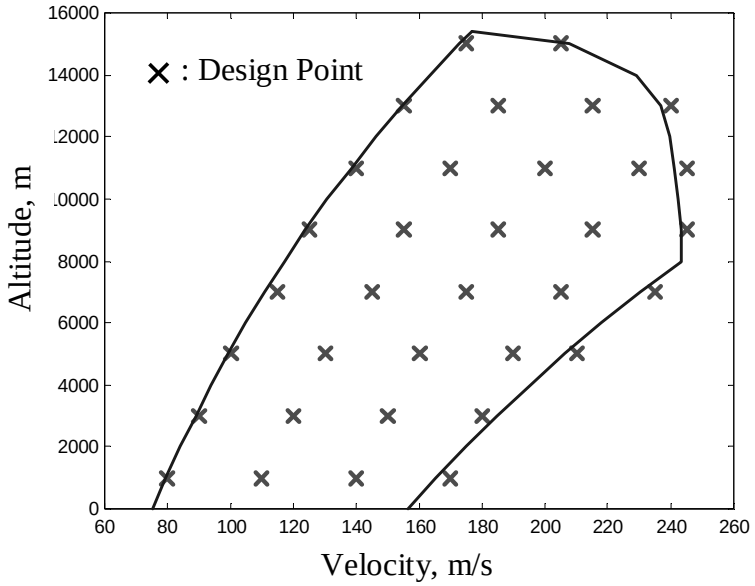

Figure 15. Business jet aircraft steady-level flight envelope (*IR*) and set *OP* of design operating points used for the neural network pre-training phase.

The flight envelope is designed by considering the stall speed, the thrust/power required and available, compressibility effects, and the maximum allowable dynamic pressure to prevent structural damage [64]. The set *OP* consists of thirty-four design points chosen from the boundaries and the interior of the flight envelope, corresponding to the region *IR*, and both sets are shown in Fig. 15. As introduced in Section 2.2, eq. 1 can be

linearized about each equilibrium or operating point in *OP* by holding the scheduling vector, $\mathbf{a}_0 = [V_0 \; H_0]^T$, fixed.

Given the dynamic system of eq. **1**, a first-degree expansion can be written:

$$\begin{aligned}\dot{\mathbf{x}}(t) &= \dot{\mathbf{x}}_0(t) + \Delta\dot{\mathbf{x}}(t) \\ &= \mathbf{f}_0[\mathbf{x}_0(t), \mathbf{p}_m(t), \mathbf{u}_0(t)] + \Delta\mathbf{f}[\mathbf{x}_0(t), \mathbf{p}_m(t), \mathbf{u}_0(t), \Delta\mathbf{x}(t), \Delta\mathbf{u}(t)] \\ &\approx \mathbf{f}_0[\mathbf{x}_0(t), \mathbf{p}_m(t), \mathbf{u}_0(t)] + \frac{\partial\mathbf{f}}{\partial\mathbf{x}}\Delta\mathbf{x}(t) + \frac{\partial\mathbf{f}}{\partial\mathbf{u}}\Delta\mathbf{u}(t) \\ &= \mathbf{f}_0[\bullet] + \mathbf{F}\Delta\mathbf{x}(t) + \mathbf{G}\Delta\mathbf{u}(t)\end{aligned} \tag{103}$$

The perturbation model is:

$$\Delta\dot{\mathbf{x}}(t) = \mathbf{F}[\mathbf{x}_0(t), \mathbf{p}_m(t), \mathbf{u}_0(t)]\Delta\mathbf{x}(t) + \mathbf{G}[\mathbf{x}_0(t), \mathbf{p}_m(t), \mathbf{u}_0(t)]\Delta\mathbf{u}(t) \tag{104}$$

This model is almost a linear, parameter-varying (LPV) plant, "almost" because the system matrices depend on $\mathbf{x}_0(t)$, as well as the remaining variables. In most applications, effects of parameter variations are ignored because time-varying effects are small. Therefore, $\{\mathbf{F}, \mathbf{G}\}_{\kappa = 1, \ldots, 34}$ can be treated as a set of LTI plant models of the type in eq. **14**. The Jacobian matrices, **F** and **G**, are evaluated numerically at all thirty-four points in *OP*, using a MATLAB built-in function, numjac, that is based on the algorithm proposed in [79]. The nonlinear control system is pre-trained by deriving performance targets from the linear control laws corresponding to this set of LTI models and by incorporating these targets into the neural networks.

The perturbation models obtained for the aircraft state and control can be written as longitudinal, $(\bullet)_L$, and lateral-directional, $(\bullet)_{LD}$, reduced, fourth-order independent models,

$$\begin{aligned}\Delta\dot{\mathbf{x}}_L(t) &= \mathbf{F}_L\Delta\mathbf{x}_L(t) + \mathbf{G}_L\Delta\mathbf{u}_L(t) \\ \Delta\dot{\mathbf{x}}_{LD}(t) &= \mathbf{F}_{LD}\Delta\mathbf{x}_{LD}(t) + \mathbf{G}_{LD}\Delta\mathbf{u}_{LD}(t)\end{aligned} \tag{105}$$

by neglecting the cross-coupling terms in the Jacobian matrices that typically are of comparatively small magnitude [64]. The longitudinal state and control vectors are $\mathbf{x}_L = [V\ \gamma q\ \theta]^T$ and $\mathbf{u}_L = [\delta T\ \delta S]^T$, respectively; and, the lateral-directional state and control are $\mathbf{x}_{LD} = [r\ \beta p\ \mu]^T$ and $\mathbf{u}_{LD} = [\delta A\ \delta R]^T$, respectively (as indicated by the respective subscripts). As a result, the family of longitudinal LTI models, $\{\mathbf{F}_L, \mathbf{G}_L\}_{\kappa = 1, \ldots, 34}$, can be considered separately from the family of lateral-directional LTI models, $\{\mathbf{F}_{LD}, \mathbf{G}_{LD}\}_{\kappa = 1, \ldots, 34}$. Longitudinal and lateral-directional linear control gains are computed and their performance evaluated independently for each of the two families, as described in the following sections. In Sections 4.3 and 4.5. these gains are used to initialize decoupled longitudinal and lateral-directional feedback, command-integral, and critic networks.

4.1.1 Proportional-Integral Control

The Proportional-Integral (PI) controller is the multivariable linear control structure chosen to motivate the nonlinear neural network control system. A PI controller, shown in Fig. 16, modifies the stability and transient response of the system through the feedback gain matrix, $\mathbf{C}_B$, and it provides Type-1 response [80] to command inputs through the proportional gain matrix, $\mathbf{C}_F$, and the command-integral gain matrix $\mathbf{C}_I$. $\mathbf{H_u}$ and $\mathbf{H_x}$ are Jacobian matrices obtained from a first-degree expansion of eq. 2, resulting into the following linearized output equation:

$$\Delta \mathbf{y}_s(t) = \mathbf{H_x} \Delta \mathbf{x}(t) + \mathbf{H_u} \Delta \mathbf{u}(t) \tag{106}$$

The objectives of the control system can be expressed in terms of the quadratic cost function,

$$
\begin{aligned}
J &= \lim_{t_f \to \infty} \frac{1}{2} \int_0^{t_f} L[\mathbf{x}_a(\tau), \tilde{\mathbf{u}}(\tau)] d\tau \\
&= \lim_{t_f \to \infty} \frac{1}{2} \int_0^{t_f} \left[\mathbf{x}_a{}^T(\tau) \mathbf{Q}_a \mathbf{x}_a(\tau) + 2\mathbf{x}_a{}^T(\tau) \mathbf{M}_a \tilde{\mathbf{u}}(\tau) + \tilde{\mathbf{u}}^T(\tau) \mathbf{R}_a \tilde{\mathbf{u}}(\tau) \right] d\tau
\end{aligned}
\tag{107}
$$

which is minimized with respect to $\tilde{\mathbf{u}}$. $\mathbf{x}_a$ represents an augmented state that includes both the deviation, $\tilde{\mathbf{x}}$, from the commanded state,

$$
\tilde{\mathbf{x}} \equiv (\mathbf{x} - \mathbf{x}_0) - (\mathbf{x}_c - \mathbf{x}_0) = \mathbf{x} - \mathbf{x}_c \tag{108}
$$

and the time integral of the output error, ξ, i.e., $\mathbf{x}_a \equiv [\tilde{\mathbf{x}}^T \ \ \xi^T]^T$.

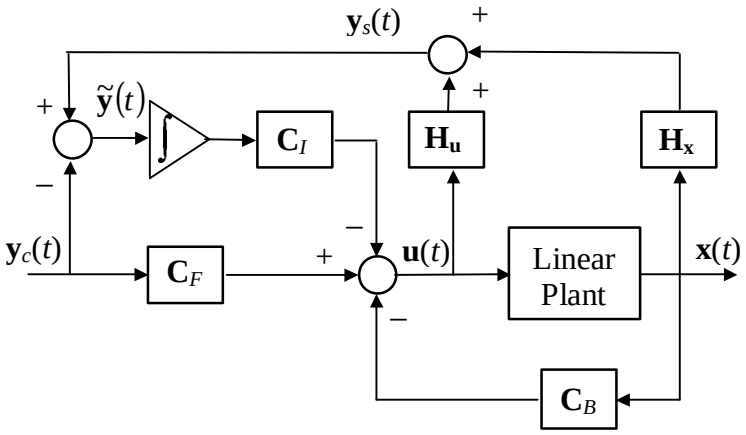

Figure 16. Example of linear proportional-integral feedback control system. (Δ's are omitted for simplicity.)

Similarly, the output error and the minimizing control are defined as deviations from the set point commanded by $\mathbf{y}_c$, i.e., $\tilde{\mathbf{y}} \equiv \mathbf{y}_s - \mathbf{y}_c$ and $\tilde{\mathbf{u}} \equiv \mathbf{u} - \mathbf{u}_c$. When the minimization of a quadratic cost function (eq. 107) is subject to a linear dynamic constraint, such as eq. 14, the LQ law eq. 21 provides for the optimal control in terms of the newly defined deviations:

$$
\tilde{\mathbf{u}}(t) = -\mathbf{C}_a \mathbf{x}_a(t) = -[\mathbf{C}_B \ \ \mathbf{C}_I] \mathbf{x}_a(t) = -\mathbf{C}_B \tilde{\mathbf{x}}(t) - \mathbf{C}_I \xi(t) \tag{109}
$$

The forward gain matrix, $\mathbf{C}_F$, can be obtained from the feedback gain matrix and set point matrices defined below [81]:

$$\mathbf{C}_F = \mathbf{B}_{22} + \mathbf{C}_B\mathbf{B}_{12} \tag{110}$$

For the linearized system (eq. 14 and 106), the set point is defined by letting the state-rate perturbation, $\Delta\dot{\mathbf{x}}$, equal zero:

$$\begin{bmatrix} \mathbf{0} \\ \Delta\mathbf{y}_c \end{bmatrix} = \begin{bmatrix} \mathbf{F} & \mathbf{G} \\ \mathbf{H_x} & \mathbf{H_u} \end{bmatrix} \begin{bmatrix} \Delta\mathbf{x}_c \\ \Delta\mathbf{u}_c \end{bmatrix} \tag{111}$$

The longitudinal-set-point state and control perturbations can be obtained from the command-input perturbation, $\Delta\mathbf{y}_c \equiv \mathbf{y}_c - \mathbf{y}_0$, through the following relation [81]:

$$\begin{bmatrix} \Delta\mathbf{x}_c \\ \Delta\mathbf{u}_c \end{bmatrix} = \begin{bmatrix} \mathbf{F} & \mathbf{G} \\ \mathbf{H_x} & \mathbf{H_u} \end{bmatrix}^{-1} \begin{bmatrix} \mathbf{0} \\ \Delta\mathbf{y}_c \end{bmatrix} \equiv \begin{bmatrix} \mathbf{B}_{11} & \mathbf{B}_{12} \\ \mathbf{B}_{21} & \mathbf{B}_{22} \end{bmatrix} \begin{bmatrix} \mathbf{0} \\ \Delta\mathbf{y}_c \end{bmatrix} = \begin{bmatrix} \mathbf{B}_{12}\Delta\mathbf{y}_c \\ \mathbf{B}_{22}\Delta\mathbf{y}_c \end{bmatrix} \tag{112}$$

The PI gains and corresponding Riccati matrix, $\mathbf{P}_a$, are obtained by solving a matrix Riccati equation [81] formulated in terms of the augmented state, $\mathbf{x}_a$, and the control deviation, $\tilde{\mathbf{u}}$. The weighting matrices $\mathbf{Q}_a$, $\mathbf{M}_a$, and $\mathbf{R}_a$, are designed using *implicit model following* (IMF), to induce the closed-loop system to follow the response of a model that satisfies established design criteria [82]. For simplicity, the ideal model, the augmented weighting matrices, and the corresponding gain matrices are obtained separately for the longitudinal and the lateral-directional models, as shown in Sections 4.1.2 and 4.1.3. The remainder of Chapter 4 deals with the initial specification of the PI neural network controller and its initialization.

4.1.2 Ideal Model

In implicit model following, the LQ law is used to induce the actual plant to follow the behavior of an ideal model specified by a linear system of the same order as the

linearized plant. The model displays desired performance characteristics that can be replicated using reasonable control usage, provided they are sufficiently close to those of the actual plant. In the case of an aircraft, the ideal performance is defined by stability and control characteristics, and handling qualities that allow the vehicle to perform its intended mission safely and in a manner that the pilot finds satisfactory. The extensive experimental studies conducted in this field have lead to findings that have been quantified into approved military specifications for different aircraft types and flight phases [83]. Table 2 summarizes the specifications pertaining to a light aircraft in a terminal flight phase (such as takeoff, approach, and landing) that requires accurate flight-path control and flying qualities that are adequate to accomplish the mission at hand. These criteria are taken into account in designing the ideal model to be followed by the business jet aircraft.

Table 2. Longitudinal and lateral military specifications for a Class I airplane in a terminal flight phase (Category C), requiring Level 1 flying qualities [83].

Longitudinal flying qualities	**Lateral flying qualities**
Phugoid mode	Roll mode
$\zeta_P > 0.04$	$\tau_{\text{roll}} \leq 1.0$
Short-period mode	Dutch roll mode
$0.35 \leq \zeta_{SP} \leq 1.30$	$\zeta_{DR} \geq 0.08$ $\zeta_{DR} \cdot \omega_{n_{DR}} \geq 0.15$ rad/s $\omega_{n_{DR}} \geq 0.4$ rad/s

The process of defining the ideal system is not an exact one; it can, however, be a decisive step in the linear control system design. As suggested by the flying qualities

specifications, it is convenient to consider the longitudinal and lateral-directional dynamics separately. Subsequently (Section 4.1.3), the longitudinal ideal model, $\mathbf{F}_{m_L}$, and the lateral-directional ideal model, $\mathbf{F}_{m_{LD}}$, can be used to design the linear control gains for the reduced systems $\{\mathbf{F}_L, \mathbf{G}_L\}_\kappa$ and $\{\mathbf{F}_{LD}, \mathbf{G}_{LD}\}_\kappa$, respectively. Both models are obtained based on aircraft linearized equations of motion and must meet the established flying qualities. Normally, the aircraft equations of motion are expressed in terms of stability derivatives which are a function of the aircraft's geometric and aerodynamic characteristics. In the case of the ideal model, the stability derivatives are chosen by considering the tradeoff between meeting desired specifications and abiding by the business jet characteristics.

4.1.2.1 Longitudinal Aircraft Model

The aircraft longitudinal dynamics are characterized by two basic modes of motion: the lightly-damped Phugoid mode and the highly-damped short period mode. The linearized longitudinal equations of motion of an airplane with fixed controls can be derived with respect to an inertial frame of reference and then expressed with respect to the state perturbation $\Delta\mathbf{x}_{L_\alpha} = [\Delta V\ \Delta\gamma\ \Delta q\ \Delta\alpha]^T$ in a simplified state space form:

$$\mathbf{F}_{L_\alpha} = \left[\begin{array}{cc:cc} TD_V & -g\cos\gamma_0 & TD_q & TD_\alpha \\ L_V/V_0 & (g/V_0)\sin\gamma_0 & L_q/V_0 & L_\alpha/V_0 \\ \hdashline M_V & 0 & M_q & M_\alpha \\ -L_V/V_0 & -(g/V_0)\sin\gamma_0 & (1-L_q/V_0) & -L_\alpha/V_0 \end{array}\right] \tag{113}$$

The coefficients of the state space model, referred to as stability derivatives, are defined according to an established convention. L_V represents the normalized derivative of the lift force with respect to velocity, TD_V represents the difference between the thrust and the drag derivatives with respect to velocity, $(T_V - D_V)$, and so on [84]. g is the

gravitational acceleration, α is the angle of attack, γ_0 and V_0 are the nominal path angle and velocity, respectively. The above is the most convenient form for investigating flying qualities based on the Jacobian matrices, because $\mathbf{F}_{L_\alpha}$ is in the most nearly block diagonal form and, hence, can be partitioned into Phugoid parameters and short-period parameters by neglecting off-block-diagonal terms [84].

As a result, the Phugoid and short period natural frequencies and damping ratios can be approximated by these conventional formulas,

$$\omega_{n_P} \approx \left[g L_V / V_0 \right]^{1/2} \tag{114}$$

$$\zeta_P \approx -TD_V / 2\omega_{n_P} \tag{115}$$

$$\omega_{n_{SP}} \approx \left[- M_\alpha \left(1 - L_q / V_0 \right) - M_q L_\alpha / V_0 \right]^{1/2} \tag{116}$$

$$\zeta_{SP} \approx \left(L_\alpha / V_0 - M_q \right) / 2\omega_{n_{SP}} \tag{117}$$

where ω_n is the mode's natural frequency. The approximations represent a more accurate depiction of the fourth-order model's (eq. 113) characteristics if the ideal model is assumed to be in level flight, i.e., $\gamma_0 = 0$, and the remaining cross-coupling terms approach zero. The following estimates can be obtained from the thirty-four linearized systems computed for the aircraft simulation over *OP* (Section4.1) : $L_q/V_0 = 0$, $TD_q = 0$, $M_V = 0$, and $L_V/V_0 = 2\ 10^{-4}$ (m^{-1}). As a consequence, the approximations in eq. 114-117 hold even when TD_α and L_α/V_0 are not zero.

The flying qualities specifications in Table 2 impose limitations on what can be considered acceptable Phugoid and short period damping ratios, ζ_P and ζ_{SP}; however, they are not stringent enough to determine the values of the remaining stability derivatives in eq. 113. From experience, it is known that good values for ζ_P and ζ_{SP} are

0.1 and 0.45, respectively. Based on these values and on eq. 114, ω_{n_P} is found to equal 0.0802 rad/s; thus, eq. 115 can be solved for the value of TD_V that produces $\zeta_P = 0.1$, i.e., –0.016 (s^{-1}). For the short period mode, the approximations leave more room for inference. A possible approach consists of letting the following stability derivatives equal to the median values of their respective distributions (obtained from $\{\mathbf{F}_L, \mathbf{G}_L\}_\kappa$): $M_\alpha = -5$, $L_\alpha/V_0 = -5$, and $TD_\alpha = -8$. In particular, TD_α affects neither the short period natural frequency nor the damping ratio; therefore, it can be chosen solely based on the aircraft's linearized models. The value of M_q varies considerably across the flight envelope; therefore, its ideal value is computed based on the short period specifications. Equation 116 is substituted into eq. 117 and, after a few iterations, it is found that $M_q = 1.7$ produces a damping ratio, ζ_{SP}, close to 0.45.

The ideal stability derivatives determined above are used with eq. 113 to form the longitudinal ideal model. However, $\mathbf{F}_{m_L}$ must be formulated in terms of the same state vector as the system to be controlled: $\Delta\mathbf{x}_L$. In purely longitudinal motion (level flight) the following relationship holds for the aircraft path angle, pitch angle, and angle of attack:

$$\gamma = \theta - \alpha \tag{118}$$

Therefore, the following matrix defines the desired transformation, $\Delta\mathbf{x}_L = \mathbf{T}_{\mathbf{x}_L}\Delta\mathbf{x}_{L_\alpha}$,

$$\mathbf{T}_{\mathbf{x}_L} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix} \tag{119}$$

and the stick-fixed equations of motion can be formulated with respect to the original state:

$$\Delta\dot{\mathbf{x}}_L = \left(\mathbf{T}_{\mathbf{x}_L}\mathbf{F}_{L_\alpha}\mathbf{T}_{\mathbf{x}_L}^{-1}\right)\Delta\mathbf{x}_L \tag{120}$$

Finally, the ideal longitudinal model used in the IMF design is,

$$\mathbf{F}_{m_L} = \begin{bmatrix} -0.016 & -1.8066 & 0 & -8 \\ 2\cdot 10^{-4} & -0.5 & 0 & 0.5 \\ 0 & 5 & -1.7 & -5 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{121}$$

and its roots are plotted in Fig. 17a. Although the coefficients of the equations of motions depend on V_0, for simplicity this ideal model is held fixed throughout the flight envelope and is used to design the PI gains for all thirty-four linearized systems.

4.1.2.2 Aircraft Lateral-directional Model

Three characteristic motions can be identified in the lateral dynamics of the aircraft; they are the slowly convergent or divergent spiral mode, the highly convergent rolling mode, and the lightly damped, low-frequency Dutch roll mode. Table 2 summarizes the limitations imposed on the lateral modes' characteristics. Furthermore, from experience, it is known that desirable values of Dutch roll natural frequency, $\omega_{n_{DR}}$, and damping ratio, ζ_{DR}, are 3.6 rad/s and 0.6, respectively. The specifications for the spiral stability involve the minimum time to double the bank angle amplitude following an initial disturbance in bank angle, μ, of up to 20 deg, and can be verified following the model design. The linearized stick-fixed lateral equations of motion are used to construct $\mathbf{F}_{m_{LD}}$. They are expressed directly in terms of $\Delta\mathbf{x}_{LD}$, using the state-space matrix,

$$\mathbf{F}_{LD} = \left[\begin{array}{cc:cc} N_r & N_\beta & N_p & 0 \\ -\left(1-Y_r/V_0\right) & Y_\beta/V_0 & Y_p/V_0 & \dfrac{g}{V_0}\cos\gamma_0 \\ \hdashline L_r & L_\beta & L_p & 0 \\ 0 & 0 & 1 & 0 \end{array}\right] \tag{122}$$

in a form that already is suitable for investigating lateral-directional flying qualities. The coefficients of the Jacobian matrix also are defined according to convention [84].

If the lateral dynamics are approximated by further reduced first and second-order models, natural frequency and damping ratio approximations can be obtained for the relevant modes of motion from the roots of the corresponding characteristic equations. The undamped Dutch roll parameters are given by the following expressions:

$$\omega_{n_{DR}} = \sqrt{\frac{Y_\beta}{V_0} N_r - \frac{Y_r}{V_0} N_\beta + N_\beta} \tag{123}$$

$$\zeta_{DR} = -\frac{1}{2\omega_{n_{DR}}} \left(\frac{Y_\beta}{V_0} + N_r \right) \tag{124}$$

The rolling mode can be approximated by a single-degree-of-freedom motion (or first-order ordinary differential equation), such that the roll time constant, τ_{roll}, is obtained in terms of the roll damping L_p:

$$\tau_{\text{roll}} = -\frac{1}{L_p} \tag{125}$$

From the characteristic equation of the system in eq. 122, it is easily shown that the above approximations also hold for the full, fourth-order lateral system, provided L_r, N_p, and g/V_0 approach zero. This can be considered as a reasonable assumption, because the same coefficients also are small for the matrices in the set $\{\mathbf{F}_{LD}\}_\kappa$.

Since the desired value of $\omega_{n_{DR}}$ is known, eq. 124 can be used to determine the desired value of the sum ($Y_\beta/V_0 + N_r$), i.e., –4.32 rad/s. A second condition can be derived from eq. 123: Y_r/V_0 usually is small or equal to zero, thus ($N_r Y_\beta/V_0 + N_\beta$) = 12.96 (rad/s)^2. N_β varies considerably across the flight envelope and is purposely chosen larger than that of the actual linearized systems, i.e., $N_\beta = 8$ s^{-2}, to provide greater directional stability. Subsequently, the values of Y_β/V_0 and N_r that satisfy both the natural

frequency and the damping ratio conditions are uniquely determined as N_r = -2.4 s^{-1} and Y_β/V_0 = -1.8 s^{-1}. The high absolute value of the yaw rate damping, N_r, improves the aircraft response; also, as is true for most airplanes, the following holds for the chosen coefficients: $|N_r| > |Y_\beta/V_0|$. This set of stability derivatives not only satisfies the Dutch roll specifications, but also produces a model that has a more stable directional motion, without differing considerably from the actual systems.

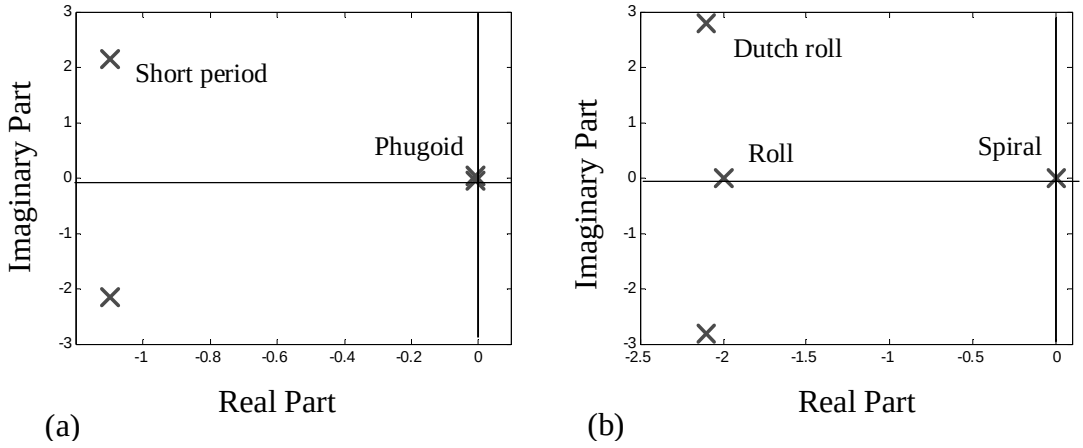

Figure 17. Characteristic roots of the longitudinal ideal model, $\mathbf{F}_{m_L}$, (a) and of the lateral-directional ideal model, $\mathbf{F}_{m_{LD}}$, (b).

The remaining coefficients are chosen equal to the best values in $\{\mathbf{F}_{LD}\}_\kappa$, that is, the dihedral effect is chosen as small as possible, i.e., L_β = –2 s^{-2}, since with N_r it provides for sufficient spiral stability. L_p is chosen equal to –2 s^{-1} to provide for the largest roll damping. Thus, the lateral-directional ideal model consists of the following matrix,

$$\mathbf{F}_{m_{LD}} = \begin{bmatrix} -2.4 & 8 & 0 & 0 \\ -1 & -1.8 & 0 & 0 \\ 0 & -2 & -2 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{126}$$

whose roots are shown in Fig. 17b. The next section demonstrates how these models are used to establish the control system's objectives through the weighting matrices $\mathbf{Q}_a$, $\mathbf{M}_a$, and $\mathbf{R}_a$, in eq. 107. The described procedure constitutes an efficient approach to designing these weighting matrices that involves both engineering practice and intuition. Other established methods and criteria also are acceptable and can be directly substituted to the next design step, without influencing the remainder of the design process.

4.1.3 Implicit Model Following

In order for the actual system to follow the behavior of the ideal model, defined in terms of the Jacobian matrix $\mathbf{F}_m$, its state rate must approach that of the model:

$$\Delta\dot{\mathbf{x}}_m(t) = \mathbf{F}_m \Delta\mathbf{x}_m(t) \tag{127}$$

The cost function to be minimized can be formulated in terms of the state rate of both systems,

$$J = \lim_{t_f \to \infty} \frac{1}{2} \int_0^{t_f} \left\{ \left[\Delta\dot{\mathbf{x}}(\tau) - \Delta\dot{\mathbf{x}}_m(\tau)\right]^T \mathbf{Q}_m \left[\Delta\dot{\mathbf{x}}(\tau) - \Delta\dot{\mathbf{x}}_m(\tau)\right] \right\} d\tau \tag{128}$$

and is equivalent to eq. 15, provided the following weighting matrices are used:

$$\mathbf{Q} = (\mathbf{F} - \mathbf{F}_m)^T \mathbf{Q}_m (\mathbf{F} - \mathbf{F}_m) \tag{129}$$

$$\mathbf{M} = (\mathbf{F} - \mathbf{F}_m)^T \mathbf{Q}_m \mathbf{G} \tag{130}$$

$$\mathbf{R} = \mathbf{G}^T \mathbf{Q}_m \mathbf{G} + \mathbf{R}_0 \tag{131}$$

F and **G** refer to the state space matrices of the system to be controlled. $\mathbf{R}_0$ is a constant matrix that represents the separate cost of control, and $\mathbf{Q}_m$ is a weighting matrix for the state-rate errors. Perfect model following can be accomplished only when the following condition is satisfied,

$$\left(\mathbf{G}\mathbf{G}^{PI} - \mathbf{I}_n\right)\left(\mathbf{F} - \mathbf{F}_m\right) = 0 \tag{132}$$

as shown in [85]. This criterion formalizes the pragmatism followed in Section 4.1.2 for the ideal model design. From eq. 132, it is easily seen that the closer $\mathbf{F}_m$ is to $\mathbf{F}$, the less control usage is needed to follow the model closely.

Since the longitudinal and lateral-directional dynamics can be decoupled, eq. 129-131 are used to compute longitudinal weighting matrices,

$$\mathbf{Q}_L = \left(\mathbf{F}_L - \mathbf{F}_{m_L}\right)^T \mathbf{Q}_{m_L} \left(\mathbf{F}_L - \mathbf{F}_{m_L}\right) \tag{133}$$

$$\mathbf{M}_L = \left(\mathbf{F}_L - \mathbf{F}_{m_L}\right)^T \mathbf{Q}_{m_L} \mathbf{G}_L \tag{134}$$

$$\mathbf{R}_L = \mathbf{G}_L^T \mathbf{Q}_{m_L} \mathbf{G}_L + \mathbf{R}_{0_L} \tag{135}$$

and lateral weighting matrices,

$$\mathbf{Q}_{LD} = \left(\mathbf{F}_{LD} - \mathbf{F}_{m_{LD}}\right)^T \mathbf{Q}_{m_{LD}} \left(\mathbf{F}_{LD} - \mathbf{F}_{m_{LD}}\right) \tag{136}$$

$$\mathbf{M}_{LD} = \left(\mathbf{F}_{LD} - \mathbf{F}_{m_{LD}}\right)^T \mathbf{Q}_{m_{LD}} \mathbf{G}_{LD} \tag{137}$$

$$\mathbf{R}_{LD} = \mathbf{G}_{LD}^T \mathbf{Q}_{m_{LD}} \mathbf{G}_{LD} + \mathbf{R}_{0_{LD}} \tag{138}$$

based on the respective models, $\mathbf{F}_{m_L}$ and $\mathbf{F}_{m_{LD}}$, designed in Section 4.1.2. When the perfect model following criteria (eq. 132) cannot be satisfied exactly, as is the case here, the weighting matrices $\mathbf{Q}_{m_L}$, $\mathbf{R}_{0_L}$, $\mathbf{Q}_{m_{LD}}$, and $\mathbf{R}_{0_{LD}}$ are chosen by the designer to achieve the best possible response, as described below.

In order to achieve PI compensation, an augmented state that includes the output error integral, ξ, is considered in the cost function (eq. 107) to be minimized. Reference [82] shows that the PI augmented weighting matrices can be defined in terms of the implicit model following matrices, eq.129-131, as

$$\mathbf{Q}_a = \begin{bmatrix} \mathbf{Q} & \mathbf{0} \\ \mathbf{0} & \mathbf{Q}_\xi \end{bmatrix} \tag{139}$$

$$\mathbf{M}_a = \begin{bmatrix} \mathbf{M} \\ \mathbf{0} \end{bmatrix} \tag{140}$$

and $\mathbf{R}_a = \mathbf{R}$. The weighting on the output error integral is represented by $\mathbf{Q}_\xi$. It follows that the longitudinal augmented matrices, $\mathbf{Q}_{a_L}$, $\mathbf{M}_{a_L}$, and $\mathbf{R}_{a_L}$ are formulated in terms of $\mathbf{Q}_L$, $\mathbf{Q}_{\xi_L}$, $\mathbf{M}_L$, and $\mathbf{R}_L$, and that the lateral augmented matrices can be similarly defined. ξ_L and ξ_{LD} are the time integrals of the longitudinal and lateral output errors, $\tilde{\mathbf{y}}_L \equiv [\tilde{V} \;\; \tilde{\gamma}]^T$ and $\tilde{\mathbf{y}}_{LD} \equiv [\tilde{\mu} \;\; \tilde{\beta}]^T$, respectively.

The PI gains for every system in the longitudinal design set $\{\mathbf{F}_L, \mathbf{G}_L\}_\kappa$ are computed by solving a Riccati equation (eq. 22), using the MATLAB built-in function lqr (part of the Control Systems Toolbox). The longitudinal PI gains $\mathbf{C}_{a_L}$ and $\mathbf{P}_{a_L}$ are computed based on the weighting matrices, $\mathbf{Q}_{a_L}$, $\mathbf{M}_{a_L}$, and $\mathbf{R}_{a_L}$, and on the corresponding longitudinal state space model defined by $\mathbf{F}_L$, $\mathbf{G}_L$, $\mathbf{H}_{\mathbf{x}_L}$, and $\mathbf{H}_{\mathbf{u}_L}$. According to the longitudinal output definition, the output matrices are,

$$\mathbf{H}_{\mathbf{x}_L} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \tag{141}$$

and $\mathbf{H}_{\mathbf{u}_L} = \mathbf{0}$. The longitudinal feedback and command-integral gain matrices, $\mathbf{C}_{B_L}$ and $\mathbf{C}_{I_L}$, are determined from $\mathbf{C}_{a_L}$, as indicated by eq. 109. Subsequently, eq. 110 can be used to compute the longitudinal forward gain matrix $\mathbf{C}_{F_L}$.

The above procedure implies that the outcome of the linear design is completely specified by the set of matrices $\mathbf{Q}_{m_L}$, $\mathbf{R}_{0_L}$, and $\mathbf{Q}_{\xi_L}$. Typically, these are diagonal

matrices whose elements are chosen by considering the role and the magnitude of the variables they weigh. The control laws corresponding to different sets of matrices are tested by simulating the following closed-loop system,

$$\begin{bmatrix} \Delta\dot{\mathbf{x}}_L(t) \\ \dot{\boldsymbol{\xi}}_L(t) \end{bmatrix} = \begin{bmatrix} \left(\mathbf{F}_L - \mathbf{G}_L\mathbf{C}_{B_L}\right) & -\mathbf{G}_L\mathbf{C}_{I_L} \\ \mathbf{H}_{\mathbf{x}_L} & \mathbf{H}_{\mathbf{u}_L} \end{bmatrix} \begin{bmatrix} \Delta\mathbf{x}_L(t) \\ \boldsymbol{\xi}_L(t) \end{bmatrix} + \begin{bmatrix} \left(\mathbf{G}_L\mathbf{C}_{B_L} - \mathbf{F}_L\right)\mathbf{B}_{12} \\ -\mathbf{H}_{\mathbf{x}_L}\mathbf{B}_{12} \end{bmatrix} \Delta\mathbf{y}_c(t) \quad (142)$$

obtained by applying the PI control law (eq. 109) to the longitudinal linearized system for which the gains have been designed. Because the same matrices $\mathbf{Q}_{m_L}$, $\mathbf{R}_{0_L}$, and $\mathbf{Q}_{\xi_L}$ are used for all points in *OP*, the designs they produce are tested throughout the envelope in Fig. 15. The flight condition (V_0, H_0) = (120 m/s, 3 000 m) is considered for illustration. The roots of eq. 142 are compared to those of the ideal model, $\mathbf{F}_{m_L}$, and to those of the open loop system, $\mathbf{F}_L$, to consider the effectiveness of the implicit model following design. The step-input aircraft response also is used to evaluate the performance of the resulting controller.

Figure 18 shows the root comparison for two possible designs obtained with different weighting matrices; "diag[•]" denotes the placement of a vector on the diagonal of a zero matrix. The closed-loop system possesses two additional characteristic roots that correspond to the integral compensation. In the first example (Fig. 18.a), the short-period roots are almost unaffected by the closed-loop design, while the phugoid roots are worsen, as they are further from the ideal roots than the open-loop system's roots. The second design (Fig. 18.b), improves the short-period roots by pushing them closer to those of the ideal model, and leaves the phugoid roots almost unaffected. However, both examples require excessive control usage, as shown in Fig. 20. After carefully comparing the characteristic roots and the step-command-input response of the closed-

loop system (eq. 142) obtained with several choices of the weighting matrices (including the examples shown in Fig.s 18 and 20) the following weighting matrices were chosen:

$$\mathbf{Q}_{m_L} = \text{diag}[0.5\ 10\ 5\ 1] \tag{143}$$

$$\mathbf{R}_{0_L} = \text{diag}[0.5\ 5] \tag{144}$$

$$\mathbf{Q}_{\xi_L} = \text{diag}[1\ 10^3] \tag{145}$$

The roots of the corresponding closed-loop system are shown in Fig. 19.

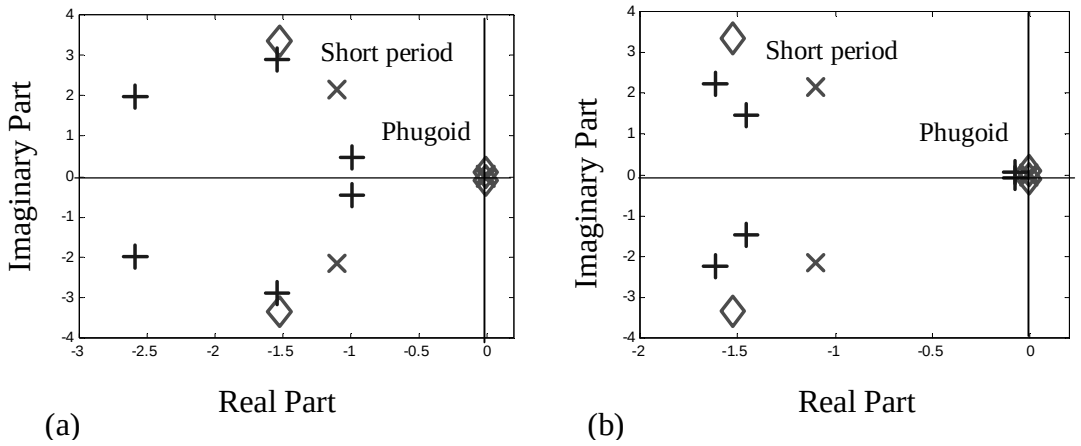

Figure 18. Characteristic roots of the longitudinal model (×), open-loop system (◊), and closed-loop system (+), obtained with two examples of weighting matrices sets: $\mathbf{Q}_{m_L}$ = diag[0.01 0.01 0.01 0.01], $\mathbf{R}_{0_L}$ = **0**, and $\mathbf{Q}_{\xi_L}$ = diag[1 1] (a), and $\mathbf{Q}_{m_L}$ = diag[10^{-3} 10^2 20 0.01], $\mathbf{R}_{0_L}$ = diag[1 1], and $\mathbf{Q}_{\xi_L}$ = diag[0.1 0.1] (b).

The rate at which the command-input error is suppressed also plays a role. The chosen set of matrices accomplishes the best compromise between matching the ideal model behavior and achieving feasible dynamic compensation. Figure 20 shows the controlled system's response to a common step command input. These results show that while the first example achieves relatively close matching of the short period root (Fig. 18a), it requires unreasonable control usage (Fig. 20b). The second example also requires

an excessive amount of throttle (Fig. 20b) and only matches the phugoid root closely (Fig. 18b). The chosen set of matrices (eq. 143-145) produce both a relatively close matching of the ideal roots and a reasonable command-input response.

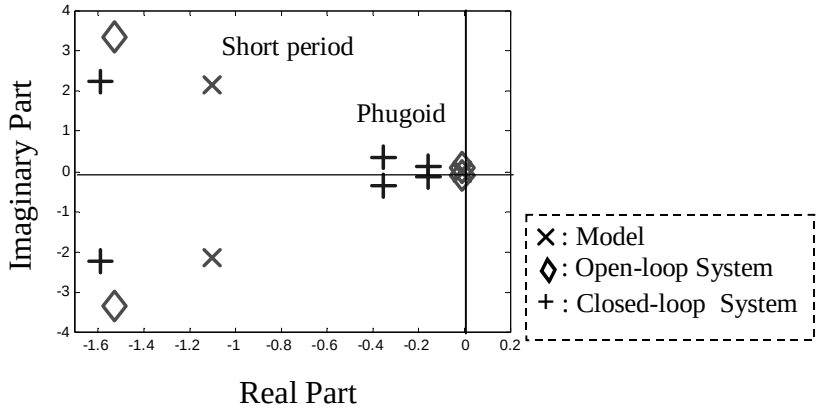

Figure 19. Characteristic roots comparison at the design point (V_0, H_0) = (120 m/s, 3 000 m), achieved with the actual weighting matrices used in all longitudinal PI designs.

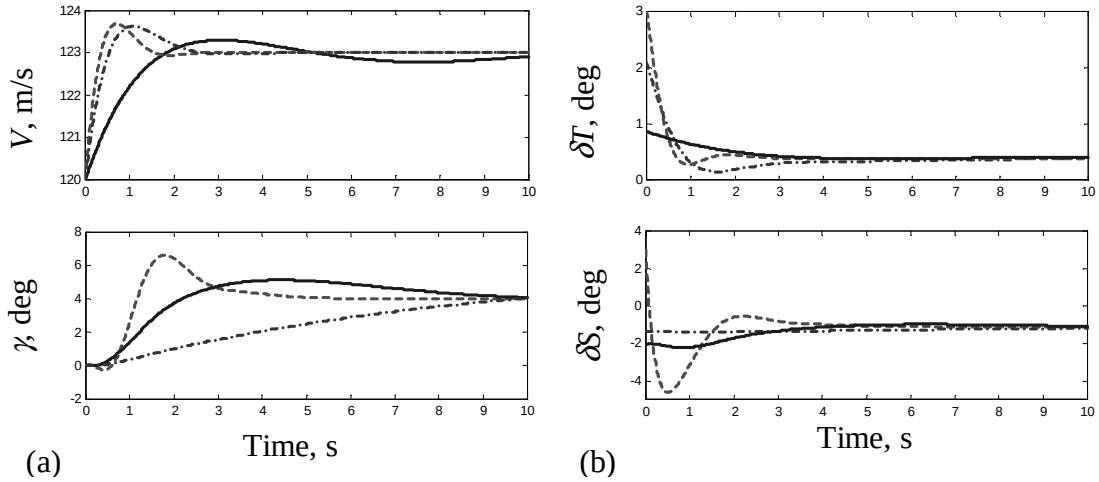

Figure 20. Longitudinal state (a) and control (b) response to a 3-m/s velocity and 4-deg path angle step command input, at the design point (V_0, H_0) = (120 m/s, 3 000 m). The actual design (solid line) is compared to a design with $\mathbf{Q}_{m_L}$ = diag[0.01 0.01 0.01 0.01], $\mathbf{R}_{0_L} = \mathbf{0}$, and $\mathbf{Q}_{\xi_L}$ = diag[1 1] (dashed line), and to a design with $\mathbf{Q}_{m_L}$ = diag[10^{-3} 10^{2} 20 0.01], $\mathbf{R}_{0_L}$ = diag[1 1], and $\mathbf{Q}_{\xi_L}$ = diag[0.1 0.1] (dashed-dotted line).

The lateral PI gains are determined through the same methodology used for the longitudinal gains, except they are based on the weighting matrices $\mathbf{Q}_{a_{LD}}$, $\mathbf{M}_{a_{LD}}$, and $\mathbf{R}_{a_{LD}}$ and on the systems $\{\mathbf{F}_{LD}, \mathbf{G}_{LD}\}_{\kappa}$. The lateral-directional output matrices are,

$$\mathbf{H}_{\mathbf{x}_{LD}} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{bmatrix} \tag{146}$$

and $\mathbf{H}_{\mathbf{u}_{LD}} = \mathbf{0}$. Therefore, once the model $\mathbf{F}_{m_{LD}}$ has been specified, as in Section 4.1.2, the lateral linear controllers depend exclusively on the choice of $\mathbf{Q}_{m_{LD}}$, $\mathbf{R}_{0_{LD}}$, and $\mathbf{Q}_{\xi_{LD}}$. The lateral equivalent of eq. 142 is used to compare designs obtained with different weighting matrices. Figures 21 and 22 show that, at a given operating point, the matrices,

$$\mathbf{Q}_{m_{LD}} = \text{diag}[10 \;\; 1 \;\; 10 \;\; 1] \tag{147}$$

$$\mathbf{R}_{0_{LD}} = \mathbf{0} \tag{148}$$

$$\mathbf{Q}_{\xi_{LD}} = \text{diag}[10 \;\; 0.1] \tag{149}$$

offers adequate matching of the model roots and good output-error damping. The linear gains based on the chosen set of weighting matrices (eq. 147-149) provide for appropriate command-input response with reasonable control usage, as shown in Fig. 23. The results shown in Fig. 21 and 22 for a representative point (V_0, H_0) = (120 m/s, 3 000 m), are typical throughout *OP* (Fig. 15).

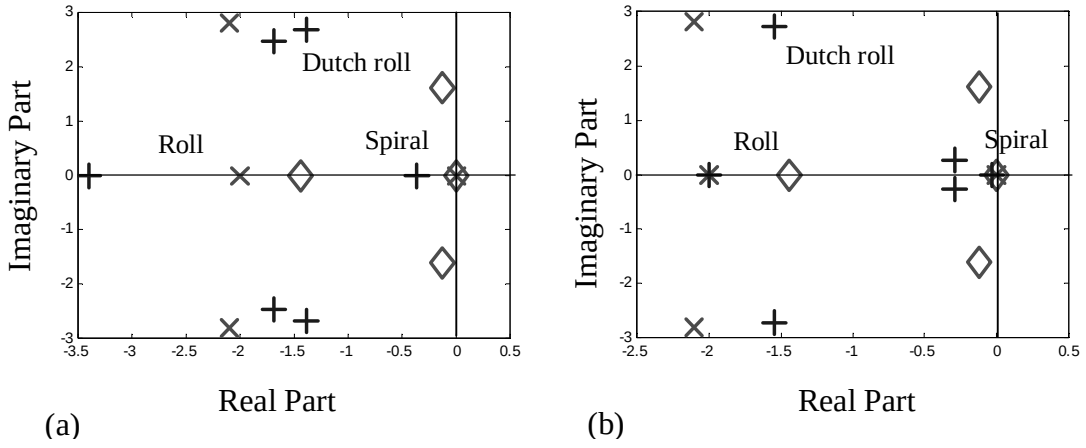

Figure 21. Characteristic roots of the lateral model (×), open-loop system (◊), and closed-loop system (+), obtained with two examples of weighting matrices sets: $\mathbf{Q}_{m_{LD}}$ = diag[0.01 0.01 0.01 0.01], $\mathbf{R}_{0_{LD}}$ = **0**, and $\mathbf{Q}_{\xi_{LD}}$ = diag[1 1] [82] (a), and $\mathbf{Q}_{m_{LD}}$ = diag[1 10 1 10^{-7}], $\mathbf{R}_{0_{LD}}$ = **0**, and $\mathbf{Q}_{\xi_{LD}}$ = diag[0.1 0.1] (b).

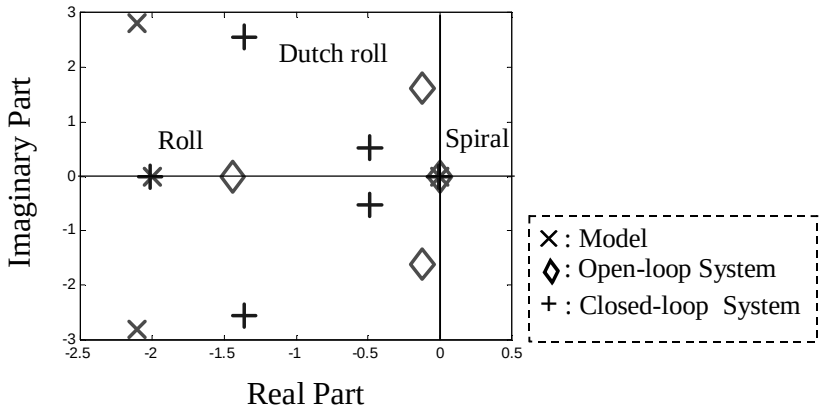

Figure 22. Characteristic roots comparison at the design point (V_0, H_0) = (120 m/s, 3 000 m), achieved with the actual weighting matrices used in all lateral PI designs.

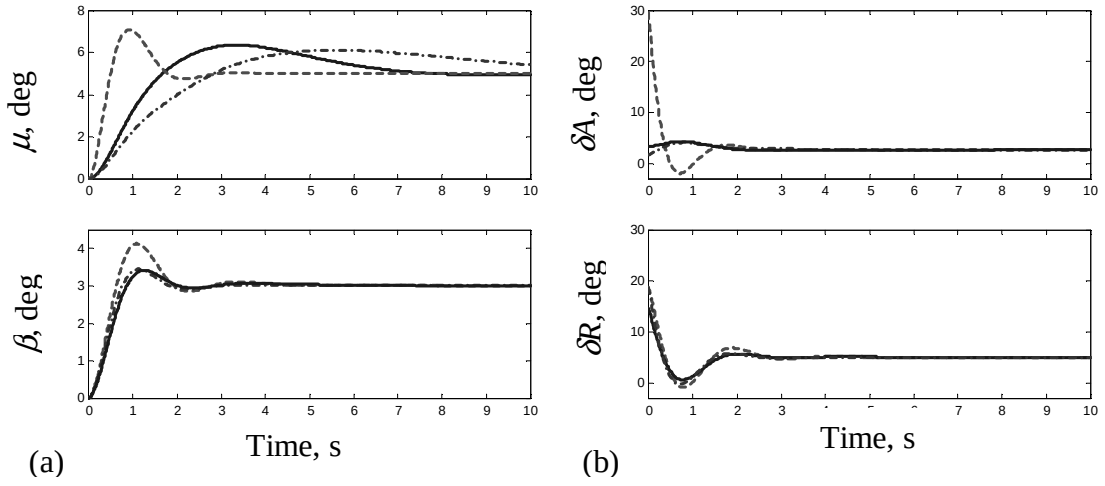

Figure 23. Lateral state (a) and control (b) response to a 5-deg bank angle and 3-deg sideslip step command input, at the design point (V_0, H_0) = (120 m/s, 3 000 m). The actual design (solid line) is compared to designs with weighting matrices $\mathbf{Q}_{m_{LD}}$ = diag[0.01 0.01 0.01 0.01], $\mathbf{R}_{0_{LD}} = \mathbf{0}$, and $\mathbf{Q}_{\xi_{LD}}$ = diag[1 1] (dashed line), and $\mathbf{Q}_{m_{LD}}$ = diag[1 10 1 10^{-7}], $\mathbf{R}_{0_{LD}} = \mathbf{0}$, and $\mathbf{Q}_{\xi_{LD}}$ = diag[0.1 0.1] (dashed and dotted line).

After the decoupled PI controllers have been tested for both reduced linear systems, they can be used to initialize the neural network controller described in the following section. The linear control laws designed above can be summarized by the longitudinal set of matrices $\{\mathbf{C}_{B_L}, \mathbf{C}_{I_L}, \mathbf{C}_{F_L}, \mathbf{P}_{a_L}\}_\kappa$ and by the lateral set of matrices $\{\mathbf{C}_{B_{LD}}, \mathbf{C}_{I_{LD}}, \mathbf{C}_{F_{LD}}, \mathbf{P}_{a_{LD}}\}_\kappa$ corresponding to the linear systems $\{\mathbf{F}_L, \mathbf{G}_L\}_\kappa$ and $\{\mathbf{F}_{LD}, \mathbf{G}_{LD}\}_\kappa$, respectively (where κ= 1, …, 34). Producing the linear design is, in itself, a time-consuming and often challenging task. Considering that many of these designs already are available in the industry and that they do entail considerable engineering knowledge and wisdom, it certainly seems important to incorporate them in the adaptive control system.

4.2 Proportional-Integral Neural Network Control

The PI Neural Network Controller (PINN) is the nonlinear structure motivated by the linear PI Controller. It is obtained by replacing each linear gain with a nonlinear neural network, $\mathbf{NN}_B$ for $\mathbf{C}_B$, $\mathbf{NN}_F$ for $\mathbf{C}_F$, and $\mathbf{NN}_I$ for $\mathbf{C}_I$, as shown in Fig. 24. The input-output structure is unchanged, and the command error is integrated, as in the linear system, to produce Type-1 response [80]. Performance targets for these networks are established locally by the respective linear gains that the networks replace. The output, $\mathbf{y}_s$, and set point, ($\mathbf{x}_c$, $\mathbf{u}_c$), are computed for the fully-coupled nonlinear system. The *Scheduling Variable Generator* (*SVG*) contains algebraic equations that produce auxiliary inputs to the neural networks based on the command input and an exogenous vector, **e**, of measured variables. The *Command State Generator* (*CSG*) uses the aircraft's kinematic equations to provide secondary elements of the state that are compatible with $\mathbf{y}_c$. It will be described in Section 4.4, because it closely interacts with the forward neural network, $\mathbf{NN}_F$.

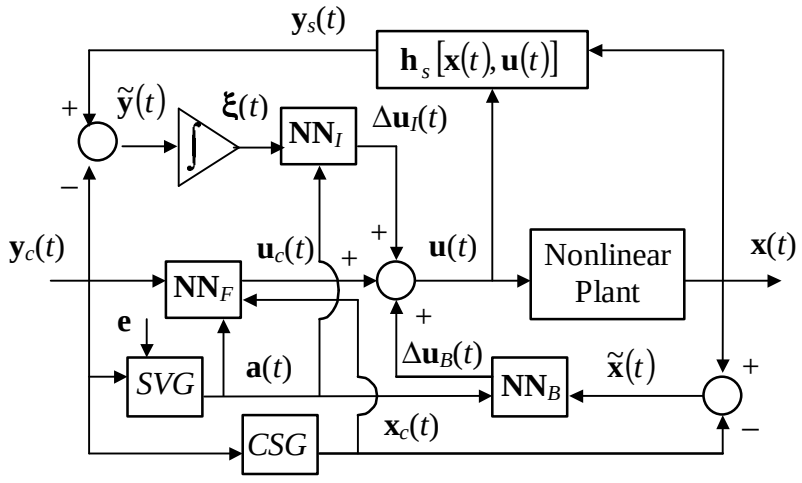

Figure 24. Nonlinear proportional-integral neural network control system.

All auxiliary inputs to the neural networks are included in the scheduling vector and can be updated in real time by the on-board instrumentation, i.e., $\mathbf{a}(t) = [V(t)\ H(t)]^T$. In addition, the networks $\mathbf{NN}_B$, $\mathbf{NN}_F$, and $\mathbf{NN}_I$ are provided with the state deviation, $\tilde{\mathbf{x}}(t)$, the command input, $\mathbf{y}_c(t)$, and the command-error integral, $\boldsymbol{\xi}(t)$, respectively. Each of these vector-output networks contributes to the total control, $\mathbf{u}(t)$,

$$\mathbf{u}(t) = \mathbf{u}_C(t) + \Delta\mathbf{u}_B(t) + \Delta\mathbf{u}_I(t) \tag{150}$$

where,

$$\begin{aligned} \mathbf{u}_c(t) &= \mathbf{NN}_F[\mathbf{y}_c(t),\ \mathbf{a}(t)] \equiv \mathbf{z}_c(t) \\ \Delta\mathbf{u}_B(t) &= \mathbf{NN}_B[\tilde{\mathbf{x}}(t),\ \mathbf{a}(t)] \equiv \mathbf{z}_B(t) \\ \Delta\mathbf{u}_I(t) &= \mathbf{NN}_I[\boldsymbol{\xi}(t),\ \mathbf{a}(t)] \equiv \mathbf{z}_I(t) \end{aligned} \tag{151}$$

As previously indicated, $\mathbf{z}$ denotes the network output. In order to perform as well as an equivalent linear structure, the nonlinear controller must satisfy a set of requirements that are imposed on the neural network equations.

As anticipated in Section 2.3, the classical and neural control systems can be synthesized by recognizing that the gradients of the nonlinear neural networks must correspond to the linear gains they replace, at selected nominal conditions (e.g., *OP*). Computational feedforward neural networks of the type shown in Fig. 5 with one hidden-layer of sigmoidal functions are employed for all neural blocks. Each vector-output neural network is obtained by joining scalar-output networks, as inferred by the input/output relation being modeled. The remainder of this chapter shows how each neural block is algebraically designed and trained, based on requirements generated from the linear PI control laws designed in Section 4.1.

4.3 Feedback and Command-Integral Neural Networks

The feedback neural network must provide for regulation in the control system and produce zero output when the state, $\mathbf{x}(t)$, approaches the commanded state, $\mathbf{x}_c(t)$, i.e., $\tilde{\mathbf{x}}(t) \rightarrow \mathbf{0}$. Therefore, at each nominal operating point $\kappa \in OP$ considered in the linear control design, the following must hold:

$$\mathbf{z}_B\left[\tilde{\mathbf{x}}(t),\ \mathbf{a}(t)\right]_\kappa = \mathbf{z}_B\left(\mathbf{0},\ \mathbf{a}^\kappa\right) = \mathbf{0} \tag{152}$$

The deviation from the commanded control, $\mathbf{u}_c(t)$, equals the sum of the feedback and command-integral neural networks' outputs, from eq. 150:

$$\tilde{\mathbf{u}}(t) = \Delta\mathbf{u}_B(t) + \Delta\mathbf{u}_I(t) = \mathbf{z}_B(t) + \mathbf{z}_I(t) \tag{153}$$

Differentiating eq. 109 and eq. 153 with respect to the neural network input $\tilde{\mathbf{x}}(t)$ reveals that, at the κ^{th} nominal flight conditions, the gradients of the feedback neural network, $\mathbf{NN}_B$, can be obtained from the feedback matrix designed for those conditions,

$$\left.\frac{\partial \mathbf{z}_B(t)}{\partial \tilde{\mathbf{x}}(t)}\right|_\kappa = \left.\frac{\partial \tilde{\mathbf{u}}(t)}{\partial \tilde{\mathbf{x}}(t)}\right|_\kappa = -\mathbf{C}_B^\kappa \tag{154}$$

where $\tilde{\mathbf{x}}(t) = \mathbf{0}$ and $\mathbf{a}(t) = \mathbf{a}^\kappa$.

A total of four scalar neural networks, of the type shown in Fig. 5, is used to form $\mathbf{NN}_B$, i.e., one for each control element:

$$\begin{bmatrix} \Delta\delta T(t) \\ \Delta\delta S(t) \\ \Delta\delta A(t) \\ \Delta\delta R(t) \end{bmatrix}_B = \begin{bmatrix} NN_{B_{L_1}}\left[\tilde{\mathbf{x}}_L(t),\ \mathbf{a}(t)\right] \\ NN_{B_{L_2}}\left[\tilde{\mathbf{x}}_L(t),\ \mathbf{a}(t)\right] \\ NN_{B_{LD_1}}\left[\tilde{\mathbf{x}}_{LD}(t),\ \mathbf{a}(t)\right] \\ NN_{B_{LD_2}}\left[\tilde{\mathbf{x}}_{LD}(t),\ \mathbf{a}(t)\right] \end{bmatrix} \tag{155}$$

Initially, these neural networks are decoupled and compute their respective outputs independently of each others. In Section 5.1.1, they will be joined algebraically to form a

vector-output network with equivalent off-line performance, that is further trained on-line. The two longitudinal feedback neural networks are fed with the same input, which includes the longitudinal state deviation, $\tilde{\mathbf{x}}_L$, defined in Section 4.1. The lateral-directional state deviation, $\tilde{\mathbf{x}}_{LD}$, is part of the input to the lateral feedback networks that produce the lateral portion of the control vector $\Delta\mathbf{u}_B$ in eq. 155.

A gradient-based training set can be obtained for each of the scalar networks in eq. 155, from the feedback requirements in eq. 152 and 154. Clearly, the output condition in eq. 152 extends to all feedback scalar neural networks, because $\tilde{\mathbf{x}} = [\tilde{\mathbf{x}}_L^T \;\; \tilde{\mathbf{x}}_{LD}^T]^T$: $z_B(0, \mathbf{a}^\kappa) = 0$, for $\forall\ \kappa$. Also, known gradient vectors can be obtained from the longitudinal and lateral feedback gain matrices computed in Section 4.1.3,

$$\begin{aligned}
\mathbf{c}_{B_{L_1}}^\kappa &= \mathbf{C}_{B_L}^\kappa (1,\ \bullet)^T \\
\mathbf{c}_{B_{L_2}}^\kappa &= \mathbf{C}_{B_L}^\kappa (2,\ \bullet)^T \\
\mathbf{c}_{B_{LD_1}}^\kappa &= \mathbf{C}_{B_{LD}}^\kappa (1,\ \bullet)^T \\
\mathbf{c}_{B_{LD_2}}^\kappa &= \mathbf{C}_{B_{LD}}^\kappa (2,\ \bullet)^T
\end{aligned} \tag{156}$$

where the argument $(l, \bullet)$ refers to the l^{th}-row in the matrix. These vectors contain the partial-derivative information defined by $\mathbf{c}^k$ in eq. 38, pertaining to $NN_{B_{L_1}}$, $NN_{B_{L_2}}$, $NN_{B_{LD_1}}$, and $NN_{B_{LD_2}}$, respectively. It follows that a training set of the type described in Section 3.1.1 can be defined for each feedback scalar neural network. Every training set is formed using the gradient vectors in eq. 156 and the scheduling vectors $\{\mathbf{a}\}_\kappa$ corresponding to all thirty-four points in *OP*. For example, the training set for $NN_{B_{L_1}}$ can be formulated as $\{[\mathbf{0}\ \mathbf{a}^{\kappa^T}]^T, 0, \mathbf{c}_{B_{L_1}}^\kappa\}_{\kappa = 1, \ldots, 34}$.

Therefore, the gradient-based algebraic procedure described in Section 3.1.1 can be used to determine the weights of the neural networks in eq. 155. Exact matching of the training data is obtained by designing each scalar network with thirty-four nodes in its hidden layer. For each of these networks, the parameters to be determined consist of the input weights associated with the state-deviation input (such as $\tilde{\mathbf{x}}_L$ or $\tilde{\mathbf{x}}_{LD}$), $\mathbf{W}_{\tilde{\mathbf{x}}}$, the input weights associated with the scheduling-vector input, $\mathbf{W_a}$, the output weights, **v**, and the biases, **d** and *b*. The linear systems in eq. 58, 59, and 63 are used to determine these weights, according to the algorithm detailed in Appendix B. Figure 25 shows the final architecture used for $NN_{B_{L_1}}$, where $\mathbf{W} = [\,\mathbf{W}_{\tilde{\mathbf{x}}} \,|\, \mathbf{W_a}\,]$; a similar one is used for all scalar feedback neural networks, with inputs and outputs defined as in eq. 155.

The final architecture is motivated by the pre-training phase, always providing zero output for zero state deviations. The direct contribution of the scheduling variables to the output is subtracted using a mirror image of the initialized network (with zero state perturbations), in order to eliminate any bias they might produce away from nominal training conditions. However, the effect of the scheduling variables with respect to the network gradients remains unaltered and, as expected, **a** schedules the gain interpolation across the flight envelope. Depending on the units of the input and output elements, the neural network gradients may be too sensitive to deviations from the nominal $\tilde{\mathbf{x}}$-input, i.e. **0**. In this case, $\tilde{\mathbf{x}}$ and $\tilde{\mathbf{u}}$ can be rescaled simply by multiplying $\mathbf{W}_{\tilde{\mathbf{x}}}$ by a small factor, e.g., $f_{\tilde{\mathbf{x}}} = 10^{-7}$, and **v** by its inverse, $1/f_{\tilde{\mathbf{x}}}$. From the weight equations, it can be seen that this factor cancels out, leaving their solution unaltered.

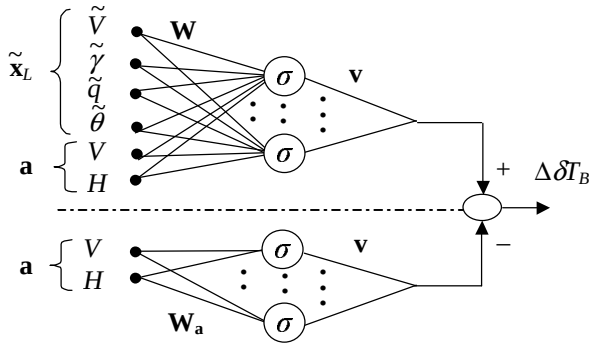

Figure 25. Final architecture for the pre-trained network $NN_{B_{L_1}}$. A similar architecture is used for all scalar feedback neural networks (biases **d** and *b* are not shown for simplicity).

The command-integral neural network, $\mathbf{NN}_I$, is expected to minimize integrals of the command-vector error in order to reduce the long-term effect of uncertain parameters or constant disturbances on the set point. It is pre-trained based on the approach introduced above for the feedback neural network. The contribution it provides, $\Delta\mathbf{u}_I$, must vanish when the command-error integrals do so as well:

$$\mathbf{z}_I\left[\boldsymbol{\xi}(t),\ \mathbf{a}(t)\right]\Big\|_{\kappa} = \mathbf{z}_I\left(\mathbf{0},\ \mathbf{a}^{\kappa}\right) = \mathbf{0} \tag{157}$$

A relationship between the gradients of the command-integral neural network, $\mathbf{NN}_I$, and $\mathbf{C}_I$ can be found by differentiating eq. 109 and eq. 153 with respect to $\boldsymbol{\xi}(t)$ and evaluating the derivative at the κ^{th} nominal flight condition,

$$\left.\frac{\partial \mathbf{z}_I(t)}{\partial \boldsymbol{\xi}(t)}\right|_{\kappa} = \left.\frac{\partial \tilde{\mathbf{u}}(t)}{\partial \boldsymbol{\xi}(t)}\right|_{\kappa} = -\mathbf{C}_I^{\kappa} \tag{158}$$

where $\boldsymbol{\xi}(t) = \mathbf{0}$ and $\mathbf{a}(t) = \mathbf{a}^{\kappa}$.

Initially, $\mathbf{NN}_I$ is composed of four independent scalar neural networks (Fig. 5),

$$\begin{bmatrix} \Delta\delta T(t) \\ \Delta\delta S(t) \\ \Delta\delta A(t) \\ \Delta\delta R(t) \end{bmatrix}_I = \begin{bmatrix} NN_{I_{L_1}}[\xi_L(t),\ \mathbf{a}(t)] \\ NN_{I_{L_2}}[\xi_L(t),\ \mathbf{a}(t)] \\ NN_{I_{LD_1}}[\xi_{LD}(t),\ \mathbf{a}(t)] \\ NN_{I_{LD_2}}[\xi_{LD}(t),\ \mathbf{a}(t)] \end{bmatrix} \tag{159}$$

where $\xi_L(t)$ and $\xi_{LD}(t)$ are defined as in Section 4.1. Although perfectly capable of operating in this configuration, they later are coupled to form a unique vector-output network that learns from the full aircraft dynamics on line. The command-integral training set also is of the form described in Section 3.1.1. The output condition $z_I(\mathbf{0}, \mathbf{a}^\kappa) = 0$ for $\forall\ \kappa$ is implied by eq. 157 , since $\boldsymbol{\xi} = [\,\xi_L^T \ \ \xi_{LD}^T\,]^T$. The gradient information is obtained from the longitudinal and lateral command-integral gain matrices computed in Section 4.1.3,

$$\begin{aligned} \mathbf{c}^\kappa_{I_{L_1}} &= \mathbf{C}^\kappa_{I_L}(1,\ \bullet)^T \\ \mathbf{c}^\kappa_{I_{L_2}} &= \mathbf{C}^\kappa_{I_L}(2,\ \bullet)^T \\ \mathbf{c}^\kappa_{I_{LD_1}} &= \mathbf{C}^\kappa_{I_{LD}}(1,\ \bullet)^T \\ \mathbf{c}^\kappa_{I_{LD_2}} &= \mathbf{C}^\kappa_{I_{LD}}(2,\ \bullet)^T \end{aligned} \tag{160}$$

with each vector's subscript designating the corresponding scalar network.

This gradient-based training data is used to initialize the command-integral scalar networks. For each NN_I, the algebraic technique introduced in Section 3.1.1 computes the input weights associated with the command-integral input (such as ξ_L or ξ_{LD}), $\mathbf{W}_\xi$, as well as the remaining weights: $\mathbf{W_a}$, $\mathbf{v}$, and $\mathbf{d}$. Figure 26 shows the final architecture used for $NN_{I_{L_1}}$, where $\mathbf{W} = [\,\mathbf{W}_\xi \,|\, \mathbf{W_a}\,]$. At this point, both $\mathbf{NN}_B$ and $\mathbf{NN}_I$ have been initialized and can be implemented in the PINN controller of Fig. 24. The pre-trained neural network controller performs identically to its linear counterpart when operating in the

neighborhood of one of the equilibria in *OP*. When operating between design equilibria, in *IR*, the pre-trained controller automatically interpolates between known gains, thanks to the neural networks' generalization abilities.

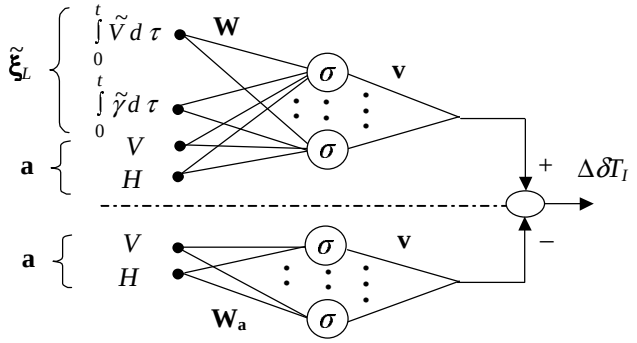

Figure 26. Final architecture for the pre-trained network $NN_{I_{L_1}}$. A similar architecture is used for all scalar command-integral networks (biases are not shown for simplicity).

The neural networks' performance is tested by applying the PINN structure for the control of the step-command-input aircraft response, over the steady-level flight envelope *IR* (Fig. 15). Three cases are illustrated; in each case, the same command input is provided to the full nonlinear simulation of the aircraft (eq. 1) flying at either a design or an interpolation operating point. The time histories of the relevant state elements and of the control deflections subject to $\mathbf{y}_c$ are used as performance evaluations and are plotted with a solid line. The state response is judged against that of a linear-PI-controller (Fig. 16), represented by a dashed line, that is specifically designed for the operating point under consideration. Prior to pre-training the forward neural network (Section 4.4), the set point ($\mathbf{x}_c$, $\mathbf{u}_c$) can be determined from linearized dynamic and output equations as shown in eq. 112, considering that $\mathbf{x}_c = \Delta\mathbf{x}_c + \mathbf{x}_0$.

Case 1: Response at a Design Point

The neural controller response is tested for a longitudinal and a lateral-directional command input at the design point (V_0, H_0) = (200 m/s, 11, 000 m), considered in the pre-training phase, to demonstrate the effectiveness of the algebraic technique implemented. Both the state and control time histories are plotted in Fig. 27 for a 2-deg-path angle command, and in Fig. 28 for a 5-deg-bank angle and 3-deg sideslip command. As expected, the response obtained with the neural controller (represented by a dashed line) is identical to that obtained with the linear controller (represented by a solid line), because all linear gains designed for *OP* are matched exactly.

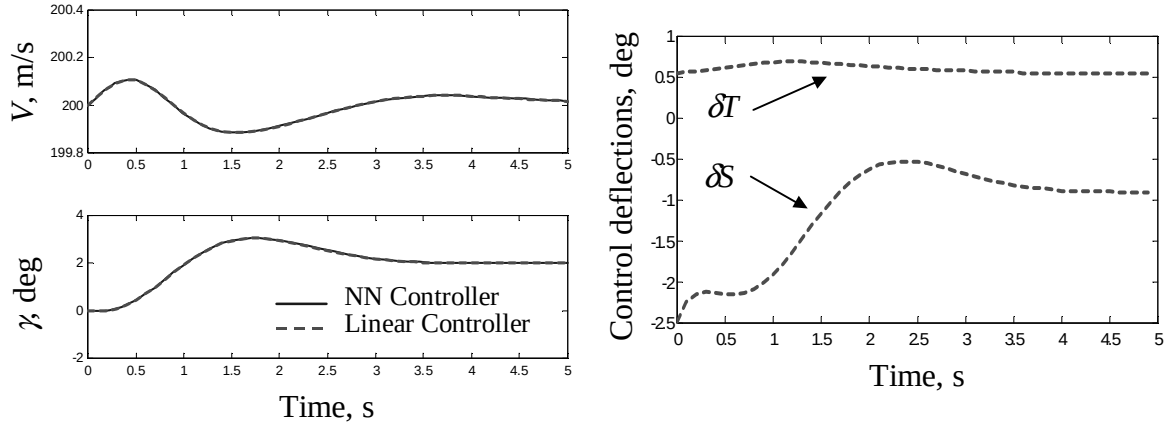

Figure 27. Relevant aircraft state and control response to 2-deg path angle step command, at the design point (V_0, H_0) = (200 m/s, 11 Km).

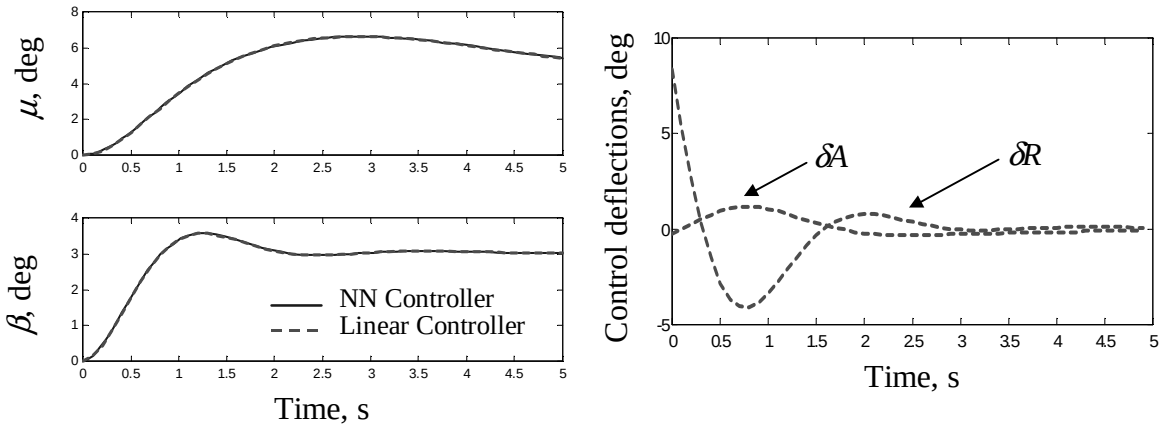

Figure 28. Relevant aircraft state and control response to 5-deg bank angle and 3-deg sideslip step command, at the design point (V_0, H_0) = (200 m/s, 11 Km).

Case 2: Longitudinal Response at an Interpolation Point

The response of the aircraft flying at an interpolation point, (V_0, H_0) = (95 m/s, 2 000 m), subject to a 97-m/s-velocity and 3-deg-path angle command input is shown in Fig. 29. The neural controller performance is compared to that of a linear controller specifically designed for these flight conditions.

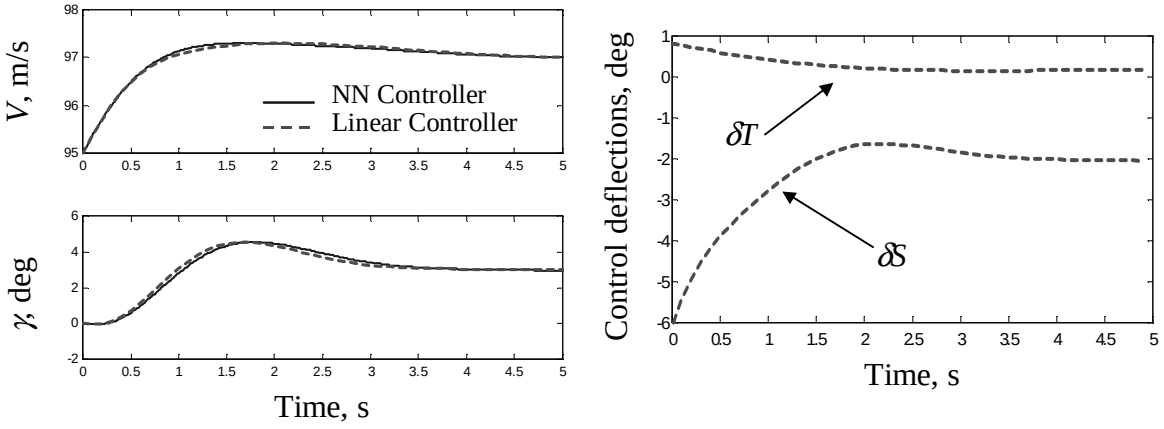

Figure 29. Relevant aircraft state and control response to 97-m/s-velocity and 3-deg-path angle step command, at the interpolation point (V_0, H_0) = (95 m/s, 2 Km).

Because this operating point was not considered in the pre-training phase, the neural networks must interpolate between the available linear controllers to produce the control. Nevertheless, the response is very close to that of a linear controller that is specifically designed for these flight conditions.

Case 3: Lateral-Directional Response at an Interpolation Point

The aircraft response is evaluated for a second interpolation point, (V_0, H_0) = (140 m/s, 6 000 m), and a purely-lateral command input of 6-deg-roll angle, to demonstrate the consistency of the results for different flight conditions. This case also shows that the pre-training phase is effective for both the longitudinal and the lateral-directional neural network controllers. The response of the same neural control structure and that of a linear controller computed for the operating point are plotted in Fig. 30, for comparison.

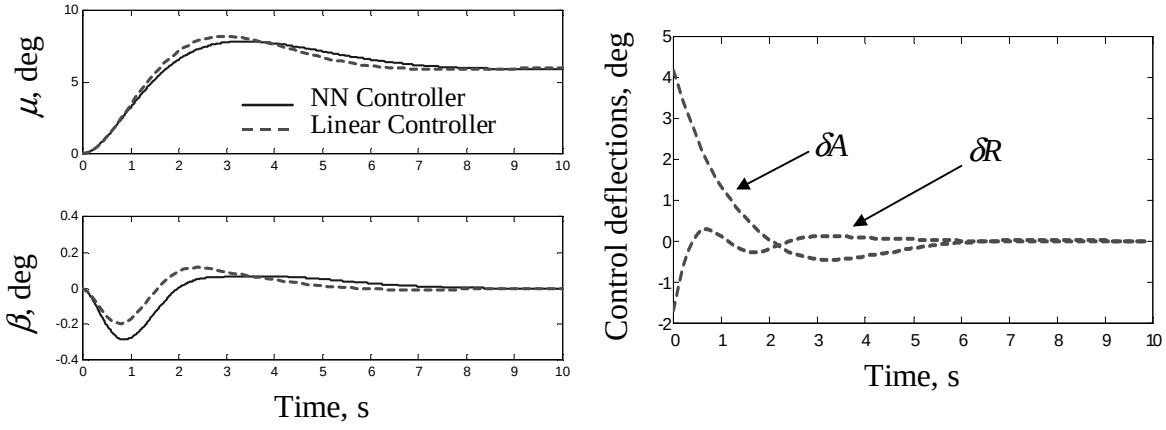

Figure 30. Relevant aircraft state and control response to 6-deg-roll angle step command, at the interpolation point (V_0, H_0) = (140 m/s, 6 Km).

The network's generalized response is again seen to be close to the linear one. While it only is feasible to design linear controllers for a finite set $OP \subset IR$, the neural controller handles the entire envelope, *IR*, with a performance analogous to that of a linear

controller specifically generated for the given flight condition. If the match at any given test point is not satisfactory, the neural network control response can be improved by adding that point to the design set.

4.4 Forward Neural Network

The role of the forward neural network in the nonlinear control system (Fig. 24) is to approximate the trim map of the business jet aircraft. The trim map represents an inversion of the aircraft nonlinear model (eq. 1) and, in series with the aircraft, it provides a feed-forward path [86]. Given the command input, $\mathbf{y}_c$, $\mathbf{NN}_F$ must produce corresponding control settings, $\mathbf{u}_c$, that trim the aircraft about the desired steady maneuver. Under perfect conditions of exact coincidence between the nonlinear model and the actual plant, the control is provided solely by the forward network (i.e., $\tilde{\mathbf{u}} = \mathbf{0}$). Perturbation feedback signals, $\Delta\mathbf{u}_B$ and $\Delta\mathbf{u}_I$, that compensate for inaccuracy in the aircraft model and for external disturbances are processed by feedback and command-integral neural networks, as accomplished in Section 4.3.

Trim control settings can be defined for a given command input,

$$\mathbf{u}_c(t) = \mathbf{g}_c[\mathbf{y}_c(t)] \tag{161}$$

such that:

$$\mathbf{f}[\mathbf{x}_c(t),\ \mathbf{p}_m(t),\ \mathbf{u}_c(t)] = \mathbf{f}[\mathbf{x}_c(t),\ \mathbf{p}_m(t),\ \mathbf{g}_c[\mathbf{y}_c(t)]] = \mathbf{0} \tag{162}$$

The *command values*, V_c, γ_c, μ_c, and β_c, in $\mathbf{x}_c$ are directly specified by $\mathbf{y}_c$. The remaining *secondary values* of the state, q_c, θ_c, r_c, and p_c, are computed so as not to oppose the steady maneuver commanded by $\mathbf{y}_c$. In the present context, a steady maneuver always is defined by zero Euler roll and pitch rates, i.e., $\dot{\phi}_c = \dot{\theta}_c = 0$, and by zero body

accelerations. Hence, it may correspond to cruising flight, steady climb or descent, or a coordinated (possibly helical) turn, such as the one illustrated in Fig. 31. The body axes system, fixed to the airplane, is shown in Fig. 32 together with an inertial reference frame that is fixed to the Earth. The latter is denoted by the subscript *r* and the former by the subscript *b*.

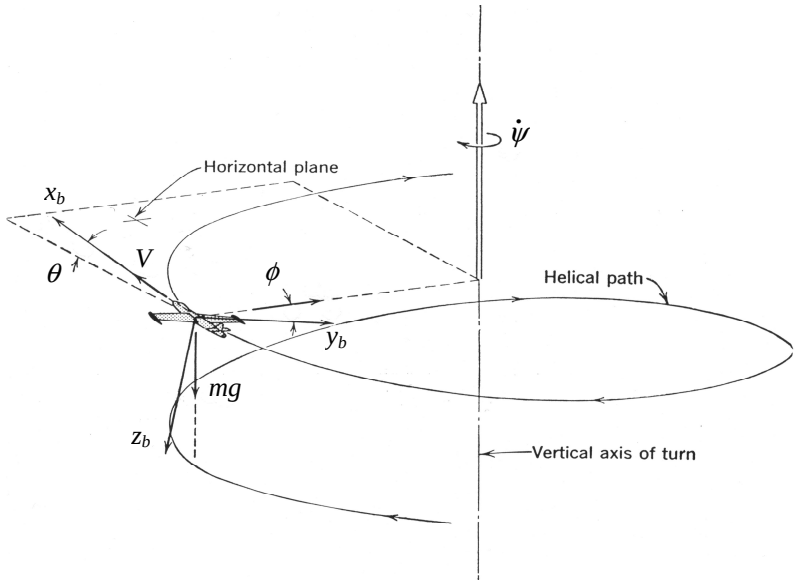

Figure 31. Steady-climbing coordinated turn, taken from [87].

The airplane velocity can be resolved into three components, *u*, *v*, and *w*, along the x_b, y_b, and z_b body axes, respectively. The angular velocities about these axes are referred to as roll rate, *p*, pitch rate, *q*, and yaw rate, *r*. Thus, in a steady maneuver, the following must hold: $\dot{u}_c = \dot{v}_c = \dot{w}_c = 0$ and $\dot{p}_c = \dot{q}_c = \dot{r}_c = 0$. Through conventional rotations [88], defined in terms of the three Euler angles, ϕ, θ, and ψ, the transformation between these two frames of reference can be derived [88] and the relationship between the Euler and the body angular rates identified as:

$$\begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} 1 & 0 & -\sin\theta \\ 0 & \cos\phi & \cos\theta\sin\phi \\ 0 & -\sin\phi & \cos\theta\cos\phi \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \tag{163}$$

The body velocities can be expressed in terms of the aerodynamic angles, β and α, and the airspeed, V [88] to relate them to the elements of the state **x**:

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} V\cos\alpha\cos\beta \\ V\sin\beta \\ V\sin\alpha\cos\beta \end{bmatrix} \tag{164}$$

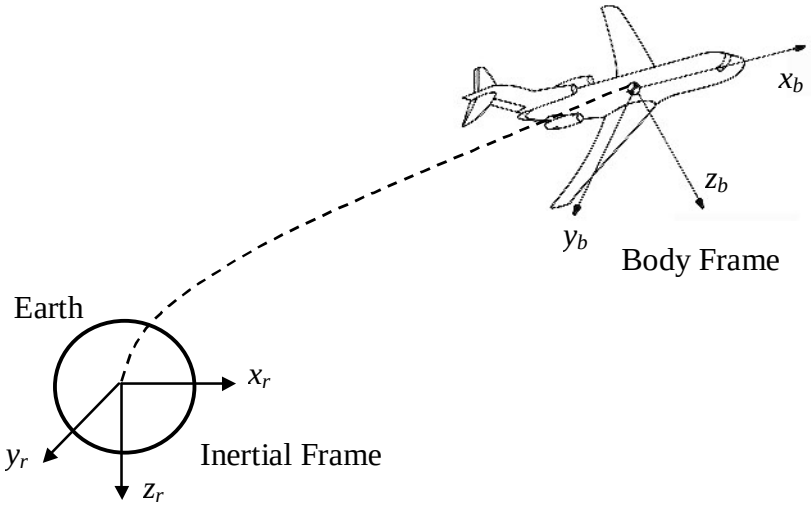

Figure 32. Body and inertial axes systems, adapted from [88].

Given the value of the command input, $\mathbf{y}_c$, at a specified altitude, the settings $\mathbf{u}_c$, θ_c, and $\dot{\psi}_c$, capable of trimming the aircraft (i.e., satisfying eq. 13) are obtained numerically from the equations of motion, by setting the body velocities and angular rates equal to zero and employing the transformations above. For three-dimensional maneuvering flight, the commanded angle of attack and Euler roll angle are fully specified by $\mathbf{y}_c$ and θ_c, as suggested by the following spherical trigonometric relationships [89]:

$$\alpha = \cos^{-1}\left(\frac{\cos\gamma\cos\theta + \sin\gamma\sin\theta}{\cos\beta}\right) \tag{165}$$

$$\phi = \sin^{-1}\left(\frac{\cos\gamma\sin\mu\cos\beta - \sin\gamma\sin\beta}{\cos\theta}\right) \tag{166}$$

Subsequently, the commanded body axes and angular rates (including the secondary values of the state, q_c, r_c, and p_c) can be obtained from eq. 164 and eq. 163, respectively, with eq. 163 simplifying to,

$$\begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} -\dot{\psi}\sin\theta \\ \dot{\psi}\cos\theta\sin\phi \\ \dot{\psi}\cos\theta\cos\phi \end{bmatrix} \tag{167}$$

under the assumption that $\dot{\phi}_c = \dot{\theta}_c = 0$.

Following the above procedure, the control settings $\mathbf{u}_c$, the state $\mathbf{x}_c$, and parameters $\mathbf{p}_m$ that satisfy eq. 161–162 for the command input $\mathbf{y}_c$ and altitude H_c can be determined. The aircraft trim map, U_c, is defined as the sampled set of control settings that trim the aircraft over its full operational domain, *OR*:

$$U_c(\mathbf{x}_c,\ \mathbf{p}_m) = \left\{\mathbf{u}_c^k : \left(\mathbf{x}_c^k,\ \mathbf{p}_m^k\right) \in OR,\ \mathbf{f}\left(\mathbf{x}_c^k,\ \mathbf{p}_m^k,\ \mathbf{u}_c^k\right) = \mathbf{0},\ k = 1,\ 2,\ \ldots\right\} \tag{168}$$

The mapping from *OR* to U_c can be assumed to be bounded and one to one. In practice, U_c is obtained by solving the steady-state equation (eq. 13) for values of the command input sampled over *OR*. For the airplane, *OR* is defined as the set of all possible maneuvers, i.e., $OR = \{V, H, \gamma, \mu, \beta\}$, and is initially estimated based on what is already known about the vehicle's capabilities, the passengers' comfort, and other operational constraints. Ultimately, the operating domain is defined as the state and parameter space for which there exists a trim solution. Therefore, its boundaries can be determined by solving the same steady-state equation that defines the aircraft trim map, eq. 161–162.

Computing the trim map is a task that is computationally challenging in itself and whose algorithmic details are omitted for simplicity. The approach taken here consists of specifying limits for what are known to be reasonable ranges of γ, μ, and β:

$$OR: \begin{cases} -6 \text{ deg} \le \gamma \le 6 \text{ deg} \\ -21 \text{ deg} \le \mu \le 21 \text{ deg} \\ -5 \text{ deg} \le \beta \le 5 \text{ deg} \end{cases} \tag{169}$$

Several combinations, $\{\gamma, \mu, \beta\}$, are considered within these limits to sample the space in a non-symmetrical fashion, as explained in Appendix D. The maximum allowable flight envelope is taken to be that for steady-level flight. Where the flight path angle is negative, the envelope could be expanded; however, we do not allow that to happen. In other cases, such as increased flight path angle, non-zero roll or sideslip angles, the $\{V, H\}$ envelope is decreased. Trim solutions are sought below an estimated ceiling of 15, 000 m, progressing downwards to sea level (0 m) in 1, 000 m intervals (Fig. 15). For each of these altitudes, the lower and upper bounds on velocity are determined based on the existence of trim solutions, as schematized in Fig. 33.

Because there are many control variables and the equilibrium is quite sensitive, numerical problems may be encountered in solving eq. 162, particularly along the low-velocity boundary. For some combinations of path angle and bank angle, (γ, μ), the amount of throttle required to trim the aircraft at a given altitude does not vary monotonically with respect to airspeed. Hence, the MATLAB fsolve routine fails to provide reliable results. For example, it may not identify valid solutions at some interior points, and it may not provide a smooth boundary at other flight conditions. The conservative approach taken here is to define a reduced flight envelope that eliminates the problem points. The collection of the $\{V, H\}$ envelopes corresponding to all chosen

combinations of path angle, bank angle, and sideslip, $\{\gamma, \mu, \beta\}$, constitutes the five-dimensional flight envelope $OR = \{V, H, \gamma, \mu, \beta\}$, whose boundaries are partly projected onto the three-dimensional space in Fig. 34, for illustration.

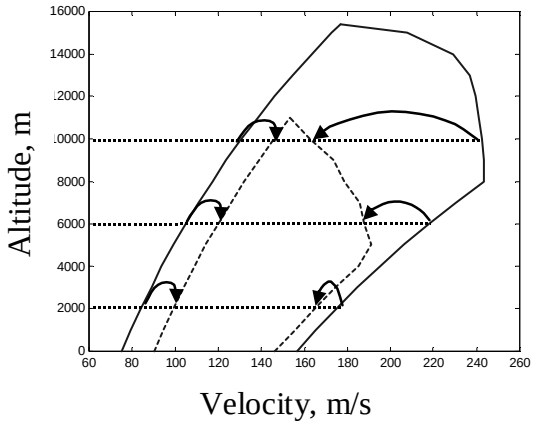

Figure 33. Search of reduced $\{V, H\}$ envelope associated with one combination of values (γ, μ, β) (dashed line), starting from the steady-level envelope (solid line). The search process is schematized for three sample altitudes.

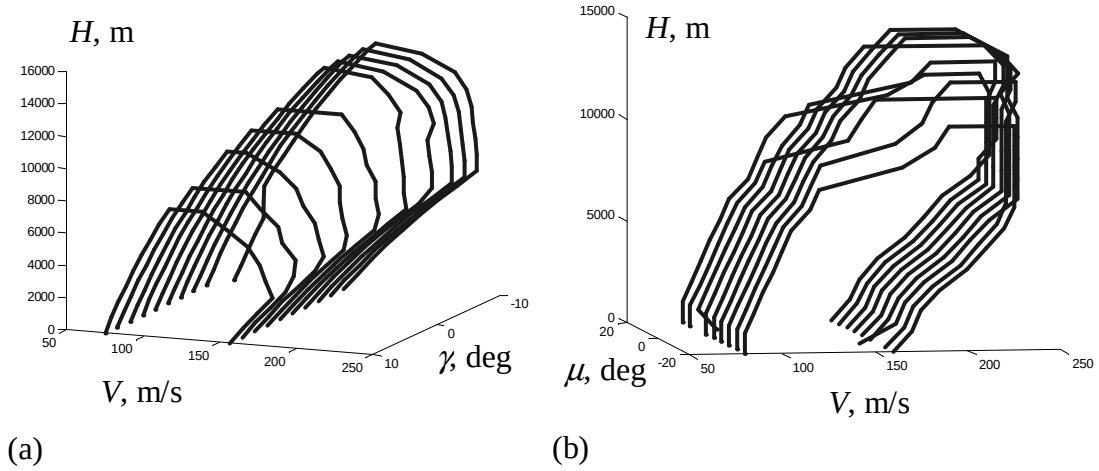

Figure 34. $\{V, H, \gamma\}$ envelope for $(\mu, \beta) = (20^o, 5^o)$ (a), and $\{V, H, \mu\}$ envelope for $(\gamma, \beta) = (0^o, -5^o)$ (a)

The trim map, U_c, and the corresponding points in *OR* are stored in the multi-dimensional array structures described in Appendix D. U_c is used to pre-train the forward neural network, $\mathbf{NN}_F$, whose objective is to approximate the nonlinear mapping in eq. 161 over the compact input space *OR*. To every value of the trim-control vector, $\mathbf{u}_c$, there corresponds a unique pair of values θ_c and $\dot{\psi}_c$ that, together with $\mathbf{u}_c$, specify the maneuver commanded by $\mathbf{y}_c$. These two parameters are needed by the Command State Generator (*CSG*) in Fig. 24, in order for it to provide the control system with the secondary values of the state, q_c, θ_c, r_c, and p_c, that are compatible with $\mathbf{y}_c$. Hence, the forward neural network, characterized by the architecture shown in Fig. 35, must learn the following trim data:

$$\left\{ \begin{bmatrix} \mathbf{y}_c^k \\ H_c^k \end{bmatrix}, \begin{bmatrix} \mathbf{u}_c^k \\ \theta_c^k \\ \dot{\psi}_c^k \end{bmatrix} \right\}_{k=1,\ \ldots,\ 2696} \tag{170}$$

This equation expresses the trim data to be used for training in a form that is equivalent to the input/output training set treated in Section 3.1.2-3.1.3. The only difference is that it refers to a vector output network and, thus, can be generalized by a set of the form $\{\mathbf{y}^k, \mathbf{u}^k\}_{k\,=\,1,\,\ldots,\,p}$.

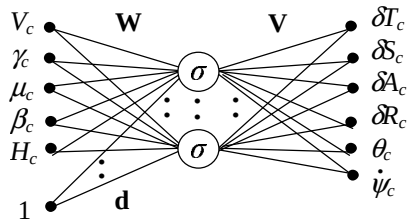

Figure 35. Forward neural network architecture, with a generic number of nodes.

Because there is a total of 2, 696 training pairs in eq. 170, it is convenient to seek only an approximate matching of the data, synthesizing this information by using less nodes than there are operating points. Also, since the multi-dimensional surface in eq. 161 is only slightly nonlinear, gradient information ($\mathbf{C}_F$) is not required to achieve good interpolation properties, as demonstrated below. Optimization-based algorithms, such as the Levenberg-Marquardt (LM) and the Resilient Backpropagation (RPROP), are not easily implemented for a training set this large, as explained in Section 3.1.3. Therefore, $\mathbf{NN}_F$ is pre-trained based on all of the trim data available, following the algebraic approach introduced in Section 3.1.3. The full set (eq. 170) is divided into smaller subsets that are individually used to train small neural networks with the same input/output structure as $\mathbf{NN}_F$. These small networks, with less nodes than there are operating points, are trained using an optimization-based algorithm and, later, combined algebraically to obtain the final forward neural network parameters.

The LM algorithm generally displays excellent performance for a number of training pairs of order ~$O(100)$ [41]. Twenty subsets are obtained from all twenty (μ, β) combinations described in the Appendix D. Each subset contains approximately 135 trim points corresponding to the three-dimensional envelope $\{V, H, \gamma\}$ and to one combination (μ, β). Then, the LM algorithm is used to train twenty vector-output neural networks (each with an input/output structure shown in Fig. 35), based on the individual subsets. It is easily found that each subset can be approximated by a ten-node network, achieving excellent generalization properties and, typically, a final mean-square error of 5×10^{-6} rad or rad/s. For each training pair the network error is defined as the difference between the ideal output vector and the actual neural network output. Because the values of μ and

β are held constant within each training subset, the inputs μ_c and β_c are essentially equivalent to input biases for the corresponding ten-node network. However, their values varying across the subsets renders them equivalent to the inputs V_c, H_c, and γ_c within the full network, obtained by superimposing all twenty ten-node networks.

Using the notation introduced in Section 3.1.3, it can be deduced that $s_1 = \ldots = s_m = 10$, and that the full network must have a number of nodes, s, equal to 200. Also, according to the procedure described in this section, the full network's input weights, **W** and **d**, are obtained algebraically from the input weights of the m ten-node networks ($m = 20$), as indicated by eq. 76-77. In the case of a vector-output network, the matrix **V** of output weights is obtained from the vector-output equivalent of eq. 80:

$$\mathbf{V} = \left(\mathbf{S}^{PI}\,\mathbf{U}\right)^T \tag{171}$$

v_{li} denotes the element in the l^{th}-row and the i^{th}-column of **V** and represents the interconnection weight between the i^{th}-node and the l^{th}-output in the network. The sigmoidal matrix **S** is defined in the usual fashion, based on eq.47, and **U** contains all of the output training data from eq. 170,

$$\mathbf{U} = \begin{bmatrix} \mathbf{u}_c^1 & \cdots & \mathbf{u}_c^p \\ \theta_c^1 & \cdots & \theta_c^p \\ \dot{\psi}_c^1 & \cdots & \dot{\psi}_c^p \end{bmatrix}^T \tag{172}$$

with a number of training pairs, p, equal to 2, 696. In this case, the error brought about by this superposition is not zero, but amounts to a mean-square error of 2×10^{-5} rad or rad/s, which is an acceptable error for this application. This full network, with architecture shown in Fig. 35 and parameters **W**, **d**, and **V**, constitutes $\mathbf{NN}_F$, as it

approximates the full trim-data training set (eq. 170), corresponding to the full envelope $OR = \{V, H, \gamma, \mu, \beta\}$.

The generalization capabilities of the forward neural network are tested throughout *OR* by computing the mean-square error at points not included in the training set, and by projecting the neural mapping onto three-dimensional space. Two sets of additional trim data are produced for validation purposes: one with 39, 764 points and one with 2, 629 points, all from conditions in *OR* that were excluded from the set in eq. 170 (Appendix D). The trim settings at the validation points are compared to those computed by the pre-trained forward neural network; the mean-square error is found to be approximately 3×10^{-5} rad or rad/s for both sets. Hence, good generalization properties are obtained consistently across *OR*, indicating that overfitting does not occur. Furthermore, the surface approximated by $\mathbf{NN}_F$ is plotted in Fig. 36 and compared to trim data from the first validation set (Fig. 37) by holding γ_c, μ_c, and β_c constant and computing the output over a fine-grid V_c-H_c input space. Figures 38 and 39 show a similar comparison obtained by holding the inputs H_c, γ_c, and β_c constant.

These results show that the neural surfaces obtained through algebraic superposition are smooth and that the approximation error is concentrated in areas corresponding to zero outputs, such as $\mu_c = \beta_c$, to which there corresponds an output $\dot{\psi}_c = 0$, for any value of the inputs V_c, H_c, and γ_c. This can also be observed from the linear system in eq. 171, where the trim data produces an inconsistency [74] that is reflected on the rank condition. If the input/output training data were fully consistent, eq. 171 would possess an exact solution and the algebraic superposition would bring about zero error (although some error would likely persist from the optimization-based training of the small *m* networks).

The pre-trained forward neural network is implemented in the nonlinear control system to compute the trim settings corresponding to the desired command input. Its parameters are held fixed during the on-line adaptation, with the remaining neural networks accounting for possible parameter variations. It is conceivable that the forward neural network also be adapted to account for changes or parameter variations in the plant to be controlled. In which case, it is recommended that a model network also be used, as explained in Section 6.3.

Given $\mathbf{y}_c$ and the corresponding values of θ_c and $\dot{\psi}_c$ approximated by the forward neural network (Fig. 35), the Command State Generator computes the secondary elements of the state that are consistent with the commanded maneuver. The altitude at which the maneuver is to take place, H_c, is approximated by the current altitude, H. The exact value of H_c is virtually impossible to compute, because the trajectory and the time required to bring the aircraft from the present altitude, H, to the commanded one, H_c, are not known *a priori*. Based on the available information, the *CSG* computes the Euler roll angle, ϕ_c, from eq. 166 and, subsequently, all body rates, p_c, q_c, and r_c, from eq. 167. The combination of $\mathbf{NN}_F$ and *CSG* is responsible for computing the set point ($\mathbf{x}_c$, $\mathbf{u}_c$) for the nonlinear aircraft, as anticipated by the block diagram in Fig. 24.

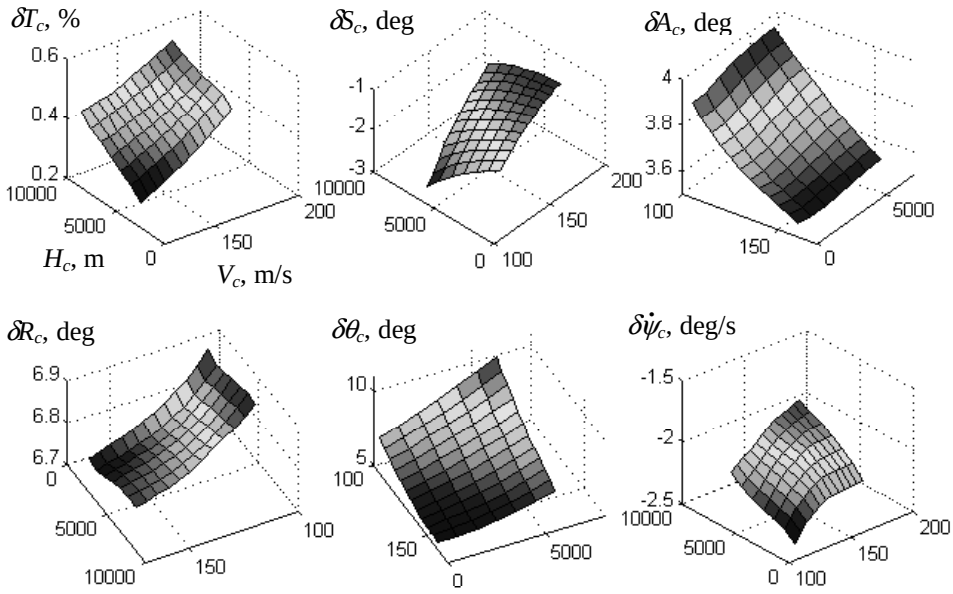

Figure 36. Trim control surfaces modeled by the forward neural network, plotted over a $\{V_c, H_c\}$-input space by holding the remaining inputs fixed at $(\gamma_c, \mu_c, \beta_c) = (3^o, 14^o, 4^o)$.

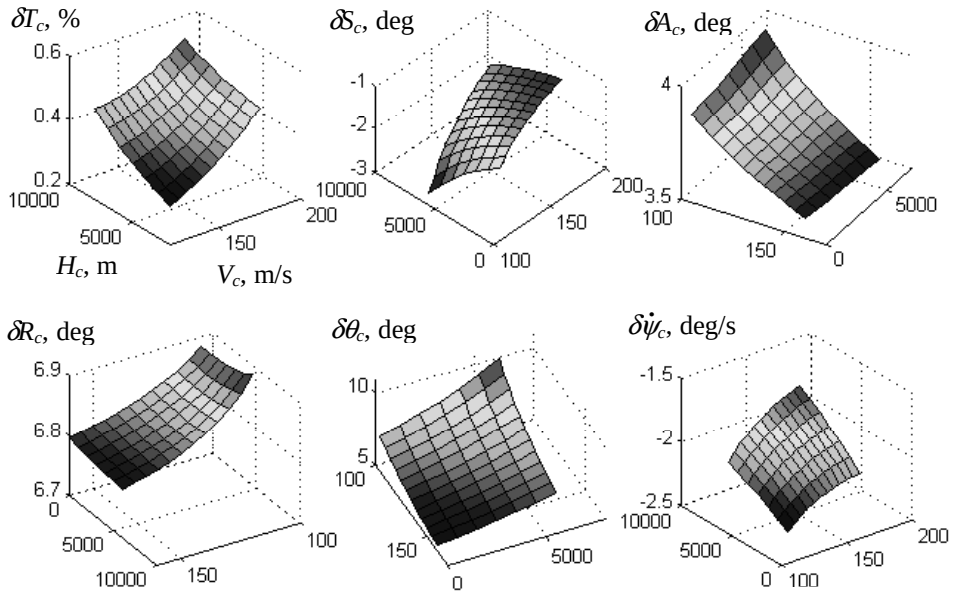

Figure 37. Actual trim control surfaces plotted over a $\{V_c, H_c\}$-input space by holding the remaining inputs fixed at $(\gamma_c, \mu_c, \beta_c) = (3^o, 14^o, 4^o)$.

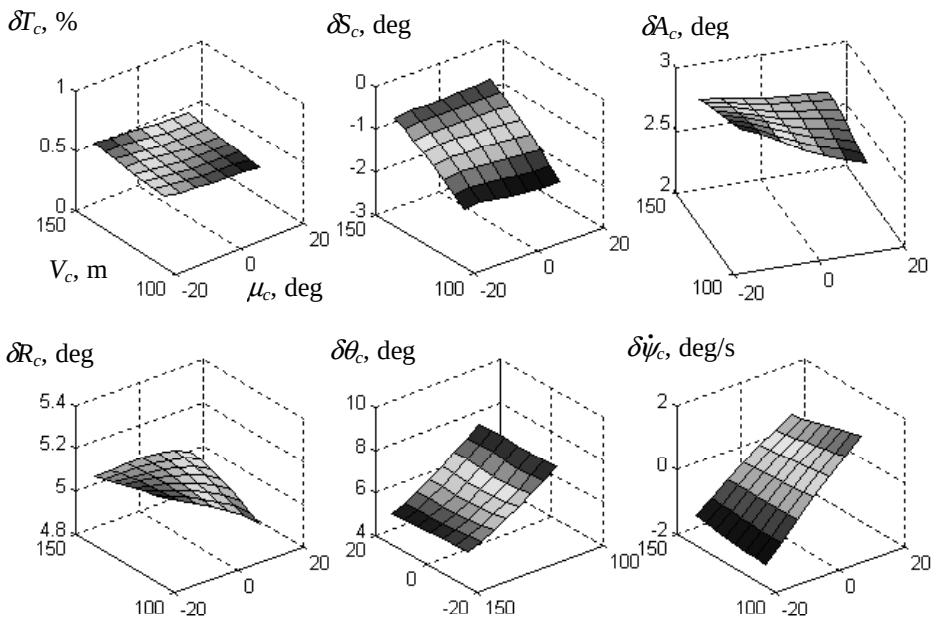

Figure 38. Trim control surfaces modeled by the forward neural network, plotted over a $\{V_c, \mu_c\}$-input space by holding the remaining inputs fixed at (H_c, γ_c, β_c) = (5 Km, 4°, 3°).

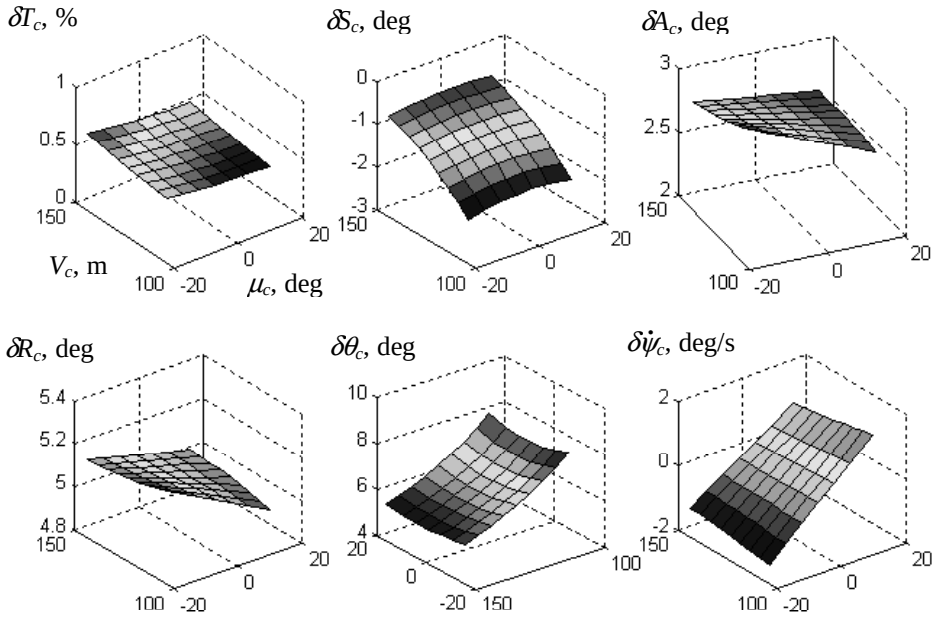

Figure 39. Actual trim control surfaces plotted over a $\{V_c, \mu_c\}$-input space by holding the remaining inputs fixed at (H_c, γ_c, β_c) = (5 Km, 4°, 3°).

4.5 Critic Neural Network

As anticipated in Section 2.1, a critic network is used to evaluate the neural controller performance on line, within the DHP adaptive critic architecture. In particular, the critic approximates the derivative of the value function with respect to the state in the optimality condition criterion (eq. 36), which is to be minimized by the control strategy. The critic network, $\mathbf{NN}_C$, also is pre-trained based on knowledge available from the corresponding linear control design. In fact, it was first established in Section 2.3 that there exists a correspondence between the critic network gradient and the Riccati matrix, at nominal operating conditions. In the LQ optimal control problem, the Riccati matrices are used to obtained the feedback and command-integral gains, but they do not appear explicitly in the PI control structure (Fig. 16). Similarly, the critic needs not be implemented for the pre-trained PINN controller (Fig. 24) to operate based on linear control knowledge. It is, however, required during the on-line phase (Chapter 5), where it aids the control networks ($\mathbf{NN}_B$ and $\mathbf{NN}_I$) in their adaptation to improve performance for large-angle and coupled maneuvers, new operating conditions, and unforeseen failures that were unaccounted for by the pre-training phase.

The same PI cost function, eq. 107, optimized in the pre-training phase, is optimized during the on-line phase. Therefore, the corresponding optimal cost-to-go or value function can be defined as in eq. 11, based on the PI Lagrangian $L[\mathbf{x}_a(t), \tilde{\mathbf{u}}(t)]$, and minimized with respect to the control deviation, $\tilde{\mathbf{u}}$. For an infinite-horizon problem, the terminal cost can be assumed to be equal to zero (as explained in Section 2.1). Hence, the value function can be differentiated with respect to the augmented state, $\mathbf{x}_a$, to obtain the PI performance measure equivalent to eq. 12,

$$\boldsymbol{\lambda}_a[\mathbf{x}_a(t)] \equiv \frac{\partial V[\mathbf{x}_a(t)]}{\partial \mathbf{x}_a(t)} \tag{173}$$

(omitting the asterisks for simplicity) that is to be approximated by the critic network:

$$\boldsymbol{\lambda}_a(t) = \mathbf{NN}_C[\mathbf{x}_a(t),\ \mathbf{a}(t)] \equiv \mathbf{z}_C(t) \tag{174}$$

The vector $\mathbf{a}(t)$ contains all relevant auxiliary inputs identified during the pre-training phase and used for all other neural networks (eq. 151).

The linear designs obtained in Section 4.1 for the set *OP* of operating points, shown in Fig. 15, are used to pre-train the critic network. At every design point indexed by $\kappa \in OP$, the critic output must equal the locally optimal value function derivative,

$$\frac{\partial V[\mathbf{x}_a(t)]}{\partial \mathbf{x}_a(t)} = \mathbf{x}_a{}^T(t)\mathbf{P}_a(t) \tag{175}$$

obtained by reformulating eq. 19 in terms of the PI augmented state and Riccati matrix, and by differentiating it with respect to $\mathbf{x}_a$. Furthermore, when the flight conditions are truly nominal, by definition both the state deviation $\tilde{\mathbf{x}}$ and the output-error integral $\boldsymbol{\xi}$ are identically equal to zero; hence, from eq. 175, the following must hold:

$$\mathbf{z}_C[\mathbf{x}_a(t),\ \mathbf{a}(t)]_\kappa = \mathbf{z}_C(\mathbf{0},\ \mathbf{a}^\kappa) = \mathbf{0} \tag{176}$$

The gradient of $\mathbf{NN}_C$ at the κ^{th} operating point is found to correspond to the Riccati matrix $\mathbf{P}_a^\kappa$ by differentiating both eq. 175 and eq. 174 with respect to $\mathbf{x}_a$:

$$\left.\frac{\partial \mathbf{z}_C(t)}{\partial \mathbf{x}_a(t)}\right|_\kappa = \left.\frac{\partial \boldsymbol{\lambda}_a(t)}{\partial \mathbf{x}_a(t)}\right|_\kappa = \mathbf{P}_a^\kappa \tag{177}$$

Initially, each element in λ_a is modeled by a scalar sigmoidal network of the type shown in Fig. 5. Based on the decoupled linear designs of Section 4.1, these networks can be characterized as longitudinal critic networks,

$$\begin{bmatrix} \partial \mathrm{V}[\mathbf{x}_a(t)]/\partial \tilde{V}(t) \\ \partial \mathrm{V}[\mathbf{x}_a(t)]/\partial \tilde{\gamma}(t) \\ \partial \mathrm{V}[\mathbf{x}_a(t)]/\partial \tilde{q}(t) \\ \partial \mathrm{V}[\mathbf{x}_a(t)]/\partial \tilde{\theta}(t) \\ \partial \mathrm{V}[\mathbf{x}_a(t)]/\partial \left(\int \tilde{V}(\tau) d\tau\right) \\ \partial \mathrm{V}[\mathbf{x}_a(t)]/\partial \left(\int \tilde{\gamma}(\tau) d\tau\right) \end{bmatrix} = \begin{bmatrix} NN_{C_{L_1}}[\mathbf{x}_{a_L}(t),\ \mathbf{a}(t)] \\ NN_{C_{L_2}}[\mathbf{x}_{a_L}(t),\ \mathbf{a}(t)] \\ NN_{C_{L_3}}[\mathbf{x}_{a_L}(t),\ \mathbf{a}(t)] \\ NN_{C_{L_4}}[\mathbf{x}_{a_L}(t),\ \mathbf{a}(t)] \\ NN_{C_{L_5}}[\mathbf{x}_{a_L}(t),\ \mathbf{a}(t)] \\ NN_{C_{L_6}}[\mathbf{x}_{a_L}(t),\ \mathbf{a}(t)] \end{bmatrix} \tag{178}$$

or as lateral-directional critic networks:

$$\begin{bmatrix} \partial \mathrm{V}[\mathbf{x}_a(t)]/\partial \tilde{r}(t) \\ \partial \mathrm{V}[\mathbf{x}_a(t)]/\partial \tilde{\beta}(t) \\ \partial \mathrm{V}[\mathbf{x}_a(t)]/\partial \tilde{p}(t) \\ \partial \mathrm{V}[\mathbf{x}_a(t)]/\partial \tilde{\mu}(t) \\ \partial \mathrm{V}[\mathbf{x}_a(t)]/\partial \left(\int \tilde{\mu}(\tau) d\tau\right) \\ \partial \mathrm{V}[\mathbf{x}_a(t)]/\partial \left(\int \tilde{\beta}(\tau) d\tau\right) \end{bmatrix} = \begin{bmatrix} NN_{C_{LD_1}}[\mathbf{x}_{a_{LD}}(t),\ \mathbf{a}(t)] \\ NN_{C_{LD_2}}[\mathbf{x}_{a_{LD}}(t),\ \mathbf{a}(t)] \\ NN_{C_{LD_3}}[\mathbf{x}_{a_{LD}}(t),\ \mathbf{a}(t)] \\ NN_{C_{LD_4}}[\mathbf{x}_{a_{LD}}(t),\ \mathbf{a}(t)] \\ NN_{C_{LD_5}}[\mathbf{x}_{a_{LD}}(t),\ \mathbf{a}(t)] \\ NN_{C_{LD_6}}[\mathbf{x}_{a_{LD}}(t),\ \mathbf{a}(t)] \end{bmatrix} \tag{179}$$

Similarly, the augmented state input $\mathbf{x}_a$ can be partitioned into the longitudinal vector $\mathbf{x}_{a_L} = \left[\tilde{\mathbf{x}}_L^T \ \ \xi_L^T\right]^T$ and the lateral-directional vector $\mathbf{x}_{a_{LD}} = \left[\tilde{\mathbf{x}}_{LD}^T \ \ \xi_{LD}^T\right]^T$. Prior to the on-line phase, these neural networks can be algebraically joined into a single vector-output network, $\mathbf{NN}_C$, with equivalent off-line performance, as will be shown in Section 5.1.1.

The critic requirements in eq. 176-177 are equivalent to those obtained for the feedback and command-integral neural networks, eq. 152-154 and eq. 157-158, respectively. Equation 176 is extended to all scalar critic networks that must satisfy an equivalent requirement: $z_C(\mathbf{0}, \mathbf{a}^{\kappa}) = 0$. Gradient information can be obtained from the longitudinal and lateral Riccati matrices designed in Section 4.1.3 and organized such that they provide known gradient vectors for the corresponding longitudinal critic networks (eq. 178),

$$
\begin{aligned}
\mathbf{c}^{\kappa}_{C_{L_1}} &= \mathbf{P}^{\kappa}_{a_L}(1,\ \bullet) \\
\mathbf{c}^{\kappa}_{C_{L_2}} &= \mathbf{P}^{\kappa}_{a_L}(2,\ \bullet) \\
\mathbf{c}^{\kappa}_{C_{L_3}} &= \mathbf{P}^{\kappa}_{a_L}(3,\ \bullet) \\
\mathbf{c}^{\kappa}_{C_{L_4}} &= \mathbf{P}^{\kappa}_{a_L}(4,\ \bullet) \\
\mathbf{c}^{\kappa}_{C_{L_5}} &= \mathbf{P}^{\kappa}_{a_L}(5,\ \bullet) \\
\mathbf{c}^{\kappa}_{C_{L_6}} &= \mathbf{P}^{\kappa}_{a_L}(6,\ \bullet)
\end{aligned}
\tag{180}
$$

and for the corresponding lateral critic networks (eq. 179):

$$
\begin{aligned}
\mathbf{c}^{\kappa}_{C_{LD_1}} &= \mathbf{P}^{\kappa}_{a_{LD}}(1,\ \bullet) \\
\mathbf{c}^{\kappa}_{C_{LD_2}} &= \mathbf{P}^{\kappa}_{a_{LD}}(2,\ \bullet) \\
\mathbf{c}^{\kappa}_{C_{LD_3}} &= \mathbf{P}^{\kappa}_{a_{LD}}(3,\ \bullet) \\
\mathbf{c}^{\kappa}_{C_{LD_4}} &= \mathbf{P}^{\kappa}_{a_{LD}}(4,\ \bullet) \\
\mathbf{c}^{\kappa}_{C_{LD_5}} &= \mathbf{P}^{\kappa}_{a_{LD}}(5,\ \bullet) \\
\mathbf{c}^{\kappa}_{C_{LD_6}} &= \mathbf{P}^{\kappa}_{a_{LD}}(6,\ \bullet)
\end{aligned}
\tag{181}
$$

It follows that a gradient-based training set of the type described in Section 3.1.1 can be obtained for each scalar critic network. According to the procedure described in Section 3.1.1, a number of nodes equal to the number of design points (thirty-four) allows for exact matching of the training data. All network parameters are determined algebraically, in one step, using the algorithm in Appendix B. For each scalar critic network NN_C, the initialized parameters consist of the weights associated with the state-deviation input ($\tilde{\mathbf{x}}_L$ or $\tilde{\mathbf{x}}_{LD}$), $\mathbf{W}_{\tilde{\mathbf{x}}}$, the weights associated with the command-integral input ($\boldsymbol{\xi}_L$ or $\boldsymbol{\xi}_{LD}$), $\mathbf{W}_{\boldsymbol{\xi}}$, the weights associated with the scheduling-vector input, $\mathbf{W}_{\mathbf{a}}$, the output weights, $\mathbf{v}$, and the biases, $\mathbf{b}$ and d. The final architecture of one critic network, $NN_{C_{L_1}}$, is shown in Figure 40, where $\mathbf{W} = [\,\mathbf{W}_{\tilde{\mathbf{x}}} \,|\, \mathbf{W}_{\boldsymbol{\xi}} \,|\, \mathbf{W}_{\mathbf{a}}\,]$. This type of architecture always provides zero output for zero augmented state input and is representative of all

critic networks, whose input/output structure is detailed in eq. 178-179. The performance of these algebraically pre-trained critic networks can be tested by comparing their gradients to corresponding Riccati matrices specifically designed for any of the interpolating points in *IR* (Fig. 15). However, this step is not necessary as the gradient-based algorithm always provides excellent generalization properties, thanks to the strict formulation of the requirements it satisfies.

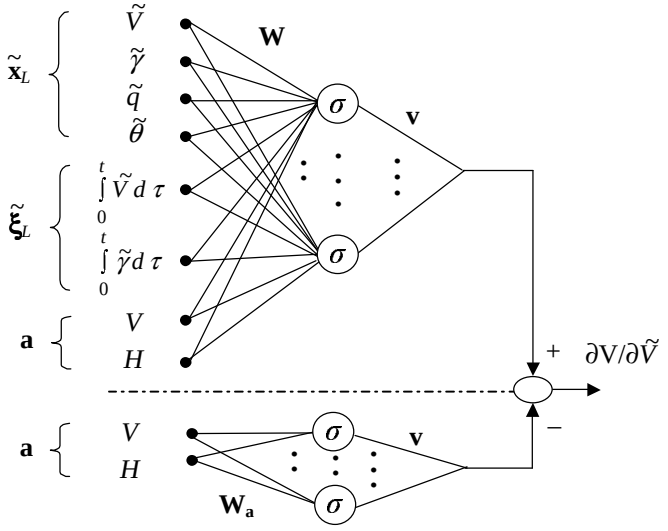

Figure 40. Final architecture for the pre-trained network $NN_{C_{L_1}}$. A similar architecture is used for all scalar critic networks (input biases are omitted for simplicity).

4.6 Chapter Summary

The initial specification of the nonlinear control law is obtained from a set of linear controllers that are designed at thirty-four nominal conditions, chosen from the steady-level flight envelope of the aircraft. These controllers are computed for the motivating linear control structure, the proportional-integral controller, by implicit model following, which is a well-established procedure that prescribes desired handling qualities and criteria by means of ideal linear models. This phase of the design can be carried out

independently for the longitudinal and the lateral-directional dynamics, simplifying the computation and testing of the control gains. The nonlinear controller is obtained by substituting the linear gains of the proportional-integral controller with nonlinear neural networks that bear the same name and the same input/output structure. As a consequence, an exact correspondence is found between the gradient of the nonlinear neural networks and the linear gains they are replacing.

An algebraic approach is used to determine the neural architectures and parameters that match the set of control gains exactly in one step, by solving linear systems of equations. Subsequently, the neural network controller with the initialized parameters held fixed is tested throughout the aircraft flight envelope. At the design points, its performance is identical to that of the corresponding linear controller; at the interpolation points, its performance is very close to that of a linear controller that is specifically designed for the given flight conditions. This demonstrates that the algebraic gradient-based technique not only matches the training set exactly, but also provides excellent generalization properties. A forward neural network also is trained based on the full trim map of the airplane using an approximate, input/output-based algebraic algorithm.

Chapter 5

Adaptation of the Neural Network Control System

In this chapter, the neural network controller adapts to large-angle maneuvers, control failures, and parameter variations, learning to deal with new system dynamics as they arise. The on-line adaptation is based on an approximate dynamic programming (ADP) approach. Neural networks are the parametric structures used to predict the control system's performance into the future, in an effort to reduce computational complexity. Unlike most dynamic programming algorithms, where convergence to the optimal control law and evaluation function requires multiple generation and expansion of the state vector, ADP iterates only on one value of the state and incorporates results at the corresponding time step. By using this iterative successive-approximation concept and Bellman's optimality conditions, it can be shown that, over time, the solution converges to the optimal trajectory [16].

In practice, it has been observed that when the neural control system learns by means of a scalar evaluation function (such as the cost-to-go or value function) convergence to the globally optimizing solution is slow for any reasonable-sized state vector [18, 91]. Instead, approximating the gradient of the value function by a neural network accelerates convergence, and it alleviates the need for persistence of excitation [92]. The Dual Heuristic Programming (DHP) approach [18-20] also guarantees convergence to the optimal solution over time, provided that the control law (action) and the value function gradient (critic) are adapted according to criteria that can be derived from the recurrence relation of Dynamic Programming (Section 2.4).

The use of neural networks as function approximators for the action and the critic functionals allows both for incremental adaptation and efficient system initialization. Prior control knowledge is incorporated off line solely by initializing the neural network parameters, as explained in Chapter 4. On line, a plant model also is used within the DHP adaptive critic architecture to predict the state and the transition matrices one step into the future. In the following sections, both theory and simulations show that the on-line learning algorithm can retain and benefit from *a-priori* knowledge, while improving system performance incrementally over time when subject to unforeseen conditions. A necessary assumption is that the action and critic functionals to be approximated belong to the class of functions that the respective neural networks can generate for different values of their adjustable parameters. While it can be shown that the number of nodes in these networks is sufficient for approximating prior knowledge as well as additional on-line information, a proof that guarantees the latter statement to hold under all possible circumstances (e.g., for all values of the state, parameter variations, and control failures) is not available. In some sense, it seems unlikely that such a guarantee be provided for truly unforeseen conditions.

5.1 Dual Heuristic Adaptive Critic Design

The DHP adaptive critic design introduced in Section 2.3 is recaptured in Fig. 41, where the sequence of events suggested by the successive approximation approach is emphasized. This on-line logic is implemented in discrete time, with the events illustrated in Fig. 41 taking place during every time interval $\Delta t = t_{k+1} - t_k$.

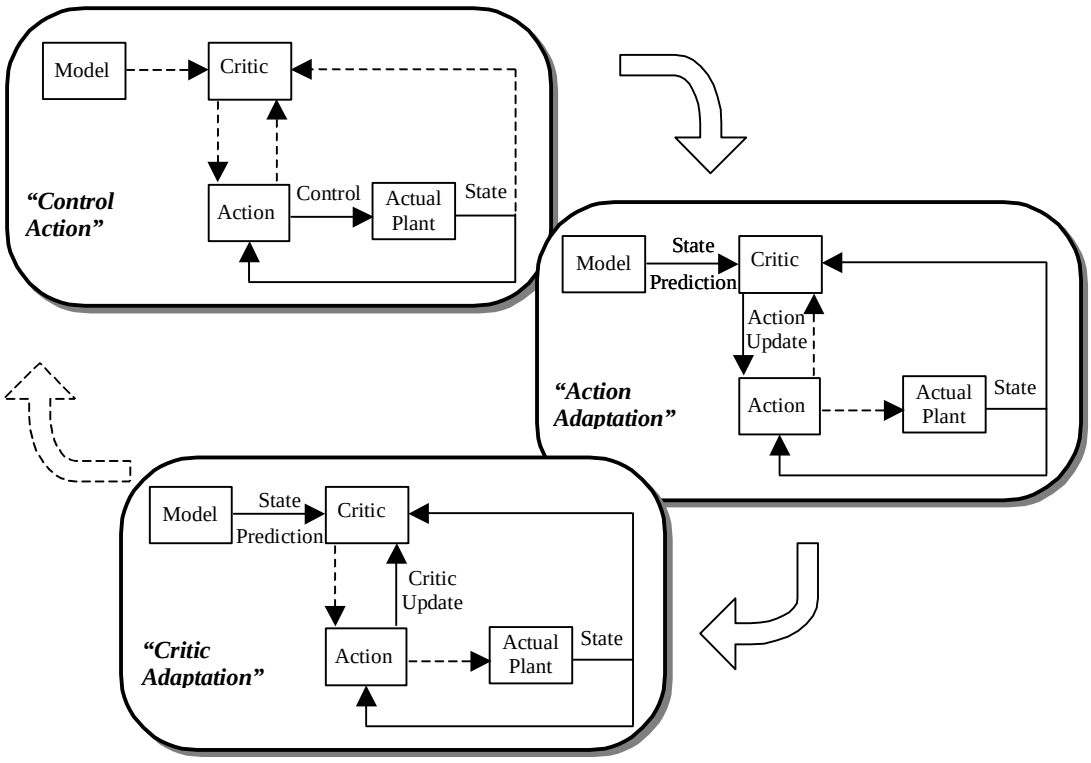

Figure 41. Event sequence performed during the time interval $\Delta t = t_{k+1} - t_k$, by the DHP adaptive critic architecture (the solid lines represent the events that are taking place).

The DHP approach can be related to the pre-trained neural control architecture specified in Chapter 4 by realizing that the same metric optimized by the linear control designs is to be optimized during the on-line adaptation. Subsequently, the input/output structure of the action and critic networks can be identified, and any other details of the adaptive critic architecture can be defined. The sampled-data *utility function* or *Lagrangian* introduced in Section 2.4 can be obtained from the PI cost function in eq. 107:

$$L_{SD}\left[\mathbf{x}_a(t_k),\quad \tilde{\mathbf{u}}(t_k)\right] = \frac{1}{2}\left[\mathbf{x}_a^T(t_k)\mathbf{Q}_a\mathbf{x}_a(t_k) + 2\mathbf{x}_a^T(t_k)\mathbf{M}_a\tilde{\mathbf{u}}(t_k) + \tilde{\mathbf{u}}^T(t_k)\mathbf{R}_a\tilde{\mathbf{u}}(t_k)\right] \tag{182}$$

The utility can also be viewed as the cost that accrues during one time interval or stage.

The augmented state $\mathbf{x}_a$ and the control deviation $\tilde{\mathbf{u}}$ are defined as in Section 4.1.1, with respect to the full state and control of the plant, **x** and **u**, respectively. Since these vectors can be partitioned into their longitudinal and lateral-directional components, as $\mathbf{x}_a = [\,\tilde{\mathbf{x}}_L^T \;\; \tilde{\mathbf{x}}_{LD}^T \;\; \xi_L^T \;\; \xi_{LD}^T\,]^T$ and $\tilde{\mathbf{u}} = [\,\tilde{\mathbf{u}}_L^T \;\; \tilde{\mathbf{u}}_{LD}^T\,]^T$, the weighting matrices in eq. 182 can be composed from the longitudinal and lateral-directional matrices obtained in Section 4.1.3:

$$\mathbf{Q}_a = \begin{bmatrix} \mathbf{Q}_L & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{Q}_{LD} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{Q}_{\xi_L} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{Q}_{\xi_{LD}} \end{bmatrix} \tag{183}$$

$$\mathbf{M}_a = \begin{bmatrix} \mathbf{M}_L & \mathbf{0} \\ \mathbf{0} & \mathbf{M}_{LD} \\ \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \tag{184}$$

$$\mathbf{R}_a = \begin{bmatrix} \mathbf{R}_L & \mathbf{0} \\ \mathbf{0} & \mathbf{R}_{LD} \end{bmatrix} \tag{185}$$

Since these matrices are designed for the set of operating points *OP* (Fig. 15), their value throughout *OR* is decided using a look-up table approach based on the scheduling vector **a**.

Following the development in Section 2.4, with eq. 182 as the utility function, the recurrence relation (eq. 34) can be formulated in terms of the augmented state and control deviation:

$$\mathrm{V}[\mathbf{x}_a(t_k)] = \mathrm{L}_{SD}[\mathbf{x}_a(t_k), \tilde{\mathbf{u}}(t_k)] + \mathrm{V}[\mathbf{x}_a(t_{k+1})] \tag{186}$$

It follows that the criteria,

$$\frac{\partial \mathrm{V}[\mathbf{x}_a(t_k)]}{\partial \tilde{\mathbf{u}}(t_k)} = \frac{\partial \mathrm{L}_{SD}[\mathbf{x}_a(t_k), \tilde{\mathbf{u}}(t_k)]}{\partial \tilde{\mathbf{u}}(t_k)} + \boldsymbol{\lambda}_a(t_{k+1}) \frac{\partial \mathbf{x}_a(t_{k+1})}{\partial \tilde{\mathbf{u}}(t_k)} = 0 \tag{187}$$

and,

$$\boldsymbol{\lambda}_a(t_k) \equiv \frac{\partial \mathrm{V}[\mathbf{x}_a(t_k)]}{\partial \mathbf{x}_a(t_k)} = \frac{\partial \mathrm{L}_{SD}[\mathbf{x}_a(t_k), \tilde{\mathbf{u}}(t_k)]}{\partial \mathbf{x}_a(t_k)} + \frac{\partial \mathrm{L}_{SD}[\mathbf{x}_a(t_k), \tilde{\mathbf{u}}(t_k)]}{\partial \tilde{\mathbf{u}}(t_k)} \frac{\partial \tilde{\mathbf{u}}[\mathbf{x}_a(t_k)]}{\partial \mathbf{x}_a(t_k)} + \boldsymbol{\lambda}_a(t_{k+1}) \frac{\partial \mathbf{x}_a(t_{k+1})}{\partial \mathbf{x}_a(t_k)} + \boldsymbol{\lambda}_a(t_{k+1}) \frac{\partial \mathbf{x}_a(t_{k+1})}{\partial \tilde{\mathbf{u}}(t_k)} \frac{\partial \tilde{\mathbf{u}}[\mathbf{x}_a(t_k)]}{\partial \mathbf{x}_a(t_k)}$$

(188)

can be used for the on-line adaptation of the PI neural network control system, provided the action and critic networks approximate the following input/output relations:

$$\begin{aligned} \tilde{\mathbf{u}}(t_k) &= \mathbf{NN}_A[\mathbf{p}_a(t_k)] \\ \boldsymbol{\lambda}_a(t_k) &= \mathbf{NN}_C[\mathbf{p}_a(t_k)] \end{aligned} \tag{189}$$

As anticipated in Section 2.3, the input to both networks includes the state and the scheduling vector; thus $\mathbf{p}_a(t_k) = [\mathbf{x}_a(t_k)^T \ \mathbf{a}(t_k)^T]^T$. Sections 5.1.2 and 5.1.3 describe the implementation of the on-line adaptation scheme derived thus far. In the following section, the architecture and the initial parameters of the action and critic networks are obtained based on the results from Sections 4.3 and 4.4.

5.1.1 Action and Critic Network Initialization

The initial parameters of the action and critic networks can be determined algebraically from the weights of the pre-trained neural networks described in Sections 4.3 and 4.4. Because of the decoupled nature of the linear designs, the pre-trained feedback, the command-integral, and the critic scalar networks were initialized independently. During testing (Section 4.3), their scalar outputs were grouped together, as in eq. 155, 159, 178, and 179, to form the desired vector outputs. The initialized action and critic networks (eq. 189) are expected to perform equivalently to these decoupled networks; also, they must have coupled inputs and outputs to, potentially, capture any possible form of the functionals in eq. 189, during on-line training. This

means that all q inputs and r outputs of the action and critic networks must be connected to all of their s hidden nodes. This approach to initializing the neural networks guarantees that they will be capable of assimilating on-line information while preserving *a-priori* knowledge, as will be demonstrated in Section 5.3.

The vector-output sigmoidal architecture shown in Fig. 42 is used for the full action and critic networks, $\mathbf{NN}_A$ and $\mathbf{NN}_C$. The output of the network is computed similarly to eq. 39, as:

$$\mathbf{z} = \mathbf{V}\boldsymbol{\sigma}\left[\mathbf{W}\mathbf{p} + \mathbf{d}\right] + \mathbf{b} \tag{190}$$

The derivative of each output with respect to each of the inputs is given by:

$$\frac{\partial z_l}{\partial p_j} = \sum_{i=1}^{s} \frac{\partial z_l}{\partial n_i} \frac{\partial n_i}{\partial p_j} = \sum_{i=1}^{s} v_{li} \sigma'(n_i) w_{ij} \quad j = 1, \ldots, q \text{ and } l = 1, \ldots, r \tag{191}$$

The vector-output neural network notation is the same as that introduced in Section 3.1, except for the output weights that are organized into a matrix $\mathbf{V} = \{v_{li}\}$, where v_{li} represents the connection weight between the i^{th}-node and the l^{th}-output of the network.

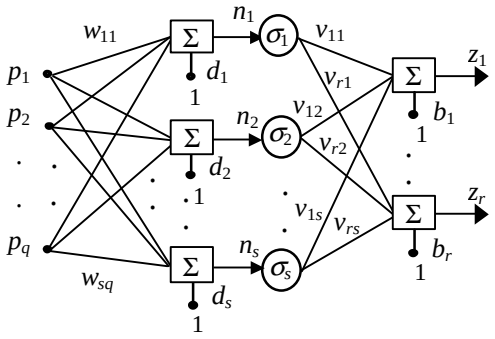

Figure 42. Sample vector-output network with q inputs, s hidden nodes, and r outputs.

Algebraic operations that combine networks with the same output and different inputs, or that superimpose networks with the same input/output structure are presented in Sections 3.2 and 3.1.3. Similarly, networks with the same input and different outputs or

with different inputs and outputs can be combined and summed together, ultimately producing an architecture such as that in Fig. 42, as illustrated below. Suppose two networks with a common input $\mathbf{x}$ and with outputs $\mathbf{u}_1$ and $\mathbf{u}_2$, respectively, have been trained (or algebraically initialized) based on corresponding training sets, producing the parameters $\mathbf{W}_1$, $\mathbf{d}_1$, $\mathbf{V}_1$, and $\mathbf{b}_1$ for the first network and the parameters $\mathbf{W}_2$, $\mathbf{d}_2$, $\mathbf{V}_2$, and $\mathbf{b}_2$ for the second network. Then, a neural network with input $\mathbf{x}$ and output $[\mathbf{u}_1{}^T\ \mathbf{u}_2{}^T]^T$, that performs identically to the two decoupled networks (i.e., that approximates both training sets) can be obtained, as suggested by Fig. 43.

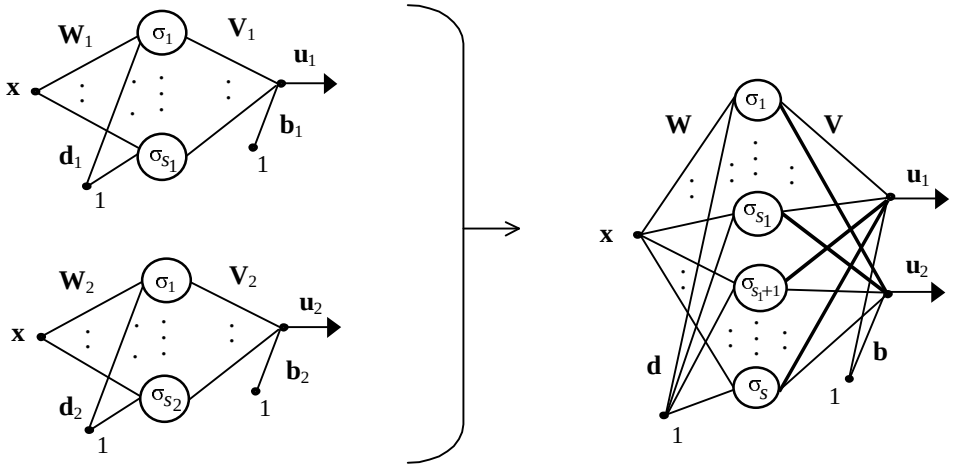

Figure 43. Two neural networks (with s_1 and s_2 nodes, respectively) are combined into one s-node network with the same input $\mathbf{x}$ and a combination of their outputs $\mathbf{u}_1$ and $\mathbf{u}_2$. The bold lines represent the new connections being introduced.

The number of nodes in the new network, s, is equal to the sum of the nodes in the original networks, s_1 and s_2, respectively ($s = s_1 + s_2$). Its initial input-weight matrix is obtained from $\mathbf{W}_1$ and $\mathbf{W}_2$,

$$\mathbf{W} = \begin{bmatrix} \mathbf{W}_1 \\ \mathbf{W}_2 \end{bmatrix} \tag{192}$$

and its initial output weight matrix is obtained from the weight matrices $\mathbf{V}_1$ and $\mathbf{V}_2$:

$$\mathbf{V} = \begin{bmatrix} \mathbf{V}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{V}_2 \end{bmatrix} \tag{193}$$

The new network biases are $\mathbf{d} = [\mathbf{d}_1{}^T \; \mathbf{d}_2{}^T]^T$ and $\mathbf{b} = [\mathbf{b}_1{}^T \; \mathbf{b}_2{}^T]^T$. The weight equations of these networks can be used to prove that the above operation preserves performance (as shown in Appendix C).

A similar operation can be performed to combine two (or more) networks with different inputs, $\mathbf{x}_1$ and $\mathbf{x}_2$, and different outputs, $\mathbf{u}_1$ and $\mathbf{u}_2$, as sketched in Fig. 44. Again, the number of nodes in the new network equals the sum of the nodes in the original networks. The initial input weight matrix can be formed from the original parameters as follows,

$$\mathbf{W} = \begin{bmatrix} \mathbf{W}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{W}_2 \end{bmatrix} \tag{194}$$

and the relations used in the previous case hold for the remaining weights, **V**, **d**, and **b**. A neural network whose input is the combination of $\mathbf{x}_1$ and $\mathbf{x}_2$, and whose output is the sum of $\mathbf{u}_1$ and $\mathbf{u}_2$, also can be obtained algebraically from the decoupled networks' parameters (Fig. 45), provided $z_1 = z_2$. The summing network's output weight matrix is $\mathbf{V} = [\mathbf{V}_1 \; \mathbf{V}_2]$. The input-weight matrix can be computed as in eq. 194, and the new biases are $\mathbf{d} = [\mathbf{d}_1{}^T \; \mathbf{d}_2{}^T]^T$ and $\mathbf{b} = \mathbf{b}_1 + \mathbf{b}_2$.

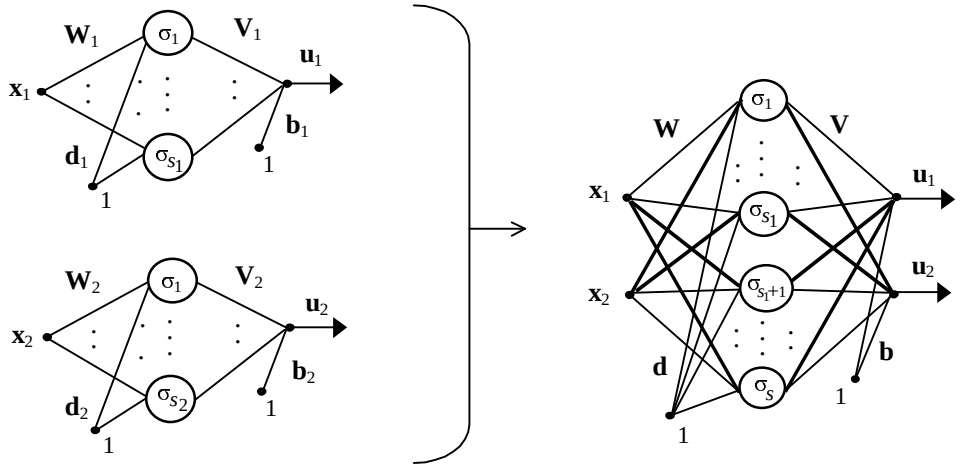

Figure 44. Two neural networks (with s_1 and s_2 nodes) are combined into one s-node network with a combination of inputs, $\mathbf{x}_1$ and $\mathbf{x}_2$, and outputs, $\mathbf{u}_1$ and $\mathbf{u}_2$. The bold lines represent the new connections being introduced.

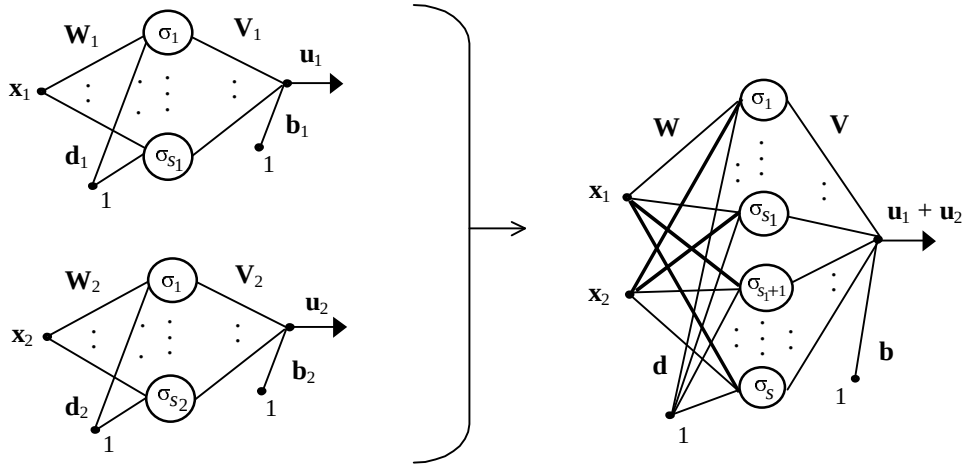

Figure 45. Two neural networks (with s_1 and s_2 nodes) are summed to produce one s-node network with inputs $\mathbf{x}_1$ and $\mathbf{x}_2$ and with output $(\mathbf{u}_1 + \mathbf{u}_2)$. The bold lines represent the new connections being introduced.

These algebraic operations can be used to obtain the parameters of $\mathbf{NN}_A$ and $\mathbf{NN}_C$, such that, their performance is initially equivalent to that of the pre-trained neural networks. That is, the initialized $\mathbf{NN}_A$ and $\mathbf{NN}_C$ in eq. 189 also satisfy the linear-control requirements in eq. 152, 154, 157, 158, 176, and 177. Hence, their initial performance will be satisfactory with respect to the design criteria established in Section 4.1. A full,

feedback neural network $\mathbf{NN}_B$ that has the required input/output structure (eq. 151) and an architecture of the type in Fig. 42 can be obtained from the scalar feedback networks in eq. 155. This neural network can be abbreviated as $\Delta\mathbf{u}_B = \mathbf{NN}_B[\,\tilde{\mathbf{x}}\,,\,\mathbf{a}]$, and is shown in Fig. 46.

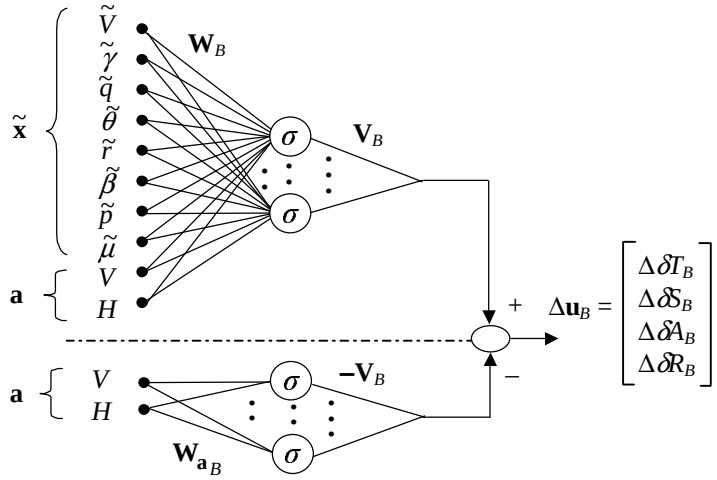

Figure 46. Architecture of the feedback neural network $\mathbf{NN}_B$ (input and output biases are not shown, for simplicity).

The initialized parameters of the scalar networks carry an homonymous subscript (according to the notation of eq. 155) and are known from Section 4.3. According to the above network operations, the initial weight matrices of $\mathbf{NN}_B$ are given by,

$$\mathbf{W}_B = \begin{bmatrix} \mathbf{W}_{\tilde{\mathbf{x}}_{B_{L_1}}} & \mathbf{0} & \mathbf{W}_{\mathbf{a}_{B_{L_1}}} \\ \mathbf{W}_{\tilde{\mathbf{x}}_{B_{L_2}}} & \mathbf{0} & \mathbf{W}_{\mathbf{a}_{B_{L_2}}} \\ \mathbf{0} & \mathbf{W}_{\tilde{\mathbf{x}}_{B_{LD_1}}} & \mathbf{W}_{\mathbf{a}_{B_{LD_1}}} \\ \mathbf{0} & \mathbf{W}_{\tilde{\mathbf{x}}_{B_{LD_2}}} & \mathbf{W}_{\mathbf{a}_{B_{LD_2}}} \end{bmatrix} \tag{195}$$

and

$$\mathbf{V}_B = \begin{bmatrix} \mathbf{v}_{B_{L_1}}^T & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{v}_{B_{L_2}}^T & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{v}_{B_{LD_1}}^T & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{v}_{B_{LD_2}}^T \end{bmatrix} \tag{196}$$

The full feedback network's input and output biases are,

$$\mathbf{d}_B = \begin{bmatrix} \mathbf{d}_{B_{L_1}} \\ \mathbf{d}_{B_{L_2}} \\ \mathbf{d}_{B_{LD_1}} \\ \mathbf{d}_{B_{LD_2}} \end{bmatrix} \tag{197}$$

and,

$$\mathbf{b}_B = \begin{bmatrix} b_{B_{L_1}} \\ b_{B_{L_2}} \\ b_{B_{LD_1}} \\ b_{B_{LD_2}} \end{bmatrix} \tag{198}$$

respectively.

Similarly, a full, command-integral neural network $\mathbf{NN}_I$ that models the desired mapping (eq. 151) and has coupled inputs and outputs, is obtained from the scalar command-integral networks in eq. 159. Its architecture can be abbreviated by $\Delta\mathbf{u}_I = \mathbf{NN}_I[\boldsymbol{\xi}, \mathbf{a}]$ (Fig. 47). Its initial input and output weight matrices are obtained from those of the scalar networks, initialized in Section 4.3, as follows,

$$\mathbf{W}_I = \begin{bmatrix} \mathbf{W}_{\xi_{I_{L_1}}} & \mathbf{0} & \mathbf{W}_{\mathbf{a}_{I_{L_1}}} \\ \mathbf{W}_{\xi_{I_{L_2}}} & \mathbf{0} & \mathbf{W}_{\mathbf{a}_{I_{L_2}}} \\ \mathbf{0} & \mathbf{W}_{\xi_{I_{LD_1}}} & \mathbf{W}_{\mathbf{a}_{I_{LD_1}}} \\ \mathbf{0} & \mathbf{W}_{\xi_{I_{LD_2}}} & \mathbf{W}_{\mathbf{a}_{I_{LD_2}}} \end{bmatrix} \tag{199}$$

and

$$\mathbf{V}_I = \begin{bmatrix} \mathbf{v}_{I_{L_1}}^T & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{v}_{I_{L_2}}^T & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{v}_{I_{LD_1}}^T & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{v}_{I_{LD_2}}^T \end{bmatrix} \tag{200}$$

The full, command-integral input and output biases are,

$$\mathbf{d}_I = \begin{bmatrix} \mathbf{d}_{I_{L_1}} \\ \mathbf{d}_{I_{L_2}} \\ \mathbf{d}_{I_{LD_1}} \\ \mathbf{d}_{I_{LD_2}} \end{bmatrix} \tag{201}$$

and,

$$\mathbf{b}_I = \begin{bmatrix} b_{I_{L_1}} \\ b_{I_{L_2}} \\ b_{I_{LD_1}} \\ b_{I_{LD_2}} \end{bmatrix} \tag{202}$$

respectively. They, too, are obtained from the command-integral pre-trained neural networks in Section 4.3, as indicated by the algebraic operations illustrated in Figs. 43 and 44.

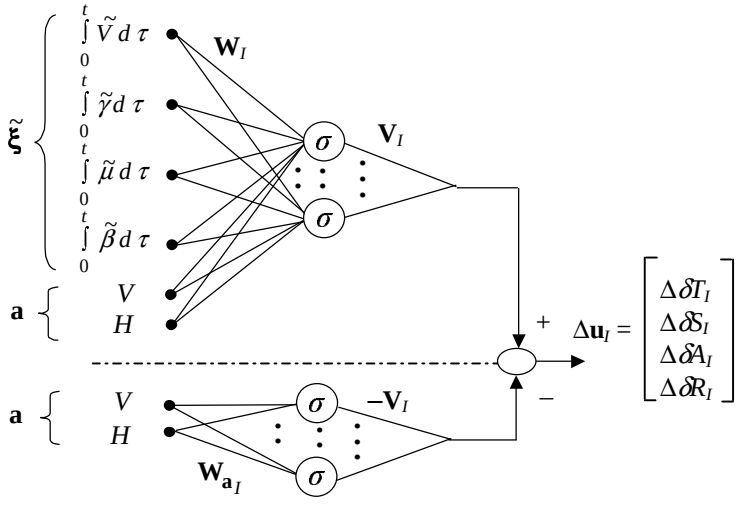

Figure 47. Architecture of the command-integral neural network $\mathbf{NN}_I$ (biases not shown).

A full critic network that models the input/output relation in eq. 174 also can be obtained from the scalar, decoupled critic networks initialized in Section 4.5. Its weights can be computed from those of the networks in eq. 178-179. The initial input-weight matrix is:

$$\mathbf{W}_C = \begin{bmatrix} \mathbf{W}_{\tilde{\mathbf{x}}C_{L_1}} & \mathbf{0} & \mathbf{W}_{\xi C_{L_1}} & \mathbf{0} & \mathbf{W}_{\mathbf{a}C_{L_1}} \\ \mathbf{W}_{\tilde{\mathbf{x}}C_{L_2}} & \mathbf{0} & \mathbf{W}_{\xi C_{L_2}} & \mathbf{0} & \mathbf{W}_{\mathbf{a}C_{L_2}} \\ \mathbf{W}_{\tilde{\mathbf{x}}C_{L_3}} & \mathbf{0} & \mathbf{W}_{\xi C_{L_3}} & \mathbf{0} & \mathbf{W}_{\mathbf{a}C_{L_3}} \\ \mathbf{W}_{\tilde{\mathbf{x}}C_{L_4}} & \mathbf{0} & \mathbf{W}_{\xi C_{L_4}} & \mathbf{0} & \mathbf{W}_{\mathbf{a}C_{L_4}} \\ \mathbf{0} & \mathbf{W}_{\tilde{\mathbf{x}}C_{LD_1}} & \mathbf{0} & \mathbf{W}_{\xi C_{LD_1}} & \mathbf{W}_{\mathbf{a}C_{LD_1}} \\ \mathbf{0} & \mathbf{W}_{\tilde{\mathbf{x}}C_{LD_2}} & \mathbf{0} & \mathbf{W}_{\xi C_{LD_2}} & \mathbf{W}_{\mathbf{a}C_{LD_2}} \\ \mathbf{0} & \mathbf{W}_{\tilde{\mathbf{x}}C_{LD_3}} & \mathbf{0} & \mathbf{W}_{\xi C_{LD_3}} & \mathbf{W}_{\mathbf{a}C_{LD_3}} \\ \mathbf{0} & \mathbf{W}_{\tilde{\mathbf{x}}C_{LD_4}} & \mathbf{0} & \mathbf{W}_{\xi C_{LD_4}} & \mathbf{W}_{\mathbf{a}C_{LD_4}} \\ \mathbf{W}_{\tilde{\mathbf{x}}C_{L_5}} & \mathbf{0} & \mathbf{W}_{\xi C_{L_5}} & \mathbf{0} & \mathbf{W}_{\mathbf{a}C_{L_5}} \\ \mathbf{W}_{\tilde{\mathbf{x}}C_{L_6}} & \mathbf{0} & \mathbf{W}_{\xi C_{L_6}} & \mathbf{0} & \mathbf{W}_{\mathbf{a}C_{L_6}} \\ \mathbf{0} & \mathbf{W}_{\tilde{\mathbf{x}}C_{LD_5}} & \mathbf{0} & \mathbf{W}_{\xi C_{LD_5}} & \mathbf{W}_{\mathbf{a}C_{LD_5}} \\ \mathbf{0} & \mathbf{W}_{\tilde{\mathbf{x}}C_{LD_6}} & \mathbf{0} & \mathbf{W}_{\xi C_{LD_6}} & \mathbf{W}_{\mathbf{a}C_{LD_6}} \end{bmatrix} \tag{203}$$

The initial output-weight matrix, $\mathbf{V}_C$, is obtained by placing the vectors $\mathbf{v}_{C_{L_1}}^T$, $\mathbf{v}_{C_{L_2}}^T$, $\mathbf{v}_{C_{L_3}}^T$, $\mathbf{v}_{C_{L_4}}^T$, $\mathbf{v}_{C_{LD_1}}^T$, $\mathbf{v}_{C_{LD_2}}^T$, $\mathbf{v}_{C_{LD_3}}^T$, $\mathbf{v}_{C_{LD_4}}^T$, $\mathbf{v}_{C_{L_5}}^T$, $\mathbf{v}_{C_{L_6}}^T$, $\mathbf{v}_{C_{LD_5}}^T$, and $\mathbf{v}_{C_{LD_6}}^T$ along the diagonal of a zero matrix, in this order, similarly to eq. 196 and 200. The input and output biases are,

$$\mathbf{d}_C = [\mathbf{d}_{C_{L_1}}^T \; \mathbf{d}_{C_{L_2}}^T \; \mathbf{d}_{C_{L_3}}^T \; \mathbf{d}_{C_{L_4}}^T \; \mathbf{d}_{C_{LD_1}}^T \; \mathbf{d}_{C_{LD_2}}^T \; \mathbf{d}_{C_{LD_3}}^T \; \mathbf{d}_{C_{LD_4}}^T \; \mathbf{d}_{C_{L_5}}^T \; \mathbf{d}_{C_{L_6}}^T \; \mathbf{d}_{C_{LD_5}}^T \; \mathbf{d}_{C_{LD_6}}^T]^T \tag{204}$$

and,

$$\mathbf{b}_C = [b_{C_{L_1}} \; b_{C_{L_2}} \; b_{C_{L_3}} \; b_{C_{L_4}} \; b_{C_{LD_1}} \; b_{C_{LD_2}} \; b_{C_{LD_3}} \; b_{C_{LD_4}} \; b_{C_{L_5}} \; b_{C_{L_6}} \; b_{C_{LD_5}} \; b_{C_{LD_6}}]^T \tag{205}$$

respectively. The full critic neural network, $\mathbf{NN}_C$, is shown in Fig. 48. Since its input/output structure is fully coupled and consistent with eq. 189, it is now ready for use in the adaptive control system.

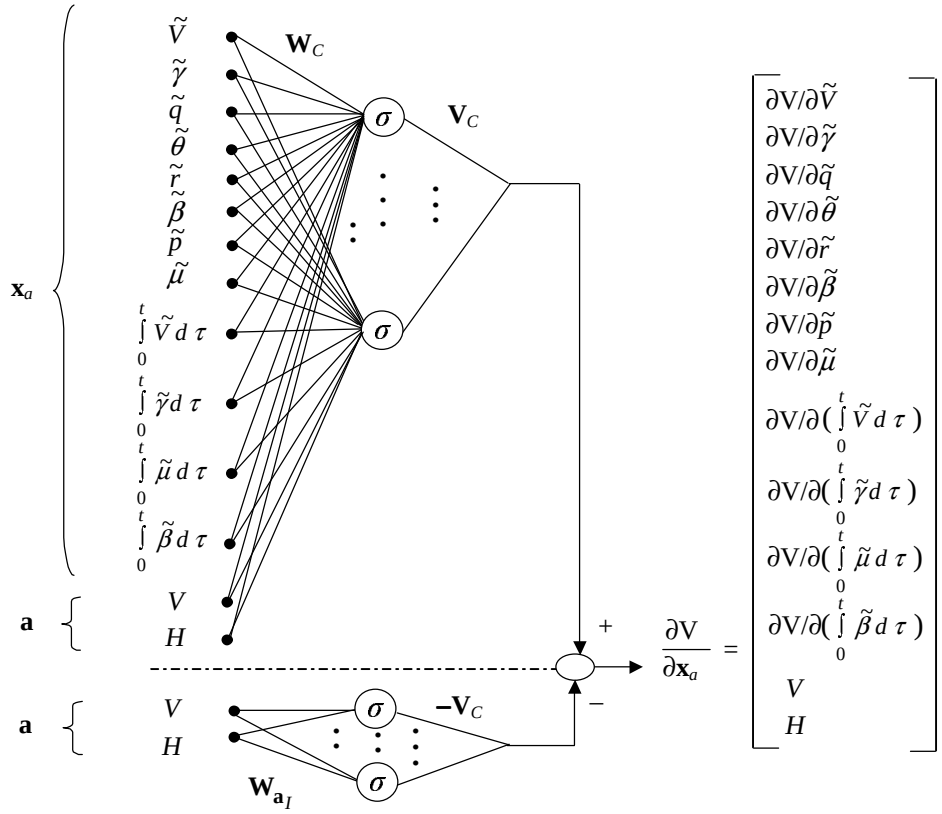

Figure 48. Architecture of the critic neural network $\mathbf{NN}_C$ (input and output biases are not shown, for simplicity).

According to eq. 189 and 153, the action-network output, $\tilde{\mathbf{u}}$, must equal the sum of the feedback and command-integral network outputs, $\Delta\mathbf{u}_B$ and $\Delta\mathbf{u}_I$. Also, the input $\mathbf{p}_a$ is a combination of the feedback and command-integral inputs, $\tilde{\mathbf{x}}$, $\boldsymbol{\xi}$, and $\mathbf{a}$. Therefore, the architecture and the initial parameters of the action network can be determined from $\mathbf{NN}_B$ and $\mathbf{NN}_I$, through the algebraic network addition described above. The action input and output weight matrices are initialized as follows:

$$\mathbf{W}_A = \left[\begin{array}{c|c|c} \mathbf{W}_{\tilde{\mathbf{x}}_B} & \mathbf{0} & \mathbf{W}_{\mathbf{a}_B} \\ \hline \mathbf{0} & \mathbf{W}_{\xi_I} & \mathbf{W}_{\mathbf{a}_I} \end{array}\right] = \left[\begin{array}{cccc|c} \mathbf{W}_{\tilde{\mathbf{x}}_{B_{L_1}}} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{W}_{\mathbf{a}_{B_{L_1}}} \\ \mathbf{W}_{\tilde{\mathbf{x}}_{B_{L_2}}} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{W}_{\mathbf{a}_{B_{L_2}}} \\ \mathbf{0} & \mathbf{W}_{\tilde{\mathbf{x}}_{B_{LD_1}}} & \mathbf{0} & \mathbf{0} & \mathbf{W}_{\mathbf{a}_{B_{LD_1}}} \\ \mathbf{0} & \mathbf{W}_{\tilde{\mathbf{x}}_{B_{LD_2}}} & \mathbf{0} & \mathbf{0} & \mathbf{W}_{\mathbf{a}_{B_{LD_2}}} \\ \hline \mathbf{0} & \mathbf{0} & \mathbf{W}_{\xi_{I_{L_1}}} & \mathbf{0} & \mathbf{W}_{\mathbf{a}_{I_{L_1}}} \\ \mathbf{0} & \mathbf{0} & \mathbf{W}_{\xi_{I_{L_2}}} & \mathbf{0} & \mathbf{W}_{\mathbf{a}_{I_{L_2}}} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{W}_{\xi_{I_{LD_1}}} & \mathbf{W}_{\mathbf{a}_{I_{LD_1}}} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{W}_{\xi_{I_{LD_2}}} & \mathbf{W}_{\mathbf{a}_{I_{LD_2}}} \end{array}\right] \tag{206}$$

$$\mathbf{V}_A = \left[\mathbf{V}_B \,|\, \mathbf{V}_I\right] = \left[\begin{array}{cccc|cccc} \mathbf{v}^T_{B_{L_1}} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{v}^T_{I_{L_1}} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{v}^T_{B_{L_2}} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{v}^T_{I_{L_2}} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{v}^T_{B_{LD_1}} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{v}^T_{I_{LD_1}} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{v}^T_{B_{LD_2}} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{v}^T_{I_{LD_2}} \end{array}\right] \tag{207}$$

The action input and output biases are $\mathbf{d}_A = [\mathbf{d}_B{}^T \; \mathbf{d}_I{}^T]^T$ and $\mathbf{b}_A = \mathbf{b}_B + \mathbf{b}_I$. Figure 49 shows the final architecture of the action network to be used in the on-line adaptive controller discussed in the following section.

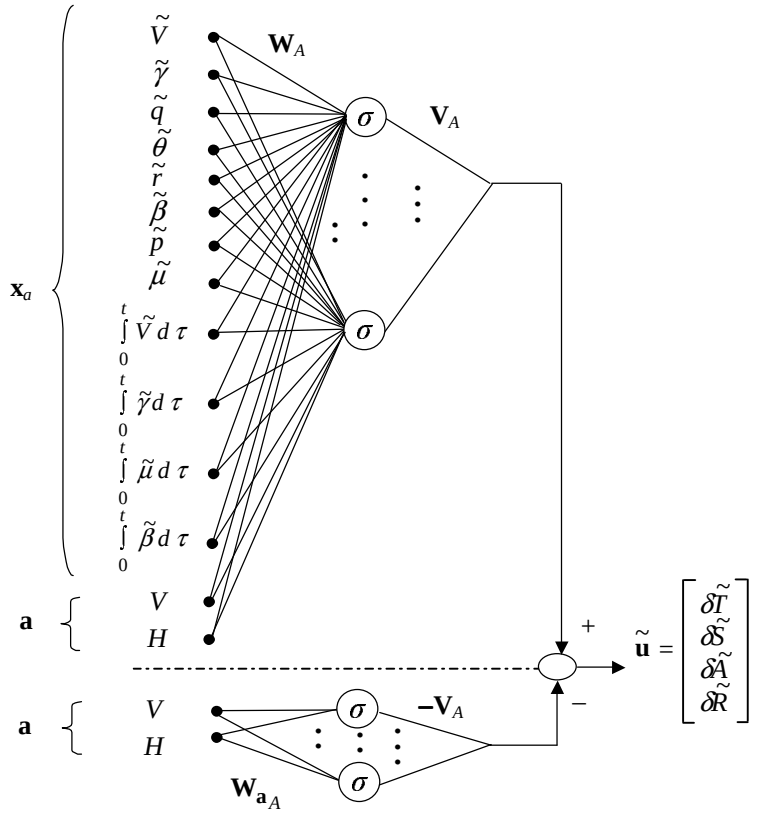

Figure 49. Architecture of the action neural network $\mathbf{NN}_A$ (input and output biases are not shown, for simplicity).

5.1.2 Action and Critic Network On-line Adaptation

A nonlinear controller that adapts on line based on the DHP adaptive critic architecture is shown in Fig. 50. The proportional-integral designs obtained in Section 4.1 are contained, implicitly, in the action and critic network's parameters initialized in the preceding section. During each time interval $\Delta t = t_{k+1} - t_k$, the networks are adapted based on the actual state of the aircraft, $\mathbf{x}(t_k)$, to more closely approximate the globally-optimal control law and value function derivatives, through the criteria in eq.s 187 and 188. Figures 51 and 52 show that the implementation of these criteria involves an on-going flow of information between the action and the critic that also is illustrated in Figs.

41 and 50. In particular, eq. 187 and 188 are used to generate desired outputs or *targets* for the action and the critic, denoted by $\tilde{\mathbf{u}}_D(t_k)$ and $\boldsymbol{\lambda}_{a_D}(t_k)$, that correspond to the present value of their input, i.e., $\mathbf{p}_a(t_k)$. The parameters of each network are updated to minimize the mean-squared error between the target and its actual output, $\mathbf{z}[\mathbf{p}_a(t_k)]$. During the first time interval, $(t_1 - t_0)$, the initialized network weights are used before each network's update. Afterward, the weights obtained during $(t_k - t_{k-1})$ are used as initial weights for the interval $(t_{k+1} - t_k)$.

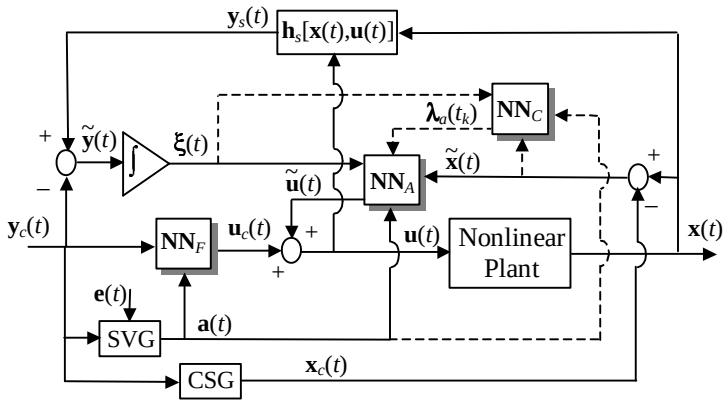

Figure 50. Action critic neural network controller. The dashed lines represent the flow of information for the adaptation, during the time interval $(t_{k+1} - t_k)$.

The action network's target, $\tilde{\mathbf{u}}_D(t_k)$, is obtained by solving the optimality condition (eq. 187), which consists of a set of nonlinear equations. The MATLAB function fsolve (part of the Optimization Toolbox) is used for solving these equations by a least-squares method. The initial guess to their solution, $\tilde{\mathbf{u}}_D(t_k)^G$, is provided by the action network using its latest weights obtained during $(t_k - t_{k-1})$, and is perturbed by the chosen algorithm (e.g., Newton-Raphson) until the default stopping condition is met. The search for $\tilde{\mathbf{u}}_D(t_k)$ can be constrained to the physically-meaningful values of $\tilde{\mathbf{u}}$ by assigning

exponential weighting to the control elements beyond the reasonable bounds. Solving the optimality condition independently of the action weights' update is considerably more effective then optimizing eq. 187 with respect to the network weights (that is, the true reinforcement-learning approach). It allows the algorithm to explore a reduced, *m*-dimensional parameter space, and to recover from bad solutions (e.g., lying outside the bounds) before the action network is affected.

Given a guess for the solution, $\tilde{\mathbf{u}}_D(t_k)^G$, and the actual value of $\mathbf{x}_a(t_k)$, the optimality condition can be evaluated from the quantities in eq. 187, as described in Fig. 51. The utility function derivative is computed analytically from eq. 182:

$$\frac{\partial \mathrm{L}_{SD}[\mathbf{x}_a(t_k),\ \tilde{\mathbf{u}}(t_k)]}{\partial \tilde{\mathbf{u}}(t_k)} = \mathbf{x}_a^T(t_k)\mathbf{M}_a + \tilde{\mathbf{u}}^T(t_k)\mathbf{R}_a \tag{208}$$

The following equation represents the sampled-data model for the augmented system:

$$\mathbf{x}_a(t_{k+1}) = \mathbf{f}_{a\,SD}[\mathbf{x}_\mathbf{a}(t_k), \mathbf{p}_m(t_k), \tilde{\mathbf{u}}(t_k)],\ \ \mathbf{x}_a(t_0) \text{ given} \tag{209}$$

It is obtained numerically by using a Runga-Kutta algorithm [90] to integrate the aircraft simulation (eq. 1) described in Chapter 4 and the ordinary differential equation governing the behavior of $\boldsymbol{\xi}$ in continuous time, i.e.:

$$\dot{\boldsymbol{\xi}}(t) = \begin{bmatrix} \mathbf{H}_{\mathbf{x}_L} & \mathbf{0} \\ \mathbf{0} & \mathbf{H}_{\mathbf{x}_{LD}} \end{bmatrix} \begin{bmatrix} \tilde{\mathbf{x}}_L(t) \\ \tilde{\mathbf{x}}_{LD}(t) \end{bmatrix} = \mathbf{H}_\mathbf{x}\tilde{\mathbf{x}}(t) \tag{210}$$

Then, the discrete model in eq. 209 is perturbed numerically, using the MATLAB function numjac [79], to obtain the transition matrix $\partial \mathbf{x}_a(t_{k+1})/\partial \tilde{\mathbf{u}}(t_k)$. This model also is needed to predict $\mathbf{x}_a(t_{k+1})$, based on $\tilde{\mathbf{u}}_D(t_k)$ and $\mathbf{x}_a(t_k)$. Once $\mathbf{x}_a(t_{k+1})$ is known, the critic network can be used to compute $\boldsymbol{\lambda}_a(t_{k+1})$ in eq. 187, based on the latest value of the neural parameters. After the optimality condition has been solved for $\tilde{\mathbf{u}}_D(t_k)$, this target can be

used to update the weights of the action network, by means of the on-line training algorithm (Section 5.1.3).

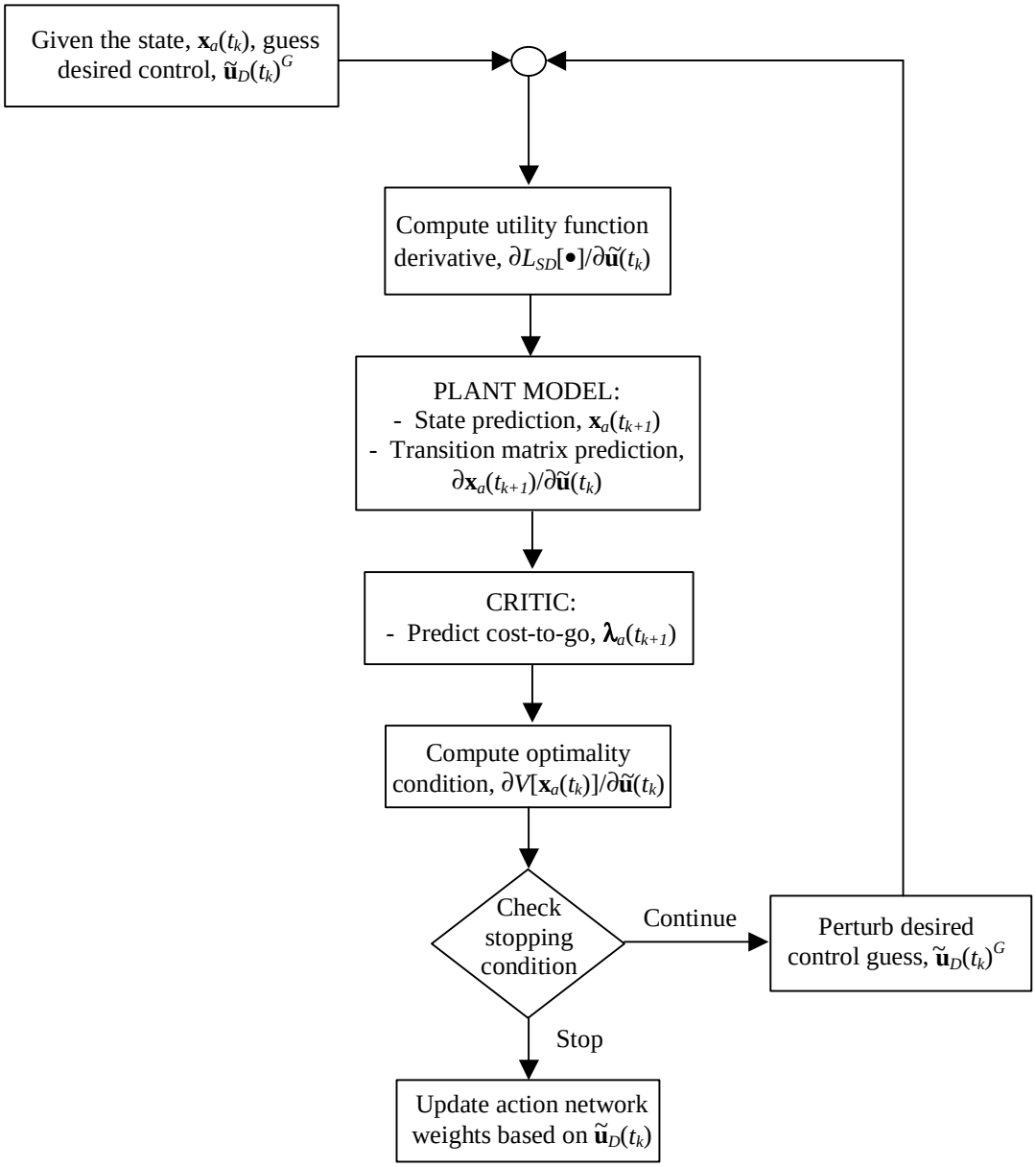

Figure 51. Dual heuristic programming action network adaptation, during $\Delta t = t_{k+1} - t_k$.

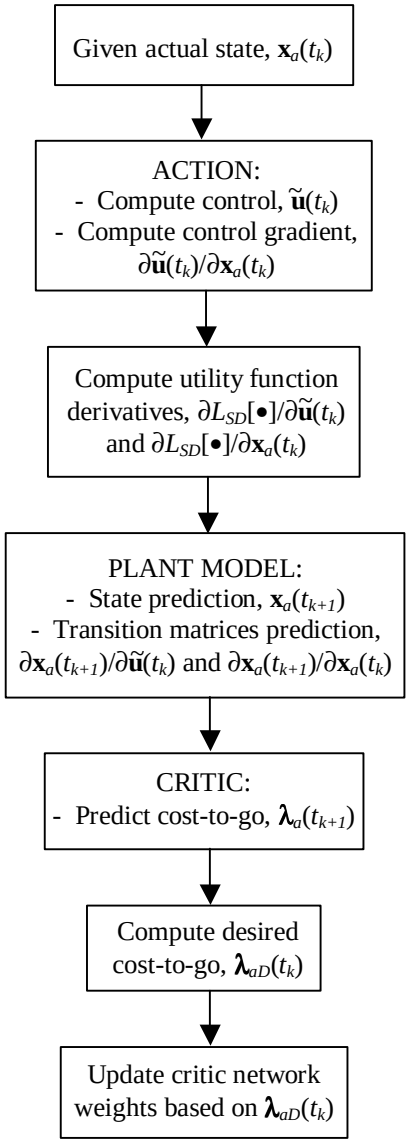

Figure 52. Dual heuristic programming critic network adaptation, during $\Delta t = t_{k+1} - t_k$.

Following the update of the action network's parameters, the critic's desired output $\boldsymbol{\lambda}_{a_D}(t_k)$ is computed from eq. 188 based on $\mathbf{x}_a(t_k)$, as shown in Fig. 52. At this point, the actual values of $\tilde{\mathbf{u}}(t_k)$ and $\partial\tilde{\mathbf{u}}(t_k)/\partial\mathbf{x}_a(t_k)$ also are known. In the case of the critic, no iteration is needed to compute its desired output, which is merely an approximation to

$\boldsymbol{\lambda}_a(t_k)$. In order to determine the right-hand side of eq. 188, eq. 208 must be re-evaluated together with the utility derivative,

$$\frac{\partial \mathrm{L}_{SD}\left[\mathbf{x}_a(t_k),\ \tilde{\mathbf{u}}(t_k)\right]}{\partial \mathbf{x}_a(t_k)} = \mathbf{x}_a^T(t_k)\mathbf{Q}_a + \left[\mathbf{M}_a\tilde{\mathbf{u}}(t_k)\right]^T \tag{211}$$

which also is obtained from eq. 182. The transition matrices $\partial\mathbf{x}_a(t_{k+1})/\partial\tilde{\mathbf{u}}(t_k)$ and $\partial\mathbf{x}_a(t_{k+1})/\partial\mathbf{x}_a(t_k)$ both are computed numerically from eq. 209. Finally, the critic network is needed to produce $\boldsymbol{\lambda}_a(t_{k+1})$ in eq. 187, based on the latest prediction of $\mathbf{x}_a(t_{k+1})$ and on the critic's parameters obtained during $(t_k - t_{k-1})$. After the target $\boldsymbol{\lambda}_{a_D}(t_k)$ becomes available, it can be used to update the critic network through the on-line training algorithm described in the following section.

5.1.3 Neural Network On-line Training Algorithm

As part of the adaptation taking place during the time interval $\Delta t = t_{k+1} - t_k$, the action and the critic neural parameters are updated based on their respective targets ($\tilde{\mathbf{u}}_D(t_k)$ or $\boldsymbol{\lambda}_{a_D}(t_k)$), as computed by the algorithms in Figs. 51 and 52. In particular, each of these networks is updated to more closely approximate its desired output, denoted in general by $\mathbf{z}_D$, as soon as it becomes available. The modified resilient backpropagation (RPROP) algorithm introduced in Section 3.2 is used for this purpose. Given the network input $\mathbf{p}_a(t_k)$, an error function of the type in eq. 86 can be defined with respect to a vector of ordered weights, $\mathbf{w} = \{w_\ell\}$, i.e.:

$$E(\mathbf{w}) \equiv \frac{1}{2}\left\|\mathbf{z}_D - \mathbf{z}(\mathbf{w})\right\|^2 \tag{212}$$

Where, $\mathbf{z}$ denotes the actual network output corresponding to $\mathbf{p}_a(t_k)$. At the time t_k, $\mathbf{w}$ is obtained from the input and output weights of either the action or the critic networks

(depending on which one is being updated). Then, during Δt, $\mathbf{w}(t_k)$ is modified by the RPROP algorithm, ultimately producing the network parameters $\mathbf{w}(t_{k+1})$, for the next moment in time, as sketched in Fig. 53. Based on the idea of backpropagation learning [76], at each iteration or epoch -- indexed by (i) -- the algorithm modifies the value of each weight $w_\ell^{(i)}$ by a small increment of size $\Delta_\ell^{(i)}$ that is based on corresponding derivative information, $\partial E(\mathbf{w})/\partial w_\ell$ (as explained in Section 3.2).

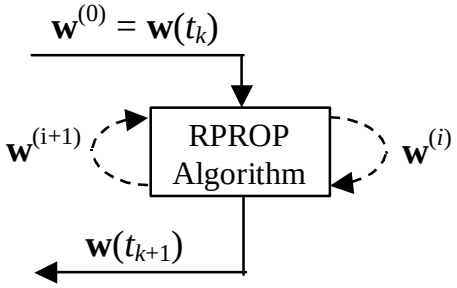

Figure 53. Conceptual illustration of on-line training by a resilient backpropagation algorithm that updates the weights through a number of epochs (i), during $\Delta t = t_{k+1} - t_k$.

Thanks to the pre-training phase, the weights are close to their optimal values, and the minimization can be kept local. The on-line adaptation is effective as well as reliable when the error (eq. 212) is decreased at the on-set of training, indicating an immediate descent toward a nearby minima, and when the update algorithm does not disregard nor degrade prior network weights, which already contain valuable information. Also, the action and critic networks, with the architecture and parameters initialized in Section 5.1.1 are fairly large (having between 102 and 408 nodes), and have weights that are highly dissimilar in magnitude. Therefore, the modified RPROP algorithm obtained in Section 3.2 is particularly well suited for this application. For both networks, the user-specified parameters are: $\eta^+ = 1.2$, $\eta^- = 0.5$ [77], $f_w \sim O(10^{-5})$, and $f_0 << 1$.

To demonstrate the effect of the proposed modifications, the action network $\mathbf{NN}_A$ (initialized in Section 5.1.1) is trained at $t_k = 0.2$ sec using both the MATLAB's RPROP training function trainrp based on the original algorithm [77], but without backtracking, and the proposed modified version. The effectiveness of both updates is tested by carrying out training until its convergence, when the mean square of the network error, i.e.,

$$\mathbf{e}_{\mathbf{NN}} \equiv \mathbf{z}_D - \mathbf{z}(\mathbf{w}) \tag{213}$$

ceases to change. The network error is plotted vs. the number of epochs as a measure of performance in Fig. 54. The results are typical among a number of simulations that involved both the action and the critic networks.

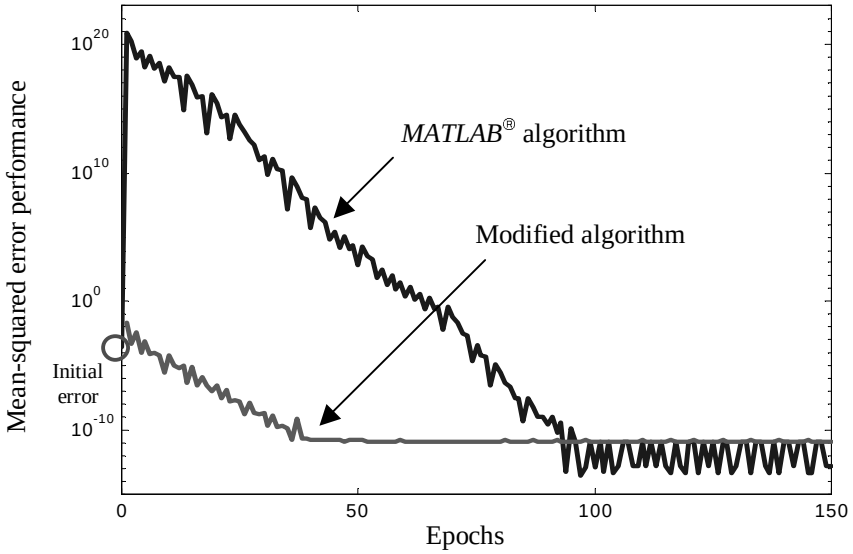

Figure 54. Performance comparison between the *MATLAB*® resilient backpropagation algorithm and its modified version, for the action network training at $t_k = 0.2$ sec.

The modified algorithm begins decreasing the error by the third epoch, and it converges to almost the same performance as the MATLAB function in half the number of epochs. Although the initial error is relatively small in magnitude, the MATLAB

function begins with too large an increment, and degrades the initial weights by looking for minima that are far away because of the absence of the backtracking feature. As a consequence, the error increases considerably (in this case, of approximately twenty orders of magnitude) before the appropriate increment size is found. Using eq. 91 for the initial increment size, $\Delta_\ell^{(0)}$, typically diminishes the number of epochs required to adapt Δ_ℓ; meanwhile, backtracking prevents the algorithm from degrading the weights. This can be verified, for example, by monitoring the size of the increments and the number of weights for which backtracking is performed at every epoch.

The final network error alone is not a key component for the comparison of the two algorithms, because it only is based on local information, i.e., on one piece of input/output data, $\{\mathbf{p}_a(t_k), \mathbf{z}_D\}$, rather than on global knowledge. Therefore, a smaller value of $E(\mathbf{w})$ may actually imply a worse global performance, as is the case for the MATLAB-trained action network in Fig. 54. On the other hand, an algorithm that decreases $E(\mathbf{w})$ while preserving initialization knowledge as much as possible can improve performance locally and globally over time by exploring new regions of the input space. The final simulations in Section 5.2 demonstrate that this is, indeed, achieved satisfactorily by the modified RPROP algorithm. Also, as a first step in the validation process, the values of the action network weights updated in Fig. 54 are compared to their initial values in Fig. 55. After 150 epochs, the initialized weights have been modified to a much greater extent by the MATLAB algorithm than by the modified RPROP. The latter algorithm achieves a performance similar to the former, but exploits mainly the neural parameters that were initialized to zero in Section 5.1.1, and only slightly perturbs the remaining ones. The MATLAB algorithm not only alters the zero

parameters by about ten orders of magnitude more than the modified algorithm, but also brings about a noticeable change in the non-zero ones. Section 5.3 shows how this result can be explained theoretically, and how the algorithm can be further modified to guarantee the preservation of *a-priori* knowledge.

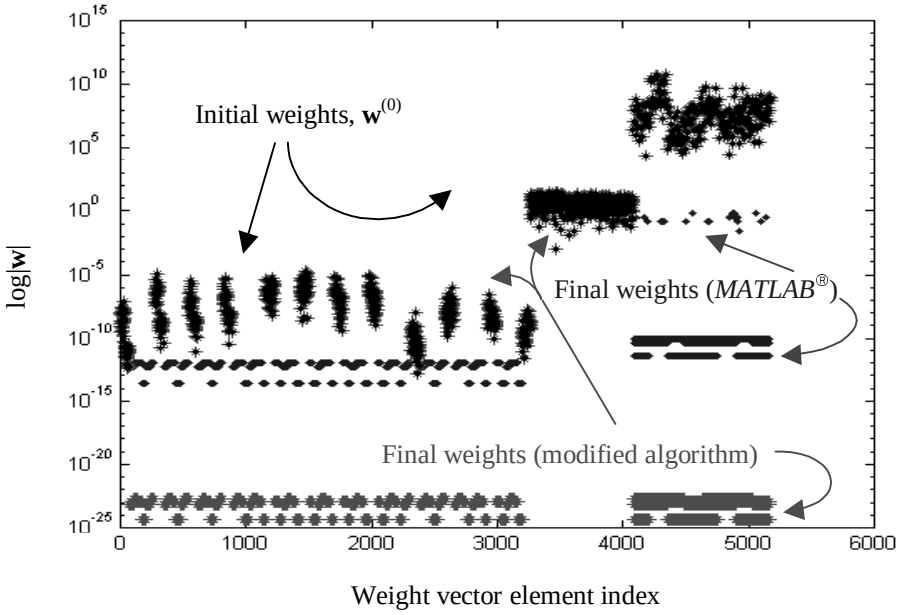

Figure 55. Comparison of the action network's weights trained with the *MATLAB*® resilient backpropagation algorithm and with its modified version. The initial weights $\mathbf{w}^{(0)}$ are selected at t_k = 0.2 sec and trained for 150 epochs, producing the final weights.

A key design attribute, that also is motivated by the above considerations about local vs. global performance, consists of terminating the action and critic network on-line training before reaching convergence. Convergence often is loosely identified with a flattening of the network-error curve during training (see also Fig. 54), but, in actuality, it cannot be so easily recognized. In this implementation, for every input/output pair $\{\mathbf{p}_a(t_k), \mathbf{z}_D\}$, the network update terminates after the mean-squared network error has decreased by 10% and at least three epochs have elapsed, allowing the increment-size adaptation to take place. When more than a few epochs are needed to decrease the error

by this amount, it means that eq. 91 is not producing a satisfactory initial increment. Thus, the terminating value of Δ_ℓ is saved and used as $\Delta_\ell^{(0)}$ for the next time interval's update (for all ℓ). This stopping rule also guarantees that the terminating weights, $\mathbf{w}(t_{k+1})$, to be used during $\Delta t = t_{k+2} - t_{k+1}$, will deliver better local performance (E) than the previous weights, $\mathbf{w}(t_k)$. Several epochs may be required to update the action and the critic during the first one or two time intervals or when significant changes in the overall controller's performance occur, that is, when the increment size needs substantial improvement. Otherwise, three epochs typically are sufficient to decrease the network error by a significant amount.

Using this *early-termination* rule not only is computationally less expensive than waiting for local convergence, but it also eliminates problems such as divergence and overfitting. Employing the sign instead of the value of the derivatives also brings about considerable computational savings and efficiency, as anticipated in Section 3.2. This partly is true because of the savings associated with storing and computing the quantity $\text{sgn}[\partial E(\mathbf{w})/\partial w_\ell]$ instead of the value $\partial E(\mathbf{w})/\partial w_\ell$, for all ℓ. Only the action and critic input and output weights are updated on line, i.e.,

$$\mathbf{w} \equiv \left[\text{Vec}(\mathbf{W_x})^T \,|\, \text{Vec}(\mathbf{V})^T \right]^T = \{w_\ell\}_{\ell=1,\,2,\,\ldots} \tag{214}$$

anticipating the results of Section 5.3. The input and output biases, **d** and **b**, and $\mathbf{W_a}$ are kept constant. Then, the performance derivatives needed in RPROP's eq. 89 and 90 can be defined as,

$$\frac{\partial E}{\partial \mathbf{w}} \equiv \left[\text{Vec}\left(\frac{\partial E}{\partial \mathbf{W}}\right)^T \;\middle|\; \text{Vec}\left(\frac{\partial E}{\partial \mathbf{V}}\right)^T \right]^T = \left\{\frac{\partial E}{\partial w_\ell}\right\}_{\ell=1,\;\;2,\;\;\ldots} \tag{215}$$

and, the remaining quantities were previously introduced. For the "signum" and "Vec" operators the following holds:

$$\operatorname{sgn}\left(\frac{\partial E}{\partial \mathbf{w}}\right) = \left[\operatorname{Vec}\left[\operatorname{sgn}\left(\frac{\partial E}{\partial \mathbf{W}}\right)\right]^T \;\middle|\; \operatorname{Vec}\left[\operatorname{sgn}\left(\frac{\partial E}{\partial \mathbf{V}}\right)\right]^T \right]^T \tag{216}$$

Thanks to the sigmoidal function's property $\operatorname{sgn}[\sigma(n)] = \operatorname{sgn}(n)$ (Section 3.1), the vector of gradient signs in eq. 217 can be obtained from the signs of the input weights,

$$\operatorname{sgn}\left(\frac{\partial E}{\partial \mathbf{W}}\right) = \operatorname{sgn}\left(-\mathbf{V}^T \mathbf{e}_{\mathbf{NN}}\right)\operatorname{sgn}\left(\mathbf{x}_a^T\right) \tag{217}$$

and from the signs of the output weights,

$$\operatorname{sgn}\left(\frac{\partial E}{\partial \mathbf{V}}\right) = \operatorname{sgn}\left(-\mathbf{e}_{\mathbf{NN}}\right)\operatorname{sgn}\left(\mathbf{n}^T\right) \tag{218}$$

where **n** is defined as in eq. 39. Therefore, sigmoidal-function evaluations only are required to compute the network output, **z**, for use in the action/critic implementation and in eq. 213. Finally, an example of modified-RPROP algorithm similar to the one employed for all network updates is shown in Appendix B.

5.2 Adaptive Flight Control Results

The adaptive PI neural network controller (Fig. 50) is implemented for the control of a full, six-degree-of-freedom simulation of a business-type twin-jet aircraft. The controlled aircraft follows a sequence of set points (i.e., a trajectory) specified by the command input, as by intent of the pilot or of a guidance law. PI dynamic compensation improves performance in the presence of constant or slowly-varying disturbances and parameter variations [81]. The simulation is allowed to explore any region of the aircraft operational domain (*OR*), defined as the envelope for which trim control settings exist,

and to temporarily leave this domain, for example, due to emergency situations and abrupt maneuvers. On-line adaptation by a DHP architecture (described in Section 5.1) learns new simulated aircraft dynamics, based on full-state feedback. A metric that expresses stability and control characteristics as well as handling qualities that are safe and pleasant to the pilot (Section 4.1.2) is optimized during both off-line and on-line learning. The resulting control system is as conservative as the scheduled linear designs incorporated during the pre-training phase, and as effective as the global nonlinear controller.

The action and critic network parameters were initialized based only on linear control knowledge obtained for a set of thirty-four operating points (*OP*). Any remaining information is incorporated entirely on line. In fact, the discrete model (eq. 209) used in the DHP architecture and the forward neural network, $\mathbf{NN}_F$, obtained from the aircraft simulation (eq. 1), only are utilized by the on-line adaptation. Therefore, in principle, they also could be updated on line, based on the actual aircraft state. In this application, they always are held fixed; whereas, the aircraft simulation that represents the real plant is modified in Section 5.2.2 and 5.2.3, to reproduce unexpected control failures and parameter variations. In the following section, the controller learns the simulation's nonlinear and coupling effects that were missed by the linearized longitudinal and lateral-directional models incorporated *a priori*. In every instance, the simulated equations of motion with feedback control provided by the nonlinear neural controller are integrated using a 4^{th}-order Runga-Kutta (RK) routine [90] with a constant time step of 0.1 sec. The adaptation time interval, Δt, also is chosen equal to 0.1 sec.

5.2.1 Full-Envelope Maneuvers

The histories of the state elements directly commanded by $\mathbf{y}_c$ are used to asses the adaptive controller's performance, similarly to Sections 4.1.3 and 4.3. During every interval Δt, the adaptation also is locally evaluated by monitoring the optimality condition (eq. 187) as well as the progress of the mean-squared network error ($\mathbf{e}_{\mathbf{NN}}$) of both the action and the critic. The optimality condition is solved numerically for the action network target, $\tilde{\mathbf{u}}_D(t_k)$, as explained in Section 5.1.2; because the terms in eq. 187 are approximate, the MATLAB function fsolve usually terminates before finding its exact solution, returning a final objective-function value that is greater than zero. As the DHP adaptation converges to the optimal policy over time, the quantities in eq. 187 also converge to their optimal value and the objective-function value decreases considerably. This scalar objective constitutes a convenient means for monitoring the temporal behavior of the solution to eq. 187. In addition, the network error is a direct indication of how well the network is doing with respect to the desired target ($\mathbf{z}_D$). Since the network input ($\mathbf{p}_a$) represents the deviation from the desired trajectory (commanded by $\mathbf{y}_c$), the network output also is expected to decrease in time, at least until a new command input is provided. Thus, a network error that consistently decreases in time over a newly explored region of the input space suggests that the adaptation is taking place efficiently. Finally, in order to obtain a fair evaluation of the effects of on-line learning, the state response is judged against that of the initialized PI controller tested in Section 4.3, whose parameters are held fixed.

Case 1: Adaptive Control During a Coupled Maneuver

The aircraft response is considered during a large-angle asymmetric maneuver, for which coupling effects between the longitudinal and lateral-directional dynamics are significant. Initially, the aircraft is flying steady and level, at a nominal airspeed V_0 of 95 m/s and an altitude H_0 of 2, 000 m, i.e., at an interpolation point where $(V_0, H_0) \not\subset OP$. At time $t = 0$, a step command consisting of 5-deg climb angle and 30-deg roll angle is initiated, as would be required to perform a climbing steady turn. The response of the aircraft subject to the on-line adaptive controller is plotted with a solid line in Fig. 56, together with that of the aircraft subject to the initialized PI controller (represented by a dashed line).

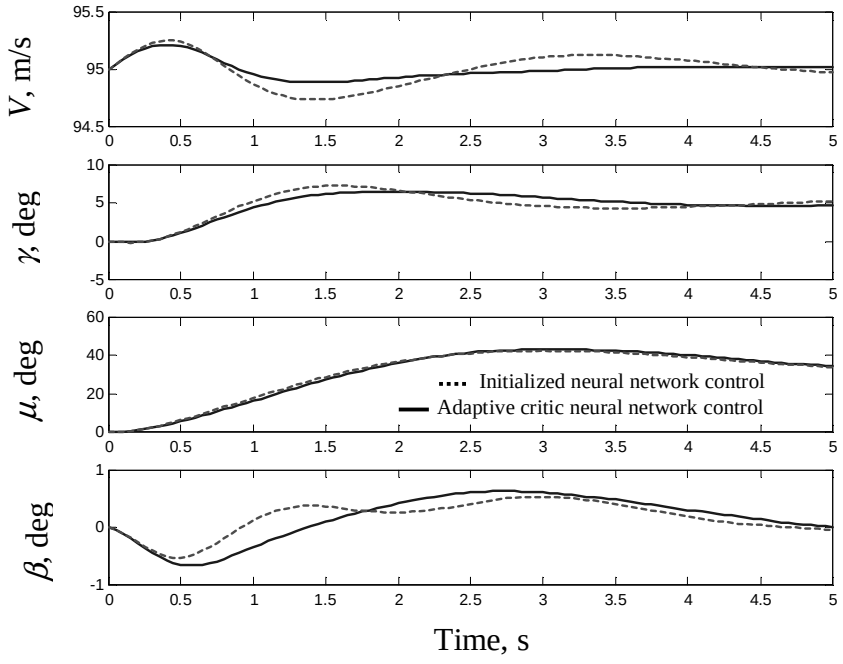

Figure 56. Comparison between the on-line trained adaptive critic neural network controller and the initialized neural network controller subject to 5-deg climb angle and 30-deg roll angle step command, at (V_0, H_0) = (95 m/s, 2 Km).

This comparison shows that the on-line adaptation taking place every 0.1 sec brings about an improvement with respect to the scheduled linear controllers. The ideal characteristics of the aircraft response to a step-command input are defined in Section 4.1.2. Following this maneuver, additional tests show that the resulting controller not only is characterized by improved performance over this region of the state space but also has preserved prior knowledge over the remaining input space (*IR*). When the final parameters are used for simulations that originate from different operating points in *IR*, the initial performance of the adaptive controller is equivalent to that of the pre-trained controller. Also, over *OP*, the gradients of the networks are found to be very close to the linear gains obtained during the pre-training phase. For example, when the final gradients of $\mathbf{NN}_A$ are compared to $\mathbf{C}_B^{\kappa}$ and $\mathbf{C}_I^{\kappa}$, for all $\kappa \in OP$, following the maneuver in Fig. 56, the total mean-squared error is found to be only 1.81×10^{-4}.

The adaptive control's time history is compared to that of the initialized controller in Fig. 57, showing a minor difference in control usage. The weights of the action and critic network undergo a small change, similarly to Fig. 55, except this change takes place over time rather than after many training epochs based on a single piece of input/output information. Figures 58 and 59 show how the on-line training algorithm modifies the network error at different stages in the adaptation. Initially (Fig. 58), eq. 91 is used to determine the increment size for the network weights; a flat curve of this kind usually indicates that the majority of the weight increments are too small and, thus, are being adapted (as explained in Sections 3.2 and 5.1.3). At later times (e.g., Fig. 59), a few epochs are sufficient to decrease the network error in the direction of a nearby minima, thanks to the modified RPROP algorithm described in Section 5.1.3.

The mean-squared network error constitutes a scalar representation of the progress made during training. However, extensive numerical simulations have demonstrated that a smooth, monotonically decreasing training curve is obtained for mse($\mathbf{e}_{NN}$) only when appropriate increment sizes (Δ_ℓ) are found, and both eq. 89 and 90 are converging.

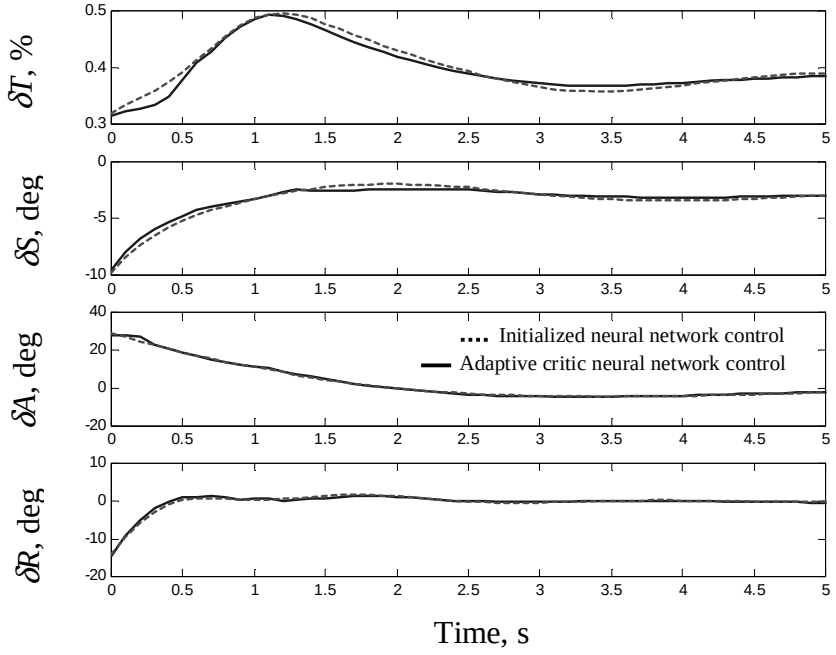

Figure 57. Comparison between the on-line trained adaptive critic neural network control history and the initialized neural network control history subject to 5-deg climb angle and 30-deg roll angle step command (Fig. 56), at (V_0, H_0) = (95 m/s, 2 Km).

The behaviors of the network error and of the optimality condition over time also show that the adaptation is drawing closer to the optimal policy. The action and critic network errors decrease similarly to Fig. 59 during every time interval $\Delta t = t_{k+1} - t_k$, with $t_k > 0.1$ sec. An overall-decreasing objective-function value (eq. 187) is shown in Table 3 for representative moments in time, at the on-set of Δt, i.e., at t_k.

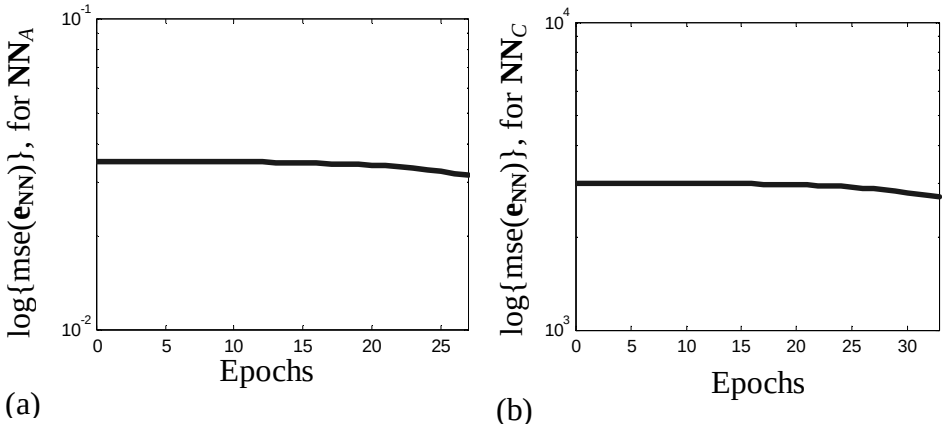

Figure 58. Mean-squared network error for the action (a) and the critic (b) versus the number of on-line training epochs, for the coupled maneuver in Fig. 56-57, at t_k = 0 sec.

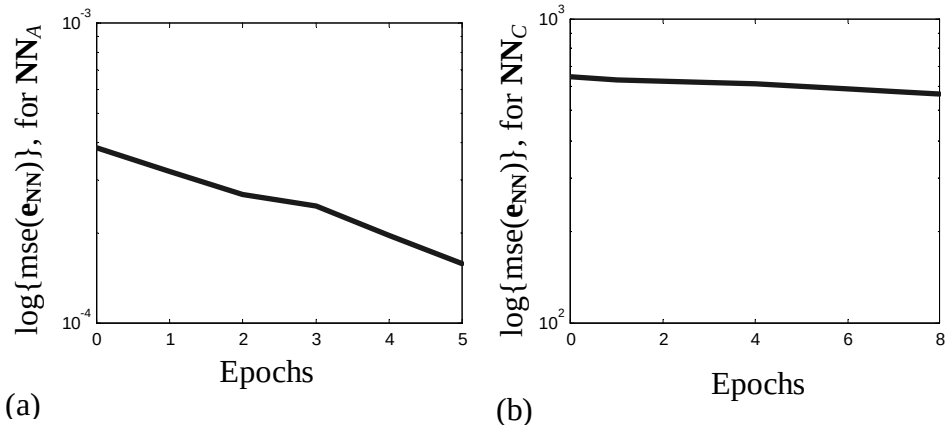

Figure 59. Mean-squared network error for the action (a) and the critic (b) versus the number of on-line training epochs, for the coupled maneuver in Fig. 56-57, at t_k = 0.4 sec.

t_k(sec)	0	0.1	0.5	1.0	1.5	2.0	3.0	4.0	4.5	5.0
$\|\partial V/\partial \tilde{\mathbf{u}}\|$	$2\ 10^4$	$7\ 10^3$	$2\ 10^3$	492	279	19	48	$6\ 10^3$	721	660

Table 3. Temporal behavior of the optimality condition at sample time intervals, for the coupled maneuver in Fig. 56-57.

The DHP on-line phase takes advantage of prior information and improves upon it, without compromising it for later use. This is easily verified by using the final weights from the present adaptation, $\mathbf{w}(t_f)$, as initial weights, $\mathbf{w}(t_0)$, for any of the maneuvers presented in the following sections. In this case, the results -- to be introduced shortly -- remain virtually the same. On the other hand, if the maneuver performed were the same, one could expect a further improvement in the performance the second time around, as will be demonstrated in Section 5.2.2. Hence, the optimization truly is global.

Case 2: Adaptive Control During a Large-Angle Maneuver

The adaptive controller is implemented on a large-angle maneuver for which the nonlinear coupling effects are so significant as to, otherwise, lead to closed-loop instability. To demonstrate this capability, a –70-deg turn is commanded, while the aircraft is flying steady and level at the nominal airspeed and altitude of 160 m/s and 7, 000 m, respectively. Typically, the maximum steady bank angle for a general aviation or transport aircraft does not exceed 60 deg. Beyond such an angle, the aircraft cannot produce sufficient lift to maintain altitude, and the coupled dynamics become so significant as to compromise any decoupled control design; also, it becomes more difficult to coordinate the turn. While a pilot normally would refrain from performing such a maneuver, abiding to safety regulations, it is conceivable for these conditions to come about in an emergency situation (or in aerobatic flight). In fact, very-large bank-angle turns near the ground contributed to many fatal accidents.

In this case, the initialized controller, represented by a dashed line in Fig. 60, causes the aircraft to become unstable and, possibly, to enter a spin. The roll and climb angles increase beyond acceptable limits; after about 4 sec, the angle-of-attack time history

shows that the aircraft enters a stall. At this point, the simulation can no longer be considered a faithful representation of the aircraft dynamics, as post-stall aerodynamic effects have not been modeled. Still, the uncoupled control design causes the aircraft to gyrate wildly, and it is not capable of recovering from this maneuver. Figure 60 also shows the response of the adaptive controller (in a solid line), for the same flight conditions. In this case, the control system learns the relevant nonlinear neural network weights on line, preventing loss of stability.

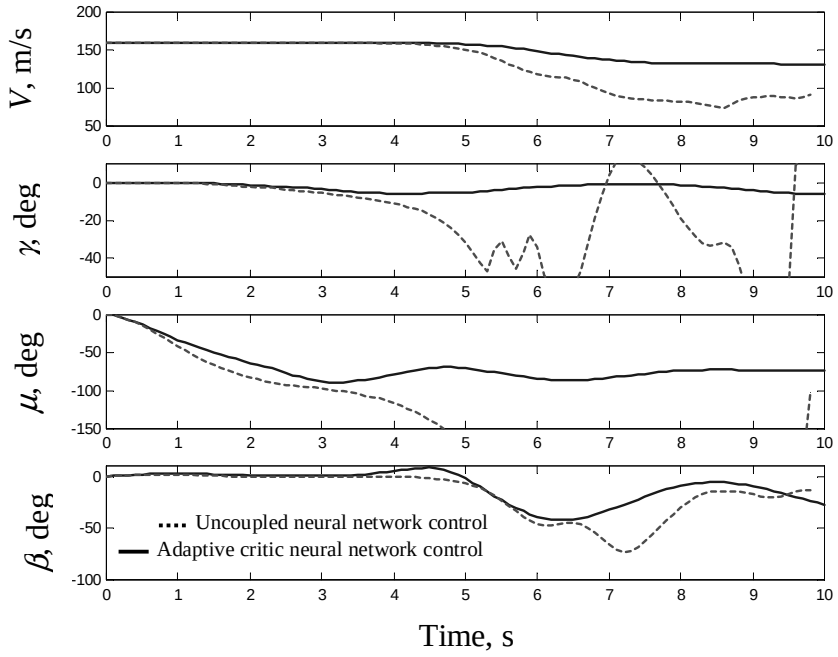

Figure 60. Comparison between the on-line trained adaptive critic neural network controller and the initialized neural network controller subject to −70-deg roll angle step command, at (V_0, H_0) = (160 m/s, 7 Km).

Under challenging circumstances, the tendency is for the system to demand unreasonable control usage. While control bounds cannot always be easily incorporated in the control design, they can be accounted for in the adaptive critic architecture simply by modifying the weighting matrices in eq. 183-185. In this implementation, the control-

weighting matrix $\mathbf{R}_a$ in eq. 187 is modified such that the search for the action target, $\tilde{\mathbf{u}}_D(t_k)$, (Fig. 51) is constrained by physically meaningful values of the (total) controls. The original matrix, eq. 185, can be used everywhere else in the implementation (Appendix E) without loss of generality, thereby preventing numerical blow ups.

The soft control constraints used in the large-angle simulation of Fig. 60, and in the simulations hereafter, are plotted in Fig. 61 for the throttle input and stabilator deflection. When the guess $\tilde{\mathbf{u}}_D(t_k)^G$ leads to a throttle input, δT^G, that is greater than the upper bound, $\delta T_{\max}$, or that is smaller than the lower bound, $\delta T_{\min}$, the throttle-weighting element in $\mathbf{R}_a$ takes the exponential form,

$$\mathbf{R}_a(1,\ 1) = e^{10\left|2\delta T^G - 1\right|} \tag{219}$$

where, $\delta T_{\max}$ = 1 %, $\delta T_{\min}$ = 0 %, and $\mathbf{R}_a(i, j)$ represents the element in the i^{th}-row and the j^{th}-column of the matrix $\mathbf{R}_a$. Similarly, when the target guess leads to a stabilator deflection, $\delta \mathcal{S}^G$, that is greater than its physical upper bound, $\delta \mathcal{S}_{\max}$, or smaller than its lower bound, $\delta \mathcal{S}_{\min}$, then the stabilator-weighting element in $\mathbf{R}_a$ is given by,

$$\mathbf{R}_a(2,\ 2) = e^{15\left|\delta \mathcal{S}^G\right|} \tag{220}$$

where, $\delta \mathcal{S}_{\max}$ = 0.6 rad and $\delta \mathcal{S}_{\min}$ = –0.6 rad. A relation of the form in eq. 220 also is used for the elements weighting the aileron and rudder deflections, $\mathbf{R}_a(3, 3)$ and $\mathbf{R}_a(4, 4)$,when the following bounds are exceeded: –0.6 rad < δA < 0.6 rad, and –0.6 rad < δR < 0.6 rad. The simulation representing the actual aircraft (eq. 1), prevents the controls from exceeding their physical limitations, regardless of the control law's outcome.

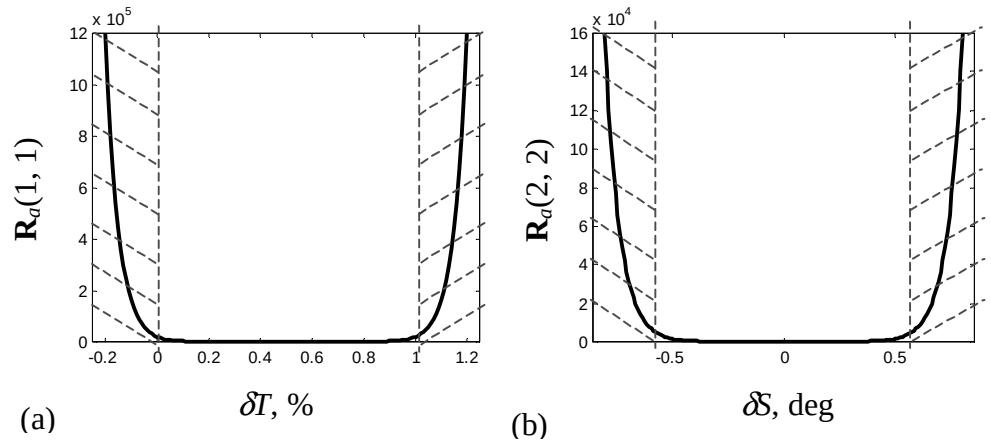

Figure 61. Exponential weighting on the throttle (a) and on the stabilator (b) controls, producing the bounds represented by the dashed bars. The weighting function in (b) also is used for the aileron and rudder controls.

The time histories of the initialized and adaptive controllers that produce the aircraft response in Fig. 60 are plotted in Fig. 62. These results show how the DHP architecture modifies the control law with respect to the one designed off line, considerably improving performance in time. The adaptive controller is capable of learning the control bounds on line, meaning that it learns how to optimize the control law for the actual plant, subject to soft constraints that are known but not accounted for *a priori*. This can be seen as another level of integration of *a-priori* and *a-posteriori* knowledge. Although its control inputs also are bounded by the simulation, to represent the physical limitations of the control surfaces, the initialized controller can not become aware of these constraints, nor can it cope with them in real time. The result is that while the adaptive control can sustain the desired banked turn, the uncoupled classical design leads to hazardous maneuver, as illustrated by the trajectories in Fig. 63.

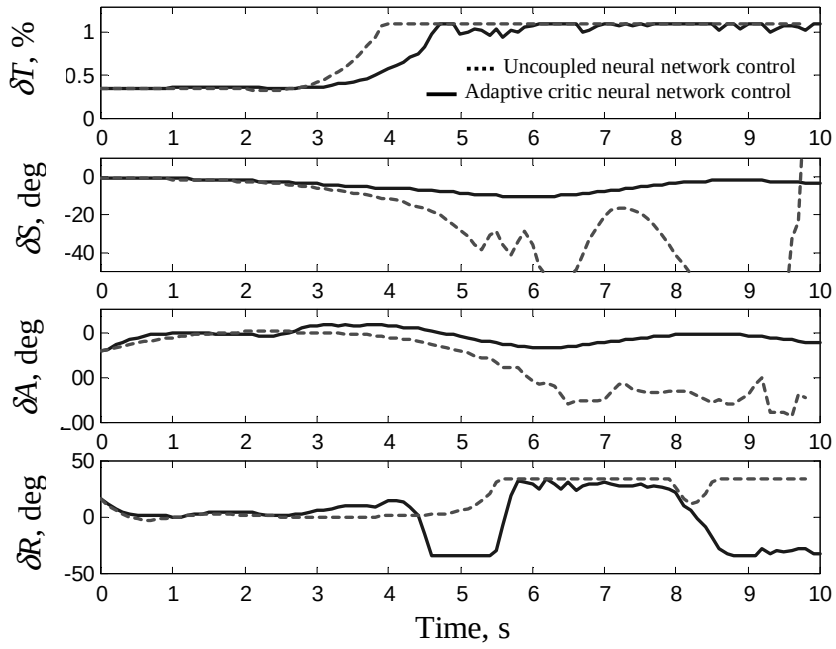

Figure 62. Comparison between the adaptive critic neural network control history and the initialized neural network control history subject to –70-deg roll angle step command, at (V_0, H_0) = (160 m/s, 7 Km).

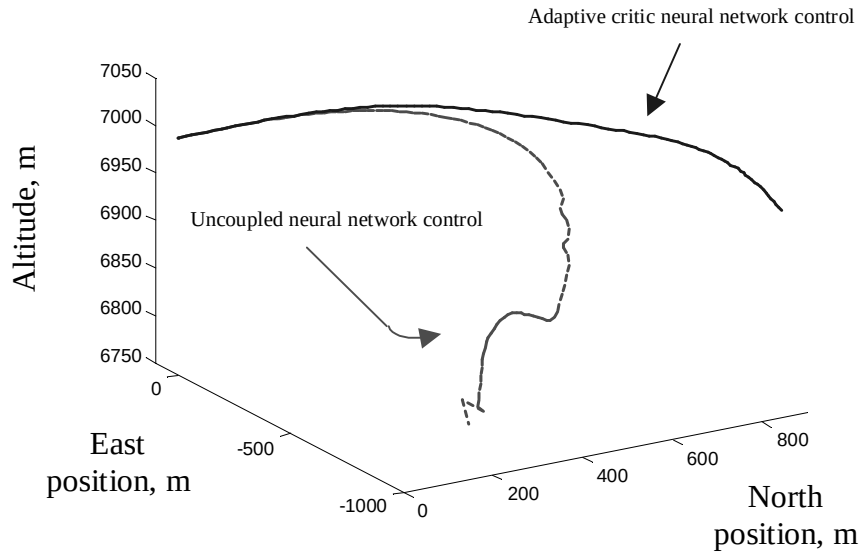

Figure 63. Comparison of the trajectories obtained with the on-line trained adaptive critic neural network controller and with the initialized neural network controller subject to a –70-deg roll angle step command, at (V_0, H_0) = (160 m/s, 7 Km).

5.2.2 Control System Failure

The adaptive controller's response to control failure, possibly due to physical damage or actuator malfunctioning, is evaluated using the tools and metrics introduced in Section 5.2.1. The failures are simulated by modifying the aircraft equations of motion (eq. 1). They are not included in the discrete model (eq. 209), as they are assumed to be unforeseen conditions. The DHP architecture proves capable of accounting for them on line, based on the observed aircraft dynamics ($\mathbf{x}_a$).

Case 3: Adaptive Control During Multiple Control Failures

The capability of the adaptive control system to handle a near-emergency situation is considered by simulating control failures during an approach for landing. The aircraft, initially flying steady level at (V_0, H_0) = (100 m/s, 3, 000 m), begins its final approach by decreasing its velocity and performing a descending turn, following the step command input $\mathbf{y}_c$ = [90 (m/s) –6 (deg) 50 (deg) 0 (deg)]^T for ten seconds. During this time, multiple control failures occur, impairing control of the aircraft. The rudder and stabilator are temporarily stuck at 0 deg, for 5 sec $\leq t_k \leq$ 10 sec; during 0 sec $\leq t_k \leq$ 5 sec the rudder also is stuck at -34 deg, and both engines produce no thrust (δT = 0 %), at all times. As a consequence, by t_k = 10 sec, the airplane has entered a steep dive with a large roll angle and fast accelerations. The state response and the control history during these first ten seconds, as produced by the initialized controller, are shown in Figs. 64 and 65, respectively. This critical situation is simulated in order to compare the adaptive and the initialized control systems during a recovery maneuver with reduced but sufficient control authority.

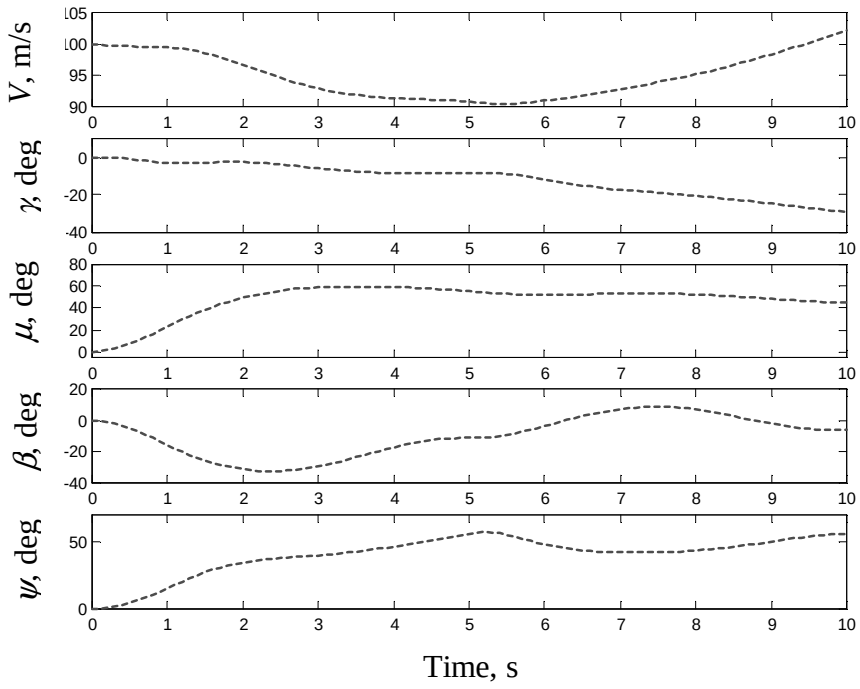

Figure 64. Uncoupled neural network controller response in the presence of failed control inputs, with $\mathbf{y}_c$ = [90 (m/s) –6 (deg) 50 (deg) 0 (deg)]^T and (V_0, H_0) = (100 m/s, 3 Km)

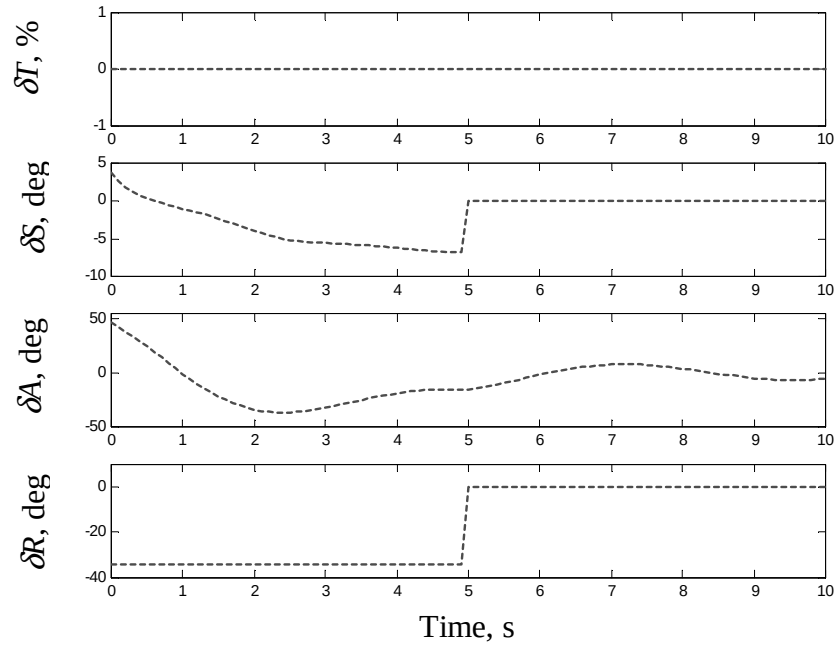

Figure 65. Uncoupled neural network control history in the presence of failed control inputs, with $\mathbf{y}_c$ = [90 (m/s) –6 (deg) 50 (deg) 0 (deg)]^T and (V_0, H_0) = (100 m/s, 3 Km).

It is presumed that 10 sec after the initial failure, the pilot or guidance logic on board the aircraft has become aware of the critical situation and has aborted landing, initiating a wings-level climb to avoid obstacles on the ground. First, the wings are brought back to level by commanding a 0-deg roll angle for 2 sec. Then, an airspeed of 95 m/s and a 5-deg path angle are commanded for climbing. In the meantime, the stabilator has become fully operational, and the available throttle is increased to 50% (as by restoring full thrust to a single engine); the rudder is stuck at – 15 deg. The response of the adaptive controller is compared to that of the initialized controller, resetting the integrator state to zero in both cases, to avoid the phenomenon known as *integrator wind-up* [93]. The error signal is said to wind-up the integrator when the integral component of the control signal has saturated and does not drop from its maximum value, even though the desired output has reached the set point and the error has changed sign.

Figure 66 shows that the adaptation improves the command-input response, at times by more than 30 %, even though these conditions are being experienced for the first time (i.e., the adaptive critic is exploring a new region of the multidimensional flight envelope *OR*). All relevant state histories, including total airspeed, are improved upon by the adaptive critic architecture. The velocity and path angle are followed more closely, despite a lesser throttle usage, because the adaptive-controlled aircraft experiences smaller angles of attack and sideslip and, hence, less drag. The adaptation also diminishes the amplitude of the roll and heading-angle oscillations. The adaptive and initialized-control time histories are shown in Fig. 67. Due to the limited (50 %) available thrust, the throttle-input profile is significantly modified and its usage is more evenly distributed over the time interval. With the rudder stuck at – 15 deg, the lateral-

directional response is improved by adapting the aileron control input. The next case shows not only that the latter adaptation has preserved prior global control knowledge, but also that revisiting this maneuver for a second time further improves local performance.

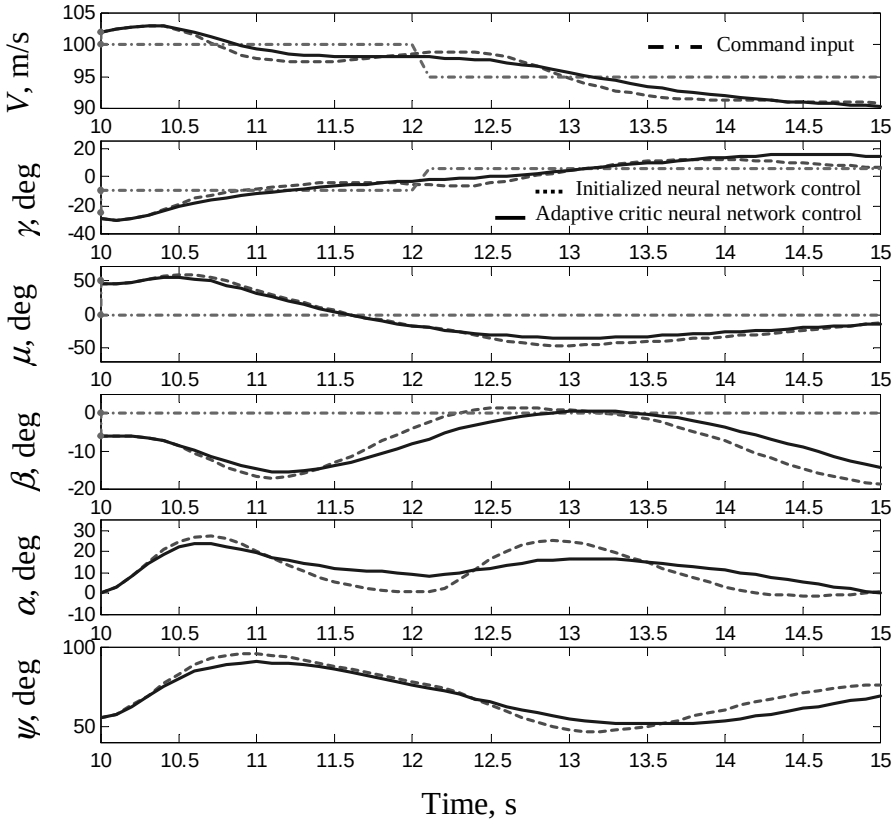

Figure 66. Comparison between the adaptive and the initialized neural controllers in the presence of multiple control failures (Fig. 67).

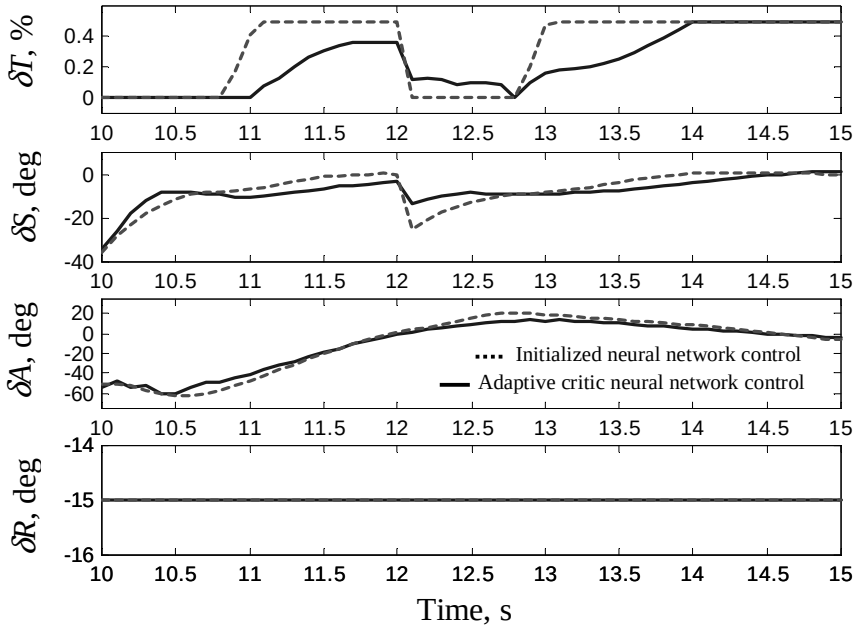

Figure 67. Adaptive and initialized neural control histories with 50 %-available thrust and the rudder stuck at –15 deg.

Case 4: Adaptive Control During a Previously-Encountered Maneuver

In the previous case, the adaptive control system was confronted with an unexplored region of the state space (*OR*) and with novel dynamics characterized both by nonlinear effects and by unexpected control failures. Given the amount of information to be assimilated on line, it is reasonable to expect that this first adaptation left room for further improvement. Therefore, the action and critic neural weights obtained at the end of the previous time period, i.e., at t_k = 15 sec, are used as initial weights for a second adaptation that is performed under the same conditions as the previous one. This scenario can be considered equivalent to a case in which the aircraft encounters the same situation (Case 3) for a second time, with no significant adaptation in between.

The state response obtained during the second adaptation (solid line) is compared to that obtained during the first adaptation (now represented by a dashed line) in Fig. 68.

The fact that initially these two responses are virtually identical signifies that, while improving performance over time, the first adaptation had not degraded prior knowledge elsewhere (in this case, over the region of the state space visited during the first few time intervals). Hence, the second adaptation inherits what was learned both through the pre-training phase and during the first on-line adaptation; moreover, it improves performance with respect to the latter and, thus, also with respect to the former. In particular, Fig. 68 shows that the updated controls (plotted in Fig. 69) considerably reduce the amplitude of the angle-of-attack and sideslip oscillations, eventually preventing stall. All this is achieved without compromising the velocity and path-angle response, implying that the control system is being reconfigured over time to deal with the failed control inputs.

Testing the adaptation for the recovery maneuver (that is, after $t_k \geq 10$ sec) demonstrates that persistence of excitation [92] is not required to learn about the failed controls. The adaptive controller copes with the failures as soon as it encounters a stimulus, without having to prepare for it ahead of time through another challenging maneuver. Therefore, if the failures were to occur during steady level flight when little or no learning takes place, the control system would still be able to account for them effectively once the need arose. Monitoring the optimality condition and network errors over time confirms that the adaptive elements are converging to a nearly optimal, global control policy. The optimality condition and the network errors during the second adaptation are smaller than during the first one. This behavior always is observed during simulations of the adaptive DHP controller, provided that the neural network input $\mathbf{p}_a$ is bounded.

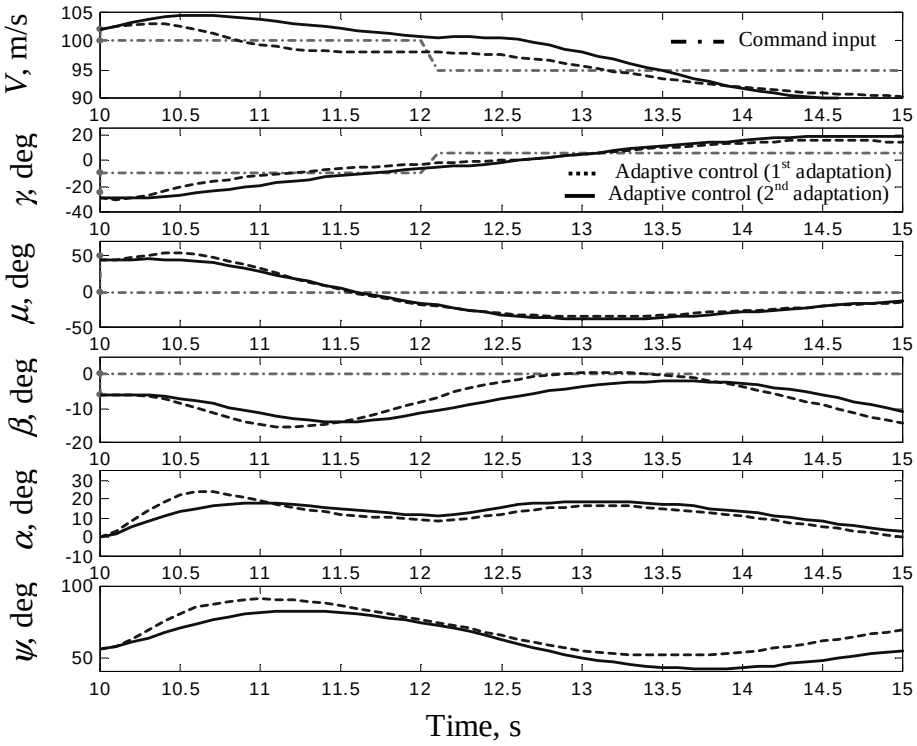

Figure 68. Adaptive controller response to a maneuver experienced for the first and second time, in the presence of multiple control failures.

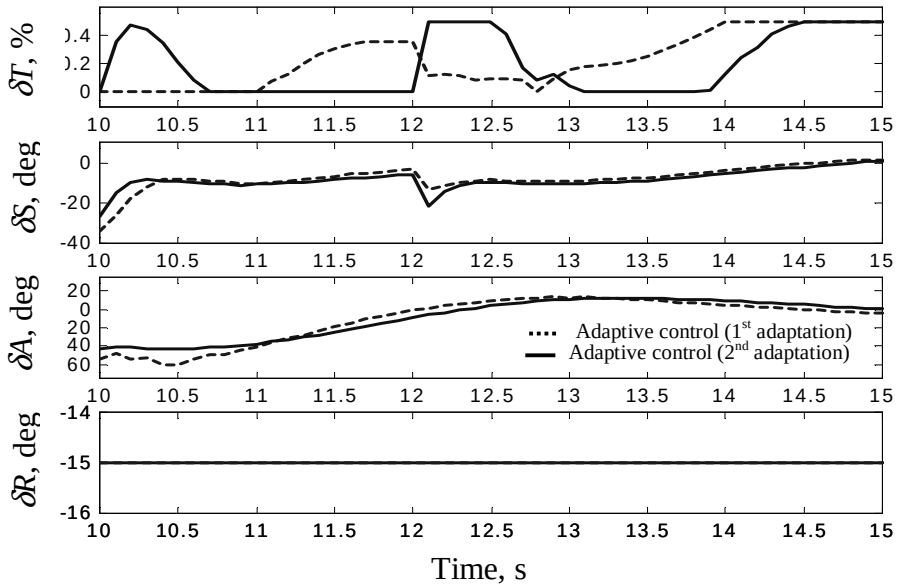

Figure 69. Adaptive control history for a maneuver experienced for the first and second time, in the presence of multiple control failures..

5.2.3 Parameter Variations

Case 5: Adaptive Control in the Presence of Parameter Variations

The adaptive controller is tested for a case in which the parameters of the simulated aircraft have changed with respect to the original model (eq. 1) that was used to design the initialized controller (Chapter 4). All control effectors are assumed to be unfailed. The pitch-rate and angle-of-attack-rate effects are decreased by 50 %, the static and directional stability coefficients are reduced by 20 % and 30 %, respectively. With the original aircraft parameters, the response of the initialized controller subject to a small-angle command input of 2 deg path angle, 5 deg roll, and 3 deg sideslip can be considered to be optimal at a design point, e.g., (V_0, H_0) = (200 m/s, 11, 000 m). Due to these modified aerodynamic effects, actual dynamic characteristics such as Dutch roll and natural frequency differ from those accounted for by the linear design (Section 4.1). Therefore, the performance of the initialized controller is degraded with respect to its original baseline, as shown in Figs. 70-71.

Although the DHP architecture employs a model that is based on the original aircraft parameters, it can learn about the new dynamics through its knowledge of the actual state. In this case, a first adaptation does not bring about a considerable improvement. But Fig. 72 shows that if the simulated aircraft with modified parameters undergoes the same maneuver a second time, the adaptation considerably improves performance and provides a response that is nearly optimal (Fig. 70). Control usage also is reduced with respect to the initialized controller, as shown in Fig. 73. The DHP implementation is found to be robust and capable of learning through an imperfect model.

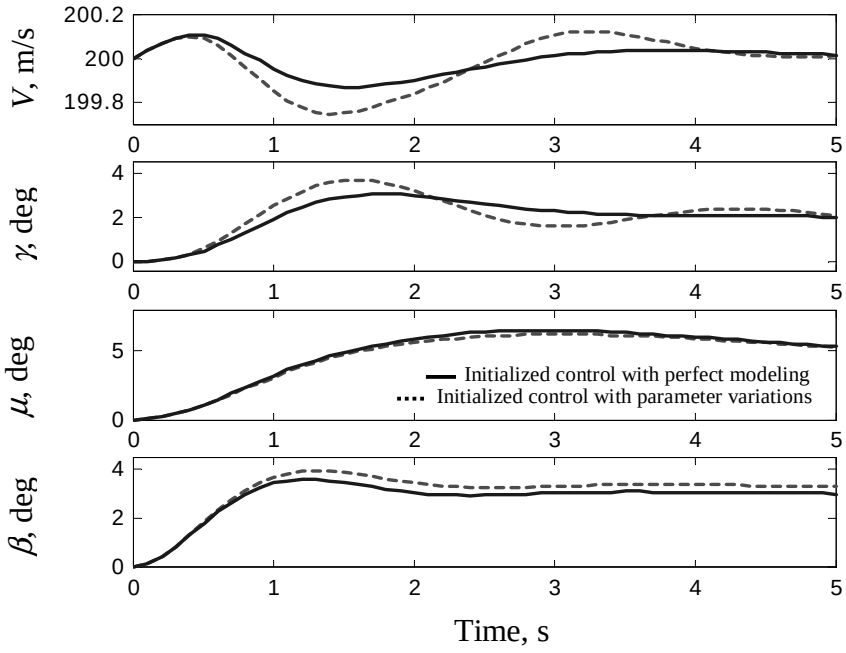

Figure 70. Initialized controller response for the perfectly-modeled aircraft and in the presence of parameter variations, with $\mathbf{y}_c$ = [200 (m/s) –2 (deg) 5 (deg) 3 (deg)]^T and at the design point (V_0, H_0) = (200 m/s, 11 Km).

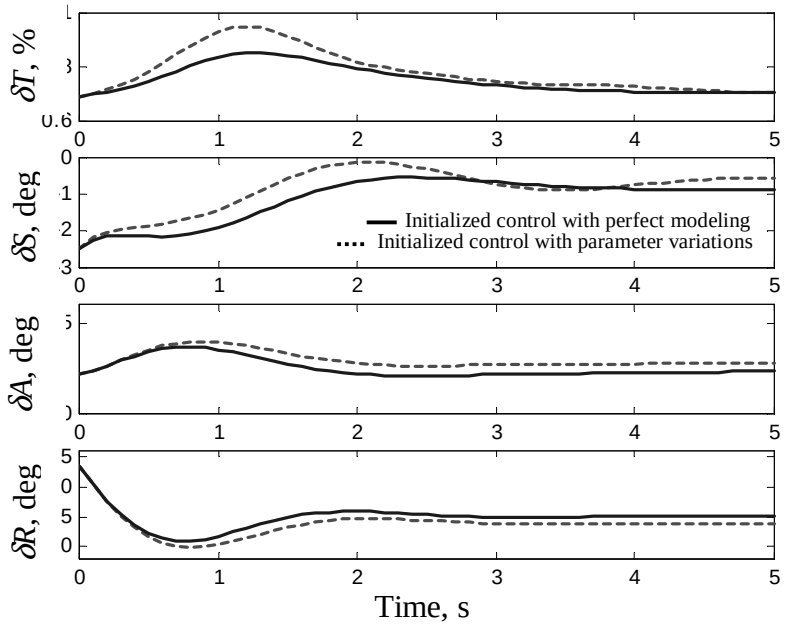

Figure 71. Initialized control history for the perfectly-modeled aircraft and in the presence of parameter variations, with $\mathbf{y}_c$ = [200 (m/s) –2 (deg) 5 (deg) 3 (deg)]^T and at the design point (V_0, H_0) = (200 m/s, 11 Km).

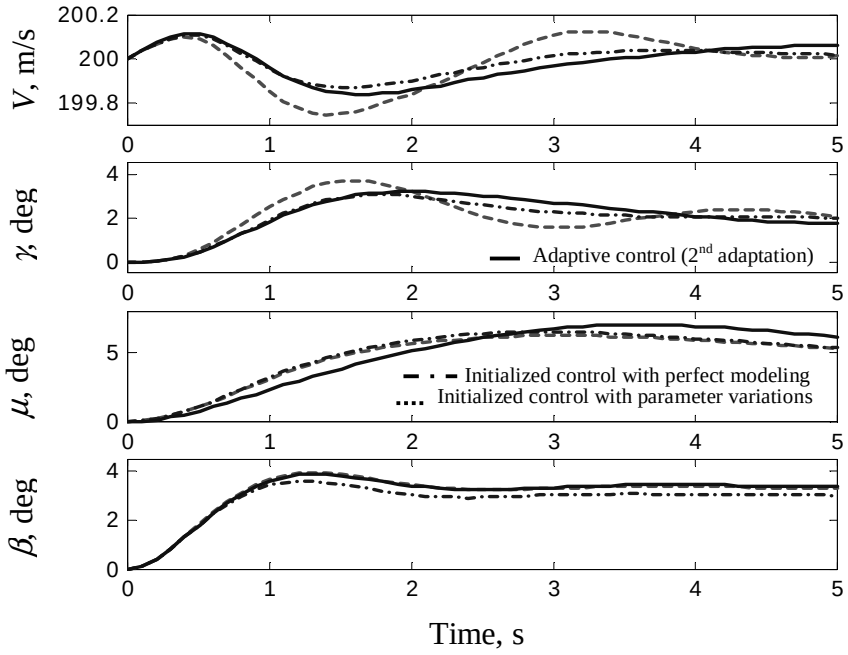

Figure 72. Comparison between the adaptive neural network controller and the initialized neural network controller in the presence of parameter variations, with $\mathbf{y}_c$ = [200 (m/s) –2 (deg) 5 (deg) 3 (deg)]^T and (V_0, H_0) = (200 m/s, 11 Km).

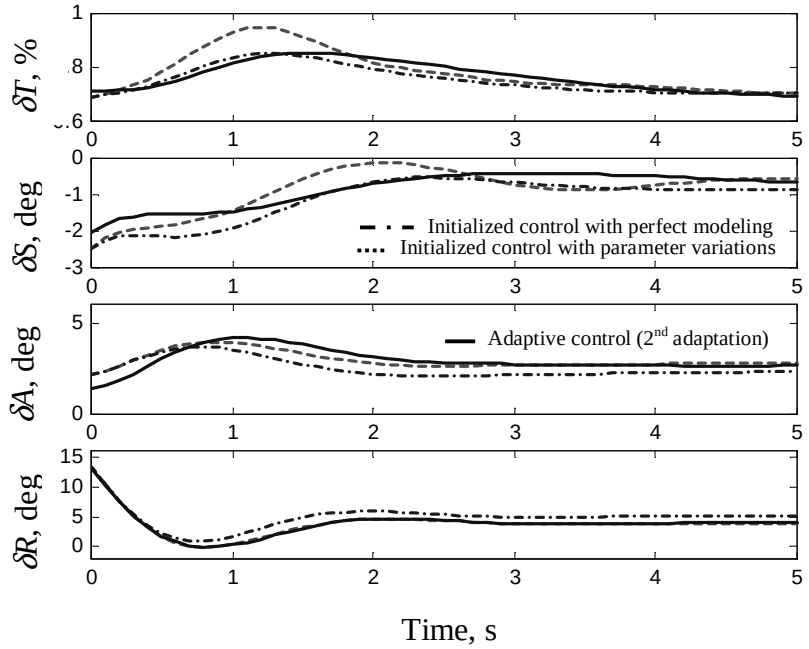

Figure 73. Control history of the adaptive neural network controller and of the initialized neural network controller in the presence of parameter variations, with $\mathbf{y}_c$ = [200 (m/s) –2 (deg) 5 (deg) 3 (deg)]^T and (V_0, H_0) = (200 m/s, 11 Km).

5.3 Algebraically Constrained Adaptive Critic Architecture

Neural network output and gradient weight equations can be used to guarantee that *a-priori* knowledge is preserved during incremental learning over time, by following a development along the lines of the one introduced in Section 3.3. A portion of the neural parameters can be used to satisfy the same set of requirements that were incorporated by initializing the neural networks off line. Meanwhile, the remaining parameters are updated incrementally through an optimization-based on-line training algorithm. Typically, the set of requirements to be incorporated off line consists of a batch training set and can be matched exactly through an algebraic training technique (Section 3.1). In on-line learning, the objective is to improve the overall neural function approximation based on local information that become available a piece at the time and, therefore, can be incorporated only incrementally. One of the main challenges of on-line learning is retaining, or remembering, previous information while assimilating new information. In particular, when global knowledge of the function being approximated is available off line, the on-line training algorithm must be able to take advantage of it before it is forgotten. This can be achieved by the action and critic networks, and prior knowledge can be maintained intact while their performance is improved incrementally over time.

A basic assumption is that the linear control knowledge obtained in Section 4.1 also holds on line; this is implied, provided the aircraft dynamics always can be approximated by the linearized models at the operating points in *OP*. When the plant to be controlled is expected to undergo considerable unforeseen changes (as due to system failure or damage), the on-line adaptation should not be constrained in this fashion; instead, it should be carried out as described in Sections 5.1 and 5.2. Nevertheless, this technique

demonstrates that the neural controller has the capability of adapting on line, while retaining an established baseline performance. In addition, it produces an algorithm that guarantees the realization of this capability, and is characterized by polynomial computational complexity. The method is referred to as *constrained* because it solves a nonlinear optimal control problem (eq. 1 and 10), subject to equality constraints on the state and the control (eq. 13 and 20).

The constrained-learning results in Section 3.3 can be extended to both the action and the critic networks initialized in Section 5.1.1; in the interest of conciseness, they are illustrated here only for the action network, $\mathbf{NN}_A$. Section 5.1.1 shows how the final action network architecture (Fig. 49) is obtained by algebraically combining the scalar networks $NN_{B_{L_1}}$, $NN_{B_{L_2}}$, $NN_{B_{LD_1}}$, $NN_{B_{LD_2}}$, $NN_{I_{L_1}}$, $NN_{I_{L_2}}$, $NN_{I_{LD_1}}$, and $NN_{I_{LD_2}}$, trained in Section 4.3. As a consequence of network combination and addition operations, the initial action input and output weight matrices (eq. 206 and 207) contain the scalar networks' pre-trained weights, as well as zero weights. Eventually, all of these parameters are modified on line. Imposing the pre-training requirements on the action network equations reveals that the action parameters obtained from the pre-trained scalar networks can be adjusted to continue to satisfy the same requirements on line, while the remaining parameters are updated to minimize the error function in eq. 212.

In other words, the action network input/output and gradient equations (eq. 190 and 191) must satisfy the feedback and command-integral requirements over *OP* (i.e., eq. 152, 154, 157, and 158) simultaneously, at every instant in time t_k. Moreover, if they do so at every training epoch, while the function in eq. 212 is being minimized with respect to the vector **w**, then the resulting optimization is constrained by these requirements at all

times. For the moment, the time and epoch indices are omitted for simplicity, implying that the following equations hold at any time and epoch. The feedback and command-integral requirements can be formulated as a unique gradient-based training set of the type introduced in Section 3.1.1, i.e., $\{[\mathbf{0}\,|\,\mathbf{a}^{\kappa T}]^T, \mathbf{0}, \mathbf{C}^{\kappa}\}_{\kappa=1,\ldots,34}$, where $\kappa \in OP$. The following matrix is known from the gains in eq. 156 and 160:

$$\mathbf{C}^{\kappa} = \begin{bmatrix} \mathbf{c}^{\kappa}_{B_{L_1}} & \mathbf{c}^{\kappa}_{B_{L_2}} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{c}^{\kappa}_{B_{LD_1}} & \mathbf{c}^{\kappa}_{B_{LD_2}} \\ \mathbf{c}^{\kappa}_{I_{L_1}} & \mathbf{c}^{\kappa}_{I_{L_1}} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{c}^{\kappa}_{I_{LD_1}} & \mathbf{c}^{\kappa}_{I_{LD_2}} \end{bmatrix} \tag{221}$$

Similarly to eq. 38, this matrix represents the Hessian of the multidimensional function to be approximated, e.g., $\mathbf{u} = \mathbf{h}(\mathbf{y})$, evaluated at selected points (*OP*), such that $\mathbf{C}^{\kappa} \equiv \partial\mathbf{u}/\partial\mathbf{y}|^{\kappa}$. It is reasonable to assume that at the equilibria *OP*, defined in Section 4.1, the longitudinal and lateral-directional dynamics are decoupled, and that eq. 153, 154, and 158 hold at all times. However, these assumption need not hold (and most likely will not hold) elsewhere in *OR*.

The above training set is matched exactly by the action network parameters when they satisfy the output weight equations,

$$\mathbf{0} = \mathbf{V}_A \boldsymbol{\sigma}\left[\mathbf{W}_{\mathbf{a}_A}\mathbf{a}^{\kappa} + \mathbf{d}_A\right] + \mathbf{b}_A, \quad \kappa = 1, \ldots, 34 \tag{222}$$

and the gradient weight equations,

$$\mathbf{C}^{\kappa} = \mathbf{W}^T_{\mathbf{x}_{aA}}\left\{\mathrm{diag}\left[\boldsymbol{\sigma}'\left(\mathbf{W}_{\mathbf{a}_A}\mathbf{a}^{\kappa} + \mathbf{d}_A\right)\right]\mathbf{V}_A{}^T\right\}, \quad \kappa = 1, \ldots, 34 \tag{223}$$

both of which are obtained from the vector-output network equation. The action network's matrix, $\mathbf{W}_A$, also can be partitioned into submatrices that bear the subscripts of the inputs they weigh, as follows:

$$\mathbf{W}_A = \left[\mathbf{W}_{\mathbf{x}_{aA}} \middle| \mathbf{W}_{\mathbf{a}_A}\right] = \left[\mathbf{W}_{\tilde{\mathbf{x}}_A} \middle| \mathbf{W}_{\xi_A} \middle| \mathbf{W}_{\mathbf{a}_A}\right] \tag{224}$$

It can be observed that if the input weights associated with the scheduling vector, $\mathbf{W}_{\mathbf{a}_A}$, and the input bias, $\mathbf{d}_A$, are held constant at all times, then the input-to-node values of the action network not only are constant but are known from the pre-training phase. In fact, the following relationship can be derived for the action network's input-to-node vector from the input-to-node variables' definition (eq. 39):

$$\mathbf{n}_A^{\kappa} \equiv \mathbf{W}_{\mathbf{a}_A}\mathbf{a}^{\kappa} + \mathbf{d}_A = \begin{bmatrix} \mathbf{W}_{\mathbf{a}_{B_{L_1}}} \\ \mathbf{W}_{\mathbf{a}_{B_{L_2}}} \\ \mathbf{W}_{\mathbf{a}_{B_{LD_1}}} \\ \mathbf{W}_{\mathbf{a}_{B_{LD_2}}} \\ \mathbf{W}_{\mathbf{a}_{I_{L_1}}} \\ \mathbf{W}_{\mathbf{a}_{I_{L_2}}} \\ \mathbf{W}_{\mathbf{a}_{I_{LD_1}}} \\ \mathbf{W}_{\mathbf{a}_{I_{LD_2}}} \end{bmatrix} \mathbf{a}^{\kappa} + \begin{bmatrix} \mathbf{d}_{B_{L_1}} \\ \mathbf{d}_{B_{L_2}} \\ \mathbf{d}_{B_{LD_1}} \\ \mathbf{d}_{B_{LD_2}} \\ \mathbf{d}_{I_{L_1}} \\ \mathbf{d}_{I_{L_2}} \\ \mathbf{d}_{I_{LD_1}} \\ \mathbf{d}_{I_{LD_2}} \end{bmatrix} = \begin{bmatrix} \mathbf{n}_{B_{L_1}}^{\kappa} \\ \mathbf{n}_{B_{L_2}}^{\kappa} \\ \mathbf{n}_{B_{LD_1}}^{\kappa} \\ \mathbf{n}_{B_{LD_2}}^{\kappa} \\ \mathbf{n}_{I_{L_1}}^{\kappa} \\ \mathbf{n}_{I_{L_2}}^{\kappa} \\ \mathbf{n}_{I_{LD_1}}^{\kappa} \\ \mathbf{n}_{I_{LD_2}}^{\kappa} \end{bmatrix} \tag{225}$$

The notation is consistent with that used in Sections 3.1.1 and 5.1.1. $\mathbf{W}_{\mathbf{a}_{B_{L_1}}}$, $\mathbf{d}_{B_{L_1}}$, and $\mathbf{n}_{B_{L_1}}^{\kappa}$ are the **a**-input weight matrix, the input bias, and the input-to-node vector of the pre-trained network $NN_{B_{L_1}}$; the remaining quantities in eq. 225 are similarly defined. Hence, the action input-to-node values can be computed and stored off line for later use.

The output weight equations, eq. 222, can be formulated as a linear system in terms of the output weights,

$$\mathbf{0} = \mathbf{S}_A\mathbf{V}_A^T + \mathbf{B}_A^T \tag{226}$$

where,

$$\mathbf{S}_A = \left[\mathbf{S}_{B_{L_1}} \;\; \mathbf{S}_{B_{L_2}} \;\; \mathbf{S}_{B_{LD_1}} \;\; \mathbf{S}_{B_{LD_2}} \;\; \mathbf{S}_{I_{L_1}} \;\; \mathbf{S}_{I_{L_2}} \;\; \mathbf{S}_{I_{LD_1}} \;\; \mathbf{S}_{I_{LD_2}}\right] \tag{227}$$

and $\mathbf{B}_A \equiv [\mathbf{b}_A \ldots \mathbf{b}_A]$ is an $m \times 34$ matrix of output biases that also are held constant. The sigmoidal matrices in this equation (defined as in eq. 47) can be computed from the pre-trained scalar networks corresponding to their subscripts, by using the appropriate input-to-node vectors in eq. 225. Thus, they are known *a priori*, and they remain constant. By rearranging eq. 226, it can be observed that the action network's output weights determined during the pre-training phase can be used to satisfy the same output requirements on the feedback and command-integral contributions, $\Delta\mathbf{u}_B$ and $\Delta\mathbf{u}_I$, for any value of the remaining output parameters. Of course now those contributions are automatically summed by $\mathbf{NN}_A$; nevertheless, they still can be identified within the output weight equations above.

With this goal in mind, the output weight matrix is partitioned as suggested by its initialized value, eq. 207,

$$\mathbf{V}_A = \begin{bmatrix} \mathbf{v}_{11}^T & \mathbf{v}_{12}^T & \mathbf{v}_{13}^T & \mathbf{v}_{14}^T & \mathbf{v}_{15}^T & \mathbf{v}_{16}^T & \mathbf{v}_{17}^T & \mathbf{v}_{18}^T \\ \mathbf{v}_{21}^T & \mathbf{v}_{22}^T & \mathbf{v}_{23}^T & \mathbf{v}_{24}^T & \mathbf{v}_{25}^T & \mathbf{v}_{26}^T & \mathbf{v}_{27}^T & \mathbf{v}_{28}^T \\ \mathbf{v}_{31}^T & \mathbf{v}_{32}^T & \mathbf{v}_{33}^T & \mathbf{v}_{34}^T & \mathbf{v}_{35}^T & \mathbf{v}_{36}^T & \mathbf{v}_{37}^T & \mathbf{v}_{38}^T \\ \mathbf{v}_{41}^T & \mathbf{v}_{42}^T & \mathbf{v}_{43}^T & \mathbf{v}_{44}^T & \mathbf{v}_{45}^T & \mathbf{v}_{46}^T & \mathbf{v}_{47}^T & \mathbf{v}_{48}^T \end{bmatrix} \tag{228}$$

and, then, substituted back into eq. 226. This results into four independent systems of equations,

$$\mathbf{0} = \begin{bmatrix} \mathbf{S}_{B_{L_1}} & \mathbf{S}_{B_{L_2}} & \mathbf{S}_{B_{LD_1}} & \mathbf{S}_{B_{LD_2}} & \mathbf{S}_{I_{L_1}} & \mathbf{S}_{I_{L_2}} & \mathbf{S}_{I_{LD_1}} & \mathbf{S}_{I_{LD_2}} \end{bmatrix} \begin{bmatrix} \mathbf{v}_{l1} \\ \mathbf{v}_{l2} \\ \mathbf{v}_{l3} \\ \mathbf{v}_{l4} \\ \mathbf{v}_{l5} \\ \mathbf{v}_{l6} \\ \mathbf{v}_{l7} \\ \mathbf{v}_{l8} \end{bmatrix} + \mathbf{B}_A(l, \bullet)^T \tag{229}$$

that each can be solved for $\mathbf{v}_{ll}$ and $\mathbf{v}_{l(l+m)}$, with $l = 1, \ldots, 4$ (as in the pre-training phase, where the remaining weights equaled zero). For example, the first two weight vectors are obtained from the solution of the first system ($l = 1$):

$$\mathbf{v}_{11} = -\left(\mathbf{S}_{B_{L_1}}\right)^{-1}\left[\mathbf{S}_{B_{L_2}} \;\; \mathbf{S}_{B_{LD_1}} \;\; \mathbf{S}_{B_{LD_2}}\right]\left[\mathbf{v}_{12}^T \;\; \mathbf{v}_{13}^T \;\; \mathbf{v}_{14}^T\right]^T - \left(\mathbf{S}_{B_{L_1}}\right)^{-1}\mathbf{b}_{B_{L_1}} \tag{230}$$

$$\mathbf{v}_{15} = -\left(\mathbf{S}_{I_{L_1}}\right)^{-1}\left[\mathbf{S}_{I_{L_2}} \;\; \mathbf{S}_{I_{LD_1}} \;\; \mathbf{S}_{I_{LD_2}}\right]\left[\mathbf{v}_{16}^T \;\; \mathbf{v}_{17}^T \;\; \mathbf{v}_{18}^T\right]^T - \left(\mathbf{S}_{I_{L_1}}\right)^{-1}\mathbf{b}_{I_{L_1}} \tag{231}$$

The vectors $\mathbf{b}_{B_{L_1}}$ and $\mathbf{b}_{I_{L_1}}$ can be obtained from the homonymous scalar output biases $b_{B_{L_1}}$ and $b_{I_{L_1}}$, as in eq. 46, because $\mathbf{b}_A = \mathbf{b}_B + \mathbf{b}_I$. The remaining output weight vectors are similarly computed, by performing a permutation of the set of values for the index l. Except for the output weights, all quantities in eq. 230 and 231 can be computed off line.

A procedure that is slightly more involved leads to the simplification of the gradient weight equations. In this case, the input weights that satisfy the gradient requirements can be computed on line, while the performance function (eq. 212) is being minimized. Again, it is convenient to partition the action network's input weight matrix as suggested by its initialized version, eq. 206,

$$\mathbf{W}_A = \left[\begin{array}{cc|cc|c} \mathbf{W}_{11} & \mathbf{W}_{12} & \mathbf{W}_{13} & \mathbf{W}_{14} & \mathbf{W}_{15} \\ \mathbf{W}_{21} & \mathbf{W}_{22} & \mathbf{W}_{23} & \mathbf{W}_{24} & \mathbf{W}_{25} \\ \mathbf{W}_{31} & \mathbf{W}_{32} & \mathbf{W}_{33} & \mathbf{W}_{34} & \mathbf{W}_{35} \\ \mathbf{W}_{41} & \mathbf{W}_{42} & \mathbf{W}_{43} & \mathbf{W}_{44} & \mathbf{W}_{45} \\ \hline \mathbf{W}_{51} & \mathbf{W}_{52} & \mathbf{W}_{53} & \mathbf{W}_{54} & \mathbf{W}_{55} \\ \mathbf{W}_{61} & \mathbf{W}_{62} & \mathbf{W}_{63} & \mathbf{W}_{64} & \mathbf{W}_{65} \\ \mathbf{W}_{71} & \mathbf{W}_{72} & \mathbf{W}_{73} & \mathbf{W}_{74} & \mathbf{W}_{75} \\ \mathbf{W}_{81} & \mathbf{W}_{82} & \mathbf{W}_{83} & \mathbf{W}_{84} & \mathbf{W}_{85} \end{array}\right] \tag{232}$$

With appropriate dimensions, the submatrices in eq. 232 assume the values of the corresponding submatrices in eq. 206 at the initial time, t_0. The submatrices in the fifth column ($\mathbf{W}_{15}$ through $\mathbf{W}_{85}$) correspond to the **a**-input and together form the matrix $\mathbf{W}_{\mathbf{a}_A}$,

which is held constant at all times for the reasons mentioned above. All of the remaining submatrices together form $\mathbf{W}_{\mathbf{x}_{aA}}$, which is modified on line at every training epoch.

With the above partition, the gradient equations (eq. 223) can be conveniently reformulated as four independent linear systems:

$$\begin{cases} \boldsymbol{\theta}_{B_{L_1}} = \boldsymbol{\sigma}'[\mathbf{N}_A]\,\mathrm{diag}[\mathbf{V}_A(1,\bullet)]\left[\mathbf{W}_{11}^T\ \mathbf{W}_{21}^T\ \mathbf{W}_{31}^T\ \mathbf{W}_{41}^T\ \mathbf{W}_{51}^T\ \mathbf{W}_{61}^T\ \mathbf{W}_{71}^T\ \mathbf{W}_{81}^T\right] \\ \boldsymbol{\theta}_{B_{L_2}} = \boldsymbol{\sigma}'[\mathbf{N}_A]\,\mathrm{diag}[\mathbf{V}_A(2,\bullet)]\left[\mathbf{W}_{11}^T\ \mathbf{W}_{21}^T\ \mathbf{W}_{31}^T\ \mathbf{W}_{41}^T\ \mathbf{W}_{51}^T\ \mathbf{W}_{61}^T\ \mathbf{W}_{71}^T\ \mathbf{W}_{81}^T\right] \end{cases} \tag{233}$$

$$\begin{cases} \boldsymbol{\theta}_{B_{LD_1}} = \boldsymbol{\sigma}'[\mathbf{N}_A]\,\mathrm{diag}[\mathbf{V}_A(3,\bullet)]\left[\mathbf{W}_{12}^T\ \mathbf{W}_{22}^T\ \mathbf{W}_{32}^T\ \mathbf{W}_{42}^T\ \mathbf{W}_{52}^T\ \mathbf{W}_{62}^T\ \mathbf{W}_{72}^T\ \mathbf{W}_{82}^T\right] \\ \boldsymbol{\theta}_{B_{LD_2}} = \boldsymbol{\sigma}'[\mathbf{N}_A]\,\mathrm{diag}[\mathbf{V}_A(4,\bullet)]\left[\mathbf{W}_{12}^T\ \mathbf{W}_{22}^T\ \mathbf{W}_{32}^T\ \mathbf{W}_{42}^T\ \mathbf{W}_{52}^T\ \mathbf{W}_{62}^T\ \mathbf{W}_{72}^T\ \mathbf{W}_{82}^T\right] \end{cases} \tag{234}$$

$$\begin{cases} \boldsymbol{\theta}_{I_{L_1}} = \boldsymbol{\sigma}'[\mathbf{N}_A]\,\mathrm{diag}[\mathbf{V}_A(1,\bullet)]\left[\mathbf{W}_{13}^T\ \mathbf{W}_{23}^T\ \mathbf{W}_{33}^T\ \mathbf{W}_{43}^T\ \mathbf{W}_{53}^T\ \mathbf{W}_{63}^T\ \mathbf{W}_{73}^T\ \mathbf{W}_{83}^T\right] \\ \boldsymbol{\theta}_{I_{L_2}} = \boldsymbol{\sigma}'[\mathbf{N}_A]\,\mathrm{diag}[\mathbf{V}_A(2,\bullet)]\left[\mathbf{W}_{13}^T\ \mathbf{W}_{23}^T\ \mathbf{W}_{33}^T\ \mathbf{W}_{43}^T\ \mathbf{W}_{53}^T\ \mathbf{W}_{63}^T\ \mathbf{W}_{73}^T\ \mathbf{W}_{83}^T\right] \end{cases} \tag{235}$$

$$\begin{cases} \boldsymbol{\theta}_{I_{LD_1}} = \boldsymbol{\sigma}'[\mathbf{N}_A]\,\mathrm{diag}[\mathbf{V}_A(3,\bullet)]\left[\mathbf{W}_{14}^T\ \mathbf{W}_{24}^T\ \mathbf{W}_{34}^T\ \mathbf{W}_{44}^T\ \mathbf{W}_{54}^T\ \mathbf{W}_{64}^T\ \mathbf{W}_{74}^T\ \mathbf{W}_{84}^T\right] \\ \boldsymbol{\theta}_{I_{LD_2}} = \boldsymbol{\sigma}'[\mathbf{N}_A]\,\mathrm{diag}[\mathbf{V}_A(4,\bullet)]\left[\mathbf{W}_{14}^T\ \mathbf{W}_{24}^T\ \mathbf{W}_{34}^T\ \mathbf{W}_{44}^T\ \mathbf{W}_{54}^T\ \mathbf{W}_{64}^T\ \mathbf{W}_{74}^T\ \mathbf{W}_{84}^T\right] \end{cases} \tag{236}$$

The matrices of gains, $\boldsymbol{\theta}$, are obtained from the p = 34 respective gradients in eq. 156 and 160 as, for example, $\boldsymbol{\theta}_{B_{L_1}} \equiv [\,\mathbf{c}^1_{B_{L_1}} \cdots \mathbf{c}^p_{B_{L_1}}\,]^T$. The function "diag" represents an operator that places the ordered elements of a column or row vector on the diagonal of a zero matrix of appropriate dimensions. From eq. 225, the action input-to-node values are known and can obtained from the **N** matrices (eq. 78) of the pre-trained scalar networks:

$$\mathbf{N}_A = \left[\mathbf{N}_{B_{L_1}}\ \mathbf{N}_{B_{L_2}}\ \mathbf{N}_{B_{LD_1}}\ \mathbf{N}_{B_{LD_2}}\ \mathbf{N}_{I_{L_1}}\ \mathbf{N}_{I_{L_2}}\ \mathbf{N}_{I_{LD_1}}\ \mathbf{N}_{I_{LD_2}}\right] \tag{237}$$

Assuming that the matrix $\mathbf{V}_A$ is known, each of the 2p gradient equations above can be solved for 2p input parameters: eq. 233 for $\mathbf{W}_{11}$ and $\mathbf{W}_{21}$, eq. 234 for $\mathbf{W}_{32}$ and $\mathbf{W}_{42}$, eq.

235 for $\mathbf{W}_{53}$ and $\mathbf{W}_{63}$, and eq. 236 for $\mathbf{W}_{74}$ and $\mathbf{W}_{84}$, consistently with the pre-training phase.

Thus, with the matrix $\mathbf{V}_A$ obtained from the new output weight equations (eq. 229), the latest form of the gradient equations can be solved for the input weight matrices indicated above. The procedure is described for eq. 233, and naturally extends to eq. 234 through 236. The first step consists of rewriting eq. 233 by parting the submatrices to be determined, $\mathbf{W}_{11}$ and $\mathbf{W}_{21}$, from the remaining input weights, as follows:

$$\begin{cases} \left[\boldsymbol{\sigma}'\left(\mathbf{N}_{B_{L_1}}\right)\mathrm{diag}\left(\mathbf{v}_{11}\right)\middle|\boldsymbol{\sigma}'\left(\mathbf{N}_{B_{L_2}}\right)\mathrm{diag}\left(\mathbf{v}_{12}\right)\right]\begin{bmatrix}\mathbf{W}_{11}\\ \mathbf{W}_{21}\end{bmatrix}=\mathbf{K}_{B_{L_1}} \\ \left[\boldsymbol{\sigma}'\left(\mathbf{N}_{B_{L_1}}\right)\mathrm{diag}\left(\mathbf{v}_{21}\right)\middle|\boldsymbol{\sigma}'\left(\mathbf{N}_{B_{L_2}}\right)\mathrm{diag}\left(\mathbf{v}_{22}\right)\right]\begin{bmatrix}\mathbf{W}_{11}\\ \mathbf{W}_{21}\end{bmatrix}=\mathbf{K}_{B_{L_2}} \end{cases} \tag{238}$$

The matrices $\mathbf{K}_{B_{L_1}}$ and $\mathbf{K}_{B_{L_2}}$ can be computed from the known gradients and from the remaining input and output weights in eq. 233. They are defined as:

$$\mathbf{K}_{B_{L_1}} \equiv \boldsymbol{\theta}_{B_{L_1}} - \boldsymbol{\sigma}'\left[\mathbf{N}_{B_{LD_1}} \;\; \mathbf{N}_{B_{LD_2}} \;\; \mathbf{N}_{I_{L_1}} \;\; \mathbf{N}_{I_{L_2}} \;\; \mathbf{N}_{I_{LD_1}} \;\; \mathbf{N}_{I_{LD_2}}\right]\mathrm{diag}\begin{bmatrix}\mathbf{v}_{13}\\ \vdots\\ \mathbf{v}_{18}\end{bmatrix}\begin{bmatrix}\mathbf{W}_{31}\\ \vdots\\ \mathbf{W}_{81}\end{bmatrix} \tag{239}$$

$$\mathbf{K}_{B_{L_2}} \equiv \boldsymbol{\theta}_{B_{L_2}} - \boldsymbol{\sigma}'\left[\mathbf{N}_{B_{LD_1}} \;\; \mathbf{N}_{B_{LD_2}} \;\; \mathbf{N}_{I_{L_1}} \;\; \mathbf{N}_{I_{L_2}} \;\; \mathbf{N}_{I_{LD_1}} \;\; \mathbf{N}_{I_{LD_2}}\right]\mathrm{diag}\begin{bmatrix}\mathbf{v}_{23}\\ \vdots\\ \mathbf{v}_{28}\end{bmatrix}\begin{bmatrix}\mathbf{W}_{31}\\ \vdots\\ \mathbf{W}_{81}\end{bmatrix} \tag{240}$$

Then, the solution of the system in eq. 238 can be written as:

$$\begin{aligned}\begin{bmatrix}\mathbf{W}_{11}\\ \mathbf{W}_{21}\end{bmatrix} &= \begin{bmatrix}\boldsymbol{\sigma}'\left(\mathbf{N}_{B_{L_1}}\right)\mathrm{diag}\left(\mathbf{v}_{11}\right) & \boldsymbol{\sigma}'\left(\mathbf{N}_{B_{L_2}}\right)\mathrm{diag}\left(\mathbf{v}_{12}\right)\\ \boldsymbol{\sigma}'\left(\mathbf{N}_{B_{L_1}}\right)\mathrm{diag}\left(\mathbf{v}_{21}\right) & \boldsymbol{\sigma}'\left(\mathbf{N}_{B_{L_2}}\right)\mathrm{diag}\left(\mathbf{v}_{22}\right)\end{bmatrix}^{-1}\begin{bmatrix}\mathbf{K}_{B_{L_1}}\\ \mathbf{K}_{B_{L_2}}\end{bmatrix} \\ &\equiv \mathbf{Z}_{B_L}^{-1}\mathbf{K}_{B_L}\end{aligned} \tag{241}$$

The subscript "B_L" refers to the input-to-node values and gain matrices that generate the matrices $\mathbf{Z}$ and $\mathbf{K}$; as, these weights are associated with the longitudinal outputs (δT and

δS) and with the input-to-node values of $NN_{B_{L_1}}$ and $NN_{B_{L_2}}$ in the gradient equations (eq. 233-236).

All of the initialized (non-zero) input and output weights that satisfy the pre-training requirements on line can be computed with low-order polynomial time (as explained in Section 5.4). The remaining parameters, initialized at zero in eq. 206 and 207, can be used to minimize eq. 212. An iterative algorithm of the type in eq. 101 can be used to accomplish both objectives incrementally. With the above developments in mind, the vector **w** is formed from the parameters allocated for the performance-function minimization:

$$\mathbf{w} \equiv \begin{bmatrix} \text{vec}\left(\begin{bmatrix} \mathbf{W}_{31} & \mathbf{W}_{41} & \mathbf{W}_{51} & \mathbf{W}_{61} & \mathbf{W}_{71} & \mathbf{W}_{81} \\ \mathbf{W}_{12} & \mathbf{W}_{22} & \mathbf{W}_{52} & \mathbf{W}_{62} & \mathbf{W}_{72} & \mathbf{W}_{82} \\ \mathbf{W}_{13} & \mathbf{W}_{23} & \mathbf{W}_{33} & \mathbf{W}_{43} & \mathbf{W}_{73} & \mathbf{W}_{73} \\ \mathbf{W}_{14} & \mathbf{W}_{24} & \mathbf{W}_{34} & \mathbf{W}_{44} & \mathbf{W}_{54} & \mathbf{W}_{64} \end{bmatrix}\right) \\ \text{vec}\left(\begin{bmatrix} \mathbf{v}_{12} & \mathbf{v}_{13} & \mathbf{v}_{14} & \mathbf{v}_{16} & \mathbf{v}_{17} & \mathbf{v}_{18} \\ \mathbf{v}_{21} & \mathbf{v}_{23} & \mathbf{v}_{24} & \mathbf{v}_{25} & \mathbf{v}_{27} & \mathbf{v}_{28} \\ \mathbf{v}_{31} & \mathbf{v}_{32} & \mathbf{v}_{34} & \mathbf{v}_{35} & \mathbf{v}_{36} & \mathbf{v}_{38} \\ \mathbf{v}_{41} & \mathbf{v}_{42} & \mathbf{v}_{43} & \mathbf{v}_{45} & \mathbf{v}_{46} & \mathbf{v}_{47} \end{bmatrix}\right) \end{bmatrix} \tag{242}$$

At every epoch (i+1), an increment $\Delta w_\ell^{(i)}$ that modifies each of the weights $w_\ell^{(i)}$ in $\mathbf{w}^{(i)}$ is computed from eq. 89 and 90 by the algorithm in Section 5.1.3. Then, all of the submatrices in eq. 242 are known from $\mathbf{w}^{(i+1)}$.

With the stated assumptions of constant biases and $\mathbf{W}_{\mathbf{a}_A}$, the solution of the output equations is formulated in terms of constant matrices that can be computed prior to the on-line adaptation. For example, the constant matrices in eq. 230 are defined as

$$\mathbf{\Gamma}_{B_{L_1}} \equiv -\left(\mathbf{S}_{B_{L_1}}\right)^{-1}\left[\mathbf{S}_{B_{L_2}} \quad \mathbf{S}_{B_{LD_1}} \quad \mathbf{S}_{B_{LD_2}}\right] \tag{243}$$

$$\mathbf{E}_{B_{L_1}} \equiv -\left(\mathbf{S}_{B_{L_1}}\right)^{-1}\mathbf{b}_{B_{L_1}} \tag{244}$$

They can be used to solve for the output weights associated with $NN_{B_{L_1}}$, i.e., $\mathbf{v}_{11}$. All of the remaining systems in eq. 229 can be formulated in terms of matrices that are similarly defined. The gradient weight equations (eq.s 233-236) can be expressed in terms of constant matrices that are easily computed *a priori* from the known input-to-node values, and in terms of input and output weights that become available on line (as shown in eq. 241). In particular, the diagonal structure of the output-weight submatrices that compose $\mathbf{Z}_{B_L}$ (eq. 241) can be exploited to reduce the amount of computation required to invert this matrix (e.g., through its LUP decomposition [94]).

Thus, the input and output weights excluded by eq. 242 are computed from the respective weight equations at every epoch. The simplified equations obtained above are used within an algebraically-constrained supervised-training algorithm to guarantee that the pre-training requirements are satisfied on line, while the weights in **w** are used to minimize the network performance, *E*. In summary, at each epoch, the action network parameters are updated according to the following sequential rules:

$$w_\ell^{(i+1)} = w_\ell^{(i)} + \Delta w_\ell^{(i)} \rightarrow \mathbf{w}^{(i+1)} \tag{245}$$

$$\mathbf{v}_{11}^{(i+1)} = \mathbf{\Gamma}_{B_{L_1}} \begin{bmatrix} \mathbf{v}_{12}^{(i+1)} \\ \mathbf{v}_{13}^{(i+1)} \\ \mathbf{v}_{14}^{(i+1)} \end{bmatrix} + \mathbf{E}_{B_{L_1}}, \quad \mathbf{v}_{15}^{(i+1)} = \mathbf{\Gamma}_{I_{L_1}} \begin{bmatrix} \mathbf{v}_{16}^{(i+1)} \\ \mathbf{v}_{17}^{(i+1)} \\ \mathbf{v}_{18}^{(i+1)} \end{bmatrix} + \mathbf{E}_{I_{L_1}} \tag{246}$$

$$\mathbf{v}_{22}^{(i+1)} = \mathbf{\Gamma}_{B_{L_2}} \begin{bmatrix} \mathbf{v}_{21}^{(i+1)} \\ \mathbf{v}_{23}^{(i+1)} \\ \mathbf{v}_{24}^{(i+1)} \end{bmatrix} + \mathbf{E}_{B_{L_2}}, \quad \mathbf{v}_{26}^{(i+1)} = \mathbf{\Gamma}_{I_{L_2}} \begin{bmatrix} \mathbf{v}_{25}^{(i+1)} \\ \mathbf{v}_{27}^{(i+1)} \\ \mathbf{v}_{28}^{(i+1)} \end{bmatrix} + \mathbf{E}_{I_{L_2}} \tag{247}$$

$$\mathbf{v}_{33}^{(i+1)} = \mathbf{\Gamma}_{B_{LD_1}} \begin{bmatrix} \mathbf{v}_{31}^{(i+1)} \\ \mathbf{v}_{32}^{(i+1)} \\ \mathbf{v}_{34}^{(i+1)} \end{bmatrix} + \mathbf{E}_{B_{LD_1}}, \quad \mathbf{v}_{37}^{(i+1)} = \mathbf{\Gamma}_{I_{LD_1}} \begin{bmatrix} \mathbf{v}_{35}^{(i+1)} \\ \mathbf{v}_{36}^{(i+1)} \\ \mathbf{v}_{38}^{(i+1)} \end{bmatrix} + \mathbf{E}_{I_{LD_1}} \tag{248}$$

$$\mathbf{v}_{44}^{(i+1)} = \mathbf{\Gamma}_{B_{LD_2}} \begin{bmatrix} \mathbf{v}_{41}^{(i+1)} \\ \mathbf{v}_{42}^{(i+1)} \\ \mathbf{v}_{43}^{(i+1)} \end{bmatrix} + \mathbf{E}_{B_{LD_2}}, \quad \mathbf{v}_{48}^{(i+1)} = \mathbf{\Gamma}_{I_{LD_2}} \begin{bmatrix} \mathbf{v}_{45}^{(i+1)} \\ \mathbf{v}_{46}^{(i+1)} \\ \mathbf{v}_{47}^{(i+1)} \end{bmatrix} + \mathbf{E}_{I_{LD_2}} \tag{249}$$

$$\begin{bmatrix} \mathbf{W}_{11}^{(i+1)} \\ \mathbf{W}_{21}^{(i+1)} \end{bmatrix} = \left\{ \mathbf{Z}_{B_L}^{(i+1)} \right\}^{-1} \mathbf{K}_{B_L} \tag{250}$$

$$\begin{bmatrix} \mathbf{W}_{32}^{(i+1)} \\ \mathbf{W}_{42}^{(i+1)} \end{bmatrix} = \left\{ \mathbf{Z}_{B_{LD}}^{(i+1)} \right\}^{-1} \mathbf{K}_{B_{LD}} \tag{251}$$

$$\begin{bmatrix} \mathbf{W}_{53}^{(i+1)} \\ \mathbf{W}_{63}^{(i+1)} \end{bmatrix} = \left\{ \mathbf{Z}_{I_L}^{(i+1)} \right\}^{-1} \mathbf{K}_{I_L} \tag{252}$$

$$\begin{bmatrix} \mathbf{W}_{74}^{(i+1)} \\ \mathbf{W}_{84}^{(i+1)} \end{bmatrix} = \left\{ \mathbf{Z}_{I_{LD}}^{(i+1)} \right\}^{-1} \mathbf{K}_{I_{LD}} \tag{253}$$

The above algorithm implies that, at the $(i)^{\text{th}}$ epoch, the update of the $\mathbf{Z}$ matrices is based on the new $(i+1)^{\text{th}}$ value of the input and output weights in eq. 239-241. The remaining matrices, $\mathbf{\Gamma}$, $\mathbf{E}$, $\mathbf{K}$, $\boldsymbol{\theta}$, and $\boldsymbol{\sigma}'(\mathbf{N}^T)^{-1}$ can be computed off line, from *a-priori* information (such as eq. 225, 156, and 160) and they are held constant thereafter. Hence, these algebraic equations constitute the criteria for updating the action network weights on line, while preserving the *a-priori* knowledge intact. A virtually identical algorithm can be obtained for the critic network, but is not shown for brevity's sake. This result proves that the neural networks involved are capable of meeting the objectives of both the pre-training phase and the on-line adaptation, providing a theoretical justification for the results obtained in the previous sections. Together with the simulations in Section 5.2, this conclusion also implies that the modified RPROP algorithm (Section 5.1.3) already is powerful enough to approximate the solution of eq. 246-253 exclusively through a gradient-based search.

5.4 A Word on Computational Complexity: Execution Time of Algebraic and Adaptive-Learning Algorithms

From a computational perspective, the use of Approximate Dynamic Programming (ADP) for solving optimization problems can be justified by the existence of many NP-complete problems for which obtaining an optimal solution is intractable. The DHP adaptive critic architecture presented in the previous sections can be considered as an approximation algorithm that seeks a near-optimal solution for the optimal control problem at hand. Although a formal analysis of the proposed algorithm is beyond the scope of this thesis, it is important to address the running time of the main subroutines introduced thus far, to begin to show that the approach produces a polynomial-time approximation algorithm. In particular, the running time of the newly developed training algorithms is discussed to emphasize their efficiency. Error-bounds are not analyzed here; however, since they constitute an integral part of approximation schemes' analysis, they are strongly recommended as a topic for future work.

The inputs to the optimization problem and, thus, to the DHP solution scheme can be identified with the state $\mathbf{x}_a$ and the control $\tilde{\mathbf{u}}$ of size $(n + e_c)$ and m, respectively. A general concern is how rapidly the running time grows with respect to these dimensions. However, in considering individual subroutines, the input size typically refers to the number of items in the input to the algorithm [94]. Therefore, the relevant dimensions depend on the context that is being investigated. Also, the running time is measured as the number of primitive operations executed, where it is understood that different operations require different execution times. Since the *order of growth* with respect to the input dimension is of major concern, the worst-case running time is independent of

the primitive execution times [94]. In this section, the bounds on the running-time function are used to judge the computational expense of algorithms. In particular, the Θ-notation, asymptotically bounding a function from above and below is adopted. For a given function $g(d)$, the notation $\Theta(g(d))$ denotes the set of functions,

$$\Theta(g(d)) = \{f(d) : \exists\, c_1, c_2, \text{and } d_0 \text{ such that } 0 \le c_1 g(d) \le f(d) \le c_2 g(d), \forall d \ge d_0\} \tag{254}$$

where c_1, c_2, and d_0 are positive constants.

It can be observed from the previous sections that the algebraic and adaptive learning techniques introduced are based entirely on matrix operations. More precisely, they are comprised of steps whose worst-case running times are associated with matrix operations, such as multiplication and inversion. Therefore, statements characterized by lower-order polynomial times, such as "diag" or "Vec" operations, can be ignored. Also, all of the learning algorithms terminate in a finite number of steps that is independent of the number of inputs. Strictly speaking, in the case of the RPROP on-line training algorithm, this is not true, because as the input dimension increases, where the input can be identified with **w** (and, hence, with the size of the network), the number of epochs needed for satisfactorily improving the performance function also is expected to increase. In this instance, error-bound analysis would be required to produce a formal estimate of the required running time. For practical purposes, if we assume that the early-termination rule of Section 5.1.3 produces a satisfactory performance bound, then this routine also terminates in a finite number of steps that is either independent of **w**, or that has a low-order-polynomial dependency with respect to its size.

The solution times of matrix multiplication and inversion are key attributes for this analysis. Typically, linear systems of equations are solved through LUP decomposition, rather than by matrix inversion, to avoid numerical instability [94]. The cost of computing the LUP decomposition of a $d \times d$ matrix is $\Theta(d^3)$. Once the LUP decomposition of a matrix is available, any corresponding linear system of equations can be solved in time $\Theta(d^2)$ [94]. The multiplication of two $d \times d$ matrices performed according to its definition requires a running time that is $\Theta(d^3)$, but it also can be obtained by the Stressen's algorithm in reduced time $\Theta(d^{2.81})$. A useful property of the Θ-notation is that if an operation with polynomial running time $\Theta(d^p)$ is repeated a number of times c, than the resulting algorithm is $\Theta(cd^p)$, or simply $\Theta(d^p)$ when c is constant with respect to d. For example, the gradient-based training algorithm in Section 3.1.1 consists of solving one p-dimensional linear system (eq. 55) and n p-dimensional linear systems (eq. 56). Provided it is well posed, the algorithm is not recurrent. Therefore, for constant n, the worst-case computation time of this subroutine is $\Theta(p^3)$. The running time for pre-training m scalar networks could be obtained in terms of the dimensions of the optimization inputs, $\mathbf{x}_a$ and $\tilde{\mathbf{u}}$, by observing that p depends on n and, in particular, on the number of scheduling variables $(n - e)$. Since p grows exponentially with $(n - e)$, this dependency is likely to be far less optimistic than the former one.

Following the pre-training phase, the neural networks are trained on line by a modified RPROP algorithm (Section 5.1.3). Because only the sign of the gradient is needed, sigmoidal-function evaluations only are required once, for computing the neural network output; and, they also require polynomial time. The sign of the gradient in eq. 217 is obtained from the matrix multiplications in eq. 217-218, in one step. The update rules in

eq. 89 and 90 are characterized by linear first-order recurrence. Therefore, a preliminary analysis shows that this algorithm also is executed in low-order polynomial time, when the appropriate conditions for local convergence are satisfied.

The constrained version of the RPROP algorithm (in Section 5.3) entails solving linear systems of equations for which the LUP decomposition can be computed and stored off line. The only exception to this statement consists of the gradient weight equations. However, in this case, the diagonal structure of the output-weight submatrices (shown in eq. 241) can be exploited to reduce the computational complexity of the solution. Therefore, the running time for the algebraically-constrained training algorithm also is low-order polynomial. In summary, breaking the optimal control structure down to sub-problems that can be solved by computationally efficient algorithms shows to be a promising approach to tackling otherwise intractable problems. On the other hand, a more extensive analysis is required to investigate the behavior of the approximation scheme and error bounds with respect to the most relevant dimensions, i.e., those of the state and control vectors.

5.5 Chapter Summary

The results of the pre-training phase are linked to the adaptive critic architecture of choice, i.e., dual heuristic programming. Consequently, the cost function and the neural parameters to be optimized on line are determined algebraically from those computed from linear control theory. The proportional-integral neural network controller is adapted on line through action and critic criteria obtained from the recurrence relation of dynamic programming. During every time interval, the adaptive critic design is used to compute

desired targets for the action and the critic networks, and a modified resilient backpropagation algorithm is implemented to update the network parameters accordingly.

The adaptive controller is tested during large-angle maneuvers and flight conditions unaccounted for by the pre-training phase. The results show that the system's performance is improved incrementally over time. The optimality condition and the network errors are monitored to verify that the action and the critic function approximations are converging to a near-optimal solution. The adaptation is so effective as to prevent closed-loop instability during a large-bank turn, and to improve performance in the presence of unforeseen maneuvers, unmodeled failures, and parameter variations. Under virtually all circumstances, the global control knowledge introduced during the pre-training phase always is well preserved; and, eventually, it is improved upon in those state-space regions that are explored by the simulation. An algebraically-constrained algorithm can be obtained to guarantee and justify the preservation of *a-priori* knowledge during on-line learning. Finally, a preliminary analysis of the training algorithms is proposed to address computational complexity and worst-case solution times.

Chapter 6

Conclusions

6.1 Summary

The primary objective of this thesis is to develop a systematic design procedure for a control system that retains the stability and robustness characteristics of the classical designs, while capitalizing on the computational capabilities of approximate dynamic programming and neural networks. The result is an adaptive control system that learns to deal with new system dynamics as they arise, improving performance during large-angle maneuvers and unforeseen conditions, such as control failures and parameter variations. The proposed design philosophy consists of constructing a nonlinear controller, comprising a network of neural networks, using a two-phase learning procedure. First, the networks' architecture, parameters, and size (i.e., number of hidden nodes) are determined from the initial specification of the control law. Secondly, on-line learning by an adaptive critic approach is expected to preserve prior control knowledge and improve performance by accounting for dynamic effects that were not captured in the initial control law.

Gain scheduling is a well-known procedure for applying linear control theory to nonlinear systems that is widely applicable, especially in the aerospace and chemical-processes industry. It consists of locally approximating the nonlinear plant as a linear-parameter-varying system at several operating points, and of designing corresponding control gains that later can be interpolated through dynamically-significant scheduling

variables. Central to this novel approach is the recognition that the gradients of a nonlinear control law represent the gain matrices of an equivalent, multivariable linear control structure that is chosen as the proportional-integral (PI) controller, for illustration. Hence, a set of satisfactory linear controllers is obtained through an established procedure known as implicit model following to satisfy well-known aircraft handling qualities and design criteria. Then, a novel gradient-based pre-training technique is used to match these linear controllers exactly by means of nonlinear neural networks in one step. A PI neural network controller is obtained by replacing the linear gains of the PI controller with the pre-trained neural networks. This architecture performs at least as well as an equivalent gain-scheduled design, immediately following the pre-training phase.

Adaptive critic designs constitute a class of approximate dynamic programming methods that optimize a short-term metric, ensuring optimization of the cost over all future times. Neural networks typically are the approximating structure of choice, because they easily handle large-dimensional input and output spaces and can learn in an incremental fashion. A dual heuristic programming adaptive critic architecture is used to adapt the pre-trained PI neural network controller over time, while a full-scale simulation of the aircraft is flying throughout its operating region. A good deal of literature has been written about the theoretical motivation behind dual heuristic programming, as a method that promises fast convergence and great potential for on-line learning. However, for several years now, the bottlenecks associated with its implementation have prevented this approach from realizing its promise.

The implementation's details and algorithms that allow for a successful implementation of dual heuristic programming (DHP) are described in this thesis. The

pre-training phase provides an excellent initialization point for the on-line phase. Furthermore, the advancement made in algebraic and adaptive learning achieve the on-line phase's objective of improving performance in the presence of unknown dynamics, while preserving the global designs incorporated off line. Classical control theory provides a unifying framework for the two training phases. The pre-training phase is based on the linear quadratic regulator; the adaptive critic approach is based on the recurrence relation of dynamic programming.

6.2 Conclusions

The foundations have been laid for a novel approach to designing nonlinear control systems that make the most of prior knowledge and experience, while capitalizing on the broader capabilities of adaptive control theory and computational neural networks. The principles introduced can be applied to any nonlinear control law that affords a gain-scheduled law formulation. During pre-training, not only the adjustable parameters, but also the network's size and architecture that are apt to meet the desired specifications are rapidly determined by solving linear algebraic equations, with no need for iteration. For example, through this novel algebraic approach it can be shown that a number of nodes equal to the number of operating points (or training pairs) achieves exact matching of the data with probability one. The approximation properties of neural networks can be investigated to a great extent by using elementary linear algebra.

The simulations show that a relatively small number of operating points, and thus of hidden neural nodes, is sufficient to obtain satisfactory neural network control throughout the steady-level flight envelope of the aircraft. The method produces a network of minimal neural networks that approximate the hypersurfaces of a global, nonlinear

control law effectively and that match equivalent, locally linearized controllers exactly. Furthermore, a number of algebraic training algorithms have been developed that can be extended to many other neural network applications, provided the training data is free of noise and available at once (also referred to as batch or off-line training).

Additional advances have been made in on-line learning techniques and in adaptive critic methods. The pre-training phase can be linked to the on-line adaptation by realizing that the same cost function is to be optimized during both phases, for consistency. Then, the cost weighting matrices and the action and critic initial parameters to be used during the on-line phase can be determined algebraically from the results of the previous phase. An existing training algorithm, resilient backpropagation, is modified to take into consideration properly-initialized parameters. It is found that the use of proportional initial increments, backstepping, sigmoidal monotonic properties, and early termination rules leads to an on-line training algorithm that preserves initial knowledge and that is characterized both by excellent convergence and computational savings.

These and other developments described in this thesis lead to a successful implementation of the nonlinear adaptive controller for the command-input control of a full-scale aircraft simulation. Relatively high-dimensional neurocontrollers adjust on line while retaining their baseline performance in unexplored regions of the state space. This behavior, observed in the simulations, can be justified theoretically by investigating on-line learning through the neural network weight equations, according to the algebraic training approach. During the on-line phase, the adaptation spontaneously utilizes those parameters that were unused during pre-training. Also, it allows the neurocontrollers to

significantly improve their performance over only one or few epochs, during each time increment. The advancements of all key design stages combined greatly increase the potential for real life applications of intelligent and reconfigurable control.

6.3 Recommendations

The main recommendation for future work is to expand upon the findings of this thesis to investigate error bounds, closed-loop stability, and robustness of the adaptive control system. Since the approach is iterative and relies heavily on computation, a rigorous analysis of the algorithms presented can be related to the first objective. In particular, it would be relevant to determine the worst-case computation times and error bounds with respect to the dimensions of the state and control, as well as the number of stages. The algebraic techniques developed, together with existing theories on Markov decision processes and on the stability of gain-scheduled designs show particular promise in this direction. Alternatively, other classical designs could be used for pre-training the controller and existing techniques (such as linear matrix inequalities) extended to include this adaptive system. The investigation of the neural approximation properties should remain a key ingredient in the process, as the neural architectures determine the class of control and cost functionals that can be approximated and, therefore, the optimal control problems that can and cannot be solved.

One of the most desirable features of this approach is its flexibility. Not only is it unrestricted by the form of the governing dynamic equation, but it allows for extensions that can deal with system identification, stochastic processes, disturbances, and state estimation, to name a few. For example, it is possible to use a *model network* to approximate the plant dynamics, allowing it to perform system identification on line. In

this case, local network gradients would be provided by the transition matrices that derive from the difference governing equation. In this case, the gradient-based algebraic training algorithm could be used to pre-train the model network as well. Subsequently, its parameters could be updated on line through the modified resilient backpropagation algorithm, as for the other networks. In this case, the forward neural network, which approximates an inversion of the plant dynamics, also should be updated on line. The rest of the design would likely remain the same, whereas the controller already learns nonlinear system dynamics on line.

A valuable extension would be to use not only the training techniques developed here, but also the entire control design process for other applications. This would be an excellent way to validate the results of the thesis, as well as to identify bottlenecks that may have been missed during this implementation. The range of possibilities is at least as diverse as are the neural network applications that already exist today in the literature. In particular, designs that can benefit both from *a-priori* and *a-posteriori* knowledge of the system would be ideal. Examples include process control and planning, routing problems in air traffic management and communication networks, pattern identification for speech/audio-recognition devices, criminal profiling, and target assignment in combat scenarios.

The optimal estimation problem can be seen as the dual of the optimal control problem. Therefore, the approach developed here for the near-optimal solution of the aircraft control problem could be extended to train a near-optimal nonlinear estimator by using, for example, Kalman filter gains during the pre-training phase [81]. Another consideration is that the sampled-time interval that was kept small in the simulations

could easily be increased to deal with a discrete-event process or to allow sufficient time for the adaptation to take place in a real-life application. In this case, a trivial extension consists of using discrete-time linear designs to pre-train the neural networks for solving either a control or an estimation problem. Since the on-line phase already is analyzed in discrete time, the same adaptive critic architecture could be used for improving the performance of a discrete controller.

In principle, adaptive critic designs should be capable of determining a near-optimal policy for a stochastic plant and/or environment. In fact, there exist important convergence proofs for the approximate dynamic programming approach that also hold in the presence of white noise and Markov noise [16, 4]. This is an important and, yet, ambitious direction of research that is likely to require substantial modifications of the approach, such as the use of stochastic approximators [95]. Finally, there is considerable interest in the field for high-dimensional problems, where the state and control have many variables [96]. Hence, the study of computation complexity should be a major focus of any solution method pursued hereon.

APPENDICES

Appendix A: Nomenclature

Symbol	Description
θ	Matrix of selected known gradient vectors
ς	Vector of all known gradients in a given training set
η	Vector of input-to-node values evaluated at all of the operating points in a given training set
ξ	Time integral of the output error
Γ	Constant matrix formed from selected sigmoidal matrices, for algebraically-constrained on-line training
$\mathbf{a}$	Scheduling vector (or vector of scheduling variables)
$\mathbf{A}$	Matrix of all scheduling vectors in a given training set
$\mathbf{b}$	Output bias of a vector-output neural network
$\mathbf{B}^k$	Constant matrix formed from the network output weights and from the sigmoid's derivatives evaluated at selected input-to-node-values
$\mathbf{C}$	Linear control gain matrix
$\mathbf{c}^k$	Known vector gradient of a scalar-output neural network, evaluated at the k^{th}-operating point in a given training set
$\mathbf{C}^k$	Known Hessian gradient-matrix of a vector-output neural network, evaluated at the k^{th}-operating point in a given training set
$\mathbf{d}$	Input bias of a neural network
$\mathbf{E}$	Constant matrix formed from the inverse of a selected sigmoidal matrix and the output bias, for algebraically-constrained on-line training
$\mathbf{e}_{\mathbf{NN}}$	Neural network vector-output error
$\mathbf{F}$	State-Jacobian matrix of a linear dynamical system
$\mathbf{G}$	Control-Jacobian matrix of a linear dynamical system
$\mathbf{K}$	Matrix of selected known quantities in an algebraically-constrained on-line training algorithm
$\mathbf{H}_\mathbf{u}$	Jacobian matrix of a linear system's output with respect to the control
$\mathbf{H}_\mathbf{x}$	Jacobian matrix of a linear system's output with respect to the state

$\mathbf{I}_n$	Diagonal matrix with n elements along the diagonal
$\mathbf{M}$	Matrix of cross-coupling weighting between the state and the controls
$\mathbf{N}$	Matrix of input-to-node values evaluated at all of the operating points in a given training set
$\mathbf{n}^k$	Vector of input-to-node values evaluated at the k^{th}-operating point in a given training set
$\mathbf{P}$	Riccati matrix
$\mathbf{p}$	Vector input of a neural network
$\mathbf{p}_m$	Parameter vector of a dynamical system
$\mathbf{Q}$	State-weighting matrix
$\mathbf{R}$	Control-weighting matrix
$\mathbf{S}$	Sigmoidal matrix
$\mathbf{T}$	Linear transformation matrix
$\mathbf{u}$	Control vector
$\mathbf{U}$	Matrix of all known output vectors in a given training set
$\mathbf{u}^k$	Neural network sampled vector-output information
$\mathbf{u}_0$	Nominal control vector
$\mathbf{v}$	Output-weight vector of a scalar-output neural network
$\mathbf{V}$	Output-weight matrix of a vector-output neural network
$\mathbf{W}$	Input-weight matrix of a neural network
$\mathbf{w}$	Vector of ordered neural network weights
$\mathbf{x}$	State vector
$\mathbf{X}$	Sparse matrix composed of the matrices $\mathbf{B}^k$, evaluated at all operating points in a given training set
$\mathbf{x}_0$	Nominal state vector
$\mathbf{Y}$	Matrix of all known input vectors in a given training set

$\mathbf{y}_c$	Command input
$\mathbf{y}^k$	Neural network sampled input information
$\mathbf{y}_s$	Output of a dynamical system
$\mathbf{z}$	Output of a neural network
$\mathbf{Z}$	Matrix of output weights and known sigmoidal-function derivatives, in an algebraically-constrained on-line training algorithm
ζ	Damping ratio of a linear system's response
τ	Time constant of a linear system's response
θ	Aircraft Euler pitch angle
α	Aircraft angle of attack
ψ	Aircraft Euler yaw angle
ω_n	Natural frequency of a linear system's response
Δt	Time increment in a sampled- time representation
f	User-defined scalar factor
H	Aircraft altitude
H_0	Nominal aircraft altitude
I_{xx}	Aircraft mass moment of inertia about the x_b axis
I_{xz}	Aircraft mass product of inertia about the y_b axis
I_{yy}	Aircraft mass moment of inertia about the y_b axis
I_{zz}	Aircraft mass moment of inertia about the z_b axis
p	Aircraft body-axis roll rate
q	Aircraft body-axis pitch rate
r	Aircraft body-axis yaw rate
t	Time variable, in a continuous-time representation

t_k	Time variable, in a sampled-time representation
u	Forward, or x-body-axis component of aircraft velocity
u^k	Neural network sampled scalar-output information
V	Total aircraft velocity or airspeed
v	Side, or y-body-axis component of aircraft velocity
V_0	Nominal airspeed
w	Downward, or z-body-axis component of aircraft velocity
w_ℓ	Adjustable ℓ^{th}-ordered parameter, or weight, of a neural network
x_b	Aircraft x body axis
x_r	Inertial x axis
y_b	Aircraft y body axis
y_r	Inertial y axis
z_b	Aircraft z body axis
z_r	Inertial z axis
Δ_ℓ	Size of the increment for the ℓ^{th}-ordered parameter of a neural network
Δw_ℓ	Increment for the ℓ^{th}-ordered parameter of a neural network
β	Aircraft-sideslip angle
δA	Aircraft-aileron deflection
δR	Aircraft-rudder deflection
δS	Aircraft-stabilator deflection
δT	Aircraft-throttle control
ϕ	Aircraft Euler roll angle

γ	Aircraft-path angle
μ	Aircraft-bank angle
$\sigma'(\bullet)$	Derivative of the sigmoidal function with respect to its argument
$\sigma(\bullet)$	Sigmoidal function
$\boldsymbol{\lambda}(t)$	Derivative of the value function with respect to the state
$\varphi[\bullet]$	Terminal state penalty-term in the cost function
$E(\bullet)$	Performance function of a neural network
$g_{b_x}(\bullet)$	x body-axis gravity component
$g_{b_y}(\bullet)$	y body-axis gravity component
$g_{b_z}(\bullet)$	z body-axis gravity component
$H[\bullet]$	Hamiltonian
$J[\bullet]$	Cost function
$J_{kN}[\bullet]$	Cost function between the k^{th} and the N^{th} stage in a multi-staged (or sampled-time) process
$L[\bullet]$	Lagrangian
$L_b(\bullet)$	Acceleration due lift
$M_b(\bullet)$	Acceleration due to pitching moment
$N_b(\bullet)$	Acceleration due to yawing moment
$V[\bullet]$	Value function or cost-to-go
$X_b(\bullet)$	Acceleration modeled in the x_b direction
$Y_b(\bullet)$	Acceleration modeled in the y_b direction
$Z_b(\bullet)$	Acceleration modeled in the z_b direction

ER	Set of extrapolating conditions in the operating region of a dynamical system
IR	Set of interpolating conditions in the operating region of a dynamical system
$\mathbf{NN}$	Vector-output mapping by a neural network
NN	Scalar-output mapping by a neural network
OP	Set of design operating points
OR	Full operating region of a dynamical system
U_c	Aircraft trim map
$(\tilde{\bullet})$	Deviation from the commanded value of a variable
$(\hat{\bullet})$	Estimated value of a variable
$(\,)^*$	Optimal value of a variable
$(\,)_{\xi}$	Variable associated with ξ, the time integral of the output error
$(\,)^{-1}$	Inverse of a matrix
$(\,)_A$	Variable associated with the action neural network
$(\,)_{\mathbf{a}}$	Variable associated with $\mathbf{a}$, the scheduling vector
$(\,)_a$	Variable associated with the augmented state, $\mathbf{x}_a$
$(\,)_B$	Variable associated with the feedback neural network
$(\,)_C$	Variable associated with the critic neural network
$(\,)_c$	Commanded variable
$(\,)_D$	Desired value of a variable
$(\,)_{DR}$	Variable associated with the Dutch Roll
$(\,)_F$	Variable associated with the forward neural network
$(\,)^G$	Initial guess for an unknown variable
$(\,)_I$	Variable associated with the command-integral neural network

$(\,)^k$, or $(\,)^\kappa$	Variable evaluated at the k^{th} training pair/triad, or at the κ^{th} operating point
$(\,)_L$	Variable associated with the aircraft longitudinal dynamics
$(\,)_{LD}$	Variable associated with the aircraft lateral-directional dynamics
$(\,)_m$	Variable associated with the ideal aircraft model
$(\,)_P$	Variable associated with the Phugoid mode of the aircraft
$(\,)^{PI}$	Left pseudo-inverse operator
$(\,)_{SD}$	Sampled-data (or discrete-time) variable
$(\,)_{SP}$	Variable associated with the short-period mode of the aircraft
$(\,)^T$	Transpose operator
$(\,)_{\mathbf{x}}$	Variable associated with **x**, the state vector
$(\dot{\cdot})$	Derivative with respect to time
$\otimes$	Element-wise multiplication between vectors of the same size
Θ	Asymptotically bounds a function from above and from below
$\Delta(\,)$	Deviation from the nominal value of the variable
diag( )	Diagonal operator, extracting the diagonal of the square-matrix argument, or placing the vector argument on the diagonal of a zero matrix
mse( )	Mean-squared error of a vector or matrix
rank( )	Rank of a matrix
sgn( )	Signum operator
Vec( )	Kronecker Vec operator, rearranging the elements of a matrix column-wise into a vector

Appendix B: Algorithms

This appendix provides the MATLAB implementation of sample algorithms developed in the thesis. The first algorithm was used in Chapter 4 to pre-train the feedback, command-integral, and critic scalar neural networks, and is based on the exact gradient-based solution introduced in Section 3.1.1. Figure B.1 illustrates how it can be coded in MATLAB. Given the gradient-based training set $\{[\mathbf{0} \mid \mathbf{a}^{k\,T}]^T, 0, \mathbf{c}^k\}_{k=1,\ldots,p}$, the matrix "A0" can be obtained from the p scheduling vectors, as:

$$A0 = [\mathbf{a}^1 \mid \cdots \mid \mathbf{a}^p] \tag{B1}$$

In Fig. B.1, the scalar "P" represents the number of training triads, p, and "c" represents the vector of known gradients, ς, defined in eq. 60. The scalar "n" is the dimension of the **x** input.

In Fig. B.1 the constants A0, P, c, and n are assumed known from the training set. All of the remaining variables are defined locally as indicated by the lines of code. The objective of the program is to compute the quantities "w_a", "v", and "w_x", representing the neural weight vectors defined in Section 3.1.1, i.e., $\mathbf{w_a}$, $\mathbf{v}$, and $\mathbf{w_x}$, respectively. The algorithm is implemented for a scalar-output sigmoidal network with one input bias, one output bias, and two scheduling variables. A user-defined subroutine "sgm" (not shown here) evaluates the sigmoidal function of its input component wise, according to the definition of the sigmoid, $\sigma(\bullet)$, introduced in Section 3.1.1. A similarly defined function "dsgm" also can be produced based on the definition of $\sigma'(\bullet)$ (Section 3.1.1).

```
% GIVEN: A0, P, c, n

% SOLVE INPUT-TO-NODE EQUATIONS
% Create the A-matrix:
A = zeros(P^2, 3*P);
for i = 1:P,
  A1 = [A0(i,1)*ones(P,1), A0(i,2)*ones(P,1), ones(P,1)];
  AP = spdiags(A1,[0 P 2*P], P, 3*P);
  A((i-1)*P+1:i*P, 1:3*P) = AP;
end

% Create the n-vector:
N = randn(P, P);
N = N - diag(diag(N));
n_vec = reshape(N, P^2,1);
w_a = pinv(A)*n_vec;
n_vec = A*w_a;

f_n = 10/(max(abs(n_vec)));
n_vec = n_vec*f_n;

% Compute the matrices N and S:
N = reshape(n_vec, P, P)';
S = sgm(N);

% Compute vector of input biases and a-input weights:
w_a = w_a*f_n;

% SOLVE OUTPUT EQUATIONS
% Create the b-vector from the output bias:
bias = 10*rand(1,1);
b = -bias*ones(P,1);

% Compute output weights:
v = inv(S)*b;

% SOLVE GRADIENT EQUATIONS
% Create the X-matrix:
X = zeros(n*P, n*P);
for i = 1:P,
  for j = 1:n,
    X((j+((i-1)*n)), (P*(j-1)+1):(P*j)) = (v.*(dsgm(N(:,i))))';
  end
end

% Compute the x-input weights:
w_x = inv(X)*c;
```

Figure B.1. Sample code for the exact gradient-based algebraic training algorithm.

A sample code is provided for the exact input/output-based solution algorithm (Section 3.1.2) in Fig. B.2. Here, the matrix "Y" and the vector "u" (corresponding to **Y**

and **u**, in Section 3.1.2) are provided to the program, based on the training set $\{\mathbf{y}^k, u^k\}_{k=1,\ldots,p}$. The number of inputs "q" and the number of training pairs "P", corresponding to *q* and *p*, also constitute inputs to the routine. The program computes the neural network weights **W** and **v** represented by the variables "W" and "v" in Fig. B.2. The output bias, *b*, can be set equal to zero, as shown by the weight equations in Section 3.1.2.

```
% GIVEN: Y, q, u

% Create matrix of input weights:
f = 3;
W = 3*randn(P,q);

% Compute input bias:
d = -diag(Y*W');

% Compute input-to-node-value (N) matrix:
N = Y*W' + ones(P,P)*diag(d);

% SOLVE OUTPUT EQUATIONS:
S = sgm(N);
v = inv(S)*u;
```

Figure B.2. Sample code for the exact input/output-based algebraic training algorithm.

When gradient information is available in the training set $\{\mathbf{y}^k, u^k, \mathbf{c}^k\}_{k=1,\ldots,p}$, it can be used to improve the generalization properties of the neural network trained by the sample code in Fig. B.2, as explained in Section 3.1.4. According to the previous routine, the number of nodes, *s*, in the network equals *p*, or "P" in the program's notation. At each step, indexed by "i", the code in Fig. B.3 compares the gradient of the neural network to the known derivatives in one training triad. Since $s = p$, the index "i" can be considered equivalent to *k* (as explained in Section 3.1.4); therefore, the algorithm always terminates in *p* steps.

```
% GIVEN: Y, u, C, W, d, v, tol_max

% Recreate N:
N = Y*W' + ones(P,P)*diag(d);

for i = 1:P,

  wi_old = W(i,:);

  % Compare neural network gradients with "ideal" ones:

  c_i = C(:,i);
  c_nn = (v'.*dsgm(N(i,:)))*W;

  if max(abs(c_i-c_nn)) > tol_max,                    %Change input weights wi)

    wi_new = W(i,:) +  (c_i-c_nn)/v(i)/dsgm(N(i,i));

    if max(abs(wi_new)) > 50,                         %Impose a "safe" upper bound
      wi_new = wi_old;                                %Retain old weights
    end

  else,

    wi_new = wi_old;                                  %Retain old weights

  end

  % Update neural network weights:
  W(i,:) = wi_new;
  d(i) = -Y(i,:)*W(i,:)';
  N(:,i) = d(i) + Y*W(i,:)';

  % Recompute output weights:
  S = sgm(N);
  v = inv(S)*u;

end
```

Figure B.3. Sample code for the approximate general solution training algorithm.

If the user-supplied gradient tolerance "tol_max" is exceeded by the gradient error at the i^{th}-step, the input weights corresponding to the i^{th}-node, $\mathbf{w}_i$, (or "wi" in the program) are modified. Consequently, the new output weights **v**, or "v", must be re-computed from the output equations. The program in Fig. B.3 utilizes a matrix of known gradients, that can be obtained from the training set as,

$$\mathrm{C} = \left[\mathbf{c}^1 \,|\cdots|\, \mathbf{c}^p\right] \tag{B2}$$

as well as the information used in Fig. B.2. The neural network weights to be refined, "W", "d" and "v", also must be provided to this approximate algorithm.

Figures B.4-B.5 provide sample code for the resilient-backpropagation (RPROP) algorithm presented in Section 3.2, and implemented by the adaptive critic architecture (Section 5.1.3). For convenience, the program is illustrated in two parts: (a) and (b). The inputs consist of the neural weights, "W", "V", and "d", and of the network's input and target, "p" and "nn_target", respectively. The algorithm modifies the weights "W" and "V", that are rearranged in the vector "w" (or **w**), as explained in Sections 3.2 and 5.1.3. For simplicity, the program is illustrated for a network with no scheduling vector (or **a** input). Part (a), in Fig. B.4, prepares the data needed by the iterative portion of the algorithm (Fig. B.5).

As anticipated in Section 5.1.3, the scheme implements a "backtracking step" [77], a gradient-sign computation (based on eq. 217-218), and an "early-termination rule" as stopping condition. Also, the increment size Δ_ℓ (or "delta_w") initially is computed through the proportional rule in eq. 91. However, since the RPROP training algorithm is called repeatedly by the adaptive critic architecture, under the stated circumstances (Section 5.1.3) the algorithm can store Δ_ℓ in the binary file "delta_file.mat". Figure B.4 also shows that, at the on-set of training, the modified-RPROP routine accesses the value of "delta_w" stored in the latter binary file, and that it uses eq. 91 to compute $\Delta_\ell^{(0)}$ only if the "delta_w" variable is empty. A file named "delta_file.mat" with an empty variable "delta_w" always should be available to this program.

```
% GIVEN: p, W, d, V, b, nn_target

% RPROP user-defined parameters:
delta_max = 50;
delta_inc = 1.2;  delta_dec = 0.5;

% Create vector of ordered weights:
[s, n] = size(W);  [m, s] = size(V);
w = [reshape(W,s*n,1); reshape(V,m*s,1)];

% Compute old-NN gradient-sign and output-error:
n_vec = W*p + d;
nn_output = V*sgm(n_vec) + b;

e_w = nn_target - nn_output;                                  %Output error
e_v = V'*e_w;
nn_mse0 = mse(e_w);

% Gradient signs:
sign_gW = sign(-e_v)*sign(p');
sign_gV = sign(-e_w)*sign(n_vec');
sign_gw = [reshape(sign_gW,s*n,1); reshape(sign_gV,m*s,1)];   %Gradient sign

% Initialize loop variables:
epochs = 0;
sign_gw_old = sign_gw;
save_delta_w = 1;  stop_condition = 0;                        %Loop flags
dw_old = zeros(length(w),1);                                  %Weight increments
f_mse = 10/100;                                               %Desired mse-change

% User-specified performance parameters:
epochs_max=5;                                                 %Maximum epochs
mse_perf_final=1e-5;                                          %Ideal performance
f2_delta_w=1e-5;

% Initialize weight-increment size:
load delta_file                                               %With: delta_w
if isempty(delta_x) == 1,
  f1_delta_w = nn_mse0*f2_delta_w;
  delta_w = f1_delta_w.*abs(w) + 1e-20;                       %Proportional rule
end
```

Figure B.4. Part (a) of a sample program based on the modified-resilient-backpropagation (RPROP) on-line training algorithm.

```
% GIVEN: p, W, d, V, b, nn_target
% GIVEN: delta_max, delta_inc, delta_dec, m, s, n, dw_old, from Part (a)
% GIVEN: stop_condition, epochs, sign_gw, sign_gw_old, from Part (a)

% Resilient Backpropagation (RPROP) Algorithm:
while stop_condition == 0,
  epochs=epochs+1;

  sign_ggw= sign_gw_old.*sign_gw;
  delta_w = ((sign_ggw>0)*delta_inc + (sign_ggw<0)*delta_dec + (sign_ggw==0)).*delta_w;

  % Bound increment size:
  delta_w = min(delta_w, delta_max);

  % "Backtracking" step:
  dw = (-sign_gw.*delta_w).*(sign_ggw>=0) + (-dw_old).*(sign_ggw<0);
  sign_gw(find(sign_ggw<0)) = 0;

  % Update old variables for next epoch:
  w_old = w;  dw_old = dw;  sign_gw_old = sign_gw;

  % Update variables for next epoch:
  w = w + dw;

  % Compute new NN gradient-sign and output-error:
  W = reshape(w(1:s*n), s, n);  V = reshape(w(s*n+1:end), m, s);
  n_vec = W*p + d;  nn_output = V*sgm(n_vec) +b;

  e_w = nn_target - nn_output;                                    %Output error
  nn_mse=mse(e_w);  e_v = V'*e_w;

  sign_gW = sign(-e_v)*sign(p');
  sign_gV = sign(-e_w)*sign(n_vec');
  sign_gw = [reshape(sign_gW,s*n,1); reshape(sign_gV,m*s,1)];     %Gradient sign

  %Check stopping condition:
  if epochs > epochs_max & nn_mse < (1-f_mse)*nn_mse0,  stop_condition=1;
  elseif nn_mse <= mse_perf_final,  stop_condition=1;
  end

  % Adaptive initial-increment-size rule:
  if nn_mse < (1-f_mse)*nn_mse0 & save_delta_w == 1 & epochs > 3,
    save delta_file.mat  delta_w                                  %Store
    save_delta_w = 0;                                             %Reset flag
  elseif nn_mse < (1-f_mse)*nn_mse0 & save_delta_w == 1 & epochs <= 3,
    save_delta_w=0;                                               %Don't store
  end

end
```

Figure B.5. Part (b) of a sample program based on the modified-resilient-backpropagation (RPROP) on-line training algorithm.

Appendix C: Proofs

Algebraic Network Operations: Output Combination

This section of the appendix shows that the algebraic operation that combines two nonlinear neural networks with the same input and different outputs, illustrated in Fig. 43, preserves performance. The proof demonstrates that, if the original networks each match the training sets $\{\mathbf{x}^k, \mathbf{u}_1{}^k, \mathbf{C}_1{}^k\}_{k = 1, \ldots, p}$ and $\{\mathbf{x}^k, \mathbf{u}_2{}^k, \mathbf{C}_2{}^k\}_{k = 1, \ldots, p}$, then a final network that matches the full set $\{\mathbf{x}^k, \mathbf{u}^k, \mathbf{C}^k\}_{k = 1, \ldots, p}$ can be obtained by the simple algebraic operations described in Section 5.1.1. The matrix of known gradients, $\mathbf{C}^k$, is defined as in Section 5.3, and the remaining quantities are defined consistently with Section 5.1.1. From Fig. 43 it can be deduced that the following relationships hold for the known outputs,

$$\mathbf{u}^k = \begin{bmatrix} \mathbf{u}_1^k \\ \mathbf{u}_2^k \end{bmatrix} \tag{C1}$$

and for the known gradients:

$$\mathbf{C}^k = \begin{bmatrix} \mathbf{C}_1^k & \mathbf{C}_2^k \end{bmatrix} \tag{C2}$$

Network weight equations can be used to show that if the full training set is matched by the final network (with weights **W**, **d**, **V**, and **b**), then the parameters of the two original networks ($\mathbf{W}_1$, $\mathbf{d}_1$, $\mathbf{V}_1$, $\mathbf{b}_1$, and $\mathbf{W}_2$, $\mathbf{d}_2$, $\mathbf{V}_2$, $\mathbf{b}_2$) also satisfy the weight equations corresponding to the original training sets. If this is the case, then the opposite argument also must apply, and the final network can be constructed from the original networks' parameters through the stated algebraic operations. A single, generic training triad indexed by *k* can be considered without loss of generality. The output weight equations for the final network can be written as the vector-output equivalent of eq. 44, i.e.,

$$\mathbf{u}^k = \mathbf{V}\boldsymbol{\sigma}[\mathbf{W}\mathbf{x}^k + \mathbf{d}] + \mathbf{b} \tag{C3}$$

Using the parameters obtained by the algebraic operations in Section 5.1.1, as well as eq. C1, the final output weight equations (eq. C3) can be reformulated as:

$$\begin{aligned}
\begin{bmatrix} \mathbf{u}_1^k \\ \mathbf{u}_2^k \end{bmatrix} &= \begin{bmatrix} \mathbf{V}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{V}_2 \end{bmatrix} \boldsymbol{\sigma}\left\{ \begin{bmatrix} \mathbf{W}_1 \\ \mathbf{W}_2 \end{bmatrix} \mathbf{x}^k + \begin{bmatrix} \mathbf{d}_1 \\ \mathbf{d}_2 \end{bmatrix} \right\} + \begin{bmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \end{bmatrix} \\
&= \begin{bmatrix} \mathbf{V}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{V}_2 \end{bmatrix} \boldsymbol{\sigma}\left\{ \begin{bmatrix} \mathbf{W}_1\mathbf{x}^k + \mathbf{d}_1 \\ \mathbf{W}_2\mathbf{x}^k + \mathbf{d}_2 \end{bmatrix} \right\} + \begin{bmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \end{bmatrix} \\
&= \begin{bmatrix} \mathbf{V}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{V}_2 \end{bmatrix} \begin{bmatrix} \boldsymbol{\sigma}\left(\mathbf{W}_1\mathbf{x}^k + \mathbf{d}_1\right) \\ \boldsymbol{\sigma}\left(\mathbf{W}_2\mathbf{x}^k + \mathbf{d}_2\right) \end{bmatrix} + \begin{bmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \end{bmatrix} \\
&= \begin{bmatrix} \mathbf{V}_1\boldsymbol{\sigma}\left(\mathbf{W}_1\mathbf{x}^k + \mathbf{d}_1\right) + \mathbf{b}_1 \\ \mathbf{V}_2\boldsymbol{\sigma}\left(\mathbf{W}_2\mathbf{x}^k + \mathbf{d}_2\right) + \mathbf{b}_2 \end{bmatrix}
\end{aligned} \tag{C4}$$

The above equation is found to be equivalent to the two output weight equations of the original networks. Thus, it can be concluded that this network operation preserves output information.

The gradient weight equations of the full network can be written as the vector-output equivalent of eq. 49, for the case in which derivatives are known with respect to all of the inputs ($e = q$), that is:

$$\mathbf{C}^{\kappa} = \mathbf{W}^T\left\{\text{diag}\left[\boldsymbol{\sigma}'\left(\mathbf{n}^k\right)\right]\mathbf{V}^T\right\} \tag{C5}$$

The vector of input-to-node values can be computed from the k^{th}-training triad, as:

$$\mathbf{n}^k = \mathbf{W}\mathbf{x}^k + \mathbf{d} \tag{C6}$$

Just as in eq. C4, it can be shown that the following holds,

$$\mathbf{n}^k = \begin{bmatrix} \mathbf{W}_1\mathbf{x}^k + \mathbf{d}_1 \\ \mathbf{W}_2\mathbf{x}^k + \mathbf{d}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{n}_1^k \\ \mathbf{n}_2^k \end{bmatrix} \tag{C7}$$

Then, the diagonal matrix in eq. C5 can be partitioned as,

$$\text{diag}\left[\boldsymbol{\sigma}'\left(\mathbf{n}^k\right)\right] = \begin{bmatrix} \text{diag}\left[\boldsymbol{\sigma}'\left(\mathbf{n}_1^k\right)\right] & \mathbf{0} \\ \mathbf{0} & \text{diag}\left[\boldsymbol{\sigma}'\left(\mathbf{n}_2^k\right)\right] \end{bmatrix} \tag{C8}$$

such that, by using the final network's parameters and eq. C2, the gradient weight equations (eq. C5) can be reformulated as:

$$\begin{aligned} \left[\mathbf{C}_1^k \mid \mathbf{C}_2^k\right] &= \mathbf{W}^T \,\text{diag}\left[\boldsymbol{\sigma}'\left(\mathbf{n}^k\right)\right]\mathbf{V}^T \\ &= \begin{bmatrix} \mathbf{W}_1^T & \mathbf{W}_2^T \end{bmatrix} \begin{bmatrix} \text{diag}\left[\boldsymbol{\sigma}'\left(\mathbf{n}_1^k\right)\right] & \mathbf{0} \\ \mathbf{0} & \text{diag}\left[\boldsymbol{\sigma}'\left(\mathbf{n}_2^k\right)\right] \end{bmatrix} \begin{bmatrix} \mathbf{V}_1^T & \mathbf{0} \\ \mathbf{0} & \mathbf{V}_2^T \end{bmatrix} \\ &= \left[\mathbf{W}_1^T \,\text{diag}\left[\boldsymbol{\sigma}'\left(\mathbf{n}_1^k\right)\right]\mathbf{V}_1^T \mid \mathbf{W}_2^T \,\text{diag}\left[\boldsymbol{\sigma}'\left(\mathbf{n}_2^k\right)\right]\mathbf{V}_2^T\right] \end{aligned} \tag{C9}$$

The above equation is expressed as two independent equations that correspond to the original networks' gradient weight equations, demonstrating that gradient information also is preserved by this operation. Similarly, it can be verified that the remaining network operations (Figs. 9, 13, 44, and 45) also preserve performance.

Appendix D: Description of Trim Data Sets

The aircraft trim map is obtained by sampling the entire operating range of the airplane, $OR = \{V, H, \gamma, \mu, \beta\}$, and by determining whether a trim solution exists at each operating point $(V, H, \gamma, \mu, \beta)$. Section 4.4 describes the philosophy behind the sampling process that also determines the boundaries of the multi-dimensional envelope *OR*. Ultimately, the trim map U_c is defined by the collection of trim control settings corresponding to the operating points for which trim solutions exist (eq. 168). This appendix describes how the space *OR* is sampled numerically, and how the corresponding trim solutions are stored once they are found. Subsequently, the trim data is reduced to obtain the training and validation sets used in Section 4.4 to train and test the forward neural network.

Initially, the outer bounds of $\{V, H\}$ are assumed to be those of the steady-level flight envelope, Fig. 15, with $\gamma = \mu = \beta = 0$. Then, different combinations of γ, μ, and β are explored within the prescribed ranges (eq. 169), redefining the *V*-*H* limits when trim solutions do not exist everywhere inside the initial boundaries. Several (γ, μ, β) combinations are obtained by sampling each of these variables. The path angle, γ, is sampled between – 6 and + 6 deg, with constant intervals $\Delta\gamma = 1$ deg. The bank angle, μ, and the sideslip, β, are sampled as shown by the Table D.1. To each combination of values (γ, μ, β) there corresponds a two-dimensional envelope $\{V, H\}$. Each of the $\{V, H\}$ envelopes is explored by sampling the altitude between 0 and 15, 000 m, at intervals of ΔH = 1, 000 m. For each altitude, the corresponding velocity range is sampled at intervals of ΔV = 5 m/s.

β (deg)	μ (deg)								
5	–20	–15	–10	–5	0	5	10	15	20
4	–21	–17	–13	–7	–2	3	8	14	19
3	–20	–16	–11	–6	–1	4	9	16	20
2	–18	–12	–8	–4	0	4	9	13	18
1	–21	–17	–13	–7	–2	3	8	14	19
0	–20	–15	–10	–5	0	5	10	15	20
–1	–19	–14	–9	–3	1	6	12	17	21
–2	–20	–16	–11	–6	–1	4	9	16	20
–3	–18	–12	–8	–4	0	4	9	13	18
–4	–21	–17	–13	–7	–2	3	8	14	19
–5	–20	–15	–10	–5	0	5	10	15	20

Table D.1. Sampled values of bank angle, μ, and the sideslip, β, used to compute the aircraft trim map, U_c.

For each combination (γ, μ, β) considered above, there exists an envelope $\{V, H\}$ of sampled V and H values and corresponding trim control settings. This suggests the following approach to storing relevant trim data in MATLAB. The full envelope of the aircraft is stored in one three-dimensional cell array, denominated "ENV", that contains nested cells with two-dimensional envelopes $\{V, H\}$. Each nested cell corresponds to one combination (γ, μ, β), with γ varying along the first dimension of the cell array, μ varying along its second dimension, and β varying along its third dimension, as shown in Fig. D.1. Then, the actual values of γ, μ, and β can be stored in three matrices, e.g., "G", "M", and "B". In each one-dimensional nested cell, the first element contains a vector of altitudes (with ΔH = 1, 000 m), up to the appropriate ceiling; the remaining elements

contain corresponding vectors of velocities sampled between the minimum and the maximum speed, with ΔV = 5 m/s (Fig. D.1).

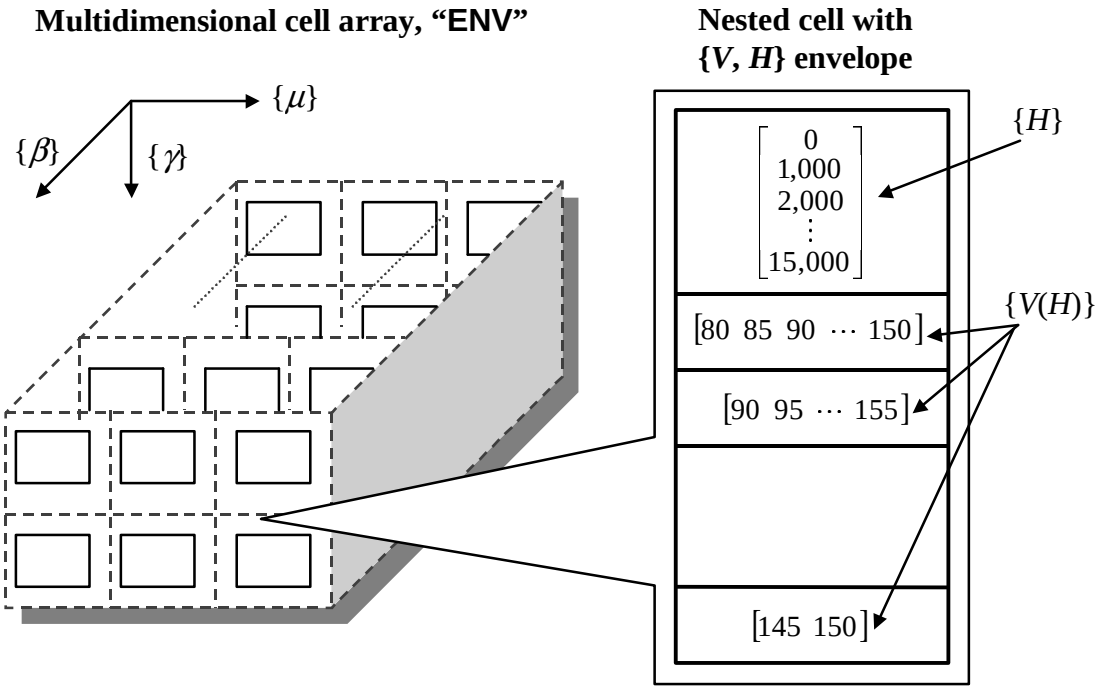

Figure D.1. Cell array structure "ENV" used to store the aircraft multidimensional flight envelope, *OR* = {*V*, *H*, γ, μ, β}.

Another multidimensional cell array, "ENVPAR", is used to store the trim control settings over the full aircraft flight envelope, as shown in Fig. D.2. This second array has the same outer correspondence to γ, μ, and β values. However, in this case each nested cell is two dimensional and contains a vectors of control settings, "TrimPar" (or $\mathbf{u}_c$), obtained for the corresponding values of *H* and *V*, stored in "ENV". *H* varies along the first dimension (column-wise) and *V* varies along the second dimension (row-wise). The cell dimensions are ordered consistently with the MATLAB convention for multi-dimensional arrays. Since the number of *V* values may be different at every altitude, *H*,

some of the vectors in the nested cells will be empty, as illustrated by the empty brackets, [ ], in Fig. D.2.

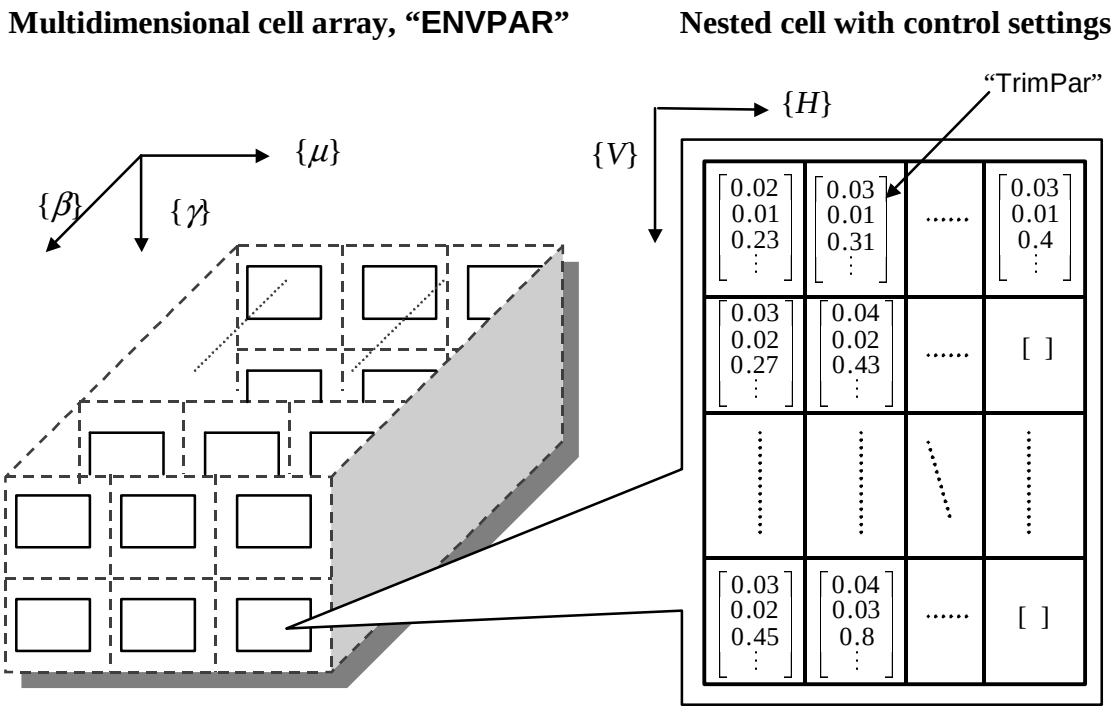

Figure D.2. Cell array structure "ENVPAR" used to store the aircraft trim map, U_c, i.e., the trim control settings, "TrimPar", corresponding to the multidimensional flight envelope, $OR = \{V, H, \gamma, \mu, \beta\}$.

The training set for the forward neural network, $\mathbf{NN}_F$, (eq. 170) is obtained from the above data by reducing it to 2, 696 operating points or, equivalently, 2, 696 (V, H, γ, μ, β) combinations. This is achieved by first reducing the number of (μ, β) combinations to the following subset:

$$\{\mu, \quad \beta\} = \left\{ \begin{bmatrix} -4 & -1 & 1 & 4 \\ -5 & -2 & 3 & 5 \\ -5 & -3 & 2 & 5 \\ -4 & 0 & 0 & 4 \\ -5 & -1 & 1 & 5 \end{bmatrix}, \begin{bmatrix} -21 & -14 & -21 & -21 \\ -15 & -6 & -1 & -15 \\ 0 & 4 & 9 & 0 \\ 14 & 0 & 15 & 8 \\ 20 & 21 & 19 & 20 \end{bmatrix} \right\} (\text{deg}) \qquad \text{(D1)}$$

For each of the (μ, β) combinations above, four to five values of γ are selected randomly within the chosen range (eq. 169). Then, for each of the resulting (γ, μ, β) combinations, the altitude values are chosen in ΔH_{train} = 2, 000 m increments, alternatively between 0 m or 1, 000 m and the applicable ceiling. For each altitude value, velocity values are selected using ΔV_{train} = 30 m/s, and retaining the minimum and maximum velocities (i.e., the envelope boundaries). This approach samples the space *OR* uniformly, while minimizing repetition in the final data set (eq. 170).

Two validation sets also are created from the full-envelope trim data. The first set is obtained by using all 1, 287 (γ, μ, β) combinations, and by selecting altitude and velocity values with the same criteria described above. Since, in general, the envelope {*V*, *H*} associated with one combination (γ, μ, β) differs from that associated with a different (γ, μ, β) combination, these sampling criteria diversify the data set with respect to *V* and *H*, as well. This validation set contains 39, 764 operating points. The second validation set contains 2, 629 operating points and is obtained by randomly picking 87 (γ, μ, β) combinations from the original possibilities, and using the same selection criteria for *V* and *H*.

The values of a variable, say μ, are said to be picked randomly when they are selected by randomly choosing among a selected number of index permutations. For example, Table D.1 shows that to different values of β there correspond different sampled sets of μ. However, all of these μ-sets contain the same number of elements, i.e., 9, (with more or less constant spacing to provide for uniform sampling). Thus, the elements in a μ-set always can be indexed by the vector [1:9] (in MATLAB notation). Here, two vectors of indices are used to define two μ-subsets: [1:2:9] and [2:2:8]. Then, random values of μ

are chosen by randomly picking between these two vectors. This approach guarantees uniform sampling, while the set is being reduced in a random fashion to minimize repetition.

Appendix E: Flight Control Software Architecture

The dual-heuristic-programming (DHP) adaptive critic design (ACD) described in Chapter 5 is implemented using a modular software comprised of user-defined MATLAB functions that take advantage of this high-level language's capabilities. The proposed modular structure has not been optimized for computational efficiency. Instead, it is designed to permit implementation changes to be performed quickly and reliably. Every function corresponds to a particular mathematical entity in the DHP architecture. An overview of the software is provided in this appendix, and the details of each function are omitted for simplicity. It is assumed that the action and critic neural parameters are initialized (according to Chapter 4 and Section 5.1.1) and stored in a MATLAB cell array, referred to as "nn_cell". The initialized cell array is stored in a binary file called "weightsA.mat" for the action network ($\mathbf{NN}_A$), and in a file called "weightsC.mat" for the critic network ($\mathbf{NN}_C$). Using the same name "nn_cell" for both the action and the critic allows the subroutines to be applicable to either networks. During every time interval, $\Delta t = t_{k+1} - t_k$, the neural parameters are modified by the DHP architecture and, as soon as they are updated to "nn_cell"(t_{k+1}), they are stored in the corresponding binary file, replacing the previous parameters "nn_cell"(t_k).

The main file, "ACDpinn.m", is used to specify a command-input time history, $\mathbf{U}_c$. However, the command-input becomes known to the DHP architecture only the present time, t_k, comes about in the simulation. The initial conditions, $\mathbf{x}_0$ and $\mathbf{u}_0$, and the time span ($t_f - t_0$) are prescribed in this file that computes the corresponding aircraft state and control histories, $\mathbf{X}$ and $\mathbf{U}$. The sampled-time history of a vector is stored column-wise in a matrix, as suggested by the MATLAB convention for ordinary differential equations.

The input/output structure of the main file and of the subroutines it implements are sketched in Fig. E.1. The global variables $\mathbf{u}_c(t_k)$, $\mathbf{a}(t_k)$, $\mathbf{x}_0$, and $\mathbf{u}_0$ can be obtained by all functions without being included in their inputs. The notation used for the input and output variables corresponds to that introduced in Chapters 1 through 5, and summarized in Appendix A. A dashed arrow indicates the function is accessing the binary file as shown, in the direction illustrated (i.e., either to store "↓" or load "↑" its contents).

The function "ClEoM.m" represents the aircraft closed-loop equations of motion. It produces the time-derivative of the augmented state, $\mathbf{x}_a(t_k)$, computing the control $\tilde{\mathbf{u}}(t_k)$ by means of the action network, with the parameters in "weightsA.mat". The function "OlEoM.m" is similarly defined, except it represents the open-loop equations of motion, therefore it also takes the control as input. Both functions simulate an ordinary differential equation and are referred to as "odefiles". A function denominated "RKstep.m" can be used to compute the Runga-Kutta [90] integration step, $\Delta\mathbf{y}(t_k)$, for a generic "odefile[$\mathbf{y}(t_k)$, t_k]", with $\mathbf{y}(t_k)$ as dependent variable and t_k as independent variable. Then, the ordinary differential equation simulated by the "odefile" can be integrated simply by computing $\mathbf{y}(t_{k+1}) = \mathbf{y}(t_k) + \Delta\mathbf{y}(t_k)$ at every interval, Δt, over the desired time span, e.g., $(t_f - t_0)$. Both "ClEoM.m" and "OlEoM.m" access the aircraft simulation described in Chapter 4, referred to as "FLIGHT.m" [64]. This program has been customized to output the aircraft parameters, $\mathbf{p}_m(t_k)$, based on the present aircraft state, $\mathbf{x}(t_k)$, and control, $\mathbf{u}(t_k)$.

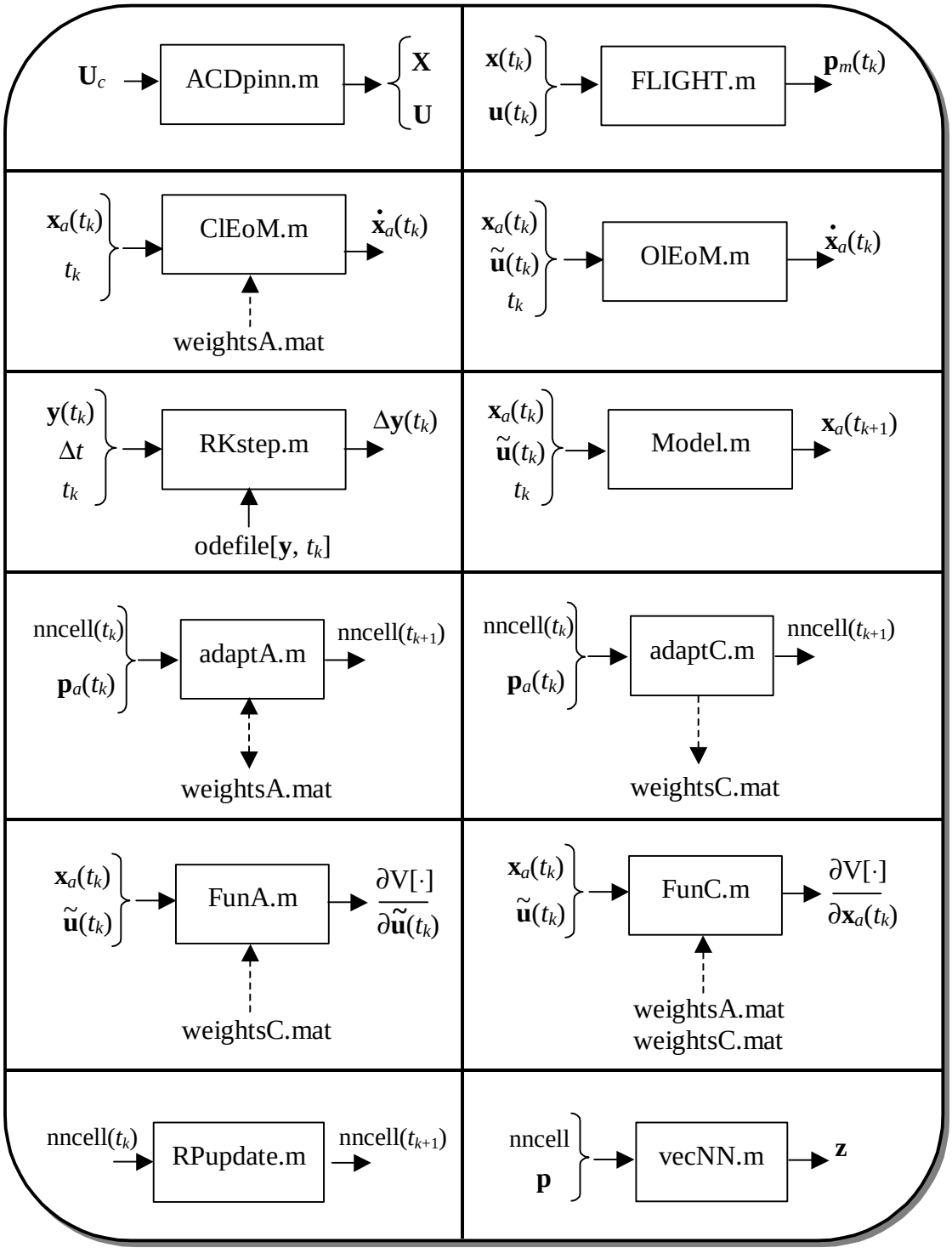

Figure E.1. Input/output structure of the user-defined functions used in the dual-heuristic-programming adaptive-critic software implementation.

The aircraft model (eq. 209) used by the DHP architecture is implemented in the function "Model.m". Given the state and control, $\mathbf{x}_a(t_k)$ and $\tilde{\mathbf{u}}(t_k)$, this function predicts the state for next time interval, $\mathbf{x}_a(t_{k+1})$, based on "OlEoM.m" and "RKstep.m". The model function also is called by the MATLAB built-in function "numjac" to predict the system transition matrices, as explained in Section 5.1.2 (eq. 187 and 188). The module for the action network adaptation, "adaptA.m", and the module for the critic network adaptation, "adaptC.m", are called by the main program once every time interval, in this order. These functions implement the flowcharts in Figures 51 and 52, respectively. The action adaptation module solves the optimality condition (eq. 36) -- represented by the function "FunA.m" -- using the MATLAB built-in function "fsolve". Then, it calls the subroutine "RPupdate.m" to update the action network parameters, by means of the resilient-backpropagation algorithm. The code in Figs. B.4-B.5 can be used to create the "RPupdate.m" function, provided it is customized to take the neural weights $\mathbf{w}(t_k)$ from the cell array "nn_cell"(t_k) and to return the updated weights $\mathbf{w}(t_{k+1})$ in the same structure "nn_cell"(t_{k+1}). Once the routine "adaptA.m" has obtained "nn_cell"(t_{k+1}) from "RPupdate.m", it returns the action parameters to "ACDpinn.m" and stores them in the binary file "weightsA.m".

Following the above action-network adaptation, the main file "ACDpinn.m" calls the module for the critic network adaptation, "adaptC.m". This function computes the critic network target by calling "FunC.m", which implements the criterion in eq. 188. Given this target ("nn_target") and the old critic parameters, "nn_cell(t_k)", "adaptC.m" calls "RPupdate.m" to compute the new parameters "nn_cell"(t_{k+1}). This updated critic network information is stored in the binary file "weightsC.m" for later use. A function

"vecNN.m" that simulates a vector-output sigmoidal neural network is used by several of the subroutines in Table D.1. It computes the network output **z** based on the available input, **p**, and network information, "nn_cell". Based on the inputs provided, "vecNN.m" plays the role of either the action or the critic network, at any moment in time t_k.

During each of the DHP sequenced events sketched in Fig. 41, the functions in Table D.1 exchange input/output information as illustrated in Fig. E.2. This diagram illustrates the implementation of the control action, Fig. E.2.(a), the action adaptation, Fig. E.2(b), and the critic adaptation, Fig. E.2(c), during the time interval Δt. The modular software architecture allows the user to perform quick implementation changes and troubleshooting. Every function has a distinct role within the adaptive critic architecture, and corresponds to a well defined mathematical entity. For example, the on-line training routine can be changed virtually instantaneously by modifying or substituting the function "RPupdate.m" alone. Another example involves the "Model.m" routine. Suppose on-line identification is to be carried out by a model neural network, as suggested by Section 6.3. Then, "Model.m" can be substituted by a function that implements "vecNN.m" to simulate the model network and that utilizes "RPupdate.m" (as well as other functions in Fig. E.1) to adapt the model parameters.

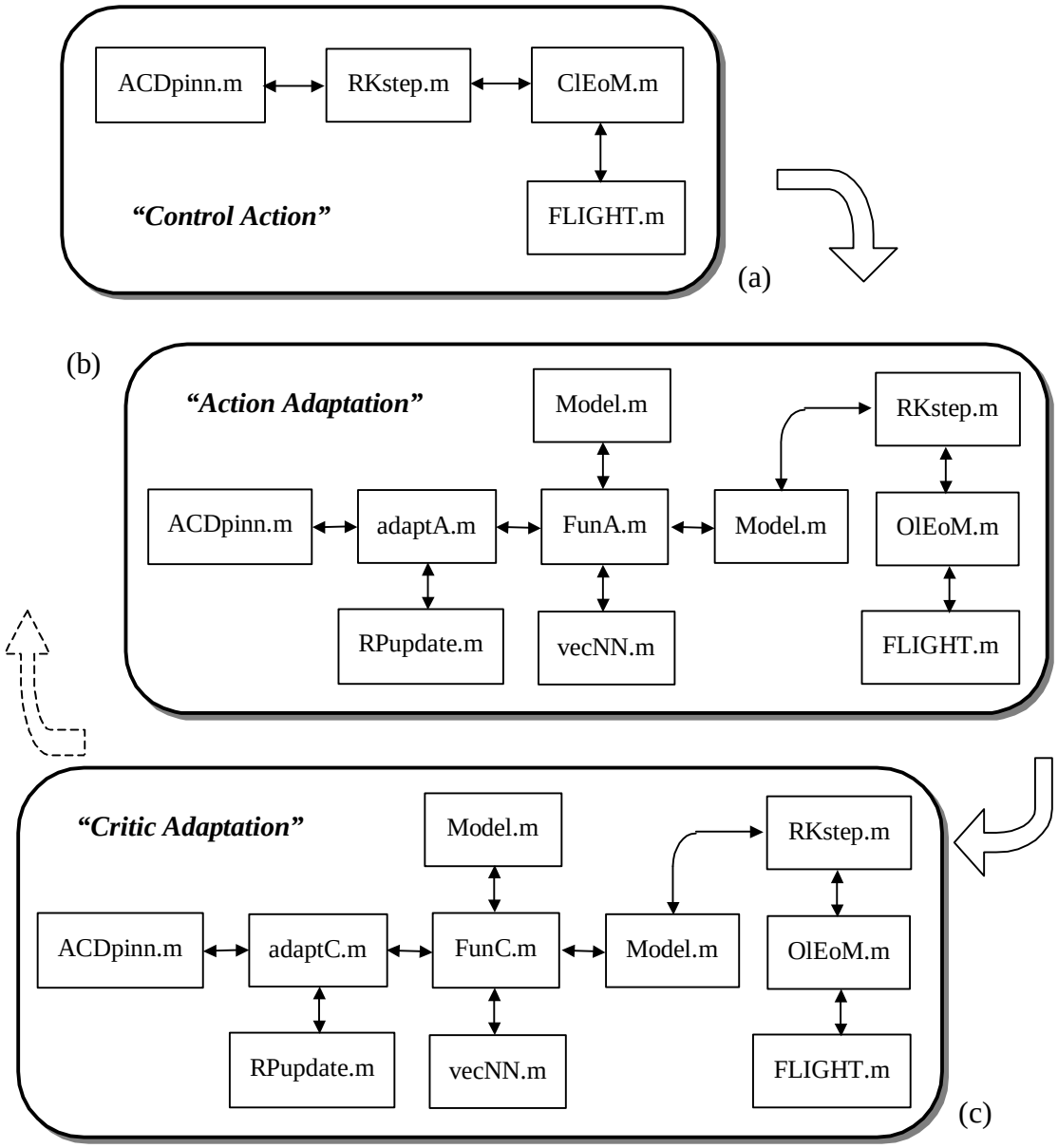

Figure E.2. Sequence of events taking place during the time interval $\Delta t = t_{k+1} - t_k$, in the dual-heuristic-programming adaptive-critic software architecture. The arrows indicate communication between functions, whose inputs and outputs are described in Fig. E.1.

Appendix F: Aircraft Model

The equations of motion and state elements that are used in the aircraft simulation (eq. 1) are reviewed in this Appendix. The simulation represents a business-type aircraft with two turbojet engines, a gross cruising weight of 4, 536 kg, and a nominal cruising Mach number of 0.79. The maximum available thrust is 26, 423 N at sea level, and 11, 735 N at 10, 000 m. The service and performance ceilings (calculated in [64]) are 15, 315 m and 15, 275 m, respectively. For a full explanation of the physical and performance characteristics modeled by the business twin-jet simulation, the reader should refer to [64]. The model estimates low-angle-of-attack Mach effects, power effects, and moments and products of inertia by using available full-scale wind tunnel data and physical characteristics, according to the methods described in [64]. The state accelerations, denoted by X_b, Y_b, Z_b, L_b, M_b, and N_b, are a function of the available thrust, and of the aerodynamic force and moment coefficients produced by the controls for the present aircraft state and wind field.

The nonlinear equations of motion are formulated with respect to the aircraft body-axis velocities and angular rates, and with respect to its position relative to an inertial frame of reference (Section 4.4), as:

$$\dot{u} = \mathrm{X}_b(\bullet) + \mathrm{g}_{b_x}(\bullet) + rv - qw \tag{F1}$$

$$\dot{v} = \mathrm{Y}_b(\bullet) + \mathrm{g}_{b_y}(\bullet) - ru + pw \tag{F2}$$

$$\dot{w} = \mathrm{Z}_b(\bullet) + \mathrm{g}_{b_z}(\bullet) + qu - pv \tag{F3}$$

$$\begin{aligned} \dot{x}_r = {} & u\cos\theta\cos\psi + v(\sin\phi\sin\theta\cos\psi - \cos\phi\sin\psi) \\ & + w(\cos\phi\sin\theta\cos\psi - \sin\phi\sin\psi) \end{aligned} \tag{F4}$$

$$\dot{y}_r = u\cos\theta\sin\psi + v(\sin\phi\sin\theta\sin\psi + \cos\phi\cos\psi) + w(\cos\phi\sin\theta\sin\psi - \sin\phi\cos\psi) \quad \text{(F5)}$$

$$\dot{z}_r = -u\sin\theta + v\sin\phi\cos\theta + w\cos\phi\cos\theta \quad \text{(F6)}$$

$$\dot{p} = \frac{q\{I_{zz}\,\mathrm{L}_b(\bullet) + I_{xz}\,\mathrm{N}_b(\bullet) - p[I_{xz}(I_{yy} - I_{xx} - I_{zz})] + r[I_{xz}^2 + I_{zz}(I_{zz} - I_{yy})]\}}{(I_{xx}I_{zz} - I_{xz}^2)} \quad \text{(F7)}$$

$$\dot{q} = \frac{[\mathrm{M}_b(\bullet) - pr(I_{xx} - I_{zz}) - I_{xz}(p^2 - r^2)]}{I_{yy}} \quad \text{(F8)}$$

$$\dot{r} = \frac{q\{I_{xz}\,\mathrm{L}_b(\bullet) + I_{xx}\,\mathrm{N}_b(\bullet) + r[I_{xz}(I_{yy} - I_{xx} - I_{zz})] + p[I_{xz}^2 + I_{xx}(I_{xx} - I_{yy})]\}}{(I_{xx}I_{zz} - I_{xz}^2)} \quad \text{(F9)}$$

$$\dot{\phi} = p + (q\sin\phi + r\cos\phi)\tan\theta \quad \text{(F10)}$$

$$\dot{\theta} = q\cos\phi - r\sin\phi \quad \text{(F11)}$$

$$\dot{\psi} = \frac{(q\sin\phi + r\cos\phi)}{\cos\theta} \quad \text{(F12)}$$

The body-axis gravity components, g_{b_x}, g_{b_y}, and g_{b_z}, and the state accelerations are a function of the state and, possibly, of the controls and wind field. The moments of inertia, I_{xx}, I_{yy}, and I_{zz}, and the product of inertia, I_{xz}, are estimated using simplified mass distributions, and are held fixed during the simulation.

The time history of the state vector $\mathbf{x} = [V\ \gamma q\ \theta r\ \beta p\ \mu]^T$ is obtained by integrating the equations of motion above. The following relations are used to compute the state elements that do not explicitly appear in the chosen formulation:

$$\begin{bmatrix} V \\ \beta \\ \gamma \end{bmatrix} = \begin{bmatrix} \sqrt{u^2 + v^2 + w^2} \\ \sin^{-1}(v/V) \\ \sin^{-1}(-w/V) \end{bmatrix} \quad \text{(F13)}$$

$$\mu = \sin^{-1}\left\{\frac{[\cos\theta\sin\phi\cos\beta + (\cos\alpha\sin\theta - \sin\alpha\cos\theta\cos\phi)\sin\beta]}{\cos\gamma}\right\} \quad \text{(F14)}$$

For small-angle maneuvers, the bank angle, μ, is very close to the roll angle, ϕ; however, while ϕ is measured about the body axis x_b, μ is measured about the velocity vector [64]. The angle of attack, α, is computed as in eq. 165. The body velocities and angular rates, and flow angles are sketched in Fig. F.1. Section 4.4 provides additional information about the body and inertial axes systems. The references [64, 87, 88] include a detailed description of all aircraft angles and coordinate transformations.

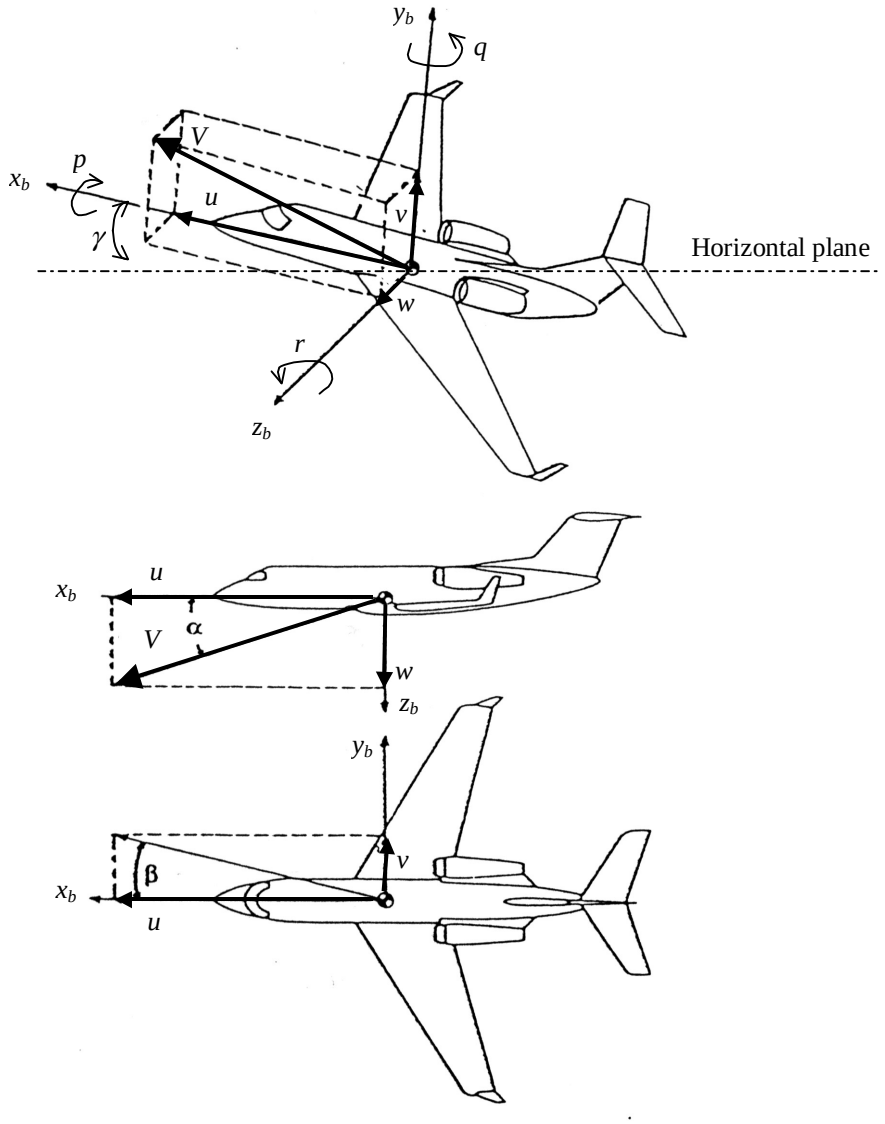

Figure F.1. Definition of path angle, angle of attack, and sideslip, adapted from [88].

References

[1] K. S. Narendra, "Adaptive Control using Neural Networks," *Neural Networks for Control*, W. T. Miller, R. S. Sutton, and P. J. Werbos, Eds., pp. 115-142, MIT Press, Cambridge, MA, 1990.

[2] L. J. Lin, "Self-improvement based on reinforcement learning, planning and teaching," *Machine Learning: Proceedings of the Eight International Workshop*, L. A. Birnbaum and G. C. Collins, Eds., pp. 323-327, Morgan Kaufmann, San Mateo, CA, 1991.

[3] A. G. Barto, "Reinforcement Learning and Adaptive Critic Methods," *Handbook of Intelligent Control*, D. A. White and D. A. Sofge, Eds., pp. 469-492, Van Nostrand Reinhold, New York, NY, 1992.

[4] D. P. Bertsekas and J. N. Tsitsiklis, *Neuro-dynamic Programming*, Athena Scientific, Belmont, MA, 1996.

[5] K. S. Narendra and K. Parthasaranthy, "Identification and control of dynamical systems using neural networks," *IEEE Trans. Neural Networks*, Vol. 1, pp. 4-27, 1990.

[6] J. Neidhoefer and K. Krishnakumar, "Nonlinear Control Using Neural Approximators with Linear Control Theory," *Proc. AIAA Guidance, Navigation and Control Conference*, New Orleans, pp. 364-372, 1997.

[7] T. Iwasa, N. Morizumi, and S. Omatu, "Temperature Control in a Batch Process by Neural Networks", *IEEE International Conference*, IEEE-0-7803-41223-8/97, V 6, no.1, pp. 2430-2433, 1997.

[8] A. J. Calise, "Neural Networks in Nonlinear Aircraft Flight Control," *IEEE Aerospace and Electronics Systems Magazine*, Vol. 11, No. 7, pp. 5-10, 1996.

[9] K.S. Narendra and O.A. Driollet, "Stochastic adaptive control using multiple models for improved performance in the presence of random disturbances," *International Journal of Adaptive Control and Signal Processing*, Vol. 15, No. 3, pp. 297-317, 2001.

[10] A. O. Esogbue, "Computational Aspects and Applications of a Branch and Bound Algorithm for Fuzzy Multistage Decision Processes," *Journal of Computers and Mathematics with Applications*, Special Issue, Vol. 21, No. 11-12, pp. 117-128, 199l.

[11] R. E. Bellman, Dynamic Programming, Princeton University Press, Princeton, NJ, 1957.

[12] D. E. Kirk, *Optimal Control Theory; an Introduction*, Prentice-Hall, Englewood Cliffs, NJ, 1970.

[13] R. Bellman, *Methods of Nonlinear Analysis: Volume II*, Academic Press, 1973.

[14] P. J. Werbos, "Building and Understanding Adaptive Systems: A Statistical/Numerical Approach for Factory Automation and Brain Research," *IEEE Trans. Syst., Man, Cybern.*, Vol. 17, No. 1, pp. 7-20, 1987

[15] D. P. Bertsekas, "Distributed Dynamic Programming," *IEEE Trans. Automatic Control*, Vol. 27, pp. 610-616, 1982.

[16] R. Howard, *Dynamic Programming and Markov Processes*, MIT Press, Cambridge, MA, 1960.

[17] P. J. Werbos, "Neurocontrol and Supervised Learning: an Overview and Evaluation," *Handbook of Intelligent Control*, D. A. White and D. A. Sofge, Eds., pp. 65-86, Van Nostrand Reinhold, New York, NY, 1992.

[18] P. J. Werbos," A Menu of Designs for Reinforcement Learning Over Time," *Neural Networks for Control*, W. T. Miller, R. S. Sutton, and P. J. Werbos, Eds., pp. 67-96, MIT Press, Cambridge, MA, 1990.

[19] P. J. Werbos, "Advanced Forecasting Methods for Global Crisis Warning and Models of Intelligence," *General Systems Yearbook*, 1997.

[20] A. Barto, R. Sutton, and C. Anderson, "Neuronlike Elements that Can Solve Difficult Learning Control Problems," IEEE Trans. Systems, Man, and Cybernetics, Vol. 3, No. 5, pp. 834-846, 1983.

[21] P. J. Werbos, "Applications of Advances in Nonlinear Sensitivity Analysis," *System Modeling and Optimization: Proceedings of the 10th IFIP Conference*, R. F. Drenick and F. Kozin, Eds., Springer-Verlag, New York, NY, 1982.

[22] C. Watkins, "Learning from Delayed Rewards," Ph.D. Thesis, Cambridge University, Cambridge, England, 1989.

[23] T. H. Wonnacott and R. Wonnacott, Introductory Statistics for Business and Economics, 2nd Ed., Wiley, New York, NY, 1977.

[24] D. Prokhorov and D. Wunsch, "Adaptive Critic Designs," *IEEE Trans. on Neural Networks*, Vol. 8, No. 5, pp. 997-1007, 1997.

[25] S. Lane and R. F. Stengel, "Flight Control Design Using Non-linear Inverse Dynamics," *Automatica*, Vol. 24, No. 4, pp. 471-483, 1988.

[26] M. G. Cox, "Practical Spline Approximation," *Lecture Notes in Mathematics 965: Topics in Numerical Analysis*, P.R. Turner, Ed., Springer Verlag, New York, NY 1982.

[27] A. Antoniadis and D. T. Pham, "Wavelets and Statistics," *Lecture Notes in Statistics 103*, Springer Verlag, New York, NY, 1995.

[28] C. K. Chui, *An Introduction to Wavelets*, Academic Press, New York, NY, 1992.

[29] T. Lyche, K. Mørken, and E. Quak, "Theory and Algorithms for Nonuniform Spline Wavelets," *Multivariate Approximation and Applications*, N. Dyn, D. Leviatan, D. Levin, and A. Pinkus, Eds., Cambridge University Press, Cambridge, UK, 2001.

[30] J. H. Friedman, "Multivariate adaptive regression splines," *The Annals of Statistics*, Vol. 19, pp. 1-141, 1991.

[31] S. Karlin, C. Micchelli, and Y. Rinott, "Multivariate splines: A probabilistic perspective," *Journal of Multivariate Analysis*, Vol. 20, pp. 69-90, 1986.

[32] C. J. Stone, "The use of polynomial splines and their tensor products in multivariate function estimation," *The Annals of Statistics*, Vol. 22, pp. 118-184, 1994.

[33] G. Cybenko, "Approximation by Superposition of a Sigmoidal Function", *Math. Contr., Signals, Syst.*, Vol. 2, pp. 359-366, 1989.

[34] K. Hornik, M. Stichcombe, and H. White, "Multilayer Feedforward Networks are Universal Approximators," *Neural Networks*, Vol. 2, No. 5, pp. 359-366, 1989.

[35] A. R. Barron, "Universal Approximation Bounds for Superposition of a Sigmoidal Function," *IEEE Transactions on Information Theory*, Vol. 39, No. 3, pp. 930-945, 1993.

[36] S. Ferrari and R. F. Stengel, "Algebraic Training of a Neural Network," *Proc. American Control Conference*, pp.1605-1610, Arlington, VA, 2001.

[37] S. Ferrari and R. F. Stengel, "Classical/Neural Synthesis of Nonlinear Control Systems," *J. Guidance, Control and Dynamics*, Vol. 25, No. 3, pp. 442-448, 2002.

[38] H. Demuth and M. Beale, "Radial Basis Networks," *Neural Network Toolbox – For Use with MATLAB*, Version 3, The MathWorks Inc., Natick, MA, pp. 6.2-6.19, 1998.

[39] K. J. Hunt, D. Sbarbaro, R. Zbikowski, and P. J. Gawthrop, "Neural Networks for Control Systems – A Survey," *Automatica*, Vol. 28, No. 6, pp. 1083-1112, 1992.

[40] D. Marquardt, "An Algorithm for Least Squares Estimation of Nonlinear Parameters," *J. Soc. Ind. Appl. Math*, pp. 431-441, 1963.

[41] M.T. Hagan and M. B. Menhaj, "Training feedforward networks with the Marquardt algorithm," *IEEE Trans. Neural Networks*, Vol. 5, No. 6, pp. 989 – 993, 1994.

[42] R. F. Stengel, J. Broussard, and P. Berry, "Digital Controllers for VTOL Aircraft," *IEEE Trans. Aerospace and Electronic Systems*, Vol. AES-14, No. 1, pp. 54-63, 1978.

[43] R. F. Stengel, J. Broussard, and P. Berry, "Digital Flight Control Design for a Tandem-Rotor Helicopter," *Automatica*, Vol. 14, No. 4, pp. 301-311, 1978.

[44] R. F. Stengel, P. Berry, and J. Broussard, "Evaluation of Digital Flight Control Design for VTOL Approach and Landing, Guidance and Control Design Considerations for Low Altitude and Terminal Area Flight," *AGARD CP-240*, October 1977.

[45] W. T. Baumann and W. J. Rugh, "Feedback Control of a Nonlinear System by Extended Linearization," *IEEE Trans. Auto. Control*, Vol. AC-31, No. 1, pp. 40-46, 1986.

[46] W. T. Baumann and W. J. Rugh, "Feedback Control of a Nonlinear System by Extended Linearization: The Multi-Input Case," *Proc. 7th Int'l Symp. on Math. Theory of Networks and Systems*, pp. 107-113, Stockholm, Sweden, 1985.

[47] J. B. Plant, Y. T. Chan, and D. A. Redmond, "A Discrete Tracking Control Law for Nonlinear Plants," *Proc. IFAC 8th Triennial World Congress – Control Science and Technology*, pp. 55-60, Kyoto, Japan, 1981.

[48] H. Nijmeijer, A. J. Van der Schaft, *Nonlinear Dynamical Control Systems*, Springer Verlaf, New York, NY, 1990.

[49] D. J. Bugajski, D. F. Enns, and M. R. Elgersma, "A Dynamic Inversion Based Control Law with Application to the High Angle of Attack Research Vehicle," *Proc. AIAA Guidance, Navigation, and Control Conf.*, pp. 20-22, 1990.

[50] S. A. Snell, D. F. Enns, and W. L. Garrard, "Nonlinear Inversion Flight Control for a Supermaneuverable Aircraft," *J. Guidance, Control and Dynamics*, Vol. 15, No. 4, pp. 976-984, 1992.

[51] J. M. Buffington, A. G. Sparks, and S. S. Banda, "Full Conventional Envelope Longitudinal Axis Flight Control with Thrust Vectoring," *Proc. American Control Conf.*, pp. 415-419, 1993.

[52] Q. Wang and R. F. Stengel, "Robust Nonlinear Control of a Hypersonic Aircraft," *J. Guidance, Control, and Dynamics*, Vol. 23, No. 4, pp. 577-585, 2000.

[53] J. S. Brinker and K. A. Wise, "Stability and Flying Qualities Robustness of a Dynamic Inversion Aircraft Control Law," *J. Guidance, Control, and Dynamics*, Vol. 19, No. 6, pp. 1270-1277, 1996.

[54] R. J. Adams and S. S. Banda, "An Integrated Approach to Flight Control Design Using Dynamic Inversion and μ-Synthesis," *Proc. American Control Conf.*, pp. 1385-1389, 1993.

[55] J. M. Buffington, R. J. Adams, and S. S. Banda, "Robust Nonlinear High Angle of Attack Control Design for a Supermaneuverable Vehicle," *Proc. of the AIAA Guidance, Navigation, and Control Conf.*, pp. 690-700, 1993.

[56] S. Golpaswami and J. K. Hedrick, "Robust Adaptive Nonlinear Control of a High Performance Aircraft," *Proc. American Control Conf.*, pp. 1279-1283, 1990.

[57] J. J. E. Slotine, "Sliding Controller Design for Nonlinear Systems," *Int. J. Control*, Vol. 40, No. 2, pp. 421-434, 1984.

[58] S. H. Lane, "Theory and Development of Adaptive Flight Control Systems Using Nonlinear Inverse Dynamics," Ph.D. Thesis, Princeton University, Princeton, NJ, 1988.

[59] K. S. Narendra, "Adaptive Control of Discrete-time Systems Using Multiple Models," *IEEE Trans. Auto. Control*, Vol. 45, No. 9, pp. 1669-1686, 2000.

[60] E. Ferreira and B. Krogh, "Switching Controllers Based on Neural Networks Estimates of Stability Regions and Controller Performance," *Lecture Notes on Computer Science, Special Issue: Hybrid Systems VI*, Springer Verlag, 1998.

[61] B. S. Kim and A. J. Calise, "Nonlinear Flight Control Using Neural Networks," *J. Guidance, Control, and Dynamics*, Vol. 20, No. 1, pp. 26-33, 1997.

[62] A. J. Calise and R. T. Rysdyk, "Nonlinear Adaptive Flight Control Using Neural Networks," *IEEE Control Systems Magazine*, pp. 14-25, December 1998.

[63] K.A. Wise, et al., "Direct Adaptive Reconfigurable Flight Control for a Tailless Advanced Fighter Aircraft," *Int. J. Robust and Nonlinear Control*, Vol. 9, pp. 999-1009, 1999.

[64] R. F. Stengel, *Flight Dynamics*, (manuscript in preparation).

[65] K. S. Narendra, "Neural Networks for Control: Theory and Practice", *Proc. of The IEEE*, Vol. 84, No.10, pp. 1385-1406, 1996.

[66] R. F. Stengel and C. Marrison, "Design of Robust Control Systems for Hypersonic Aircraft," *J. Guidance, Control, and Dynamics*, Vol. 21, No.1, pp.58-63, 1998.

[67] R. F. Stengel and L. R. Ray, "Stochastic Robustness of Linear-Time Invariant Control Systems," *IEEE Trans. Automatic Control*, Vol. 36, No. 1, pp. 82-87, 1993.

[68] R. F. Stengel and L. R. Ray, "A Monte Carlo Approach to the Analysis of Control System Robustness," *Automatica*, Vol. 29, No. 1, pp. 229-236, 1993.

[69] R. F. Stengel and C. Marrison, "Stochastic Robustness Synthesis Applied to a Benchmark Control Problem," *Int'l. J. Robust and Nonlinear Control*, Vol. 5, No. 1, pp. 13-31, 1995.

[70] R. F. Stengel and C. Marrison, "Robust Control System Using Random Search and Genetic Algorithms," *IEEE Trans. Automatic Control*, Vol. 42, No. 6, pp. 835-839, 1997.

[71] A. N. Kolmogorov, "On the Representation of Continuous Functions of Several Variables by Superposition of Continuous Functions of One Variable and Addition," *Dokl. Akad. Nauk SSSR*, Vol. 114, pp. 953-956, 1957.

[72] D. Linse and R. F. Stengel, "Identification of Aerodynamic Coefficients Using Computational Neural Networks," *J. Guidance, Control, and Dynamics*, Vol. 16, No. 6, pp. 1018-1025, 1993.

[73] A. Graham, *Kronecker Products and Matrix Calculus: with Applications*, Ellis Horwood Ltd, Chichester, UK, 1981.

[74] G. Strang, *Linear Algebra and Its Applications*, 3rd Ed., Harcourt, Brace, Janovich, San Diego, 1988.

[75] D. Nguyen and B. Widrow, "Improving the Learning Speed of 2-Layer Neural Networks by Choosing Initial Values of the Adaptive Weights," *Proc. Intl. Joint Conf. on Neural Networks*, San Diego, CA, Vol. III, pp. 21-26, 1990.

[76] P. J. Werbos, "Backpropagation Through Time: What It Does and How To Do It," *Proc. of the IEEE*, Vol. 78, No. 10, pp. 1550-1560, 1990.

[77] M. Reidmiller and H. Braun, "A Direct Adaptive Method for Faster Backpropagation Learning: The RPROP Algorithm," *Proc. IEEE Int. Conf. on NN (ICNN)*, pp. 586-591, San Francisco, CA, 1993.

[78] R. J. Vanderbei, *Linear Programming: Foundations and Extensions*, Kluwer Academic Publisher, Boston/London/Dordrecht, 1997.

[79] D.E. Salane, "Adaptive Routines for Forming Jacobians Numerically," *SAND86-1319*, Sandia National Laboratories, 1986.

[80] J. J. D'Azzo and C. H. Houpis, "Nyquist, Bode, and Nichols Plots," *The Control Handbook*, W. S. Levine, Ed., pp. 173-181, CRC Press, Boca Raton, FL, 1996.

[81] R. F. Stengel, *Optimal Control and Estimation*, Dover Publications, New York, NY, 1994.

[82] C. Huang and R. F. Stengel, "Restructurable Control Using Proportional-Integral Model Following," *J. Guidance, Control, and Dynamics*, Vol. 13, No. 2, pp. 303-309, 1990.

[83] *Flying Qualities of Piloted Airplanes, Military Specifications MIL-F-8785C*, USAF ASD, Wright Patterson AFB, November 1980.

[84] R. F. Stengel, "A Unifying Framework for Longitudinal Flying Qualities Criteria," *J. Guidance, Control, and Dynamics*, Vol. 6, No. 2, pp. 84-90, 1983.

[85] H. Erzberger, "Analysis and Design of Model Following Control Systems by State Space Techniques," *Proc. of the 1968 Joint Automatic Control Conf.*, pp. 572-581, June 1968.

[86] L. S. Cicolani, B. Sridhar, and G. Meyer, "Configuration Management and Automatic Control of an Augmentor Wing Aircraft with Vectored Thrust," *NASA Technical Paper, TP-1222*, 1979.

[87] B. Etkin, *Dynamics of Atmospheric Flight*, John Wiley & Sons, Inc., Toronto, 1972.

[88] R. C. Nelson, *Flight Stability and Automatic Control*, McGraw-Hill, Inc., New York, NY, 1989.

[89] J. Kalviste, "Spherical Mapping and Analysis of Aircraft Angles for Maneuvering Flight," *J. Aircraft*, Vol. 24, No. 8, pp. 523-530, 1987.

[90] L. F. Shampine and M. K. Gordon, *Computer Solutions of Ordinary Differential Equations*, W. H. Freeman & Co., 1975.

[91] P. J. Werbos, "Approximate Dynamic Programming for Real-time Control and Neural Modeling," *Handbook of Intelligent Control*, D. A. White and D. A. Sofge, Eds., pp. 493-526, Van Nostrand Reinhold, New York, NY, 1992.

[92] K. Narendra and A. M. Annaswamy, *Stable Adaptive Systems*, Prentice Hall, Englewood Cliffs, NJ, 1989.

[93] B. Friedland, "Observers," *The Control Handbook*, W. S. Levine, Ed., pp. 607-618, CRC Press, Boca Raton, FL, 1996.

[94] T. H. Cormen, *Introduction to Algorithms*, 2nd Ed., MIT Press, Cambridge, MA, 2001.

[95] A. Gelb, Ed., *Applied Optimal Estimation*, MIT Press, Cambridge, MA, 1974.

[96] *NSF Workshop on Learning and Dynamic Programming*, Playacar, MX, April 2002.

Postscriptum

"Know thyself" is all science. Only when he will have finished knowing all things man will have known himself. Things, in fact, are only man's limits.

— Friedrich Nietzsche, *The Gay Science*