2024

Online learning for adaptive control : stable learning and control for aerospace and robotics

Jacob G. Elkins

ONLINE LEARNING FOR ADAPTIVE CONTROL: STABLE LEARNING AND CONTROL FOR AEROSPACE AND ROBOTICS

Jacob G. Elkins

A DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in
Mechanical Engineering
to
The Graduate School
of
The University of Alabama in Huntsville

August 2024

Approved by:

Dr. Farbod Fahimi, Research Advisor/Committee Chair
Dr. Avimanyu Sahoo, Committee Member
Dr. Rohan Sood, Committee Member
Dr. Howard Chen, Committee Member
Dr. Chang-kwon Kang, Committee Member
Dr. George Nelson, Department Chair
Dr. Shankar Mahalingam, College Dean
Dr. Jon Hakkila, Graduate Dean

Abstract

ONLINE LEARNING FOR ADAPTIVE CONTROL: STABLE LEARNING AND CONTROL FOR AEROSPACE AND ROBOTICS

Jacob G. Elkins

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy

Mechanical and Aerospace Engineering

The University of Alabama in Huntsville

August 2024

Aerospace and robotic systems perform various tasks in uncertain, dynamic environments. Further, the data that a system encounters in the real-world is often the most valuable. The most advanced aerospace and robotic systems of the future will be able to learn online, during operation, from this real-world data. Aerospace and robotic systems are often expensive and difficult to model, and the controllers of these systems thus require rigorous control proofs and safety guarantees. Recently, general research in artificial intelligence and machine learning (AI/ML) has made significant strides in developing learning-based systems and controllers. However, much of this research has focused on optimizing control performance, robustness, or prediction accuracy, without considering the stability and safety requirements for control of real-world aerospace and robotic systems. Additionally, the optimization objectives of adaptive control and machine learning, adapting parameters over time to achieve some desired goal or performance, are often closely related. Thus, the theory and mathematical rigor of adaptive control can be used to augment popular AI/ML tools for stability guarantees and online learning. This dissertation discusses

research progress in utilizing AI/ML tools, namely deep neural networks, together with adaptive control to achieve provably-stable online learning and optimization. Part I first describes "learning for control," where neural networks are stably used in a fully online model-based nonlinear controller. The derived controller is shown to desirably control robotic arms, spacecraft, and quadcopters under various disturbances and model uncertainties with limited *a priori* modeling. Next, Part II describes "control for learning," where control-theoretic techniques are used to stably update deep neural network parameters online. The proposed update law is shown to give desirable performance when deep neural network outputs are used in predicting or controlling dynamical systems, especially under domain shift from the training distribution to the target distribution, common in forecasting and sim-to-real transfer of control policies. Throughout this dissertation, the connections between machine learning and adaptive control are explored, with each field acutely poised to benefit the other.

Acknowledgements

First, thank you to the Army Research Office for selecting me to serve as a National Defense Science and Engineering Graduate Fellow. The chance to pursue a doctoral degree is the opportunity of a lifetime in itself, and it fills me with pride knowing that my work up to and beyond this point has the potential to positively influence the things that I care about. This fellowship has allowed me to develop the skills and knowledge that I needed to meaningfully contribute to the defense of the United States in the critical technological area of intelligent systems. I will never forget the day I was selected as a Fellow – thank you to the Department of Defense, the U.S. Army, and my Army Research Lab mentor, Dr. Vernon J. Lawhern.

To my doctoral advisor, Prof. Farbod Fahimi: thank you for your time, expertise, patience, and guidance. I am fortunate to have studied robotics with you, and the progress shown in this dissertation would not have been possible without you.

To my advisor at the University of Alabama, Prof. Rohan Sood: thank you for letting me join your laboratory all those years ago. Thank you for your guidance – technical, professional, and personal – from the beginning of graduate school to now. This dissertation would not have been possible without you pushing me to learn new skills and get out of my comfort zone. I am deeply grateful.

Thank you to my doctoral committee members: Prof. Avimanyu Sahoo, thank you for your hard work. Your courses, discussions, and help have given me the thorough knowledge of nonlinear and adaptive control that I sought in doctoral study. Prof. Howard Chen, thank you for your advice and help on my dissertation. I am sure our paths will cross again. Prof. Chang-kwon Kang, thank you for your hard work and advice in my coursework and my dissertation.

Thank you to Matt Seaman for taking a chance on me and believing in me, when I wasn't even sure I believed in myself yet. My time as a data science intern was my first foray into the world of AI/ML, and I never looked back. I cannot overstate how much I learned and grew during my summers across the country as an intern. Thank you to those at LM who helped me along the way: Mike, Zach, and Norris, to name a few – I enjoyed learning from all of you.

Thank you to Josh Darrow for taking a chance on me and letting me take my AI skills to the next level. My experiences at Pax River have formed my technological and personal foundations more than you know. I believe many of my opportunities to this point have resulted from you giving me an opportunity. I am grateful for you and my Navy team: Matt, Jason, Adam, Dewey, Joe – it was a pleasure.

Thank you to my previous teachers and professors who have had significant influence on my development: Harold Wright, Mike Pope, Derek Irons, Sheri Humphrey, Amanda Bittinger, and countless others. I would not be who I am today without each of you.

To my many friends who have helped me and supported me over my years of school: Thank you to my close friend Leete Skinner, for helping me through my first summer at LM and for continuing to lend some of the most valuable advice I have ever received. Thank you to my close friend and freshman year roommate Daniel Scruggs, for always being there to listen, to offer advice, and to help me with anything. Thank you to my former ASRL labmates William Ledbetter and Ari Rubinsztejn, for mentoring me and showing me the research ropes during my time as a graduate student in Tuscaloosa.

Last yet foremost, thank you to my family. Thank you to my older brother, Tyler, for your loving support as a brother. Thank you to my twin sister, Jessie, for your loving support and help, both technically and emotionally. I am so glad I got to spend time at the same college with both of you – that memory is worth the world.

Thank you to my home lab colleague, Chief, for the unwavering support while lying under my desk all these years. Thank you to my father, Jeff, for your sacrifice to our family and for teaching me toughness and strength.

Above all, thank you to my mother, Kathy. You literally taught me everything I know, from our homeschooling as a kid to our discussions and phone calls as an adult. You instilled in me a relentless drive to do my best and an appreciation for the beauty in knowledge and learning. It truly all begins with looking at Mars through a telescope in the driveway when I was a kid. I hope this achievement helps demonstrate how much I appreciate your sacrifice and your seemingly tireless effort towards me and my siblings.

Dedication

To my mother, for a debt that cannot be repaid. I love you, Mom.

Table of Contents

List of Figures

List of Tables

[Hephaestus] went to the doorway limping, and in support of their master moved his attendants [...]. There is intelligence in their hearts, and there is speech in them and strength, and from the immortal gods they have learned how to do things.

- Homer, *The Iliad*, Book XVIII, l.416-421 (Lattimore)

Chapter 1. Introduction

1.1 Connecting Control and Learning

The fields of artificial intelligence and machine learning (AI/ML) have recently enjoyed an explosion of research interest and development, largely due to the vast increase in data generation and available computational power [1]. This increase in data and computational power has allowed AI/ML researchers to develop the early neural network theory of Turing, McCulloch, Pitts, Rosenblatt, Minsky, Papert, and Hopfield into the powerful nonlinear function approximators that comprise the backbone of most AI/ML breakthroughs in the last quarter century, known as *deep learning* [2–6]. Improving the optimization, stability, and explainability of these black-box function approximators is an ongoing research topic, with the most important tool of this optimization being gradient-based backpropagation, used to iteratively adjust function approximator parameters to minimize prediction error via numerous data points [7]. Further, increased computational power has led to the use of high-fidelity simulation being used to drive AI/ML innovations, namely through reinforcement learning [8]. These innovations have already made significant impacts on many fields, such as medical diagnosis, finance, drug discovery, and retail, to name a few.

Aerospace and robotic systems are well-poised to benefit greatly from recent AI/ML development in many ways. Aerospace systems generate large amounts of data via sensors, are often complicated to control and difficult to model, and operate in uncertain and ever-changing environments with disturbances. The controllers of aerospace and robotic systems must be robust to internal and external disturbances and be able to adapt to unforeseen events and changing environments. Thus, the field of adaptive control is a vibrant and critical research field for aerospace and robotic systems – in fact, the genesis of adaptive control lies in the design of autonomous flight controllers for airplanes in the 1950s. Adaptive control is primarily concerned with accomplishing various control goals (such as trajectory tracking) in the presence of disturbances and uncertainties; adaptive controllers may or may not have learning elements in the control law [9]. For instance, the conventional formulation of sliding mode control, a robust control paradigm utilized throughout this dissertation, can control systems under parameter uncertainty without any update laws or learning elements. However, the notation of "learning" is often simply nomenclature, as many adaptive control methods involve parameter estimation, which can easily be argued as learning from an AI/ML perspective [10].

Adaptive control, due to its criticality to the safe and efficient operation of aerospace and robotic systems, is likely the avenue of greatest benefit for integrating AI/ML into aerospace and robotics. This comes from a critical theory of this dissertation: ***the data that a system encounters online, during operation, is the most important for learning.*** As aforementioned, mod-

ern AI/ML is notoriously computationally and data intensive, through the optimization methods used and the simulation and data processing required to train models and agents. There remains much work to be done in incorporating these AI/ML innovations into smart, adaptive aerospace controllers that are provably stable and improve over time. For controllers to improve as they operate, the controller must learn and evolve during operation to changing conditions and environments. This field is generally called *online learning*, which is intimately related to adaptive control with update rules for adjusting parameters. Online learning for control comes with many challenges, such as computational cost of updates while controlling, exploration without jeopardizing a control goal, operation under high levels of system uncertainty, and overoptimizing to recent inputs, called *catastrophic forgetting*. ***The primary goal of this dissertation is to explore the connections between adaptive control and machine learning in the online learning problem, and how each field could potentially be used to improve the other.*** Put simply, the objective of this dissertation research was to study ways to guarantee the training and performance of AI/ML elements such that those elements can safely be used in aerospace and robotic systems, among other applications.

1.2 Organization

This dissertation is divided by theme into two primary parts: Part I, "Learning for Control," and Part II, "Control for Learning."

Part I: Learning for Control

Part I describes the progress made in developing adaptive controllers that utilize and stably update AI/ML elements (namely, neural networks) online. The motivation of this section is to explore how the adaptive control theories of Lyapunov, LaSalle, and Barbalat can be used to derive stability guarantees on neural networks, while improving adaptive control through nonlinear function approximation.

Part II: Control for Learning

Part II describes the developments made in using control-theoretic techniques to update AI/ML instruments online. Using control theory to directly update AI/ML function approximator parameters can give performance guarantees on prediction error of the AI/ML elements being updated online. Considering the goals of adaptive control theory and AI/ML to be generally aligned, the motivation of this section is to go beyond basic backpropagation – to improve performance and stability of AI/ML elements when deployed online, especially when the online data distribution is shifted from the training data distribution.

1.3 Summary and Contributions

In Chapter 2, the primary tool of modern AI/ML, the neural network (NN) is stably incorporated into an online adaptive controller [11]. A dynamics model common to many real-world robotic systems is assumed, and a NN is used

together with direct parameter estimation to learn the relevant functions and parameters for control of the system fully online. The NN uses no pretraining, which makes controller initialization an issue. To overcome this issue, a nonlinear term in the control law is used – sliding mode control (SMC), a robust model-based control paradigm. During development of the stability proof, it was found that the learning error acts on the system equivalently to an external disturbance, allowing disturbance-rejecting nonlinear control terms to stabilize the system during initial learning. ***The neural network term is found to increase control accuracy while alleviating chatter, the primary drawback of conventional sliding mode control.*** The controller is verified by simulating a non-planar 3 degree-of-freedom robotic arm picking up a load, and the developed controller outperforms conventional SMC, especially when the load margin built into the conventional model-based SMC controller is exceeded. Further, Chapter 2 describes extending the controller to incorporate stable estimation of positive definite coefficient matrices, which simplifies the learning problem for the NN.

In Chapter 3, the general online NN-based adaptive controller developed in [11] is applied to two challenging aerospace control problems: rigid-body spacecraft attitude control and full quadcopter control. As in Chapter 2, ***the learning elements allow adaptation, while the nonlinear control elements stabilize the system during initial learning and reject disturbances after learning converges.*** Spacecraft attitude control is chosen to show that careful consideration of the sliding variable leads to minimum-time slews in quaternion trajectory tracking. Quadcopter control is chosen to exemplify the use of virtual

control inputs and multiple variable-order subsystems to control a complicated, underactuated system such as a quadcopter. ***Both applications of the controller require decreased tuning and modeling than traditional model-based approaches by simplifying complicated state functions into NN approximations, rendering a highly adaptive and robust controller.***

Chapter 4 concludes Part I with discussion and conclusions on the progress made in using AI/ML instruments inside adaptive controllers, discussed in Chapters 2 and 3. The chapter concludes with some possible avenues of future work, such as developing recursive rules for arbitrary-depth DNNs or adaptive-gain rules for the nonlinear control terms.

Chapter 5 discusses theoretical and experimental developments in utilizing the mathematical rigor and stability proofs of control theory to update the parameters of AI/ML instruments directly [12]. An arbitrary-depth feed-forward deep neural network (DNN) is considered, pretrained on a dataset that experiences domain shift when deployed online. The DNN is formulated as a continuous-time dynamical system to be controlled, and it is shown that ***a novel super-twisting-based update law on the output layer of the DNN can guarantee online prediction performance under domain shift.*** It is also shown that knowledge of the time derivative of the DNN input vector is required to drive online prediction error to zero. This derivative is often approximated via numerical differentiation, or noisy through estimation. ***Theoretical analysis and experiment show that training the DNN with spectral normalization can decrease the upper bound of prediction error when the time derivative of the***

DNN input vector is noisy or estimated. An important application of this includes trivial implementation of simulation-trained controllers onto real-world systems via online adaptation and learning (known as "sim2real" transfer [13]). A sim2real transfer example is described for a model reference adaptive control problem of a robot arm, where ***the DNN learns online to compensate for unmodeled robot dynamics using only 10 seconds of simulation data.***

Chapter 6 concludes Part 2 with discussion and conclusions on the theoretical developments and simulation results of Chapter 5. The chapter discusses potential avenues of future work, such as update rules for other layers of the DNN, update rules for different neural network architectures outside of an arbitrary-depth feedforward DNN, and provably stable real-world control policies fully derived from reinforcement learning.

Part I: Learning for Control

Stable Online Learning-Based Adaptive Control of Robotic Manipulators, Spacecraft, and Quadcopters

Chapter 2. Improving Model-Based Control with Online Adaptation of Neural Networks

This chapter presents progress in stably incorporating the most popular tool in modern AI/ML, the neural network, inside of online adaptive controllers. The neural network is used to learn a suitable model for control fully online, where a nonlinear model-based control term is used to stabilize the system during initial learning. This chapter is adapted from the work "Online Neural Sliding Mode Control with Guaranteed Stability," by Jacob G. Elkins and Farbod Fahimi, published in the International Journal of Control in 2024 [11].

2.1 Introduction

Engineers utilize model-based control systems to leverage known system information and behavior, often resulting in improved control accuracy or robustness. However, these system models can change over time, be difficult to obtain for complex systems, or fail to capture higher-order effects in the system. In this chapter, the highly robust model-based control methodology of sliding mode control is extended to simplify the *a priori* system modeling. A neural network learns an approximate system model fully online, with no pretraining. The neural network term increases control accuracy while allowing lower gains

on the discontinuous terms in the control law, thereby alleviating chatter, the primary drawback of conventional sliding mode control. The controller is verified by simulating a non-planar 3 degree-of-freedom robotic arm picking up a load, and the proposed controller is shown to outperform conventional sliding mode control with no learning elements, especially when the load margin on the end effector built into the conventional model-based controller is exceeded.

2.2 Background and Literature Review

As modern systems increase in complexity, the models required for effective model-based control become more difficult and time-consuming to derive and obtain. Further, as aerospace and robotic systems operate in uncertain dynamical environments, the controller must be robust to disturbances and model discrepancies. There has been a multitude of research in the design and implementation of adaptive and robust controllers, particularly motivated by aerospace applications. Some notable nonlinear control methodologies include, but are not limited to, backstepping, feedback linearization, $\mathcal{H}_\infty$ control, and sliding mode control (SMC). Often, these controllers combine some form of (non-adaptive) control with a form of parameter estimation or system identification [14, 15]. SMC has been shown to effectively control many nonlinear dynamical systems in the presence of disturbances and model uncertainties [16–19]. SMC involves defining a sliding manifold, which defines the desired error behavior (often exponential). Once the sliding manifold (or "mode") is reached, the error will follow the prescribed desired behavior – one of the primary advantages of SMC is the controller's insen-

sitivity to system parameter deviations and disturbance rejection once the sliding mode is reached.

Conversely, the primary drawback of SMC is chattering in the control input to the system, due to discontinuity in the control law. This discontinuity can potentially harm real-world acuators and lead to other undesirable side effects. Chatter in SMC-based controllers has been attenuated in various ways, such as using a continuous approximation of the discontinuous term (*i.e.*, boundary layer interpolation) or integrating the discontinuous term [9, 14, 16, 20].

Conventional SMC requires an *a priori* dynamical model and bounded model uncertainty assumptions to assign convergence and stability guarantees. As aforementioned, obtaining dynamical models of complicated nonlinear systems can be difficult and expensive, often requiring experimental system identification or simplifications in the model [21, 22]. Since SMC is robust to model uncertainties, simplified dynamic models of the underlying system can often be used effectively [23, 24]. Further, learning-based models have been shown to render improved control accuracy by incorporating novel techniques from machine learning to capture complex effects that are difficult to model [25–27].

The recent explosion of research interest large-scale data science and machine learning has impacted almost all fields in science and engineering. Notable examples of the success of modern machine learning include high-level reinforcement learning and planning [8], drug discovery in medicine [28], and image/text generation [29]. These innovations are driven by one of the most popular and effective instruments of learning-based models in machine learning, the neural

network, which has been shown to have significant power in function approximation [30]. While neural network research in control theory is not new [31–34], the data and computing power available today have allowed neural networks to make large strides in machine learning problems. Thus, a critical avenue of controls research is studying how to effectively incorporate cutting-edge machine learning research into the design, analysis, and verification of control systems.

Adaptive controllers utilizing neural networks have been extensively studied in various control design scenarios with promising results [31–41]. However, some adaptive control methodologies have been shown to suffer in performance and stability when using learned models [42]. Recently, there has been increased research interest in incorporating stability guarantees and safety into these learned models. Some large themes of stability guarantees in learned models include using Lyapunov theory to directly design [43] and train [44–46] neural networks, contraction theory [42], and machine learning techniques such as spectral normalization to establish Lipschitz-continuity bounds [25]. These works, along with the research of this dissertation as a whole, have shown that considerable connections exist among modern machine learning, nonlinear systems, and control theory. Further, these works have highlighted that learning-based controllers should be carefully designed to have rigorous performance and stability guarantees under careful assumptions.

This chapter focuses on online learning in adaptive control, when the controller must adapt as data is acquired during operation. Adaptive control and online learning are closely related, as both are generally concerned with converging

on a parameter or set of parameters to achieve some goal. Previous works using neural networks adapted online have shown effective control of nonlinear systems in both discrete time [36–39] and continuous time [20, 35, 40, 41, 47]. Early previous works utilize neural networks with only one layer of parameters (no hidden layer) [36]. However, neural networks with only an input and output, formerly known as a two-layer perceptron, have been described by the universal approximation theorem to be incapable of effectively representing functions outside of a specific class [5, 30]. Other works using neural networks evolved online in adaptive control have used radial basis function (RBF) networks [37, 40, 47–52]. While RBF networks have been theoretically guaranteed to exhibit universal approximation [53], their popularity in modern machine learning has declined due to the success of feed-forward neural networks with computationally-simple activation functions (such as sigmoid or ReLU). Other works have utilized terminal SMC with RBF networks effectively [40, 52]. Terminal sliding mode with robust exact differentiation and multilayer neural networks have been used with promising results for quadcopter control [41]. Recurrent neural networks have been shown to be effective tools in online adaptation for an integral sliding mode controller [54], though the work assumes some knowledge of system parameters. Another work in [45] used a recurrent neural network to directly identify the system, as opposed to controlling the system. Other works have used Lyapunov-derived neural network update laws outside the scope of SMC, where [46] assumes a simplified system with an identity control effectiveness matrix. Concurrent work to this chapter also showed that multilayer feed-forward neural networks are effective in an in-

tegral sliding mode control scheme [20], where the state regulation problem is considered. It is shown in [20] that the states of the system are ultimately upper bounded around the zero state equilibrium point.

With the considerations of modern machine learning described above, this chapter connects one of the primary tools of machine learning, the multilayer feed-forward neural network, to the conventional SMC robust control design. The general control design framework presented is extensible to other neural network architectures and real-world systems to be controlled. This chapter presents a novel online-learning-based controller that needs no *a priori* system parameter information, is robust to external disturbances, and gives guaranteed tracking performance and stability. This chapter is some of the first work to provide a conventional sliding mode controller using adaptive control, neural networks, and direct parameter estimation with no prior knowledge of system parameters on a general control-affine system.

2.3 Contributions

The full contributions of this chapter are as follows:

(1) This chapter presents generalized n^{th}-order Lyapunov stability proofs to derive real-time neural network update laws that are closely related to general neural network backpropagation. This method results in globally asymptotically stable trajectory-tracking error, which is an improvement compared to other work in the literature that achieves ultimately upper bounded state convergence to a zero equilibrium point [20]. In addition, the work in this chapter addresses

systems with an unknown constant control effectiveness matrix, which builds upon previous work that assumes the control effectiveness matrix to be identity [46].

(2) This chapter combines the robustness to disturbances and uncertainty of SMC and direct adaptive control with the approximation power of multilayer neural networks in a novel, general, lightweight framework that learns a sufficient system dynamic model for control fully online. It is shown that the neural network learning can reduce chattering by decreasing the discontinuous control gains, which is widely considered the primary drawback of conventional SMC.

(3) The proposed controller's effectiveness, generality, and ease of implementation is validated in an adaptive control problem by controlling a simulated robotic arm, using no neural network pretraining or further *a priori* dynamic modeling. The developed method learns the new system dynamics when an arbitrary load on the end effector is applied. The method developed in this chapter is compared to conventional model-based SMC with a designed load margin on the end effector, and it is shown that the developed method outperforms conventional model-based SMC when the load applied exceeds the margin built into the conventional SMC controller. In this scenario, the conventional SMC controller fails, where the developed controller adapts to the new system dynamics.

(4) The general controlled is extended by a novel eigenspace projection rule on the coefficient matrix, preserving the symmetric, positive definite property of the assumed system dynamics. This method is validated in simulation and compared to the general controller without the eigenspace projection, and it is

found that the eigenspace projection on the coefficient matrix seems to attenuate chatter in some scenarios.

2.4 Notation and Preliminaries

This section presents the notation used in this chapter, the assumed dynamics for control derivation, and the basics of neural network function approximation.

2.4.1 Notation

The notation used in this chapter is specific to this chapter. The set of real numbers is denoted as $\mathbb{R}$, and the set of positive real numbers is denoted as $\mathbb{R}^+$. $\mathbb{R}^n$ denotes the set of real vectors of dimension $n \times 1$, and $\mathbb{R}^{n \times m}$ denotes the set of real matrices of dimension $n \times m$. The n^{th} time t derivative of a variable $y(t)$ is denoted as $y^{(n)} = \frac{d^{(n)}y}{dt^{(n)}}$ for $n \in \mathbb{R}^+$, where the first time derivative is denoted simply as $\dot{y} = y^{(1)} = \frac{dy}{dt}$. The vector L^2 norm is denoted as $|| \cdot ||_2$. The diagonal matrix $Q \in \mathbb{R}^{n \times n}$ with values of the vector $q \in \mathbb{R}^n$ along the main diagonal is denoted as $\texttt{diag}(q)$. For vectors $a, b \in \mathbb{R}^n$, the elementwise product is denoted as $a \odot b$.

2.4.2 Control Preliminaries

This chapter considers system dynamics of the form

$$My^{(n)} + f(x) + d(t) = u(t), \tag{2.1}$$

where $M \in \mathbb{R}^{m \times m}$ is an unknown diagonal constant coefficient matrix, $y^{(n)} \in \mathbb{R}^m$ is the n^{th} time derivative of system output, $f(x) : \mathbb{R}^p \to \mathbb{R}^m$ is some unknown nonlinear system function of state $x \in \mathbb{R}^p$ to be learned online, $d \in \mathbb{R}^m$ is an additive bounded environmental/internal disturbance term; and $u \in \mathbb{R}^m$ is the control input.

The tracking error is defined as

$$e = y_d - y, \tag{2.2}$$

where y_d is the desired system trajectory. As discussed above, sliding mode control is the robust nonlinear control paradigm used in the work of this chapter. This requires defining a sliding manifold (or variable) that prescribes desired error performance. That is, when the sliding variable is driven to zero, the error follows prescribed performance in time – known as 'reaching" the sliding mode. The sliding variable used is the filtered tracking error s, written as

$$s = e^{(n-1)} + \sum_{i=0}^{n-2} \binom{n-1}{i} \lambda^{n-i-1} e^{(i)}, \tag{2.3}$$

where λ is a designed diagonal, positive definite matrix. Using the assumed system in (2.1), the desired controller should drive $s \to 0$ in time. When $s = 0$, the controller has reached the sliding mode, where the error dynamics in (2.3) exponentially converge to zero.

2.4.3 Online Function Approximation Using Neural Networks

While constant coefficients such as M in (2.1) can be learned directly through online parameter adaptation, general nonlinear functions (such as the state-function portion $f(x)$ in (2.1)) must be learned using a nonlinear function approximator. Easily-trainable models for general nonlinear function approximation is a highly active research area in the field of machine learning, and many different methods of nonlinear function approximation and modeling exist throughout mathematics. In learning-based controls, an ideal function approximator is accurate, easy to update/tune, and computationally efficient. The work in this chapter uses a neural network to learn $f(x)$. Neural networks have adequately modeled many nonlinear functions in a variety of data-driven machine learning settings, with policy and value function representation in reinforcement learning being of particular relevance to the work described in this chapter.

The simple neural network used in this chapter consists of three layers: an input layer, a hidden layer, and an output layer. Neural networks of this architecture can universally approximate a nonlinear function to an arbitrary error bound ϵ_B, as described by the universal approximation theorem [30]. Mathematically, this is represented as

$$f(x) = W^T \sigma\left(V^T x\right) + \epsilon\left(x\right), \tag{2.4}$$

where $x \in \mathbb{R}^p$ is the input vector, $V \in \mathbb{R}^{p \times n_H}$ is the matrix of weights connecting the input layer and the hidden layer, $\sigma(\cdot)$ is a nonlinear activation function, and

$W \in \mathbb{R}^{n_H \times m}$ is the matrix of weights connecting the hidden layer and the output layer, and $\epsilon(x)$ is an approximation error bounded by $||\epsilon(x)||_2 \leq \epsilon_B$. Almost all modern artificial neural networks employ neurons that contain a multiplicative weight and an additive bias, modeled after the perceptron [4]. The work in this chapter follows [35], appending a 1 to the input vector x and a constant first-term of 1 in the activation vector $\sigma\left(V^T x\right)$ to incorporate bias terms in the network. This allows the update rules of the weights W and V to also update the respective bias vectors. For simplicity of notation, it is hereafter assumed that the dimension of the state vector $x \in \mathbb{R}^p$ includes the appended 1.

Using this neural network to approximate the state function gives

$$\hat{f}(x) = \hat{W}^T \sigma\left(\hat{V}^T x\right), \tag{2.5}$$

where $\hat{W}$ and $\hat{V}$ are the current weight estimates. The work of this chapter uses the sigmoid function for the activation function $\sigma(\cdot)$, written as

$$\sigma(z) = \frac{1}{1+e^{-z}}. \tag{2.6}$$

The activation function is performed elementwise on its input vector. The sigmoid function is commonly used due to its continuity, its boundedness, and its simple derivative, which is needed for the update rule of the neural network weights. The derivative of the sigmoid function with respect to argument z is

$$\sigma'(z) = \sigma(z)\left(1 - \sigma(z)\right), \tag{2.7}$$

which is easily implemented and computationally efficient, since the quantity $\sigma(z)$ is already calculated from the forward pass of the neural network.

The update rules for $\hat{W}$ and $\hat{V}$ will be developed and discussed in the following sections.

2.5 Controller

As described above, the desired controller drives the sliding variable $s \to 0$ in time by learning online the nonlinear system function $f(x)$ and constant parameter matrix M. When designing an online-learning based controller, the controller must be stable and performing within error tolerances while the controller is still learning about the system (*i.e.*, before the learned parameters/functions have converged). This is a crucial aspect of applying learning-based controllers to unstable aerospace systems, such as quadcopters. As will be seen in Chapter 3, if there is no neural network pretraining or nonlinear control term in the control law, the quadcopter would initially fail to fly. Considering the controller's current estimates of system parameters and functions as system modeling errors, a model-based control term that is robust to model uncertainties can be used to stabilize the system in the outer loop before (and during) learning convergence.

In this chapter, it is only assumed that the system dynamics follow the form of (2.1) and that the disturbance term $\delta(t) = d_a(t) + d$, to be defined below, is bounded by a constant vector $\Delta = [\Delta_1, \Delta_2, \ldots, \Delta_m]^T$, such that $|\delta_i(t)| \leq \Delta_i$ for $\forall t \geq 0; i = 1, 2, \ldots, m$. To simplify notation, a reference output y_r is defined such that

$$y_r^{(n)} = y_d^{(n)} + \sum_{i=0}^{n-2} \binom{n-1}{i} \lambda^{n-i-1} e^{(i+1)}. \tag{2.8}$$

The model-based control paradigm used in this controller is sliding mode control, which is a well-developed model-based control methodology that is robust to model uncertainties and disturbances [16]. In conventional variable structure/sliding mode control, to achieve the sliding mode goal of $s = 0$, s is typically evolved by a nonlinear, discontinuous function. Typically, the signum function $\dot{s} = -K\texttt{sign}(s)$ is used, where K is some positive gain. The discontinuity in $\dot{s}$ can cause chattering in real-world actuators due to their operation on discrete time intervals. Chatter can be mitigated by using a *boundary layer*, which resolves the discontinuity in $\dot{s}$ by linearizing $\dot{s}$ about some small boundary layer thickness ϕ. The $\texttt{sat}(\cdot)$ function, shown below in (2.9) and used in the controller developed in this chapter, concisely implements this idea:

$$\texttt{sat}(s/\phi) = \begin{cases} \texttt{sign}(s) & \text{if } |s| > \phi \\ \frac{s}{\phi} & \text{if } |s| \leq \phi. \end{cases} \tag{2.9}$$

Note that the boundary layer thickness ϕ is a hyperparameter, tuned for desired accuracy while attenuating chatter.

The control law proposed in this chapter is

$$u = \hat{M} y_r^{(n)} + \hat{f}(x) + \eta \odot \texttt{sat}(s/\phi), \tag{2.10}$$

where $\hat{M}$ is the current estimate of M, $\hat{f}(x)$ is the current estimate of the system state function $f(x)$, and $\eta \in \mathbb{R}^m$ is a positive gain vector tuned for performance and assisting in initial learning convergence.

The current estimate of M is adapted by

$$\dot{\hat{M}} = H y_r^{(n)} s^T, \tag{2.11}$$

where $H \in \mathbb{R}^{m \times m}$ is a diagonal, positive definite gain matrix tuned for the learning and stability of M.

The system state function $\hat{f}(x)$ is approximated using the neural network given in (2.5) with sigmoid activation functions as described in Section 2.4.3. Denoting the hidden layer activation function with current weight estimates as $\hat{\sigma} = \sigma(\hat{V}x)$ and its derivative as $\hat{\sigma}' = \sigma'(\hat{V}x) = \hat{\sigma}(1 - \hat{\sigma})$, the online update rules for the neural network weight matrices are given as

$$\dot{\hat{W}} = F \hat{\sigma} s^T \tag{2.12}$$

$$\dot{\hat{V}} = G x s^T \hat{W}^T \hat{\sigma}', \tag{2.13}$$

where $F \in \mathbb{R}^{n_H \times n_H}$ and $G \in \mathbb{R}^{p \times p}$ are diagonal, positive definite update gain matrices tuned for learning and stability, and x is the input vector to the neural network. The hidden layer activation derivative is implemented using matrix multiplication as $\hat{\sigma}' = \texttt{diag}(\hat{\sigma})(I - \texttt{diag}(\hat{\sigma}))$, where I is the identity matrix. A block diagram of the controller structure is shown in Figure 2.1. The following

section shows the derivation and equilibrium stability of these update rules by the Lyapunov direct method of stability analysis.

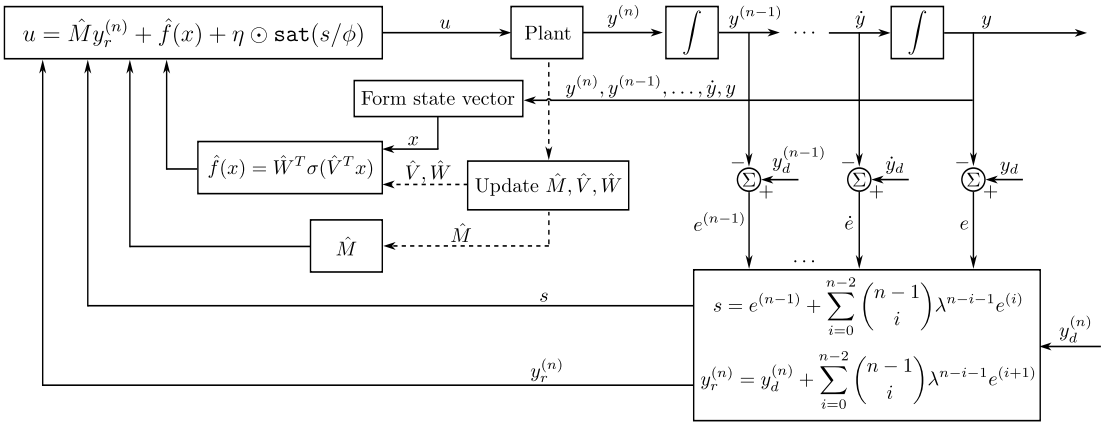

Figure 2.1: Block diagram of the online neural sliding mode controller structure.

2.6 Stability Analysis of the Developed Controller

To construct a Lyapunov function candidate, consider the objective for the controller: to drive the sliding variable $s \to 0$ by learning relevant system parameters. The relevant system parameters set for learning are $\hat{M}$ and $\hat{f}(x)$, where $\hat{f}(x)$ is represented by the neural network $\hat{f}(x) = \hat{W}^T\sigma(\hat{V}^T x)$. Thus, the goal of this controller is to drive the sliding variable $s \to 0$ by evolving $\hat{M}$, $\hat{W}$, and $\hat{V}$ such that $\hat{M} \to M$, $\hat{W} \to W$, and $\hat{V} \to V$. Lyapunov function candidates must be scalar and positive definite. Thus, a scalar quantification of matrix error is needed for the following Lyapunov analysis. The matrix trace operator is related to the Frobenius norm by

$$||A||_F^2 = \mathrm{Tr}\left(A^T A\right) = \sum a_{ij}^2, \tag{2.14}$$

where $\mathrm{Tr}\,(\cdot)$ denotes the matrix trace operator and a_{ij} is the $(i,j)^{th}$ component of matrix A. Thus, a scalar measure of matrix error between the "ideal" matrix A and the current estimate $\hat{A}$ can be written as $||\tilde{A}||_F^2 = ||A - \hat{A}||_F^2 = \mathrm{Tr}\left(\tilde{A}^T\tilde{A}\right)$.

Theorem 2.6.1. *Let the desired system trajectory, y_d, and its n time derivatives be continuous and bounded. Let $\delta(t) = d_a(t) + d(t)$ be the sum of approximation/learning error and external disturbance, to be defined. Assuming $\delta(t)$ is upper bounded such that $\forall t \geq 0 : |\delta_i(t)| \leq \Delta$, for $i = 1, 2, \ldots, m$; the dynamical system in* (2.1)*, the control input in* (2.10)*, and update rules in* (2.11)*,* (2.12)*, and* (2.13)*; the estimates $\hat{M}$, $\hat{W}$, and $\hat{V}$ are bounded and $s \to 0$ as $t \to \infty$.*

Proof. Considering the controller objectives described above, the positive definite Lyapunov function candidate used for the controller in this chapter is given by

$$L = \frac{1}{2}\left(s^T M s + \mathrm{Tr}\left(\tilde{M}^T H^{-1}\tilde{M}\right) + \mathrm{Tr}\left(\tilde{W}^T F^{-1}\tilde{W}\right) + \mathrm{Tr}\left(\tilde{V}^T G^{-1}\tilde{V}\right)\right), \quad (2.15)$$

where $\tilde{M} = M - \hat{M}$, $\tilde{W} = W - \hat{W}$, and $\tilde{V} = V - \hat{V}$.
Differentiating (2.15) with respect to time gives

$$\dot{L} = s^T M\dot{s} + \mathrm{Tr}\left(\tilde{M}^T H^{-1}\dot{\tilde{M}}\right) + \mathrm{Tr}\left(\tilde{W}^T F^{-1}\dot{\tilde{W}}\right) + \mathrm{Tr}\left(\tilde{V}^T G^{-1}\dot{\tilde{V}}\right) \quad (2.16)$$

since it is assumed that $\dot{M} = 0$. Differentiating (2.3) with respect to time and using the relation in (2.8), $\dot{s}$ can be written as

$$\dot{s} = y_r^{(n)} - y^{(n)}, \tag{2.17}$$

where the system in (2.1) can be rewritten in an affine form as

$$y^{(n)} = M^{-1}\left(u(t) - f(x) - d(t)\right). \tag{2.18}$$

Substituting the control law in (2.10) into the system in (2.18) and simplifying, the relation for $\dot{s}$ in (2.17) becomes

$$\dot{s} = y_r^{(n)} + M^{-1}\left(-\hat{M}y_r^{(n)} + f(x) - \hat{f}(x) - \eta \odot \mathtt{sat}(s/\phi) + d(t)\right), \tag{2.19}$$

which is substituted into the Lyapunov derivative in (2.16) to get

$$\begin{aligned}\dot{L} = s^T M y_r^{(n)} - s^T \hat{M} y_r^{(n)} + \mathrm{Tr}\left(\tilde{M}^T H^{-1}\dot{\tilde{M}}\right) + s^T\left(f(x) - \hat{f}(x)\right) + \mathrm{Tr}\left(\tilde{W}^T F^{-1}\dot{\tilde{W}}\right) \\ + \mathrm{Tr}\left(\tilde{V}^T G^{-1}\dot{\tilde{V}}\right) - s^T \eta \odot \mathtt{sat}(s/\phi) + s^T d(t).\end{aligned} \tag{2.20}$$

The fourth term of (2.20) can be written as

$$f - \hat{f} = W^T\sigma(V^T x) - \hat{W}^T\sigma(\hat{V}^T x) + \epsilon(x) \tag{2.21}$$

from (2.4) and (2.5), where the function arguments of $f = f(x)$ and $\hat{f} = \hat{f}(x)$ have been dropped for brevity. Adding and subtracting $W^T\sigma(\hat{V}^T x)$ to (2.21) and simplifying gives

$$f - \hat{f} = W^T(\sigma(V^T x) - \sigma(\hat{V}^T x)) + (W^T - \hat{W}^T)\sigma(\hat{V}^T x) + \epsilon\,(x)\,. \tag{2.22}$$

The shorthand notations of $\tilde{\sigma} = (\sigma(V^T x) - \sigma(\hat{V}^T x))$ and $\hat{\sigma} = \sigma(\hat{V}^T x)$ allows rewriting (2.22) as

$$f - \hat{f} = W^T\tilde{\sigma} + \tilde{W}^T\hat{\sigma} + \epsilon\,(x)\,, \tag{2.23}$$

where adding and subtracting another $\hat{W}^T\tilde{\sigma}$ gives

$$f - \hat{f} = \tilde{W}^T\tilde{\sigma} + \hat{W}^T\tilde{\sigma} + \tilde{W}^T\hat{\sigma} + \epsilon\,(x)\,. \tag{2.24}$$

Here, as in [35], a Taylor series expansion is used for the hidden layer activation function about a given input x:

$$\sigma(V^T x) = \sigma(\hat{V}^T x) + \sigma'(\hat{V}^T x)\tilde{V}^T x + \mathcal{O}\left((\tilde{V}^T x)^2\right), \tag{2.25}$$

where $\hat{\sigma}'$ denotes the hidden layer activation derivative and $\mathcal{O}\left((\tilde{V}^T x)^2\right)$ denotes higher-order terms from the Taylor series expansion. Subtracting $\sigma(\hat{V}^T x)$ in (2.25) and using our shorthand notation gives

$$\tilde{\sigma} = \hat{\sigma}'\tilde{V}^T x + \mathcal{O}\left((\tilde{V}^T x)^2\right). \tag{2.26}$$

Substituting (2.26) into the first and second terms of (2.24) renders

$$f - \hat{f} = \hat{W}^T\hat{\sigma}'\tilde{V}^T x + \tilde{W}^T\hat{\sigma} + d_a(t), \tag{2.27}$$

where $d_a(t)$ is the internal "disturbance" due to neural network approximation errors and higher-order Taylor series terms, written as

$$d_a(t) = \tilde{W}^T\hat{\sigma}'\tilde{V}^T x + W^T\mathcal{O}\left((\tilde{V}^T x)^2\right) + \epsilon(x). \tag{2.28}$$

Substituting (2.27) back into the Lyapunov derivative in (2.20) gives

$$\begin{aligned}\dot{L} = s^T M y_r^{(n)} - s^T\hat{M}y_r^{(n)} + \mathrm{Tr}\left(\tilde{M}^T H^{-1}\dot{\hat{M}}\right) + s^T\hat{W}^T\hat{\sigma}'\tilde{V}^T x + s^T\tilde{W}^T\hat{\sigma} \\ + \mathrm{Tr}\left(\tilde{W}^T F^{-1}\dot{\tilde{W}}\right) + \mathrm{Tr}\left(\tilde{V}^T G^{-1}\dot{\tilde{V}}\right) - s^T\eta \odot \mathtt{sat}(s/\phi) \\ + s^T(d_a(t) + d(t)).\end{aligned} \tag{2.29}$$

The last term in (2.29) shows that the approximation disturbance $d_a(t)$ act on the system similar to the external disturbances $d(t)$. This is the motivation of using a disturbance-rejecting nonlinear control term alongside learning elements – the learning/approximation error can be considered as a disturbance and rejected by a robust control term.

In general, for column vectors $\vec{\alpha}$ and $\vec{\beta}$, the inner product can be written as the matrix trace of the outer product, $\vec{\alpha}^T\vec{\beta} = \mathrm{Tr}\,(\vec{\beta}\vec{\alpha}^T)$. This matrix trace property allows reordering matrix terms such that unknown variables or parameters can be factored out and combined with other terms. Various terms in (2.29) can thus be rewritten as

$$s^T M y_r^{(n)} = \mathrm{Tr}\,(M y_r^{(n)} s^T) \tag{2.30}$$

$$s^T \hat{M} y_r^{(n)} = \mathrm{Tr}\,(\hat{M} y_r^{(n)} s^T) \tag{2.31}$$

$$s^T \hat{W}^T \hat{\sigma}' \tilde{V}^T x = \mathrm{Tr}\,(\tilde{V}^T x s^T \hat{W}^T \hat{\sigma}') \tag{2.32}$$

$$s^T \tilde{W}^T \hat{\sigma} = \mathrm{Tr}\,(\tilde{W}^T \hat{\sigma} s^T). \tag{2.33}$$

Substituting the relations in (2.30)-(2.33) back into (2.29) gives

$$\begin{aligned}\dot{L} = \mathrm{Tr}\left(M y_r^{(n)} s^T\right) - \mathrm{Tr}\left(\hat{M} y_r^{(n)} s^T\right) + \mathrm{Tr}\left(\tilde{M}^T H^{-1} \dot{\tilde{M}}\right) + \mathrm{Tr}\left(\tilde{V}^T x s^T \hat{W}^T \hat{\sigma}'\right) \\ + \mathrm{Tr}\left(\tilde{W}^T \hat{\sigma} s^T\right) + \mathrm{Tr}\left(\tilde{W}^T F^{-1} \dot{\tilde{W}}\right) + \mathrm{Tr}\left(\tilde{V}^T G^{-1} \dot{\tilde{V}}\right) \\ - s^T \eta \odot \mathtt{sat}(s/\phi) + s^T (d_a(t) + d(t)).\end{aligned} \tag{2.34}$$

The matrix trace operator is a linear operator. Thus, for square matrices A and B of equal dimension, $\mathrm{Tr}\,(A)+\mathrm{Tr}\,(B)=\mathrm{Tr}\,(A+B)$. This additive trace property is used to combine like terms in (2.34) to get

$$\begin{aligned}\dot{L} = \mathrm{Tr}\left(\tilde{M}^T y_r^{(n)} s^T + \tilde{M}^T H^{-1}\dot{\tilde{M}}\right) + \mathrm{Tr}\left(\tilde{V}^T x s^T \hat{W}^T \hat{\sigma}' + \tilde{V}^T G^{-1}\dot{\tilde{V}}\right) \\ + \mathrm{Tr}\left(\tilde{W}^T \hat{\sigma} s^T + \tilde{W}^T F^{-1}\dot{\tilde{W}}\right) - s^T \eta \odot \mathtt{sat}(s/\phi) + s^T(d_a(t)+d(t)), \quad (2.35)\end{aligned}$$

where the fact $\tilde{M}^T = \tilde{M} = M - \hat{M}$ was used to simplify the first term in (2.35). (2.35) can be factored and rewritten as

$$\begin{aligned}\dot{L} = \mathrm{Tr}\left(\tilde{M}^T\left(y_r^{(n)} s^T + H^{-1}\dot{\tilde{M}}\right)\right) + \mathrm{Tr}\left(\tilde{V}^T\left(x s^T \hat{W}^T \hat{\sigma}' + G^{-1}\dot{\tilde{V}}\right)\right) \\ + \mathrm{Tr}\left(\tilde{W}^T\left(\hat{\sigma} s^T + F^{-1}\dot{\tilde{W}}\right)\right) - s^T \eta \odot \mathtt{sat}(s/\phi) \\ + s^T(d_a(t)+d(t)). \quad (2.36)\end{aligned}$$

Since the "ideal" values of M, V, and W are all assumed to be constant, $\dot{\tilde{M}} = -\dot{\hat{M}}$, $\dot{\tilde{V}} = -\dot{\hat{V}}$, and $\dot{\tilde{W}} = -\dot{\hat{W}}$. These relations are used to write

$$\dot{L} = \mathrm{Tr}\left(\tilde{M}^T\left(y_r^{(n)}s^T - H^{-1}\dot{\hat{M}}\right)\right) + \mathrm{Tr}\left(\tilde{V}^T\left(xs^T\hat{W}^T\hat{\sigma}' - G^{-1}\dot{\hat{V}}\right)\right) + \mathrm{Tr}\left(\tilde{W}^T\left(\hat{\sigma}s^T - F^{-1}\dot{\hat{W}}\right)\right) - s^T\eta \odot \mathtt{sat}(s/\phi) + s^T(d_a(t) + d(t)). \quad (2.37)$$

Substituting the update rules (2.11), (2.12), and (2.13) into (2.37) gives

$$\dot{L} = -s^T\eta \odot \mathtt{sat}(s/\phi) + s^T\delta(t), \quad (2.38)$$

which, by evaluating the expanded scalar form, can be combined and rewritten as

$$\dot{L} = s^T\mathtt{sat}(s/\phi) \odot (\delta(t) - \eta), \quad (2.39)$$

where $\delta(t) = d_a(t) + d(t)$. Note that, according to the definition of $\mathtt{sat}(s/\phi)$ in (2.9), each term in the inner product $s_i \cdot \mathtt{sat}(s_i/\phi) \leq |s_i|$, for $i = 1, 2, \ldots, m$. Under the assumption $\delta(t)$ is upper bounded such that $\forall t \geq 0 : |\delta_i(t)| \leq \Delta$, for $i = 1, 2, \ldots, m$; (2.39) simplifies to the inequality relation

$$\dot{L} \leq |s|^T(\Delta - \eta), \quad (2.40)$$

where $|s|^T = [|s_1|, |s_2|, \ldots, |s_m|]$ is an elementwise absolute value. Since $L > 0$, selecting gains $\eta_i > \Delta$, $\forall i = 1, 2, \ldots, m$ forces $\dot{L} \leq 0$. Using LaSalle's invariance principle and Barbalat's lemma, $s \to 0$ as $t \to \infty$; while $\tilde{M}$, $\tilde{W}$, and $\tilde{V}$ are all

bounded in time [9, 55, 56]. Further, since $\tilde{M}$, $\tilde{W}$, and $\tilde{V}$ are bounded in time, the estimates $\hat{M}$, $\hat{W}$, and $\hat{V}$ are also bounded in time.

□

2.7 Simulation Results

To validate the developed controller and highlight its ability to extend conventional model-based SMC, a three-link robotic arm was simulated, shown in Figure 2.2.

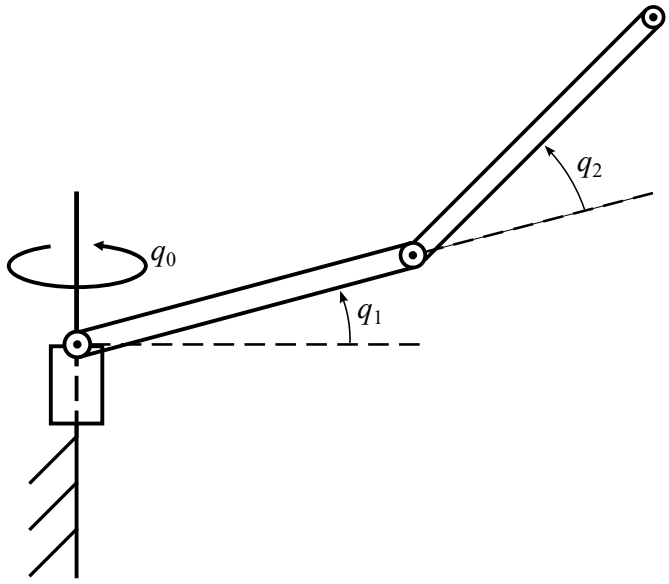

Figure 2.2: The simulated three-link manipulator with a revolute base [57].

The joints $q = [q_0, q_1, q_2]^T$ are controlled via torque control by the control law in (2.10) to track a desired trajectory in the joint space. The desired trajectories for each joint are given in Table 2.1.

Table 2.1: Desired joint trajectories during the simulation.

Joint	**Trajectory (deg)**
$q_0(t)$	$60\sin(t/10)$
$q_1(t)$	$40\sin(t/5)+40$
$q_2(t)$	$35\sin(t/8)$

The learned coefficient matrix and neural network parameters in the controller are updated by integrating (2.11), (2.12), and (2.13) at each timestep. Additional simulation details, including the robot model used, are described in Appendix A.1.

To demonstrate the adaptivity of the controller, the simulated manipulator picks up a mass with the end effector during the experiment, thereby changing the mass-moment matrix, M, and state-function, $f(x)$, of the robot's dynamics in (2.1). In conventional model-based SMC, a load margin (that the conventional SMC controller is robust to) is built into the assumed model via mass uncertainty on the end effector [19]. However, if the mass picked up by the end effector exceeds this margin, the stability guarantees on the conventional SMC controller are no longer valid. Within the developed controller, the adaptive elements simply adjust to the new system dynamics when the external load is applied.

The state vector used as input to the neural network is

$$x = \begin{bmatrix} q_0 & q_1 & q_2 & \dot{q}_0 & \dot{q}_1 & \dot{q}_2 & 1 \end{bmatrix}^T \tag{2.41}$$

since the robot dynamics are generally known to be functions of joint positions and joint velocities. The 1 is appended to incorporate a bias as described in

Section 2.4.3. The neural network is initialized with all weights sampled from a uniform distribution on the interval $[-0.01, 0.01]$ with no pretraining, and $\hat{M}$ is initialized as $\hat{M}(t = 0) = \texttt{diag}([0.1, 0.1, 0.1])$. This initialization is a rough estimate of M in the real system dynamics. The robot arm is initialized at $q(t = 0) = [10^\circ, 50^\circ, 10^\circ]^T$ at rest, giving an initial error of 10° on each joint. All hyperparameters used by the developed controller in the simulation are given in Table 2.2, which were tuned to highlight the learning elements of the controller. The controller is run at a fidelity of 100 Hz in each simulation.

Table 2.2: Controller hyperparameters used in the simulation.

Hyperparameter	**Value**
η	$[0.01, 0.001, 0.001]^T$
ϕ	$[0.06, 0.002, 0.002]^T$ rad
λ	$\texttt{diag}([1, 1, 1])$
n_H	25
F	$\texttt{diag}([1.5,] \times 25)$
G	$\texttt{diag}([0.4,] \times 7)$
H	$\texttt{diag}([0.9, 0.9, 0.1])$

The model-based conventional robust SMC, implemented for comparison and labeled "Conventional SMC" in the following plots, is built and tuned to handle an end effector load margin of up to 0.2 kg. This controller is described in detail in Appendix A.2.

Two different simulation experiments are presented in this chapter: in the first experiment, the manipulator end effector picks up a 0.15 kg mass at $t = 30$ s, which is well within the designed load margin for the conventional SMC

controller. The trajectories over time for each joint are shown in Figure 2.3, the errors over time for each joint are shown in Figure 2.4, and the sliding variables over time for each joint are shown in Figure 2.5. To see the adaptation of the proposed controller over time, the learned values of $\hat{M}$ and the Frobenius norm of the neural network layers over time are shown in Figure 2.6. Some plots are zoomed in to display controller behavior near the zero values.

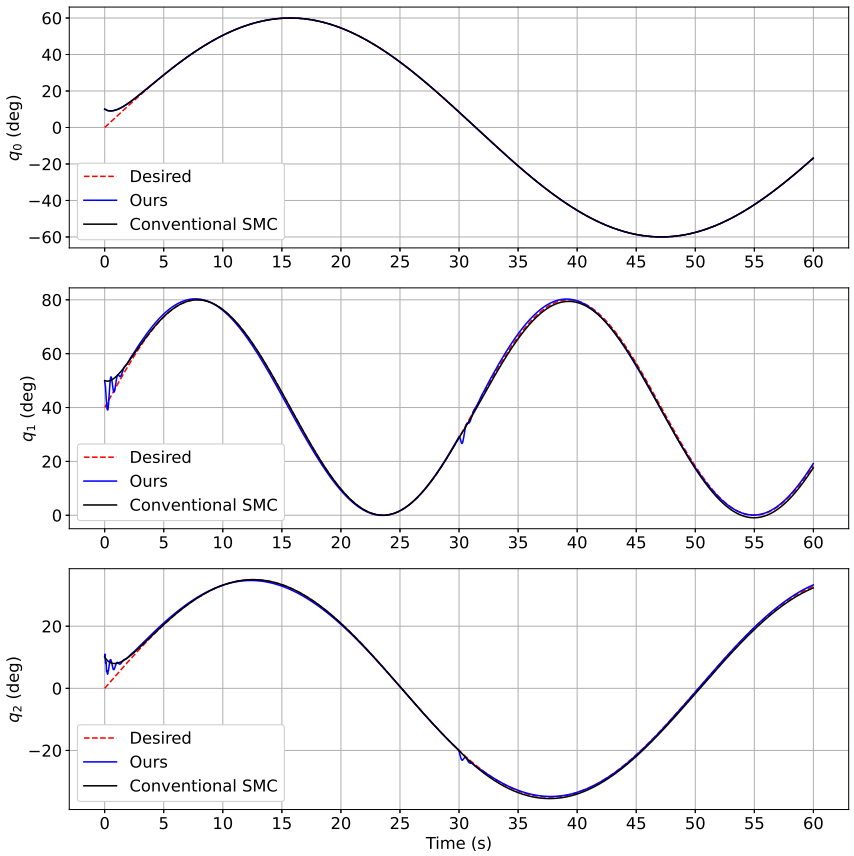

Figure 2.3: Joint trajectories over time for the robot arm simulation (0.15 kg mass loaded onto the end effector at $t = 30$ s).

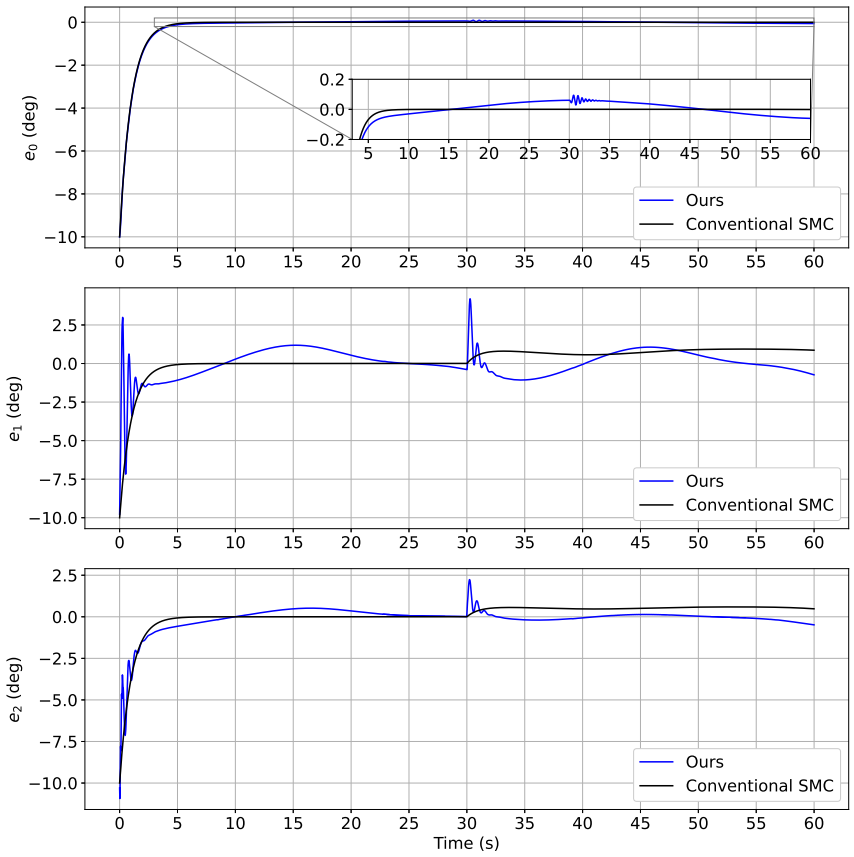

Figure 2.4: Joint errors over time for the robot arm simulation (0.15 kg mass loaded onto the end effector at $t = 30$ s).

Comparing the controller developed in this chapter with the conventional SMC controller in Figure 2.4, the conventional controller performs ideally while unloaded on the interval $t \in [0, 30)$ s, due to perfect knowledge of the system model. Once the end effector is loaded at $t = 30$ s, steady state error persists in joints 1 and 2, due to the boundary layer required in the conventional SMC controller for chatter attenuation – the primary drawback of conventional SMC. The developed controller is able to adapt to the new load and effectively control

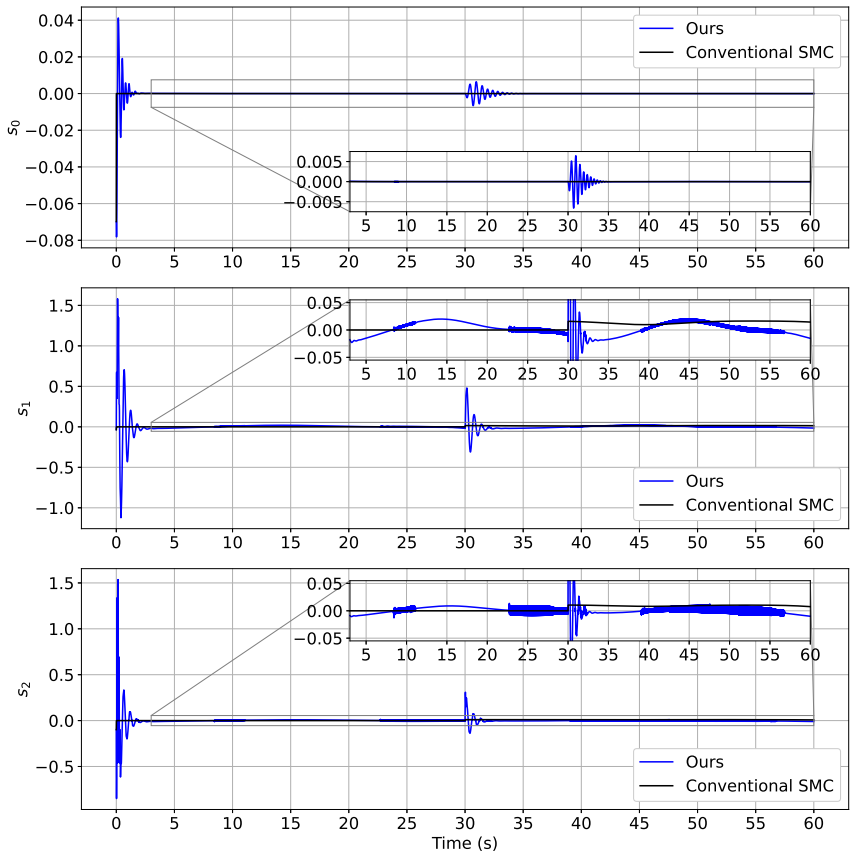

Figure 2.5: Sliding variables over time for the robot arm simulation (0.15 kg mass loaded onto the end effector at $t = 30$ s).

the arm in all joints to similar performance as the unloaded case. The adaptation can clearly be seen in Figure 2.6, where the learning elements adjust to the new load at $t = 30$ s. In this experiment, the robust advantages of conventional SMC can clearly be seen, but a strict load margin and system model (including perfect knowledge of all system parameters) are assumed known *a priori* in this case.

In the second experiment, the manipulator end effector picks up a 0.5 kg mass at $t = 30$ s, which is outside of the designed load margin for the conven-

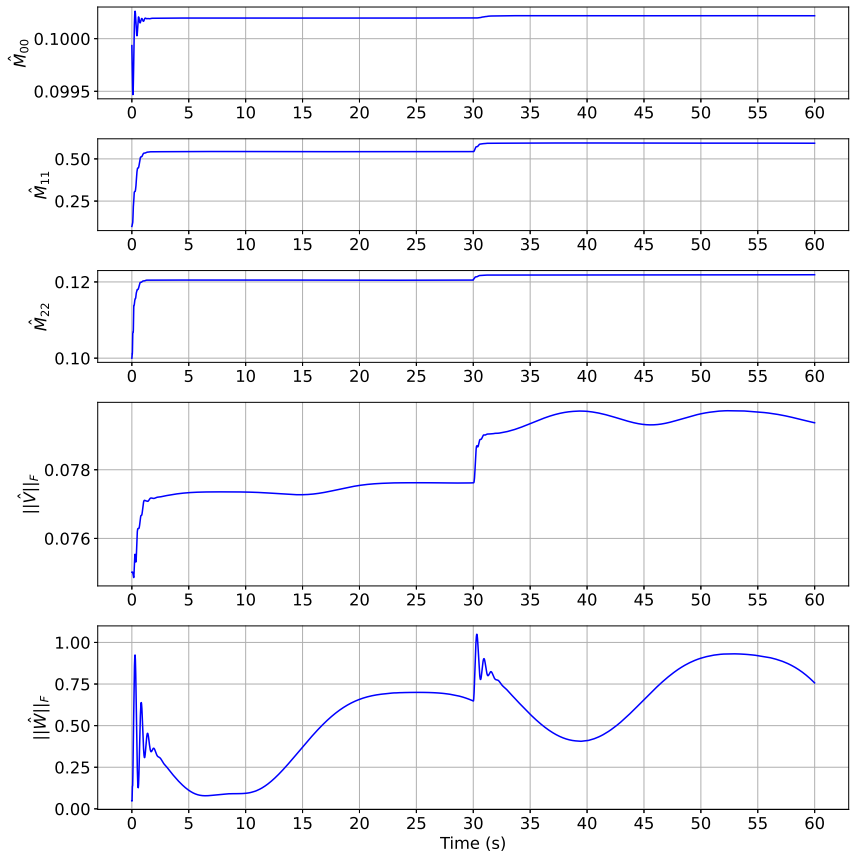

Figure 2.6: Learning updates over time for the robot arm simulation (0.15 kg mass loaded onto the end effector at $t = 30$ s).

tional SMC controller. This experiment was designed to show how the learning elements of the developed controller can extend conventional SMC, especially when controller design margins are violated. Similar to the first experiment, the trajectories over time for each joint is shown in Figure 2.7, the errors over time for each joint are shown in Figure 2.8, and the sliding variables over time for each joint are shown in Figure 2.9. The learned values of $\hat{M}$ and the Frobenius norm of the neural network layers over time are shown in Figure 2.10.

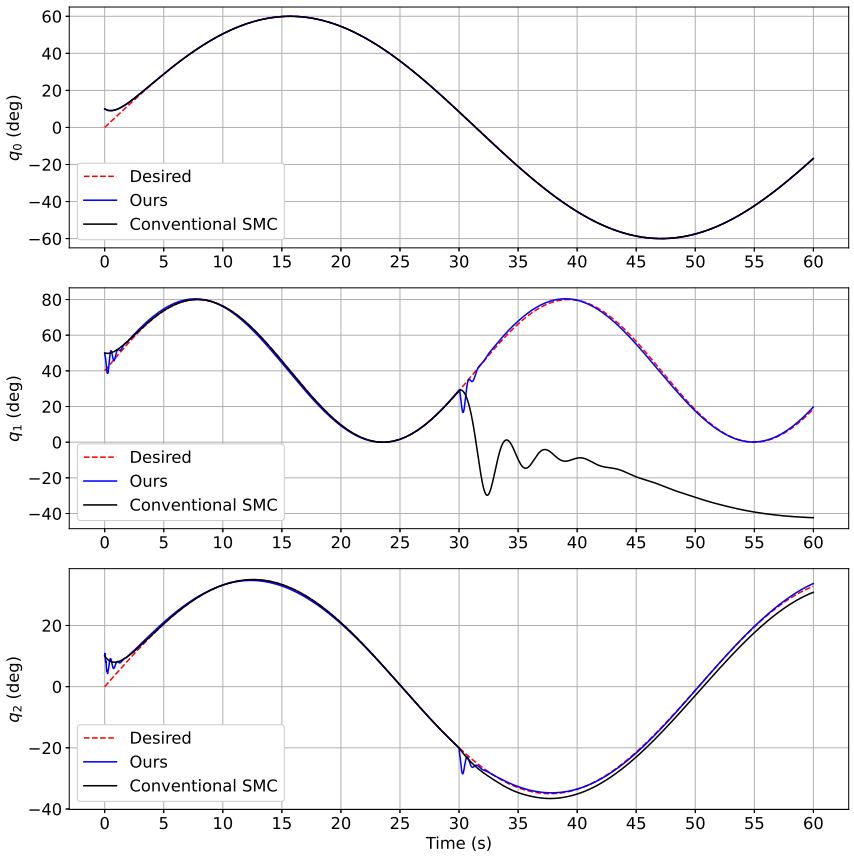

Figure 2.7: Joint trajectories over time for the robot arm simulation (0.5 kg mass loaded onto the end effector at $t = 30$ s).

In this experiment, the conventional SMC controller fails to be robust to the 0.5 kg load on its end effector, as expected, as this mass is outside of the controller's designed load margin. Figure 2.8 shows the errors of joints 1 and 2 increasing dramatically at $t = 30$ s, with joint 1 failing under the moment arm of the applied load. The developed controller is again able to resume near-nominal unloaded performance after learning converges on the new dynamics near $t = 33$ s. In Figure 2.10, both the mass moment matrix and neural network can again be

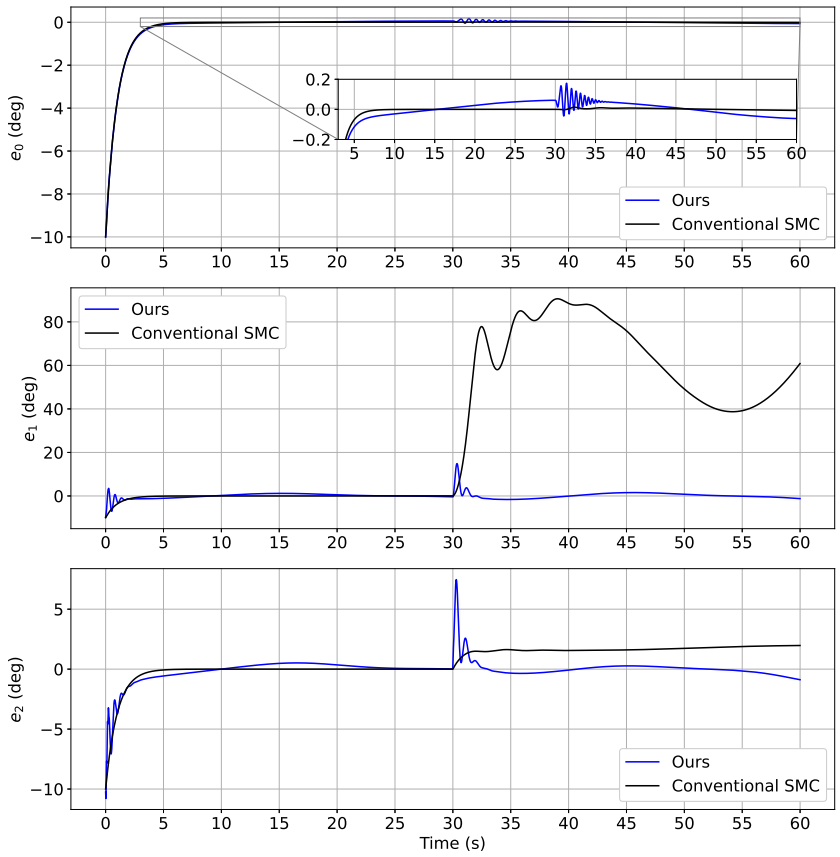

Figure 2.8: Joint errors over time for the robot arm simulation (0.5 kg mass loaded onto the end effector at $t = 30$ s).

seen to adjust to the increased load at $t = 30$ s. Note that the general shape of the updates in Figure 2.10 closely follows the shape of the updates in Figure 2.6 for the 0.15 kg case, but with differing magnitudes, since the applied load is different.

For general remarks, it should be noted that while the proposed controller assumes system dynamics following (2.1), no other *a priori* modeling of the system parameters is required to implement the developed controller. A large benefit of the developed controller comes with control of systems that are complex

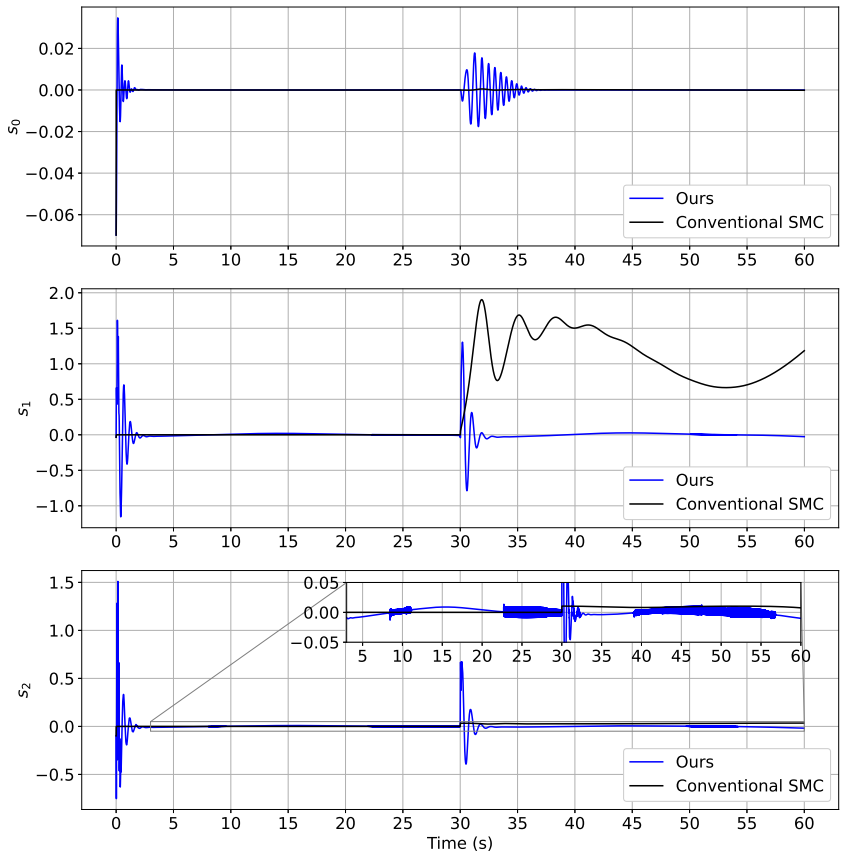

Figure 2.9: Sliding variables over time for the robot arm simulation (0.5 kg mass loaded onto the end effector at $t = 30$ s).

and difficult to model: the adaptive elements of the proposed controller learn a suitable model for control fully online, removing the need for expensive system identification.

Further, during testing of the proposed controller, the benefit of chatter attenuation using the neural network term was discovered. In the above experiments, the developed controller was tuned for adaptation over control accuracy, to showcase a possible scenario where the adaptive SMC prevails over conven-

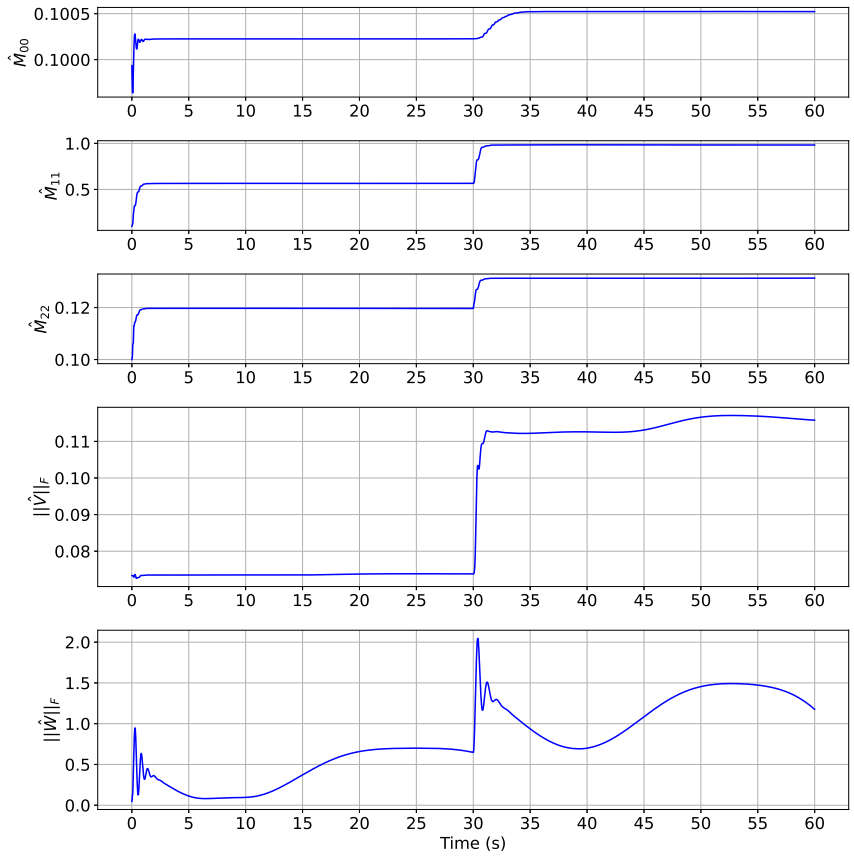

Figure 2.10: Learning updates over time for the robot arm simulation (0.5 kg mass loaded onto the end effector at $t = 30$ s).

tional robust SMC. When the developed controller is tuned for control accuracy versus adaptation, the neural network term in the controller allows a low gain, η, on the SMC term in (2.10). With a low gain on the discontinuous SMC term, a low boundary layer value, ϕ, can be used to prevent chattering. This analysis can be seen mathematically in the Lyapunov proof of Theorem 2.6.1 by assuming the dynamics of (2.1) to be slightly modified to

$$My^{(n)} + f'(x) + d(t) = u(t)$$

such that $f(x, d) = f'(x) + d(t)$ is a new grouped state function and disturbance term to be approximated online by the NN. Following similar analysis to the proof of Theorem 2.6.1 and assuming d_a is upper bounded such that $\forall t \geq 0 : |d_a| \leq \Delta_a$, for $i = 1, 2, \ldots, m$; (2.39) simplifies to the inequality relation

$$\dot{L} \leq |s|^T(\Delta_a - \eta),$$

where d_a is the approximation error of the NN due to higher-order effects and finiteness as defined in (2.28). Thus, selecting gains $\eta_i > \Delta_a$, $\forall i = 1, 2, \ldots, m$ guarantees $\dot{L} \leq 0$. The neural network term in the proposed controller effectively learns to compensate for the terms $f(x) + d(t)$ in (2.1). Further, during implementation, d_a decreases over time as the NN learns over time, thus requiring a lower gain η over time for guaranteed convergence. This lower-gain requirement is the source of the reduced chatter that was experienced during the simulation and testing of the developed controller. The methodology described in this chapter can thus result in an overall higher control accuracy while alleviating chatter, the primary drawback of switching controllers such as SMC.

2.8 Extension: Control with Nondiagonal M Estimation

Note that the controller developed in the chapter above assumes the coefficient matrix M in (2.1) to be constant and diagonal. While this assumption is

reasonable from a controls approximation perspective, it is desirable to preserve as much *a priori* structure in the dynamics as possible to simplify the learning problem. This is especially true in the online learning problem for control considered in this chapter, since efficient computation and fast convergence are necessary for effective control of real-world systems. This section thus details extending the developed controller above to include systems that are accurately modeled by (2.1) with a symmetric, positive definite coefficient matrix M. The symmetric, positive definite M is consistent with serial robotic arms and other physical systems [57].

2.8.1 Extended Control and Update Laws

This section assumes the system dynamics given in (2.1) with a constant, symmetric, positive definite, unknown coefficient matrix M. The central idea of the control law extension is to map (or *project*) the current estimate of M, $\hat{M}$, to an allowable set of symmetric positive definite matrices. The general control law of (2.10) is thus modified as

$$u = \bar{M} y_r^{(n)} + \hat{f}(x) + \eta \odot \mathtt{sat}(s/\phi), \tag{2.42}$$

where $\bar{M}$ is the current projected estimate of M, to be described below. The NN weight update laws of (2.12) and (2.13) are used, with the evolution of the estimate $\hat{M}$ modified to

$$\dot{\hat{M}} = \frac{\gamma}{2}\left(y_r^{(n)} s^T + s(y_r^{(n)})^T\right) + H(\hat{M} - \bar{M}) + (\hat{M} - \bar{M})H, \tag{2.43}$$

where $\gamma \in \mathbb{R}^+$ is a constant design gain and $H \in \mathbb{R}^{m \times m}$ is now a constant design diagonal Hurwitz matrix [58, 59]. Note that the update in (2.43) is symmetric as required. Assuming the spectral decomposition of $\hat{M} = U\hat{\Lambda}U^T$ where $\hat{\Lambda} = \text{diag}(\hat{\lambda})$ and $\hat{\lambda} = [\hat{\lambda}_1, \hat{\lambda}_2, \ldots, \hat{\lambda}_m]^T$, the projection of the estimate $\hat{M}$ to the symmetric positive definite estimate $\bar{M}$ is found as

$$\bar{M} = U\bar{\Lambda}U^T, \tag{2.44}$$

where $\bar{\Lambda} = \text{diag}(\text{clip}(\hat{\lambda}))$ is constructed by projecting the spectrum of $\hat{M}$ in the defined allowable eigenspace $\lambda_i \in [\alpha_{pd}, \beta_{pd}]$:

$$\text{clip}(\hat{\lambda}) = \begin{cases} \beta_{pd} & \text{if } \hat{\lambda}_i > \beta_{pd} \\ \alpha_{pd} & \text{if } \hat{\lambda}_i < \alpha_{pd} \\ \hat{\lambda}_i & \text{otherwise} \end{cases} \tag{2.45}$$

for $i = 1, 2, \ldots, m$; where α_{pd}, β_{pd} are positive finite constants. This update rule ensures the symmetric and positive definiteness of the estimate of M to be within the predefined eigenvalue bounds.

2.8.2 Stability Analysis of the Extended Controller

The control goals of the controller given in (2.42) with the update rules given in (2.43), (2.12), and (2.13) remain the same as in Section 2.6. The Lyapunov function candidate must be slightly modified to incorporate the boundedness of the clipped positive definite estimate, $\bar{M}$. To show this, the following

lemma is required to establish positive definiteness of the Lyapunov function candidate to be used in the stability proof.

Lemma 2.8.1. *Let the unknown coefficient matrix M of the dynamical system in* (2.1) *be symmetric and positive definite, with eigenvalues $\lambda = [\lambda_1, \lambda_2, \ldots, \lambda_m]^T$. Let $\lambda_i \in [\alpha_{pd}, \beta_{pd}]$ for $i = 1, 2, \ldots, m$; where α_{pd}, β_{pd} are positive finite constants. The function*

$$L_{pd} = \text{Tr}\left((M - \hat{M})^2 - (\hat{M} - \bar{M})^2\right) \tag{2.46}$$

is positive definite such that $L_{pd} \geq 0$, with $L_{pd} = 0$ occurring at $M = \hat{M} = \bar{M}$.

Proof. The Lyapunov function candidate in (2.46) can be expanded as

$$L_{pd} = \text{Tr}\left(M^2 - 2M\hat{M} - \bar{M}^2 + 2\hat{M}\bar{M}\right), \tag{2.47}$$

where the fact that $M = M^T$, $\hat{M} = \hat{M}^T$, and $\bar{M} = \bar{M}^T$ has been used to combine like terms. Adding and subtracting $\bar{M}^2 + 2M\bar{M}$ to (2.47) gives

$$L_{pd} = \text{Tr}\left(M^2 - 2M\hat{M} - \bar{M}^2 + 2\hat{M}\bar{M} + \bar{M}^2 + 2M\bar{M} - \bar{M}^2 - 2M\bar{M}\right), \tag{2.48}$$

which can be combined to render

$$L_{pd} = \text{Tr}\left((M - \bar{M})^2 + 2(M - \bar{M})(\bar{M} - \hat{M})\right). \tag{2.49}$$

The matrix trace is a linear mapping, which allows (2.49) to be written as

$$L_{pd} = \mathrm{Tr}\left((M - \bar{M})^2\right) + 2\,\mathrm{Tr}\left((M - \bar{M})(\bar{M} - \hat{M})\right). \tag{2.50}$$

The first term in (2.50) is quadratic and clearly positive for all $\bar{M} \neq M$. Using the spectral decompositions $M = V_m \Lambda V_m^T$, $\hat{M} = U\hat{\Lambda}U^T$, and $\bar{M} = U\bar{\Lambda}U^T$, the trace part of the second term in (2.50) can be written as

$$\mathrm{Tr}\left((M - \bar{M})(\bar{M} - \hat{M})\right) = \mathrm{Tr}\left((V_m \Lambda V_m^T - U\bar{\Lambda}U^T)(U\bar{\Lambda}U^T - U\hat{\Lambda}U^T)\right). \tag{2.51}$$

Note that, since M is real and symmetric, M is also Hermitian. Since M is Hermitian, the eigenbasis U is orthogonal, such that $UU^T = U^TU = I$. Using this fact, (2.51) can be simplified to

$$\mathrm{Tr}\left((M - \bar{M})(\bar{M} - \hat{M})\right) = \mathrm{Tr}\left((U^T V_m \Lambda V_m^T U - \bar{\Lambda})(\bar{\Lambda} - \hat{\Lambda})\right). \tag{2.52}$$

The eigenvalues of the unprojected estimate $\hat{M}$ can belong to one of three partitions on the eigenspace: $\Omega_- = \{i \mid \hat{\lambda}_i < \alpha_{pd}\}$, $\Omega = \{i \mid \alpha_{pd} \leq \hat{\lambda}_i \leq \beta_{pd}\}$, or $\Omega_+ = \{i \mid \hat{\lambda}_i > \beta_{pd}\}$ [59]. In the set Ω_-, the eigenvalues of $\bar{M}$ are projected via (2.45) to the lower bound α_{pd} such that $\bar{\lambda}_i = \alpha_{pd}$. In the set Ω, the eigenvalues of $\bar{M}$ are unprojected via (2.45) such that $\bar{\lambda}_i = \hat{\lambda}_i$. In the set Ω_+, the eigenvalues of $\bar{M}$ are projected via (2.45) to the upper bound β_{pd} such that $\bar{\lambda}_i = \beta_{pd}$. On these three partitions, the trace in (2.52) can thus be written in summation form

$$\mathrm{Tr}\left((M-\bar{M})(\bar{M}-\hat{M})\right) = \sum_{i\in\Omega_-}(\lambda_i-\alpha_{pd})(\alpha_{pd}-\hat{\lambda}_i) + \sum_{i\in\Omega}(\lambda_i-\hat{\lambda}_i)(\hat{\lambda}_i-\hat{\lambda}_i) + \sum_{i\in\Omega_+}(\lambda_i-\beta_{pd})(\beta_{pd}-\hat{\lambda}_i), \quad (2.53)$$

where it can easily be seen that the second term in (2.53) is zero. Since it is assumed that $\lambda_i \in [\alpha_{pd}, \beta_{pd}]$ and $\hat{\lambda}_i < \alpha_{pd}$ for $i \in \Omega_-$, the first term in (2.53) is nonnegative. Similarly, since $\lambda_i \in [\alpha_{pd}, \beta_{pd}]$ and $\hat{\lambda}_i > \beta_{pd}$ for $i \in \Omega_+$, the third term in (2.53) is also nonnegative. Thus, the second term in (2.50) is entirely nonnegative. This shows that the Lyapunov candidate function in (2.46) is positive definite such that $L_{pd} \geq 0$ with equality $L_{pd} = 0$ occurring at $M = \hat{M} = \bar{M}$.

□

Theorem 2.8.2. *Let the desired system trajectory, y_d, and its n time derivatives be continuous and bounded. Let $\delta(t) = d_a(t) + d(t)$ be the sum of approximation/learning error and external disturbance, to be defined. Assuming $\delta(t)$ is upper bounded such that $\forall t \geq 0 : |\delta_i(t)| \leq \Delta$, for $i = 1, 2, \ldots, m$; the dynamical system in* (2.1)*, the control input in* (2.42)*, and update rules in* (2.43)*,* (2.12)*, and* (2.13)*; the estimates $\bar{M}$, $\hat{M}$, $\hat{W}$, and $\hat{V}$ are bounded and $s \to 0$ as $t \to \infty$.*

Proof. Consider the composite Lyapunov function candidate of

$$L = \frac{1}{2}\left(s^T M s + \frac{1}{\gamma}L_{pd} + \mathrm{Tr}\left(\tilde{W}^T F^{-1}\tilde{W}\right) + \mathrm{Tr}\left(\tilde{V}^T G^{-1}\tilde{V}\right)\right), \tag{2.54}$$

where the notation $\tilde{W} = W - \hat{W}$, and $\tilde{V} = V - \hat{V}$ is again used, with L_{pd} defined in (2.46).

Differentiating (2.54) with respect to time gives

$$\dot{L} = s^T M \dot{s} + \frac{1}{2\gamma}\dot{L}_{pd} + \mathrm{Tr}\left(\tilde{W}^T F^{-1}\dot{\tilde{W}}\right) + \mathrm{Tr}\left(\tilde{V}^T G^{-1}\dot{\tilde{V}}\right). \tag{2.55}$$

The system in (2.1) can be rewritten using the sliding variable as $M\dot{s} = My_r^{(n)} + f + d - u$. Substituting the control law in (2.42) into this relation gives

$$M\dot{s} = \tilde{M}y_r^{(n)} + f(x) - \hat{f}(x) - \eta \odot \mathtt{sat}(s/\phi) + d(t), \tag{2.56}$$

where the notation $\tilde{M} = M - \bar{M}$ is used. The relation in (2.56) is substituted into the Lyapunov derivative in (2.55) to get

$$\begin{aligned}\dot{L} = s^T \tilde{M} y_r^{(n)} + \frac{1}{2\gamma}\dot{L}_{pd} + s^T\left(f(x) - \hat{f}(x)\right) + \mathrm{Tr}\left(\tilde{W}^T F^{-1}\dot{\tilde{W}}\right) \\ + \mathrm{Tr}\left(\tilde{V}^T G^{-1}\dot{\tilde{V}}\right) - s^T \eta \odot \mathtt{sat}(s/\phi) + s^T d(t).\end{aligned} \tag{2.57}$$

Following similar analysis from the proof of Theorem 2.6.1, (2.57) can be rewritten as

$$\dot{L} = \mathrm{Tr}\left(\tilde{M}^T y_r^{(n)} s^T\right) + \frac{1}{2\gamma}\dot{L}_{pd} + \mathrm{Tr}\left(\tilde{V}^T\left(x s^T \hat{W}^T \hat{\sigma}' + G^{-1}\dot{\tilde{V}}\right)\right) + \mathrm{Tr}\left(\tilde{W}^T\left(\hat{\sigma} s^T + F^{-1}\dot{\tilde{W}}\right)\right) - s^T \eta \odot \mathtt{sat}(s/\phi) + s^T(d_a(t) + d(t)), \quad (2.58)$$

where the third and fourth terms in (2.58) are equal to zero using the NN weight adaptation laws given in (2.12) and (2.13) and $d_a(t)$ is defined in (2.28). Using Lemma 2 of [59] and the update law in (2.43), the derivative of the composite part of the Lyapunov function $\dot{L}_{pd}$ is upper bounded such that

$$\dot{L}_{pd} \leq 2\gamma\, \mathrm{Tr}\left(-\frac{\tilde{M}}{2}\left(y_r^{(n)} s^T + s(y_r^{(n)})^T\right)\right). \quad (2.59)$$

Substituting the relation in (2.59) into (2.58) and the NN update rules in (2.12) and (2.13), the Lyapunov derivative is now bounded by the inequality

$$\dot{L} \leq \mathrm{Tr}\left(\tilde{M}^T y_r^{(n)} s^T\right) + \mathrm{Tr}\left(-\frac{\tilde{M}}{2}\left(y_r^{(n)} s^T + s(y_r^{(n)})^T\right)\right) - s^T \eta \odot \mathtt{sat}(s/\phi) + s^T(d_a(t) + d(t)), \quad (2.60)$$

which, combining the first two terms, can be written as

$$\dot{L} \leq \mathrm{Tr}\left(\frac{\tilde{M}^T}{2}\left(y_r^{(n)}s^T - s(y_r^{(n)})^T\right)\right) - s^T\eta \odot \mathtt{sat}(s/\phi) + s^T(d_a(t) + d(t)). \quad (2.61)$$

Note that the matrix $\frac{\tilde{M}^T}{2}$ is symmetric, and the matrix $y_r^{(n)}s^T - s(y_r^{(n)})^T$ is skew-symmetric. Since the trace of the product of a symmetric matrix and a skew-symmetric matrix is zero, the first term of (2.61) is equal to zero. Defining $\delta(t) = d_a(t) + d(t)$ to be the sum of approximation/learning error and external disturbance and assuming $\delta(t)$ is upper bounded such that $\forall t \geq 0 : |\delta_i(t)| \leq \Delta$, for $i = 1, 2, \ldots, m$; the inequality relation in (2.61) simplifies to

$$\dot{L} \leq |s|^T(\Delta - \eta). \quad (2.62)$$

Since the composite Lyapunov function L_{pd} in (2.46) is positive definite via Lemma 2.8.1, the entire Lyapunov function given in (2.54) is positive definite. The rest of the proof follows from the proof of Theorem 2.6.1, where the boundedness of $\hat{M}$ and $\bar{M}$ now come from the the composite Lyapunov function in (2.46).

□

2.8.3 Simulation Results

For validation and testing of the controller in (2.42) and the positive definite update law in (2.43), the three-link robot arm in Section 2.7 is simulated. The joints of the robot arm are again controlled via torque control to follow the

desired joint space trajectory given in Table 2.1. The control law in (2.42) is used, with the learned parameters updated via (2.12), (2.13), and the projection in (2.43).

The robot's end effector is unloaded in the simulation on the interval $t \in [0, 30)$ s. As in Section 2.7, two separate experiments are run: a 0.15 kg load placed onto the end effector at $t = 30$ s, and a 0.5 kg load placed onto the end effector at $t = 30$ s. This is done to show the effectiveness of the developed controller(s) over conventional model-based SMC with a built-in load margin. To compare the impact of the positive definite M estimation of the controller described in Section 2.8 with the diagonal M estimate of the controller described in Section 2.5, the hyperparameters of the controller with the M projection are left the same as the non-projected controller when applicable. Relevant hyperparameters of the M-projection controller are given below in Table 2.3.

Table 2.3: M-projection controller hyperparameters used in the simulation.

Hyperparameter	**Value**
η	$[0.01, 0.001, 0.001]^T$
ϕ	$[0.06, 0.002, 0.002]^T$ rad
λ	$\texttt{diag}([1, 1, 1])$
n_H	25
F	$\texttt{diag}([1.5,] \times 25)$
G	$\texttt{diag}([0.4,] \times 7)$
H	$\texttt{diag}([-5, -5, -5])$
γ	0.9
α_{pd}	0.001
β_{pd}	0.55

For the M-projection controller, the state vector input to the NN is given in (2.41). The NN is initialized with all weights sampled from a uniform distribution on the interval $[-0.01, 0.01]$ with no pretraining, with $\hat{M}(t = 0) = \texttt{diag}([0.1, 0.1, 0.1])$ and $q(t = 0) = [10°, 50°, 10°]^T$ at rest, giving an initial error of $10°$ on each joint.

In the plots below, the controller with update laws described in Section 2.5 is labeled as "ONSMC w/o Proj.," with all hyperparameters and initializations as described in Section 2.7 above. The controller with positive definite M projection described in Section 2.8 is labeled as "ONSMC w/ Proj.," and the conventional non-adaptive SMC described in Section 2.7 and Appendix A.2 is labeled as "Conventional SMC." Each controller is run at a fidelity of 100 Hz in the simulations.

The trajectories over time for each joint are shown in Figure 2.11, the errors over time for each joint are shown in Figure 2.12, and the sliding variables over time for each joint are shown in Figure 2.13. To compare parameter adaptation of each controller, the Frobenius norm of the coefficient matrix and the neural network layers over time are shown in Figure 2.14. The eigenvalues of the projected matrix $\bar{M}$ are plotted in Figure 2.15. Some plots are zoomed in to display controller behavior near the zero values.

While both of the controllers developed in this chapter tend to outperform conventional SMC in control error when the end effector is loaded, the M-projection controller does not provide any noticeable reduction in control error. The primary difference effect of the M-projection can be seen in the learning plot of Figure 2.14, where the Frobenius norm of $\hat{M}$ generally remains the same

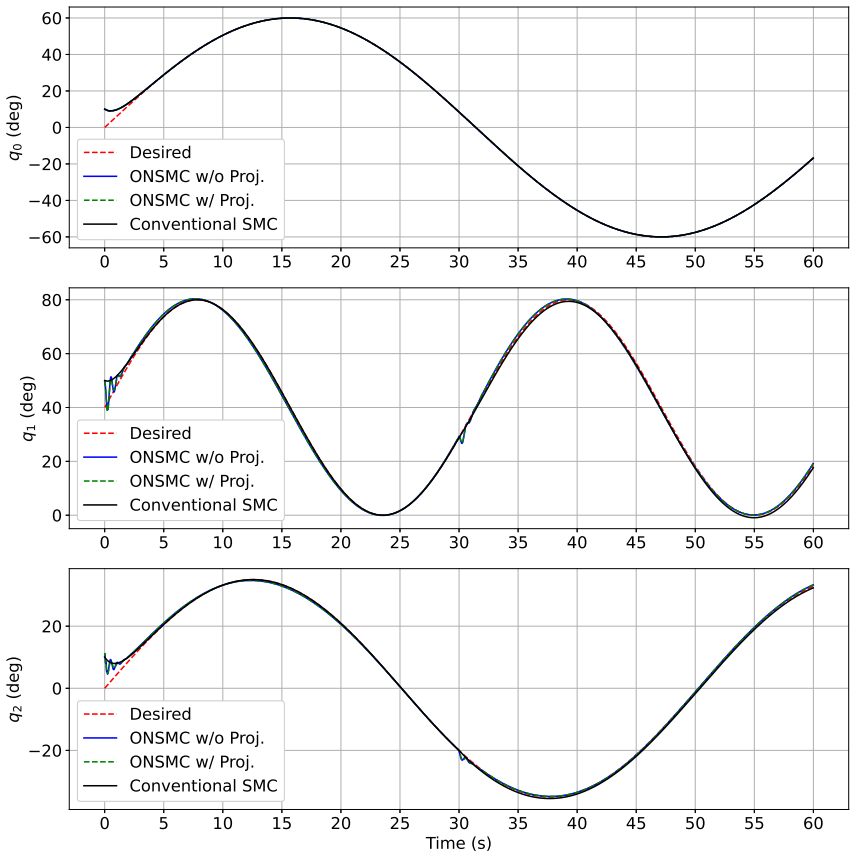

Figure 2.11: Comparison of joint trajectories over time for the robot arm simulation (0.15 kg mass loaded onto the end effector at $t = 30$ s).

throughout the simulation. Interestingly, this offloads the learning required to the first layer of the NN, $\hat{V}$. The second layer of the NN, $\hat{W}$, remains the same in its norm. The output of the first layer of the NN acts as a basis for the second layer of the NN. The behavior in Figure 2.14 shows that, while the same general shape of control input is required for the desired trajectory following, the output magnitude required from the NN term is increased to achieve the same control effort. In the controller without projection, the Frobenius norm of $\hat{M}$ clearly

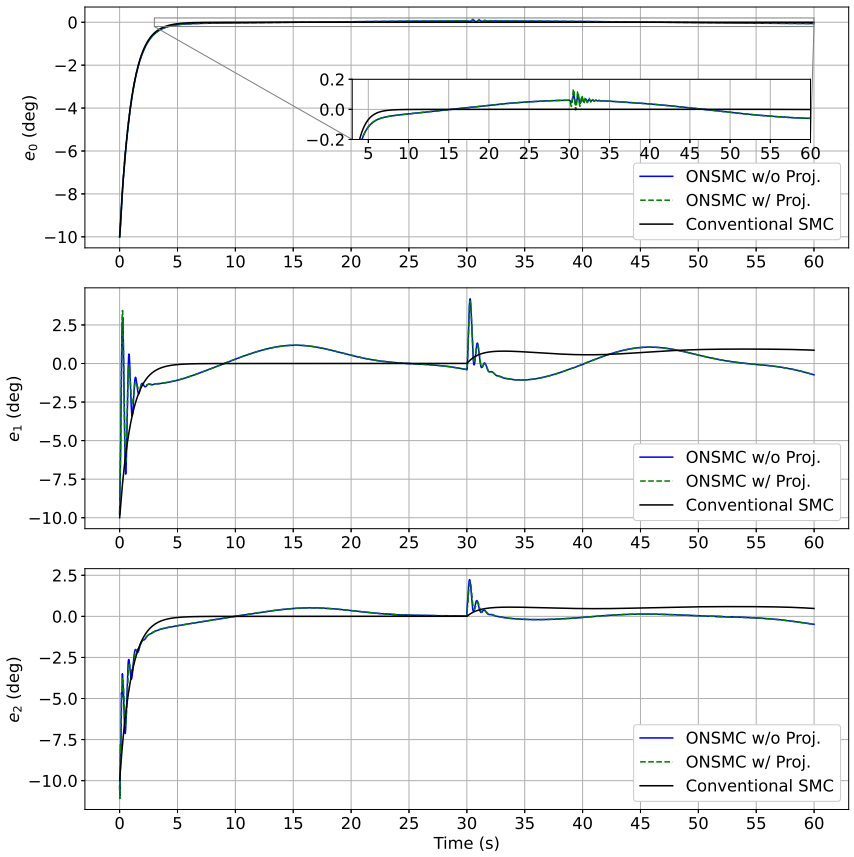

Figure 2.12: Comparison of joint errors over time for the robot arm simulation (0.15 kg mass loaded onto the end effector at $t = 30$ s).

increases when the end effector is loaded at $t = 30$ s. Each eigenvalue of $\bar{M}$ can be seen in Figure 2.15 to remain in the allowable eigenspace determined by the eigenprojection of (2.45).

Next, the 0.5 kg load on the end effector at $t = 30$ s is simulated. Similar to above, the trajectories over time for each joint are shown in Figure 2.16, the errors over time for each joint are shown in Figure 2.17, and the sliding variables over time for each joint are shown in Figure 2.18. The Frobenius norm of the coefficient

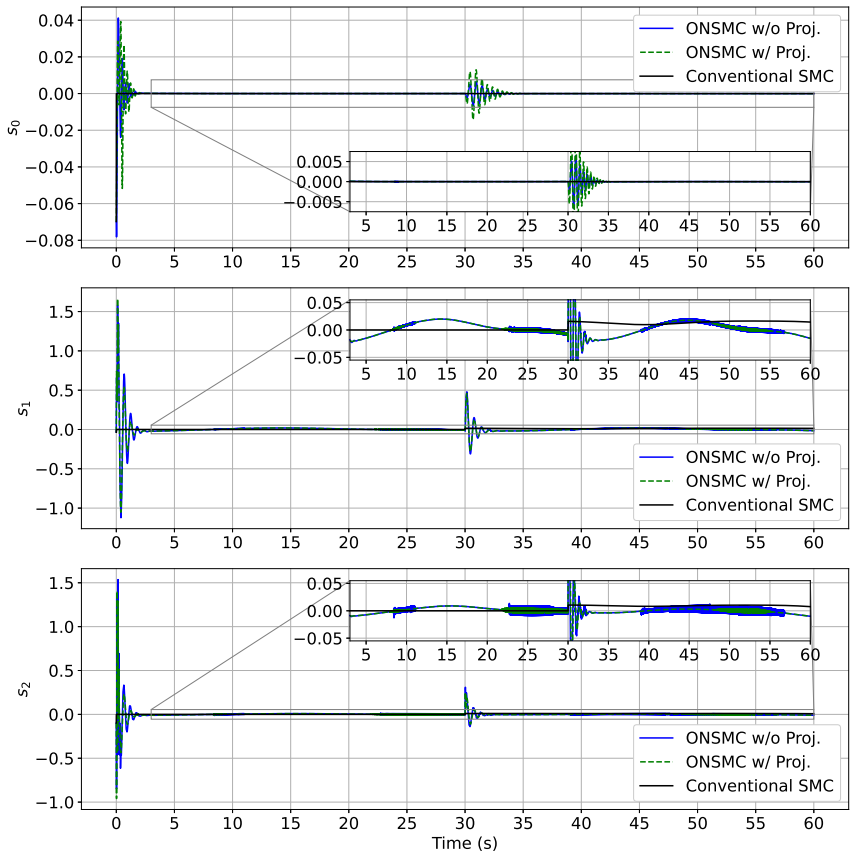

Figure 2.13: Comparison of sliding variables over time for the robot arm simulation (0.15 kg mass loaded onto the end effector at $t = 30$ s).

matrix and the neural network layers over time are shown in Figure 2.19, and the eigenvalues of the projected matrix $\bar{M}$ are plotted in Figure2.20.

For the 0.5 kg case, both of the controllers developed in this chapter again outperform conventional SMC, which fails when the end effector is loaded by 0.5 kg at $t = 30$ s. The M-projection controller does not provide any noticeable reduction in control error for this experiment. Interestingly, a slight reduction in chatter in the M-projection controller can be seen in Figure 2.18, notably near

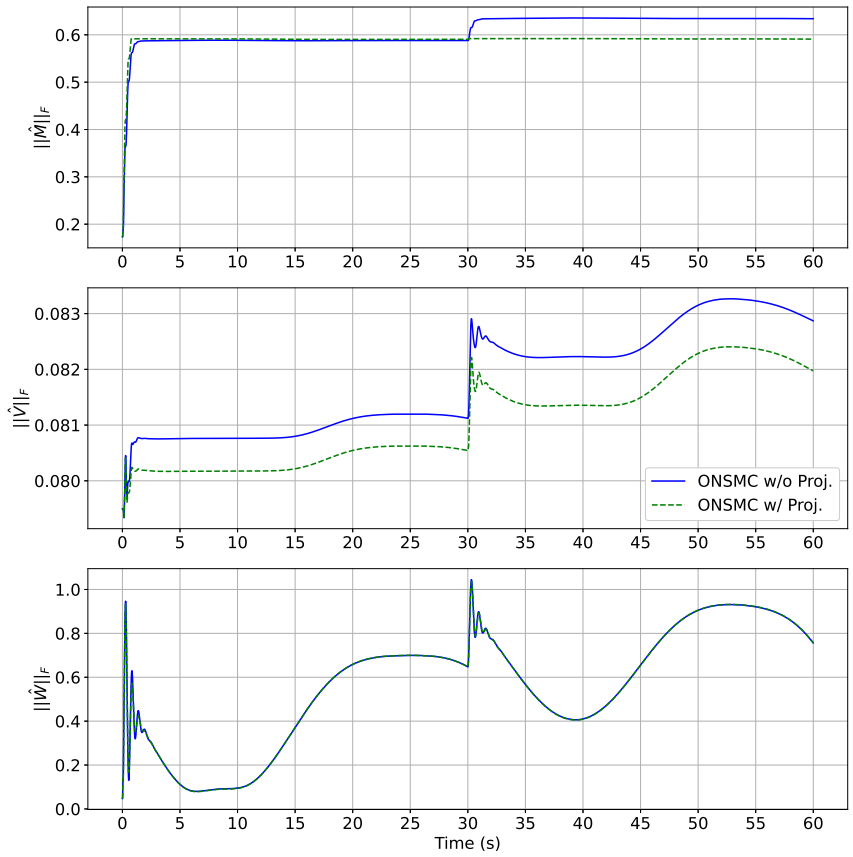

Figure 2.14: Comparison of learning updates over time for the robot arm simulation (0.15 kg mass loaded onto the end effector at $t = 30$ s).

$t = 25$ s and $t = 50$ s in s_2. Similar to the 0.15 kg experiment, Figure 2.19 again shows that the Frobenius norm of $\hat{M}$ remains the same throughout the simulation, thereby offloading the learning required to the first layer of the NN, $\hat{V}$. The second layer of the NN, $\hat{W}$, again follows the behavior of the non-projected controller. The eigenvalues of $\bar{M}$ again remain inside the allowable eigenspace determined by the eigenprojection of (2.45), as shown in Figure 2.20.

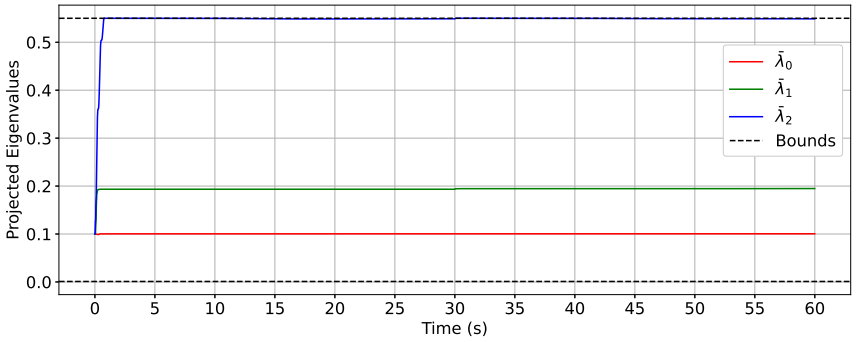

Figure 2.15: Comparison of eigenvalues of the estimate $\bar{M}$ over time for the robot arm simulation (0.15 kg mass loaded onto the end effector at $t = 30$ s).

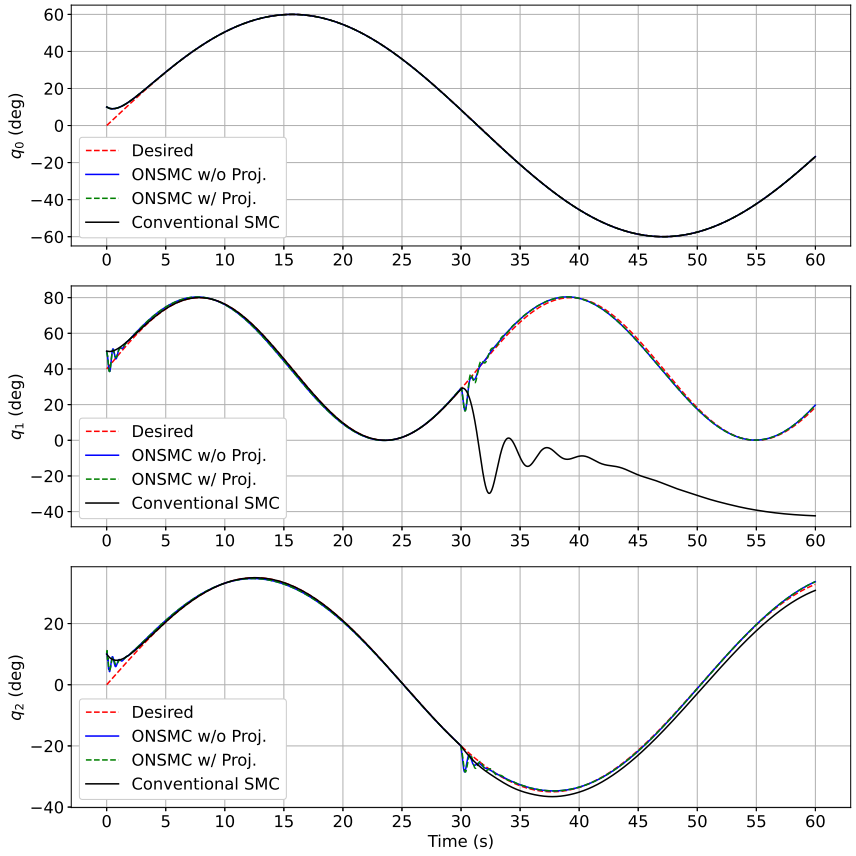

Figure 2.16: Comparison of joint trajectories over time for the robot arm simulation (0.5 kg mass loaded onto the end effector at $t = 30$ s).

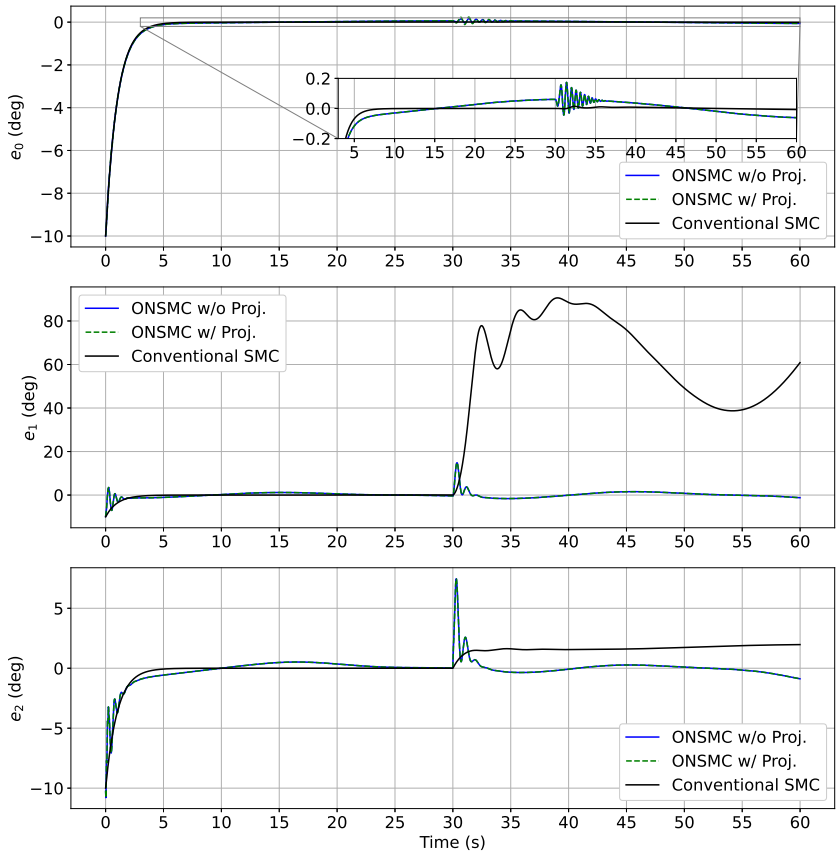

Figure 2.17: Comparison of joint errors over time for the robot arm simulation (0.5 kg mass loaded onto the end effector at $t = 30$ s).

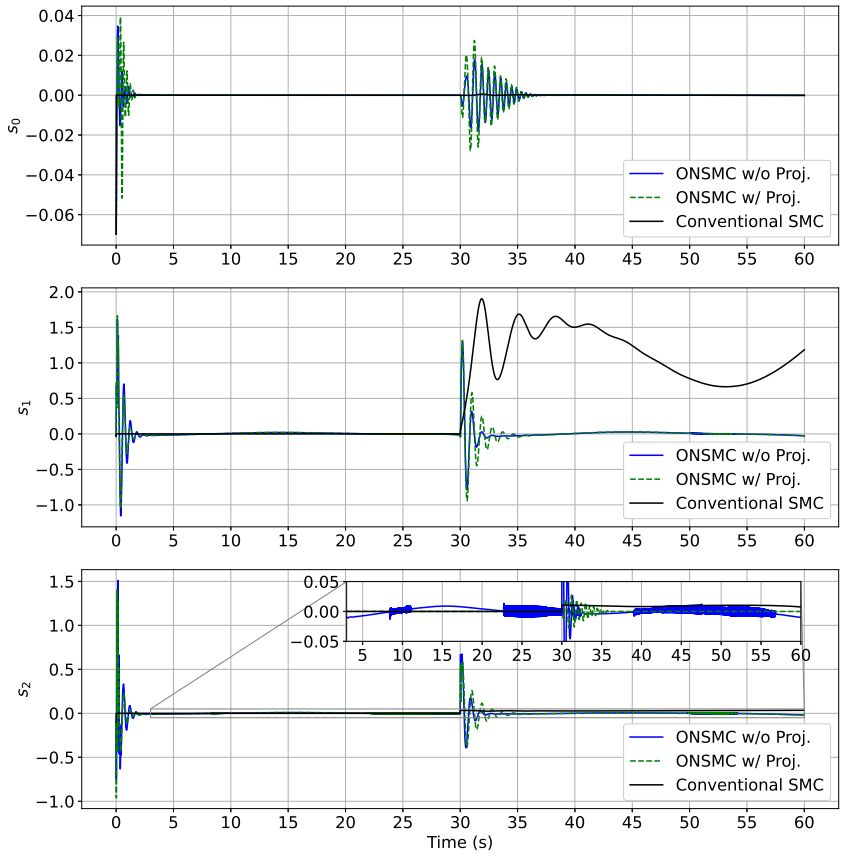

Figure 2.18: Comparison of sliding variables over time for the robot arm simulation (0.5 kg mass loaded onto the end effector at $t = 30$ s).

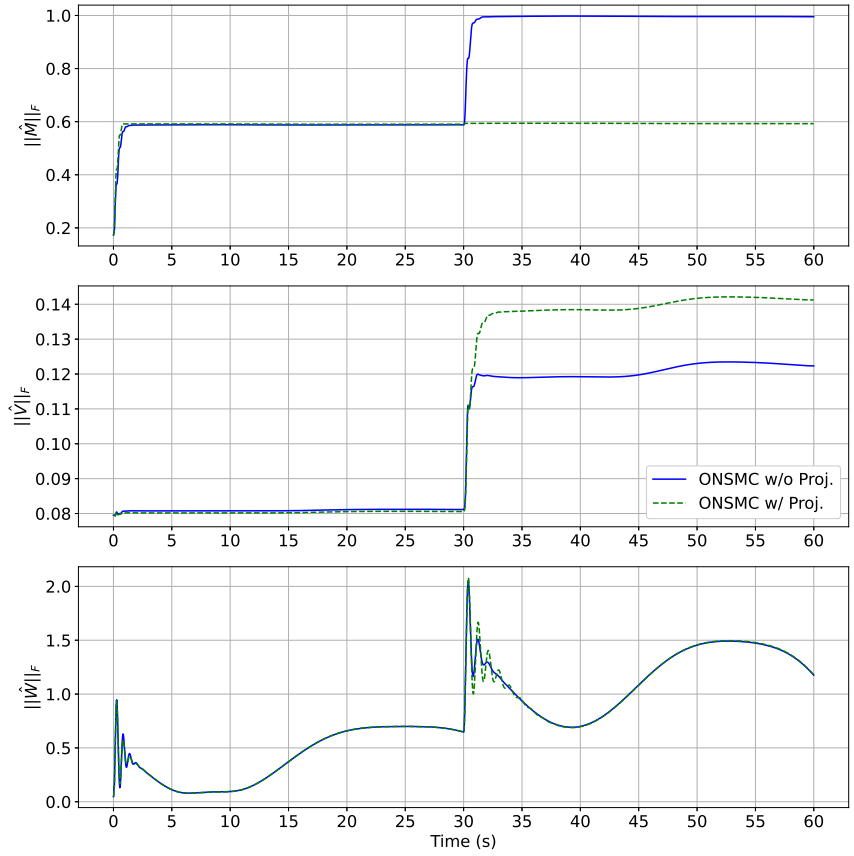

Figure 2.19: Comparison of learning updates over time for the robot arm simulation (0.5 kg mass loaded onto the end effector at $t = 30$ s).

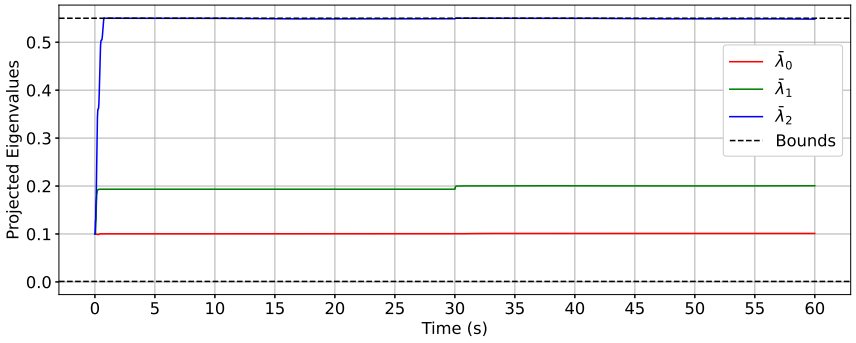

Figure 2.20: Comparison of eigenvalues of the estimate $\bar{M}$ over time for the robot arm simulation (0.5 kg mass loaded onto the end effector at $t = 30$ s).

Chapter 3. Online Learning-Based Control of Spacecraft and Quadcopters

This chapter discusses progress in expanding the AI/ML-based online adaptive controllers developed in Chapter 2 for learning-based control of aerospace systems. The two aerospace control problems discussed are rigid-body spacecraft attitude control and full quadcopter control. Both of these problems present unique aspects of the developed controller, including reformulating the sliding manifold, virtual control inputs, varying-order controllers, and particular tuning. This chapter is adapted from the work "Stable Online Learning-Based Adaptive Control of Spacecraft and Quadcopters," by Jacob G. Elkins, Farbod Fahimi, and Rohan Sood, presented at the 2024 IEEE Aerospace Conference in Big Sky, MT, where it won best paper in the Software and Computing track [60].

3.1 Introduction

Aerospace systems routinely perform difficult tasks in diverse and uncertain environments. The controllers of these systems must be adaptive to unforeseen internal and external disturbances and changes online, during operation. This chapter builds off Chapter 2's central idea of incorporating nonlinear stabilizing elements into the control law for safe and stable online learning. The

learning elements allow adaptation, while the nonlinear control elements stabilize the system during initial learning and reject disturbances after learning converges. To demonstrate the versatility and various application intricacies of the proposed controller, two common aerospace control problems are considered: rigid-body spacecraft attitude control and quadcopter control. In spacecraft attitude control, the sliding variable used must be slightly modified from the general case to guarantee minimum-distance slews for quaternion trajectory tracking. A satellite sky-scanning problem is described and simulated to verify the attitude controller. In quadcopter control, the full controller is divided into two second-order and one fourth-order sub-controllers via virtual control inputs. The quadcopter dynamics are formulated to match the assumptions of the general controller with minimal assumptions. A three-dimensional figure eight trajectory tracking problem is simulated to verify the full quadcopter controller, which effectively controls the position and yaw of the quadcopter under wind and other aerodynamic effects. Both applications of the developed controller require minimal tuning and modeling relative to conventional model-based approaches by simplifying complicated state functions into online neural network approximations, rendering a highly accurate and robust controller.

3.2 Background and Literature Review

Modern aerospace systems must be controlled with high accuracy and disturbance rejection within uncertain, dynamic environments. Spacecraft must closely follow planned trajectories to point onboard sensors and antennae for

observation and communication, while rejecting complicated disturbances such as atmospheric drag, gravity-gradient torque, and solar radiation pressure [61]. Unmanned aerial vehicles (UAVs) must stably maneuver in complex indoor and outdoor environments, under gusty winds and other aerodynamic effects, to perform a multitude of useful tasks [62]. The fields of adaptive and robust control emerged in response to development of these complex systems and their challenging control requirements, with control systems "learning" and adapting over time to new circumstances during operation [63]. Today, adaptive control remains a critical and highly active research area, with work being done to find better ways to efficiently and effectively control these complex systems under uncertainty and unknown disturbances.

As described throughout this dissertation, the two fields of AI/ML and adaptive control have many obvious connections and relations [10]. A critical piece of study in modern AI/ML is that of explainable AI, which seeks to understand and derive more information out of traditionally black-box approaches (such as deep learning [1]). In control theory, controllers must be proven mathematically to drive control variables to desired values under certain assumptions. Thus, controllers using black-box AI/ML instruments must incorporate those instruments in a provably stable manner. Common methods to stabilize and bound training in ML include various forms of regularization and normalization (as will be discussed in Chapter 5) [64, 65], with some learning stability guarantees given in works such as [25–27, 43–45, 66, 67].

In this chapter, ML and adaptive control are combined fully online specifically for the control of spacecraft and quadcopters. As described in Chapter 2, when learning a suitable model for control fully online, the phase of control before the learned parameters have converged becomes difficult, with learned models exhibiting stability and performance issues [68]. This is particularly true for systems such as quadcopters, where any instability could cause a crash during flight. Sliding mode control (SMC), the highly-robust nonlinear control methodology, is again used in this controller to stabilize the system during learning convergence [16]. Notable early work including SMC with online-learned models includes [69], which uses a neural network trained with a version of backpropagation. Other works employing similar strategies of Lyapunov-based neural network evolution and control laws include [20, 35, 52, 70]. In this chapter, the proofs and neural network update laws in [35] are extended to include the disturbance-rejecting robustness of conventional SMC in a novel online adaptation and control framework specifically for aerospace control systems.

Two common problems in aerospace control are considered in this chapter to showcase the previously-developed controller: spacecraft attitude control and quadcopter UAV control. These problems are studied due to their differences in relative degree and control variable parameterization. For example, spacecraft attitude can be represented in many different ways, such as Euler angles, quaternions, or modified Rodrigues parameters [61]. Thus, the sliding variable (or manifold) and other small factors must be chosen carefully for desirable control performance. Notable previous work in learning-based adaptive control for

spacecraft attitude includes [71], which uses a direct parameter adaptation law in a sliding spacecraft attitude controller. A basis function neural network with only one layer is used in [49], with a similar control law used to control spacecraft attitude under actuator faults and saturation. Quadcopter UAVs are increasingly popular UAV platforms across many applications, due to their high maneuverability and ease of use [72]. The control of quadcopters is not straightforward, as quadcopters are generally an underactuated system, with four inputs (the four rotor speeds) to six degrees of freedom (three position and three attitude). Related works using learning-based SMC for quadcopter control include [40, 41, 73]. In [73] and [40], radial basis function networks are used with fast-terminal SMC to achieve desirable control performance for quadcopter UAVs. In [41], a similar scheme with neural networks adapted online is incorporated with numerical differentiation.

3.3 Contributions

The contribution of this chapter is the derivation of an applicable provably-stable online learning-based control law for control of aerospace systems from the novel general controller developed in Chapter 2 [11]. Motivated by the results in [74] and [75], the spacecraft attitude controller proposed in [74] produced from pure reinforcement learning has no rigorous stability guarantees. However, the advanced and adaptive behavior of the learning-based controller proposed in [74] is desirable. Further, the quadcopter controller derived in [75] is highly adaptive. Combining these works, this chapter describes the derivation, application,

and tuning of a novel controller of spacecraft and quadcopters from the general robotic arm controller in Chapter 2. The implementation details specific to aerospace control, including reparameterizing the sliding variable and using control subsystems with virtual control inputs, are described in this chapter.

3.4 Notation and Preliminaries

This section describes the theory of the general controller, including the system dynamics assumed, error formulation, control law, update rules, and neural network basics. The proof of asymptotic convergence to the origin and learned parameter boundedness is also given in this section. This general controller will be extended to both the spacecraft attitude control problem and the full quadcopter control problem, described throughout this chapter.

3.4.1 Notation

The notation used in this chapter is specific to this chapter. The set of real numbers is denoted as $\mathbb{R}$, where $\mathbb{R}^m$ denotes a real-valued vector of dimension m and $\mathbb{R}^{m\times n}$ denotes a real-valued matrix of dimension $m \times n$. The set of positive real numbers is denoted as $\mathbb{R}^+$. The n^{th} time derivative of some variable y is denoted as $y^{(n)} = \frac{d^n y}{dt^n}$. As shorthand for time derivatives, the notation $y^{(1)} = \dot{y}$, $y^{(2)} = \ddot{y}$, $y^{(3)} = \dddot{y}$, and $y^{(4)} = \ddddot{y}$ is used. The zero matrix of dimension $m \times n$ is denoted as $0_{m\times n}$, and the identity matrix of dimension $m \times n$ is similarly denoted as $I_{m\times n}$.

3.4.2 Control Preliminaries

The general controller again assumes general n^{th} order dynamics of the form

$$My^{(n)} + f(x) + d(t) = u(t), \tag{3.1}$$

where $M \in \mathbb{R}^{m\times m}$ is a diagonal coefficient matrix, $y^{(n)} \in \mathbb{R}^m$ is the n^{th} time derivative of the system output, $f(x) : \mathbb{R}^p \to \mathbb{R}^m$ is a nonlinear function of state $x \in \mathbb{R}^p$, $d \in \mathbb{R}^m$ is a bounded disturbance; and $u \in \mathbb{R}^m$ is the system control input. While this dynamics model is typical of serial robotic arms as discussed in Chapter 2 [57], the form in (3.1) is easily reachable in spacecraft and quadcopter applications [9, 26].

This chapter considers trajectory-tracking control, since regulation can easily be derived from trajectory tracking formulations with zero (or constant) predefined trajectories. The coefficient matrix M and the state function $f(x)$ is assumed to be unknown *a priori* and the system disturbance $d(t)$ is assumed to be bounded in magnitude, described in-depth in the stability proof below.

Defining tracking error as $e = y_d - y$, where y_d is the desired system output trajectory, the n^{th}-order filtered error is written as

$$s = e^{(n-1)} + \sum_{i=0}^{n-2} \binom{n-1}{i} \lambda^{n-i-1} e^{(i)}, \tag{3.2}$$

where λ is a positive definite diagonal design matrix for desired error convergence.

Since the coefficient matrix, M, and the state function, $f(x)$, are assumed to be unknown, the control goal is to approximate these quantities fully online for a controller that drives $s \to 0$ asymptotically. As can be seen in (3.2), when $s = 0$, e approaches zero exponentially. Since the coefficient matrix, M, is assumed constant, direct parameter estimation can be used for its online approximation, while general nonlinear function approximation (such as neural networks) must be used for the state function, $f(x)$. However, with no initial pretraining of $f(x)$ or a poor initial guess of M, an online learning-based controller could be prone to initial instability and poor performance, which is highly undesirable in aerospace systems such as spacecraft and quadcopters. Thus, the controller must be robust to initial learning error while rejecting external disturbances during and after learning convergence.

For the nonlinear robustifying term in the general controller of this chapter, sliding mode control (SMC) is used [16]. The adaptivity of learning and the robustness of SMC is combined in the following general control law, extended throughout the rest this chapter:

$$u = \hat{M} y_r^{(n)} + \hat{f}(x) + (D + \eta)\, \mathtt{sat}(s/\phi), \tag{3.3}$$

where $\hat{M}$ is the learned online estimate of M, $\hat{f}(x)$ is the learned online estimate of the state function $f(x)$, $D \in \mathbb{R}^+$ is an elementwise upper bound on disturbance to be discussed in-depth later, and $\eta \in \mathbb{R}^+$ is a control gain tuned for robustness (especially during initial learning). The reference output $y_r^{(n)}$ is a desired output shifted by a variant of the filtered error, used to simplify notation, written as

$$y_r^{(n-1)} = y_d^{(n-1)} + \sum_{i=0}^{n-2} \binom{n-1}{i} \lambda^{n-i-1} e^{(i)} \tag{3.4}$$

since $\dot{s} = y_r^{(n)} - y^{(n)}$. The $\texttt{sat}(\cdot)$ function is a continuous approximation of the switching control of $\texttt{sign}(\cdot)$, designed to attenuate chatter in both simulation and real-world systems [9]:

$$\texttt{sat}(s/\phi) = \begin{cases} \texttt{sign}(s) & \text{if } |s| > \phi \\ \frac{s}{\phi} & \text{if } |s| \leq \phi, \end{cases} \tag{3.5}$$

where ϕ is a boundary layer thickness value, tuned to limit chatter for a desired control error.

As discussed above, since the state function $f(x)$ is assumed to be any general nonlinear function, a general nonlinear function approximation instrument is required. Thus, the learned state function, $\hat{f}(x)$, is parameterized by a simple neural network (NN) in this chapter, represented as

$$\hat{f}(x) = \hat{W}^T \sigma(\hat{V}^T x) \tag{3.6}$$

where $x \in \mathbb{R}^p$ is the NN input vector, $V \in \mathbb{R}^{p \times n_H}$ is the first layer of NN weights, $W \in \mathbb{R}^{n_H \times m}$ is the second layer of weights, n_H is the number of "hidden" neurons, and $\sigma(\cdot)$ is a nonlinear activation function. The sigmoid function $\sigma(z) = 1/(1 + e^{-z})$ is used as the nonlinear activation, due to its continuous differentiability and ease of derivative calculation ($d\sigma(z)/dz = \sigma' = \sigma(z)(1 - \sigma(z))$). A constant 1 is appended to the NN input x and the hidden input $\sigma(\hat{V}^T x) = \hat{\sigma}$ to account for

additive bias terms in the NN, such that any tuning to $\hat{W}$ and $\hat{V}$ also tunes the corresponding bias.

The following update rules for the learned parameters are used in this chapter:

$$\dot{\hat{W}} = F\hat{\sigma}s_{\Delta}^{T} \tag{3.7}$$

$$\dot{\hat{V}} = Gxs_{\Delta}^{T}\hat{W}^{T}\hat{\sigma}' \tag{3.8}$$

$$\dot{\hat{M}} = Hy_{r}^{(n)}s_{\Delta}^{T}, \tag{3.9}$$

where $s_{\Delta} = s - \phi\texttt{sat}(s/\phi)$ is an algebraic distance of the sliding variable to the boundary layer; $F \in \mathbb{R}^{n_H \times n_H}$, $G \in \mathbb{R}^{p \times p}$, and $H \in \mathbb{R}^{m \times m}$ are each diagonal gain matrices tuned for learning. Note that the variable s_{Δ} is used in the update rules of this chapter (versus s in (2.3) of Chapter 2). This basically stops the adaptation of learning elements at the boundary layer such that the nonlinear control term can drive the system from the boundary layer closer to the $s = 0$ equilibrium point for added stability. Further note that the NN weight update rules in Equations (3.7) and (3.8) are the familiar equations of minimizing mean squared error via backpropagation, with s_{Δ} replacing error.

The implementation of the general controller hence relies on converting the system to be controlled into the form given in (3.1). As will be shown throughout this chapter, this can sometimes be done directly (as in the spacecraft attitude control case), or using a combination of virtual control inputs and control subsystems with differing relative degrees (as in the quadcopter case). In the next

subsection, the stability of the control law in (3.3) with the update rules given in Equations (3.7)-(3.9) will be shown.

3.4.2.1 Stability of the General Controller

Theorem 3.4.1. *Let the desired trajectory, y_d, and all of its n time derivatives be continuous and bounded, with coefficient matrix M constant, such that $\dot{M} = 0_{3\times 3}$. Let $\delta(t) = d_a(t) + d(t)$ be upper bounded by D, where $d_a(t)$ is an internal "disturbance" due to NN approximation error and higher order Taylor series terms to be derived in the proof. Consider the Lyapunov function candidate*

$$L = \frac{1}{2}\left(s_\Delta^T M s_\Delta + \mathrm{Tr}\left(\tilde{M}^T H^{-1}\tilde{M}\right) + \mathrm{Tr}\left(\tilde{W}^T F^{-1}\tilde{W}\right) + \mathrm{Tr}\left(\tilde{V}^T G^{-1}\tilde{V}\right)\right), \tag{3.10}$$

where $\tilde{M} = M - \hat{M}$, $\tilde{W} = W - \hat{W}$, and $\tilde{V} = V - \hat{V}$. With the Lyapunov function candidate in (3.10)*, the system in* (3.1)*, the control input in* (3.3)*, and the update rules in* (3.7)*,* (3.8)*,* (3.9)*; there exists a positive gain $\eta > 0$ such that s_Δ, $\tilde{M}$, $\tilde{V}$, $\tilde{W}$ all approach zero as $t \to \infty$, and the estimates $\hat{M}$, $\hat{V}$, $\hat{W}$ are bounded in time.*

Proof. For proof, see Appendix B. □

3.5 Spacecraft Attitude Control

This section describes the spacecraft attitude control problem, the kinematics of quaternions, and the rigid-body dynamics used to describe spacecraft attitude. This section also describes the modification in choice of the sliding

variable and gives a simulation example of a satellite following a sky-scanning trajectory under both constant and impulsive disturbances. The spacecraft attitude control problem presents an interesting example for a typical implementation of the general controller in second order ($n = 2$), and how the control designer may choose values such as the sliding variable or output parameterization for the system being controlled.

3.5.1 Quaternion Kinematics

In this chapter, the attitude of rigid-body spacecraft is represented by the unit quaternion $q \in SO(3)$, which relates the orientation of the body-fixed frame of the spacecraft to some fixed reference frame. Quaternions are frequently used in attitude representation due to their avoidance of gimbal lock and ease of computational manipulation. Quaternions are generally evolved through time via the kinematic equation

$$\dot{q} = \frac{1}{2}\Xi(q)\omega, \tag{3.11}$$

where

$$\Xi(q) = \begin{bmatrix} q_4 & -q_3 & q_2 \\ q_3 & q_4 & -q_1 \\ -q_2 & q_1 & q_4 \\ -q_1 & -q_2 & -q_3 \end{bmatrix} \tag{3.12}$$

and $\omega \in \mathbb{R}^3$ is the angular velocity about the spacecraft's body-fixed principal axes. For attitude trajectory tracking, the spacecraft's current orientation, q, is related to the desired orientation frame, q_d, via the unit error quaternion, $\delta q \in SO(3)$, by

$$\delta q = q \otimes q_d^{-1} = \begin{bmatrix} \delta q_{1:3} \\ \delta q_4 \end{bmatrix} = \begin{bmatrix} \Xi^T(q) q_d \\ q^T q_d \end{bmatrix}, \tag{3.13}$$

where $\delta q_{1:3} = [\delta q_1, \delta q_2, \delta q_3]^T$ is the vector part of the error quaternion, $\delta q_4 \in \mathbb{R}$ is the scalar part of the error quaternion, and the operator $\otimes$ denotes quaternion multiplication [61].

The control goal is to manipulate the dynamics via the control law such that the error quaternion, δq, approaches the identity quaternion, $q_I = [0, 0, 0, 1]^T$, thereby aligning the spacecraft orientation with the desired orientation.

3.5.2 Spacecraft Attitude Dynamics

The rigid body dynamics of a rotating spacecraft are defined by Euler's rotational equations of motion

$$M\dot{\omega} + \omega^{\times} M\omega + d = \tau, \tag{3.14}$$

where M is the mass-moment of inertia matrix, $\omega = [\omega_1, \omega_2, \omega_3]^T$ is again the angular velocity about the spacecraft's body-fixed principal axes, d is an additive torque disturbance, τ is the external torque control input about the spacecraft's body-fixed principal axes, and $\omega^{\times}$ is a cross-product matrix, defined as

$$\omega^{\times} = \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix}. \tag{3.15}$$

(3.14) is integrated in simulation for the ω at the next timestep with a given control input τ.

3.5.3 Control Design

This section discusses the extension of the general controller in (3.3) to the dynamics given in (3.14). Since the spacecraft attitude is represented via quaternions, the following kinematic equation from [61] can be used to relate the spacecraft angular acceleration about the body-fixed principal axes, $\dot{\omega}$, to the second time derivative of the attitude quaternion, $\ddot{q}$:

$$\ddot{q} = \frac{1}{2}\Xi(q)\dot{\omega} - \frac{1}{4}||\omega||_2^2 q. \tag{3.16}$$

From the definition of the matrix $\Xi(q)$ in (3.12), it can be shown that $\Xi^T(q)\Xi(q) = ||q||_2^2 I_{3\times 3}$. Since unit quaternions are used thoughout this chapter, $||q||_2^2 = 1$, which implies that the pseudoinverse relation $\Xi^{\dagger}(q) = \Xi^T(q)$ holds. Solving (3.16) above for $\dot{\omega}$ using the property $\Xi^{\dagger}(q) = \Xi^T(q)$, the spacecraft angular acceleration is equal to

$$\dot{\omega} = 2\Xi^T(q)\ddot{q} + \frac{1}{2}||\omega||_2^2\Xi^T(q)q. \tag{3.17}$$

It can also be shown from (3.12) that $\Xi^T(q)q = 0_{3\times 1}$. Thus, (3.17) reduces to

$$\dot{\omega} = 2\Xi^T(q)\ddot{q}, \tag{3.18}$$

which, substituting into (3.14), gives

$$2M\Xi^T(q)\ddot{q} + \omega^{\times} M\omega + d = \tau. \tag{3.19}$$

(3.19) is in the control-affine form given in (3.1), so no virtual control inputs are required to isolate the control input from any other quantities, which will be discussed in the quadcopter control section below. (3.19) is a second-order system with respect to time, with the control input τ appearing in the second-order dynamics of our system output, $\ddot{q}$. Note that M is assumed to be constant in Theorem 3.4.1 above. However, in (3.19), the effective M is now $2M\Xi^T(q)$, which varies with attitude q. It is now further assumed that the change in time of this "effective" M is negligible, such that $\frac{d}{dt}(M\Xi^T(q)) \approx 0_{3\times 3}$.

Thus, the spacecraft attitude control law is written as

$$\tau = \hat{M}\ddot{y}_r + \hat{f}(x) + (D + \eta)\,\mathtt{sat}(s/\phi). \tag{3.20}$$

The attitude error of the spacecraft is represented by the error quaternion, δq, which is not directly differentiated for $\dot{\omega}$. Thus, the sliding surface, s, must be slightly modified to capture the desired spacecraft behavior. The sliding surface used in the spacecraft attitude controller of this chapter is written as

$$s = \omega_d - \omega + \lambda \ \texttt{sign}(\delta q_4)\delta q_{1:3}, \tag{3.21}$$

where ω_d and q_d are the desired body-fixed angular velocity of the spacecraft and the desired attitude of the spacecraft, respectively. Comparing (3.21) to the general form of (3.2), the time derivative of the error is replaced by a form of angular velocity error $\omega_d - \omega$; and the output error is replaced by the vector part of the error quaternion. However, since there are two "paths" on the sliding manifold represented by the equivalent rotations δq (shortest path) and $-\delta q$ (longest path), the inclusion of the $\texttt{sign}(\delta q_4)$ term in (3.21) represents selecting the shortest path on the sliding manifold. This analysis is derived from optimal control theory in [76]. It should be noted that the desired trajectory quantities $\{q_d, \omega_d, \dot{\omega}_d\}$ must all be consistent, with the commanded quantities derived from the same underlying desired kinematics (cf. [61]). An example of this trajectory generation using desired Euler angle rates will be shown when describing the simulation of the following section.

Similarly, the reference output, $\ddot{y}_r$, is modified to

$$\ddot{y}_r = \dot{\omega}_d + \lambda \delta \dot{q}_{1:3}, \tag{3.22}$$

where the time derivative of the vector part of the error quaternion, $\delta \dot{q}_{1:3}$, is found by the relation

$$\delta \dot{q}_{1:3} = \frac{1}{2}\delta q_4(\omega - \omega_d) + \frac{1}{2}\delta q_{1:3} \times (\omega + \omega_d) \tag{3.23}$$

where the $\times$ operator here designates simple vector cross-product [61]. Note that neither the $\texttt{sign}(\delta q_4)$ term nor its derivative appear in (3.22). During the shortest-path slew maneuver defined by the sliding manifold in (3.21), δq_4 does not change sign.

Comparing (3.19) to (3.1), it can be seen that the state function to be approximated by the NN is $f(x) = \omega^{\times} M\omega$. The NN input vector for this controller is thus

$$x = [\omega_1, \omega_2, \omega_3, 1]^T \tag{3.24}$$

where the 1 is appended for bias, as described above. A block diagram of the full spacecraft attitude controller is shown in Figure 3.1. Starting on the top left of Figure 3.1, the controller commands a torque, τ, and sends it to the plant (the spacecraft). The spacecraft state $\{q, \omega\}$ is compared to the generated desired trajectory $\{q_d, \omega_d, \dot{\omega}_d\}$ to get s and $\ddot{y}_r$ for the next control input. After each torque is commanded, the learned parameters $\hat{M}, \hat{V}, \hat{W}$ are updated via their respective update rules.

3.5.4 Simulation Example

A sky-mapping spacecraft trajectory tracking problem is simulated as in [61] by specifying the desired Euler angles $\{\phi_d, \theta_d, \psi_d\}$ to follow

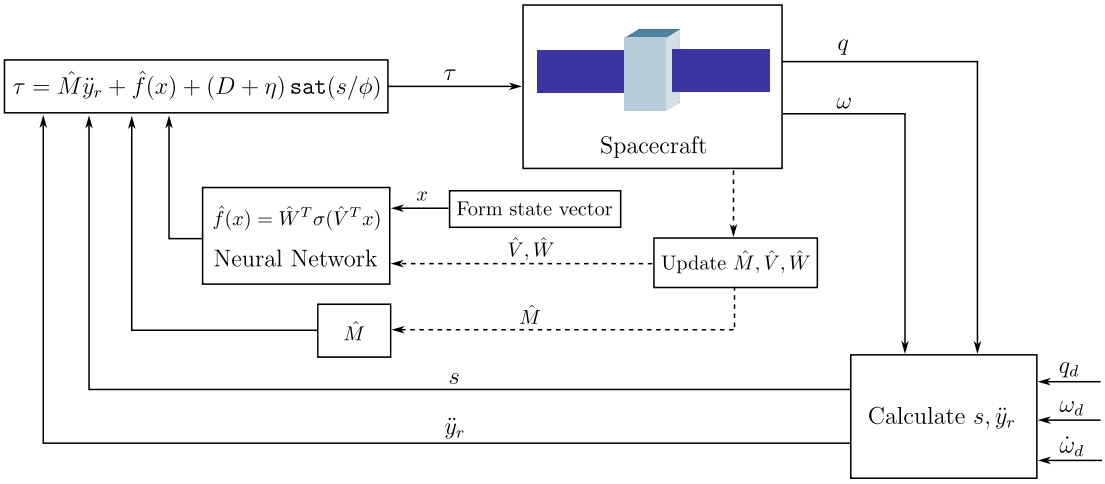

Figure 3.1: Block diagram of the learning-based spacecraft attitude controller.

$$\begin{aligned} \dot{\phi}_d &= 1 \text{ rev/h} = 0.001745 \text{ rad/s} \\ \theta_d &= 22.5^\circ = 0.3927 \text{ rad} \\ \dot{\psi}_d &= 0.464 \text{ rpm} = 0.04859 \text{ rad/s}, \end{aligned} \tag{3.25}$$

where ϕ_d and ψ_d are calculated by integrating the desired rates of $\dot{\phi}_d$ and $\dot{\psi}_d$, given above, respectively. The desired body-fixed 3-1-3 Euler angles are converted to the desired values required by the controller via

$$q_d = \begin{bmatrix} q_{d,1:3} \\ q_{d,4} \end{bmatrix} = \begin{bmatrix} \sin\left(\frac{\theta_d}{2}\right)\cos\left(\frac{\phi_d - \psi_d}{2}\right) \\ \sin\left(\frac{\theta_d}{2}\right)\sin\left(\frac{\phi_d - \psi_d}{2}\right) \\ \cos\left(\frac{\theta_d}{2}\right)\sin\left(\frac{\phi_d + \psi_d}{2}\right) \\ \cos\left(\frac{\theta_d}{2}\right)\cos\left(\frac{\phi_d + \psi_d}{2}\right) \end{bmatrix} \tag{3.26}$$

$$\omega_d = \begin{bmatrix} \dot{\phi}_d \sin\theta_d \sin\psi_d \\ \dot{\phi}_d \sin\theta_d \cos\psi_d \\ \dot{\psi}_d \end{bmatrix} \tag{3.27}$$

$$\dot{\omega}_d = \dot{\phi}_d \dot{\psi}_d \begin{bmatrix} \sin\theta_d \cos\psi_d \\ -\sin\theta_d \sin\psi_d \\ 0 \end{bmatrix}. \tag{3.28}$$

In the simulation, the mass-moment of inertia matrix for the spacecraft is assumed to be

$$M = \begin{bmatrix} 399 & -2.81 & -1.31 \\ -2.81 & 377 & 2.54 \\ -1.31 & 2.54 & 377 \end{bmatrix} \text{ kg-m}^2 \tag{3.29}$$

with the spacecraft controlled by external torques about the body-fixed principal axes at a control frequency of 100 Hz. Note that the off-diagonal components of the assumed inertia matrix are negligible compared to the diagonal components, which is consistent with the assumption of a diagonal M in (3.1) and described in the previous section. To showcase the disturbance-rejecting nature of the controller, multiple external torque disturbances are imparted on the spacecraft during the simulation. The disturbance

$$d(t) = \begin{bmatrix} 0.5\sin(0.01t) \\ 0.3 \\ 0.5\cos(0.015t) \end{bmatrix} \text{ N-m} \tag{3.30}$$

is enacted at all timesteps in the simulation. This disturbance is a larger magnitude version of the disturbance in the experiment in [61], designed to simulate a disturbance torque such as gravity gradient. At timestep $t = 40$ s, an additional impulsive disturbance of $[30, -15, 50]^T$ N-m is enacted on the spacecraft. At timestep $t = 60$ s, another impulsive disturbance of $[30, -45, 5]^T$ N-m is enacted on the spacecraft, while also changing the mass moment of inertia of the spacecraft to

$$M = \begin{bmatrix} 100 & -2.81 & -1.31 \\ -2.81 & 100 & 2.54 \\ -1.31 & 2.54 & 377 \end{bmatrix} \text{ kg-m}^2. \tag{3.31}$$

This change in moment of inertia is analogous to spacecraft operations such as deploying deputy satellites, undocking from a larger spacecraft, or retracting solar panels. Note that all disturbances above are described in the spacecraft body-fixed frame. Equations (3.11) and (3.14) are jointly integrated using fourth order Runge-Kutta with a timestep of 0.01 s for 100 s. The attitude quaternion is normalized by $\delta q = \delta q / ||\delta q||_2$ after each integration step in the simulation to preserve the unit quaternion property. Each torque command, τ, is clipped at a control bound of 10 N-m, selected to mimic the torque produced from an impulsive thrust

attitude control system (like a cold gas thruster) [77]. The spacecraft is initialized at an orientation of $q(t=0) = [0.3, 0.1, -0.1, 0.9434]^T$, with the coefficient gain matrix in the controller initialized at $\hat{M}(t=0) = \text{diag}(300, 300, 300)$, intended to represent a rough estimate of the true M in the system dynamics. The NN weight matrices, $\hat{V}$ and $\hat{W}$, are initialized by sampling from a uniform distribution on the interval $[-0.1, 0.1)$.

The time history of the spacecraft attitude trajectory, Euler angle errors, and control inputs for the simulation are shown in Figure 3.2, with the time history of the adaptive parameters shown in Figure 3.3. The Frobenius norm of the NN weight matrices is plotted to show how the magnitudes of the learned parameters in the NN change, since the weight matrices have many components. Controller hyperparameters used in the simulation are given in Table 3.1.

Table 3.1: Hyperparameters used for the simulated spacecraft attitude controller.

Hyperparameter	**Value**
η	400
D	100
ϕ	0.05
λ	$\text{diag}(4, 4, 4)$
n_H	15
F	$\text{diag}((10,) \times n_H)$
G	$\text{diag}((10,) \times 4)$
H	$\text{diag}(0.9, 0.9, 0.9)$

In Figure 3.2, the spacecraft trajectory's quaternion components $q = [q_1, q_2, q_3, q_4]^t$ are plotted, along with the desired trajectory's quaternion com-

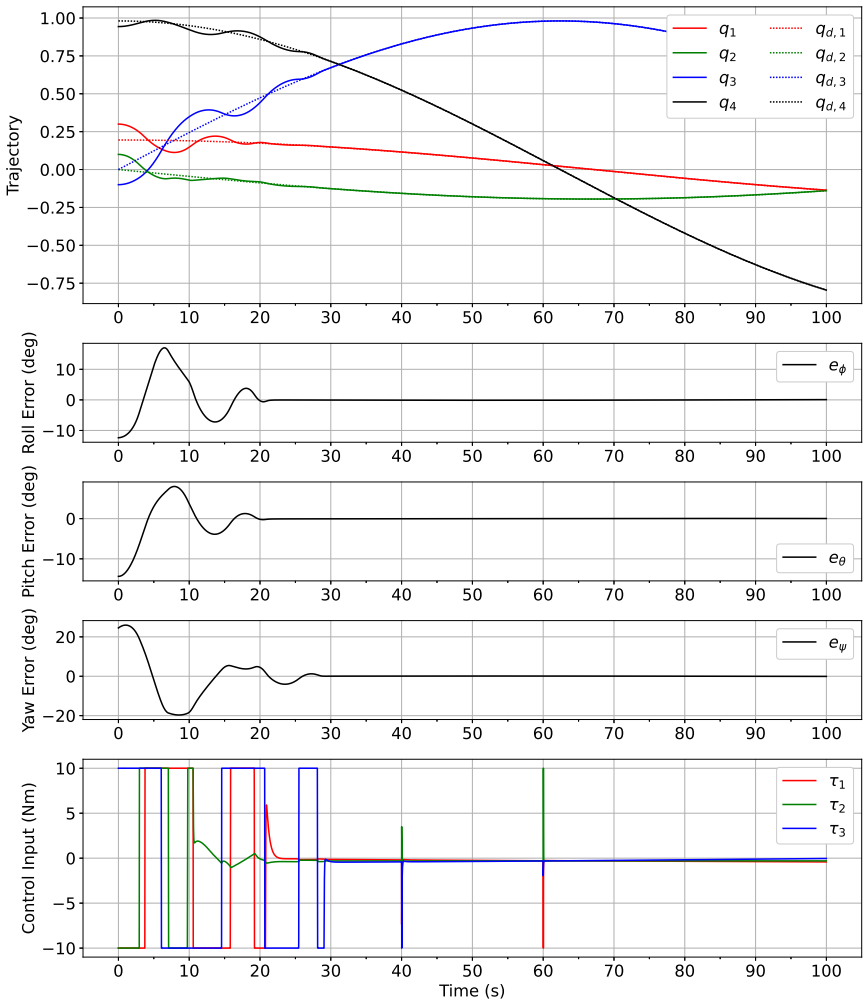

Figure 3.2: Spacecraft trajectory, error, and control input versus time for the sky-scanning attitude control simulation.

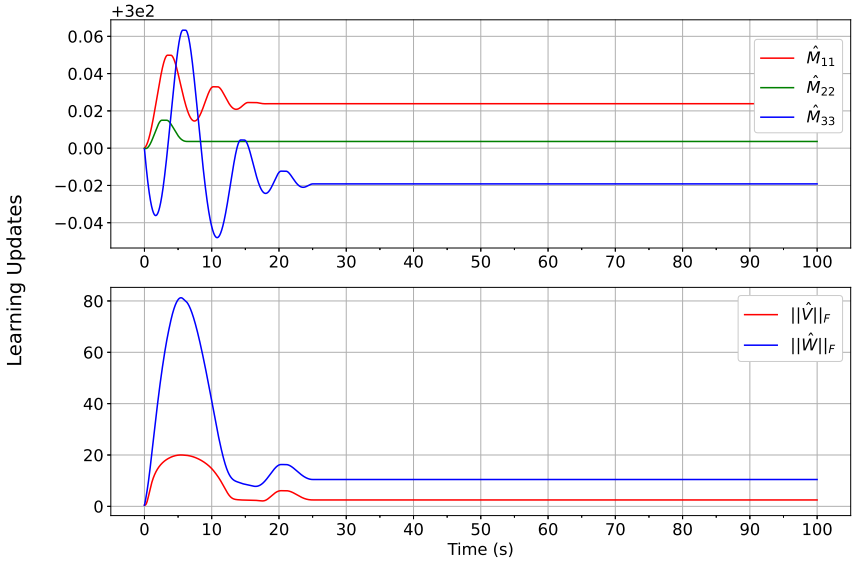

Figure 3.3: Components of $\hat{M}$ and the Frobenius norm of $\hat{V}$ and $\hat{W}$ over time for the sky-scanning attitude control simulation.

ponents $q_d = [q_{d,1}, q_{d,2}, q_{d,3}, q_{d,4}]^T$. To easily visualize the error over time in the simulation, the spacecraft's attitude is converted from the quaternion form, q, back to 3-1-3 Euler angles $\{\phi, \theta, \psi\}$. These Euler angles are compared to the desired Euler angles generated by evolving (3.25) in time to find the errors in roll, $e_\phi = \phi_d - \phi$, pitch, $e_\theta = \theta_d - \theta$, and yaw, $e_\psi = \psi_d - \psi$. The controller is able to reject the continuous disturbance at all times, while handling the impulsive disturbances at $t = 40$ s and $t = 60$ s. The settling time is due to the imposed control limit of 10 N-m. The controller is also able to handle the internal change in rotational inertia at $t = 60$ s. In Figure 3.3, the learning updates of $\hat{M}$, $\hat{V}$, and $\hat{W}$ converge to a constant value around $t = 23$ s, with each parameter bounded as proven in Theorem 3.4.1. The updates to $\hat{V}$ and $\hat{W}$ in Figure 3.3 follow a similar shape to the Euler angle errors in Figure 3.2, settling at $t = 23$ s once the

spacecraft reaches the boundary value $s_\Delta = 0$. As apparent from the simulation, the spacecraft is controlled with high accuracy and robustness to both internal and external disturbances, with only a very rough estimate of $\hat{M}(t = 0)$ for initialization, due to the proven-stable online-learning based controller developed in this portion of the chapter.

3.6 Quadcopter Control

This section describes extending the general controller in (3.3) to the quadcopter system. The section begins with discussion of the kinematics and dynamics of the assumed system, followed by the control design. The quadcopter is controlled by three subsystem controllers: two second-order controllers in vertical position and yaw, and one fourth-order controller for both horizontal position dimensions. This control formulation requires calculation of virtual control inputs and relating these virtual inputs to the final control input to the system: the four rotor speeds. The quadcopter system is thus of merit to study implementation of the general controller presented to control a complicated system of varying order, and this example showcases the system modeling or parameter estimation possibly needed for applying the developed controller to a particular system.

3.6.1 Quadcopter Kinematics

The quadcopter described in this chapter is modeled as a six degree-of-freedom rigid body, controlled by the angular speed of each of the four rotors. The quadcopter translates in space via the rotational relation

$$\dot{r} = V = R^T v, \tag{3.32}$$

where $r = [r_x, r_y, r_z]^T$ is the position vector relating the origin of the Earth-fixed inertial frame $\{X_e, Y_e, Z_e\}$ to the quadcopter-fixed body frame $\{X, Y, Z\}$. The quadcopter velocity in the Earth-fixed frame is denoted as $V = [V_x, V_y, V_z]^T$, and the quadcopter velocity in the body-fixed frame is denoted as $v = [v_x, v_y, v_z]^T$. The rotation matrix, R, relates free vectors in the Earth-fixed frame to the body-fixed frame via a body-fixed 1-2-3 rotation, defined as

$$R = \begin{bmatrix} c\theta c\psi & c\phi s\psi + s\phi s\theta c\psi & s\phi s\psi - c\phi s\theta c\psi \\ -c\theta s\psi & c\phi c\psi - s\phi s\theta s\psi & s\phi c\psi + c\phi s\theta s\psi \\ s\theta & -s\phi c\theta & c\phi c\theta \end{bmatrix}, \tag{3.33}$$

where the shorthand $c(\cdot) = \cos(\cdot)$, $s(\cdot) = \sin(\cdot)$ is used. $\Phi = [\phi, \theta, \psi]^T$ is the vector of Euler angles (roll, pitch, yaw) relating the orientation of the quadcopter body-fixed frame to the Earth-fixed frame. Both the roll angle, ϕ, and pitch angle, θ, are assumed to be restricted to the domain $(-\pi/2, \pi/2)$. The Euler angle rotation rates, $\dot{\Phi}$, are related to the angular velocity about the quadcopter body-fixed principal axes via

$$\omega = B\dot{\Phi}, \tag{3.34}$$

where

$$B = \begin{bmatrix} c\theta c\psi & s\psi & 0 \\ -c\theta s\psi & c\psi & 0 \\ s\theta & 0 & 1 \end{bmatrix}. \tag{3.35}$$

The matrix B above is invertible, since the determinant $\cos(\theta) \neq 0$ due to the pitch angle restriction $\theta \in (-\pi/2, \pi/2)$.

3.6.2 Quadcopter Dynamics

Since the quadcopter is modeled as a rigid body, its rotation dynamics are similarly defined by Euler's rotational equations of motion as in (3.14) of the spacecraft example:

$$M\dot{\omega} + \omega^{\times} M\omega + \tau_d = \tau, \tag{3.36}$$

where $\tau = [\tau_x, \tau_y, \tau_z]^T$ is the resultant control torque from the rotor inputs and τ_d is some torque disturbance. Recall from above that $\omega^{\times}$ is the cross-product matrix defined in (3.15). Translational motion of the quadcopter in the body-fixed frame is described by the equation

$$m\dot{v} + m\omega^{\times} v + F'_d = F' + F'_g, \tag{3.37}$$

where m is the mass of the entire quadcopter, F'_d is a disturbance force in the body-fixed frame, $F'_g = R[0, 0, -mg]^T$ is the force due to gravity in the body-fixed frame, and $F' = [0, 0, f]$ is the control thrust force from the rotor inputs.

The equations of translational motion in the Earth-fixed inertial frame are written via Newton's second law as

$$m\dot{V} + F_d = F + F_g = R^T \begin{bmatrix} 0 \\ 0 \\ f \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix}, \tag{3.38}$$

where $F_d = R^T F_d'$, $F = R^T F'$, and $F_g = R^T F_g'$.

The relation between the quadcopter control inputs, Ω_i (the angular speed of the i^{th} rotor), and the resultant thrust/torques to control, is written as

$$\begin{bmatrix} f \\ \tau_x \\ \tau_y \\ \tau_z \end{bmatrix} = \begin{bmatrix} c_T & c_T & c_T & c_T \\ d_4 c_T & -d_3 c_T & -d_3 c_T & d_4 c_T \\ d_1 c_T & -d_2 c_T & d_1 c_T & -d_2 c_T \\ -c_Q & -c_Q & c_Q & c_Q \end{bmatrix} \begin{bmatrix} \Omega_1^2 \\ \Omega_2^2 \\ \Omega_3^2 \\ \Omega_4^2 \end{bmatrix}, \tag{3.39}$$

where the various moment arms are $d_1 = d/\sqrt{2} + x_{CoM}$, $d_2 = d/\sqrt{2} - x_{CoM}$, $d_3 = d/\sqrt{2} + y_{CoM}$, $d_4 = d/\sqrt{2} - y_{CoM}$, c_T is a coefficient of thrust, and c_Q is a coefficient of torque. As shown in Figure 3.4, d is defined as the distance from the geometric center of the quadcopter to the center of each of the rotors, and $r_{CoM} = [x_{CoM}, y_{CoM}, z_{CoM}]^T$ is the vector relating the geometric center of the quadcopter to its center of mass. Note that the matrix in (3.39) is also invertible, due to the determinant $-8c_Q c_T^3 d^2 \neq 0$ since $\{c_Q, c_T, d\} > 0$ [78].

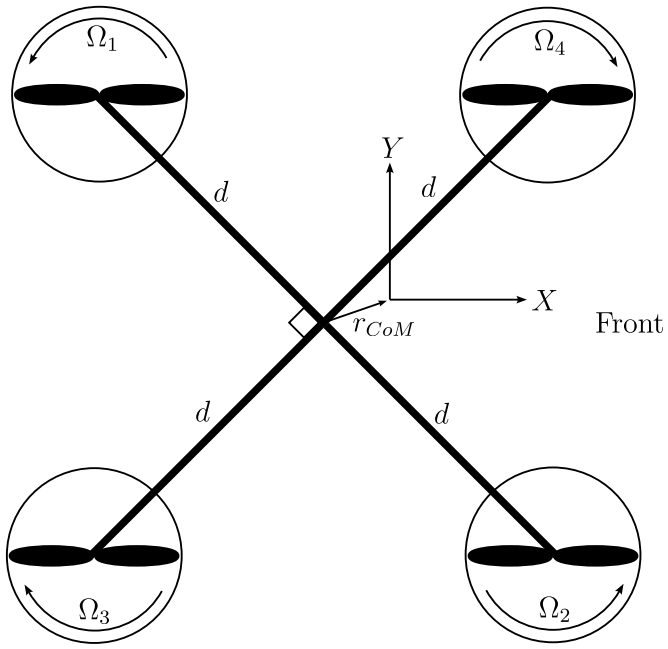

Figure 3.4: Top view of the assumed quadcopter geometry, with four counter-rotating rotors.

3.6.3 Control Design

While the general controller is designed to require no *a priori* system modeling, the trick in implementation is getting the system dynamics to the form of the assumed system in (3.1), similar to the spacecraft control section above. As aforementioned, the quadcopter is an underactuated system, having six degrees of freedom (three position and three orientation) to four control inputs (the four rotor speeds). The controller described in this subsection considers controlling the quadcopter position in space r_x, r_y, r_z and the yaw angle ψ via the four rotor speeds. The form of the dynamics required by the controller in (3.1) can be achieved by three subsystem controllers: two second-order controllers for r_z and

ψ, and one fourth-order controller for r_x and r_y [78]. Each subsystem outputs virtual control inputs related to the final control input to the system: the rotor speeds, Ω.

3.6.3.1 Control of r_z

Assuming the disturbance force F_d is unknown during control design, the dynamics of r_z from the last line in (3.38) is written as

$$m\ddot{r}_z = R_{33}\hat{f} - mg \tag{3.40}$$

which relates the nominal value of control thrust force $\hat{f}$ to the control variable r_z, where R_{ij} is the $(i,j)^{\text{th}}$ element of R in (3.33). Note that the respective component of F_d has been dropped in (3.40) above. To simplify notation, a new control input $u_1 = \hat{f}/c_T$ is defined and substituted into (3.40) for $\hat{f}$ to get

$$m\ddot{r}_z = R_{33}c_T u_1 - mg. \tag{3.41}$$

Defining the virtual control input $v_1 = R_{33}u_1$, (3.41) can be written as

$$\frac{m}{c_T}\ddot{r}_z + \frac{mg}{c_T} = v_1, \tag{3.42}$$

which is in the desired form of (3.1). The r_z controller, following (3.3), is thus

$$v_1 = \hat{M}_1\ddot{r}_{z,r} + f_1(x_1) + (D_1 + \eta_1)\mathtt{sat}(s_z/\phi_1), \tag{3.43}$$

where

$$s_z = \dot{e}_z + \lambda_1 e_z \tag{3.44}$$

from (3.2), and

$$\ddot{r}_{z,r} = \ddot{r}_{z,d} + \lambda_1 \dot{e}_z \tag{3.45}$$

from (3.4), where $r_{z,d}$ is a two-times differentiable desired trajectory of r_z. Comparing (3.1) to (3.42), the state function to be approximated by the NN is $f(x) = mg/c_T$, which is a constant. The NN input vector, nominally containing all the variables of the approximated state function, is chosen as $x_1 = [1]$ to approximate the constant term for continuity.

3.6.3.2 Control of ψ

From (3.34), the kinematics of the Euler angle rates $\dot{\Phi}$ can be written as

$$\dot{\Phi} = B^{-1}\omega, \tag{3.46}$$

where

$$B^{-1} = \begin{bmatrix} \frac{c\psi}{c\theta} & -\frac{s\psi}{c\theta} & 0 \\ s\psi & c\psi & 0 \\ -c\psi t\theta & s\psi t\theta & 1 \end{bmatrix} \tag{3.47}$$

and the shorthand $s(\cdot) = \sin(\cdot)$, $c(\cdot) = \cos(\cdot)$, and $t(\cdot) = \tan(\cdot)$ is again used. To relate the control torque about the Z-axis, τ_z, to the control variable, ψ, (3.46) is differentiated with respect to time to get

$$\ddot{\Phi} = \dot{B}^{-1}\omega + B^{-1}\dot{\omega}. \tag{3.48}$$

From the last line in (3.48), the yaw angular rate $\ddot{\psi}$ is related to the angular acceleration about the Z-axis by the form

$$\ddot{\psi} = f_B(\Phi, \dot{\Phi}, \omega, \dot{\omega}) + \dot{\omega}_z, \tag{3.49}$$

where $f_B(\Phi, \dot{\Phi}, \omega, \dot{\omega})$ is a state function capturing additional terms. Substituting the last line of (3.36) for $\dot{\omega}_z$ into (3.49) gives

$$\ddot{\psi} = f_B(\Phi, \dot{\Phi}, \omega, \dot{\omega}) + \frac{\tau_z}{M_{zz}} + \frac{(M_{xx} - M_{yy})\omega_x\omega_y}{M_{zz}}, \tag{3.50}$$

where $\{M_{xx}, M_{yy}, M_{zz}\}$ are the diagonal components of the mass-moment of inertia matrix M. Defining a virtual control input $v_4 = u_4 = \tau_z/c_Q$ for consistency, (3.50) can be written in the desired form of (3.1) by

$$\frac{M_{zz}}{c_Q}\ddot{\psi} + f_4(\Phi, \dot{\Phi}, \omega, \dot{\omega}) = v_4, \tag{3.51}$$

where $f_4(\Phi, \dot{\Phi}, \omega, \dot{\omega})$ is the new system state function in the ψ controller to be approximated online via NN. The ψ controller, following (3.3), is finally given as

$$v_4 = \hat{M}_4\ddot{\psi}_r + f_4(x_4) + (D_4 + \eta_4)\mathtt{sat}(s_\psi/\phi_4), \tag{3.52}$$

where the NN input vector, x_4, is chosen as $x_4 = [\Phi^T, \dot{\Phi}^T, \omega^T, \dot{\omega}^T]^T$ to correspond to the state functions described in Equations (3.50) and (3.51),

$$s_\psi = \dot{e}_\psi + \lambda_4 e_\psi \tag{3.53}$$

from (3.2), and

$$\ddot{\psi}_r = \ddot{\psi}_d + \lambda_4 \dot{e}_\psi \tag{3.54}$$

from (3.4). ψ_d is a two-times differentiable desired trajectory of the yaw angle, ψ.

3.6.3.3 Control of r_x, r_y

To relate the control variables r_x and r_y to the control torques τ_x and τ_y, (3.38) is differentiated twice with respect to time, assuming no disturbance force. Noting the the time derivative of a rotating frame is written as $\dot{R}^T = R^T\omega^\times$, these derivatives are written as

$$m\ddot{V} = R^T \left(\omega^\times \begin{bmatrix} 0 \\ 0 \\ \hat{f} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \dot{\hat{f}} \end{bmatrix} \right) \tag{3.55}$$

$$m\dddot{V} = R^T \left((\dot{\omega}^\times + \omega^\times \omega^\times) \begin{bmatrix} 0 \\ 0 \\ \hat{f} \end{bmatrix} + 2\omega^\times \begin{bmatrix} 0 \\ 0 \\ \dot{\hat{f}} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \ddot{\hat{f}} \end{bmatrix} \right). \tag{3.56}$$

Note that this control subsystem is now fourth order in r_x and r_y, where the r_z and ψ controllers above are second order. Substituting the Euler rotational equations in (3.36) for $\dot{\omega}_x$ and $\dot{\omega}_y$ in the first two equations in (3.56), the control variables r_x and r_y are related to the control inputs τ_x and τ_y via

$$m\dddot{V}_x + f_x(\Phi, \omega, \hat{f}, \dot{\hat{f}}, \ddot{\hat{f}}) = \frac{R_{11}\hat{f}}{M_{yy}}\tau_y - \frac{R_{21}\hat{f}}{M_{xx}}\tau_x \tag{3.57}$$

$$m\dddot{V}_y + f_y(\Phi, \omega, \hat{f}, \dot{\hat{f}}, \ddot{\hat{f}}) = \frac{R_{12}\hat{f}}{M_{yy}}\tau_y - \frac{R_{22}\hat{f}}{M_{xx}}\tau_x. \tag{3.58}$$

To isolate the two control variables, it is hereafter assumed that $M_{xx} = M_{yy} = M_{xxyy}$. This assumption is mild and generally valid for the quadcopter system, as quadcopters are typically designed near-symmetric to rotations about the body-fixed X and Y axes for stability of flight. Defining new control inputs $u_2 = \tau_x/c_T$ and $u_3 = \tau_y/c_T$ and rearranging gives

$$\frac{M_{xxyy}m}{c_T}\dddot{V}_x + f_2(\Phi, \omega, \hat{f}, \dot{\hat{f}}, \ddot{\hat{f}}) = (R_{11}u_3 - R_{21}u_2)\hat{f} \tag{3.59}$$

$$\frac{M_{xxyy}m}{c_T}\dddot{V}_y + f_3(\Phi, \omega, \hat{f}, \dot{\hat{f}}, \ddot{\hat{f}}) = (R_{12}u_3 - R_{22}u_2)\hat{f}, \tag{3.60}$$

which, defining virtual control inputs $v_2 = (R_{11}u_3 - R_{21}u_2)\hat{f}$ and $v_3 = (R_{12}u_3 - R_{22}u_2)\hat{f}$, renders the desired form of (3.1):

$$\frac{M_{xxyy}m}{c_T}\dddot{V}_x + f_2(\Phi, \omega, \hat{f}, \dot{\hat{f}}, \ddot{\hat{f}}) = v_2 \tag{3.61}$$

$$\frac{M_{xxyy}m}{c_T}\dddot{V}_y + f_3(\Phi, \omega, \hat{f}, \dot{\hat{f}}, \ddot{\hat{f}}) = v_3. \tag{3.62}$$

Since the unknown state functions f_2 and f_3 are functions of the same variables, their approximation can be combined into one NN to get the r_x, r_y controller:

$$\begin{bmatrix} v_2 \\ v_3 \end{bmatrix} = \hat{M}_{23}\begin{bmatrix} \ddddot{r}_{x,r} \\ \ddddot{r}_{y,r} \end{bmatrix} + f_{23}(x_{23}) + \begin{bmatrix} (D_2 + \eta_2)\mathtt{sat}(s_x/\phi_2) \\ (D_3 + \eta_3)\mathtt{sat}(s_y/\phi_3) \end{bmatrix}, \tag{3.63}$$

where

$$s_x = \ddot{e}_x + 3\lambda_2\ddot{e}_x + 3\lambda_2^2\dot{e}_x + \lambda_2^3 e_x \tag{3.64}$$

$$s_y = \ddot{e}_y + 3\lambda_3\ddot{e}_y + 3\lambda_3^2\dot{e}_y + \lambda_3^3 e_y \tag{3.65}$$

from (3.2), and

$$\ddddot{r}_{x,r} = \ddddot{r}_{x,d} + 3\lambda_2\,\dddot{e}_x + 3\lambda_2^2\ddot{e}_x + \lambda_2^3\dot{e}_x \tag{3.66}$$

$$\dddot{r}_{y,r} = \dddot{r}_{y,d} + 3\lambda_3 \ddot{e}_y + 3\lambda_3^2 \dot{e}_y + \lambda_3^3 e_y \tag{3.67}$$

from (3.4). $r_{x,d}$ and $r_{y,d}$ are a four-times differentiable desired trajectory of r_x and r_y. The NN input vector is nominally chosen as $x_{23} = [\Phi^T, \omega^T, \hat{f}, \dot{\hat{f}}, \ddot{\hat{f}}]^T$.

3.6.3.4 Getting Ω^2 from u

Finally, the virtual control inputs $\{v_1, v_2, v_3, v_4\}$ from each control subsystem must be converted to the rotor speed inputs $\{\Omega_1, \Omega_2, \Omega_3, \Omega_4\}$ for input to the system. The intermediate control inputs $\{u_1, u_2, u_3, u_4\}$ are related to the virtual control inputs via

$$\begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} = \begin{bmatrix} R_{33} & 0 & 0 & 0 \\ 0 & -R_{12}\hat{f} & R_{11}\hat{f} & 0 \\ 0 & -R_{22}\hat{f} & R_{12}\hat{f} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{bmatrix}, \tag{3.68}$$

where this coefficient matrix is invertible since the determinant $R_{33}\hat{f}^2(R_{11}R_{22} - R_{12}R_{21}) \neq 0$ since $\phi \in (-\pi/2, \pi/2)$ and $\theta \in (-\pi/2, \pi/2)$. Substituting the relations $u_1 = \hat{f}/c_T$, $u_2 = \tau_x/c_T$, $u_3 = \tau_y/c_T$, and $u_4 = \tau_z/c_Q$ into (3.39), the relation of the intermediate control inputs to the square of the rotor speeds is written as

$$\begin{bmatrix} \Omega_1^2 \\ \Omega_2^2 \\ \Omega_3^2 \\ \Omega_4^2 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ d_4 & -d_3 & -d_3 & d_4 \\ d_1 & -d_2 & d_1 & -d_2 \\ -1 & -1 & 1 & 1 \end{bmatrix}^{-1} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix}. \tag{3.69}$$

Note that the coefficient matrix above also uses the geometrical system parameters d_1, d_2, d_3, d_4, which must be roughly estimated *a priori*. The coefficient matrix above is invertible, as the determinant $-4(d_1d_3 + d_1d_4 + d_2d_3 + d_2d_4) \neq 0$. The full quadcopter control block diagram is shown in Figure 3.5.

Starting in the top left block in Figure 3.5, the three control subsystems intake the desired position and yaw trajectories to generate the virtual control inputs, v. The virtual control inputs are then converted to rotor speeds, Ω, via Equations (3.68) and (3.69). The rotor speeds are sent to the quadcopter plant, which outputs the state variables for position, velocity, attitude, and angular velocity. These terms are then numerically differentiated and sent back to the controller at the next timestep.

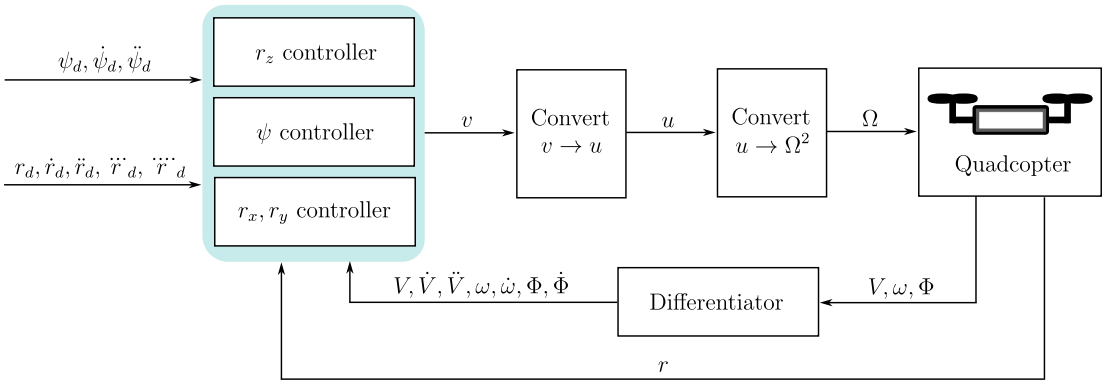

Figure 3.5: A simplified block diagram of the developed quadcopter controller.

Further note that, for the quadcopter system, the rotor speeds $\Omega_i \geq 0$ are calculated from the virtual control inputs v_i, which can be positive or negative. To avoid negative rotor speeds when solving (3.69), the v_1 control is centered about the hover mode, which is implemented as $v_1 \leftarrow v_1 + mg/c_T$.

(3.40) is solved for $\hat{f}$ for use in (3.68) above. Since $\ddot{r}_z$ is not usually known or measured, V_z is numerically differentiated for $\ddot{r}_z$ in this chapter. Numerical differentiation of V_x and V_y to get $\dddot{r}_x$, $\dddot{r}_y$, $\ddddot{r}_x$, and $\ddddot{r}_y$ for the r_x, r_y controller is also required. While other controllers (*e.g.*, [78]) have avoided the requirement of calculating higher-order derivatives using dynamic models, the developed controller learns these relations online. Thus, the trade-off for online learning of suitable dynamic models in this controller requires calculation of higher-order derivatives. While robust numerical differentiation is easily implemented for this controller, it is an extra step to tune for control performance and stability.

3.6.4 Simulation Example

To validate the developed quadcopter controller, the Bitcraze Crazyflie 2.1 quadcopter system is simulated.[1] The Crazyflie is a small lightweight drone, commonly used as a platform for quadcopter swarm research. The simulation is performed as in [78] by integrating Equations (3.36) and (3.37), with the addition of aerodynamic forces F_a and torques τ_a (considered as disturbances). The aerodynamic force and torques used in the simulation are written as

[1] https://www.bitcraze.io/products/crazyflie-2-1/

$$F_a = -q_{fs} S_{ref} \begin{bmatrix} C_L \\ C_M \\ C_N \end{bmatrix}, \; \tau_a = -q_{fs} S_{ref} d_{ref} \begin{bmatrix} C_l \\ C_m \\ C_n \end{bmatrix}, \tag{3.70}$$

where $q_{fs} = \frac{1}{2}\rho_a v_{rel}^2$ is the freestream dynamic pressure, ρ_a is the ambient air density, $v_{rel} = v - RV_w$ is the relative velocity of air on the quadcopter in the body-fixed frame, V_w is the wind velocity in the inertial frame, S_{ref} is the reference surface area of the drone, d_{ref} is the mean diameter of the body, C_L, C_M, C_N are the aerodynamic force coefficients; and C_l, C_m, C_n are the aerodynamic torque coefficients. The parameters used in the simulation are given in Table 3.2, where d_{1-4} is shorthand for $d_1 = d_2 = d_3 = d_4$.

Table 3.2: Simulation parameters used for the quadcopter.

Parameter	**Value**
d_{1-4}	0.028 m
M	$\text{diag}(1.4, 1.4, 2.17) \times 10^{-5}$ kg-m^2
c_T	3.16×10^{-10}
c_Q	7.94×10^{-12}
C_L	9.17854×10^{-7}
C_M	9.17854×10^{-7}
C_N	9.17854×10^{-7}
C_l	9.17854×10^{-7}
C_m	9.17854×10^{-7}
C_n	9.17854×10^{-7}
S_{ref}	0.006 m^2
d_{ref}	0.028 m
m	0.027 kg
V_w	$[3.53, 3.53, 0]^T$ m/s
ρ_a	1.225 kg/m^3

The four-time differentiable desired trajectory of the quadcopter for this simulation example is a three dimensional figure eight, written as

$$r_{z,d} = -0.5 \sin\left(\frac{2\pi t}{200}\right) \tag{3.71}$$

$$r_{x,d} = 0.1 \sin\left(\frac{2\pi t}{100}\right) \tag{3.72}$$

$$r_{y,d} = -0.1 \sin\left(\frac{2\pi t}{100}\right) \tag{3.73}$$

$$r_{\psi,d} = \frac{\pi}{4} \sin\left(\frac{2\pi t}{200}\right), \tag{3.74}$$

which represents a reasonably complicated trajectory for showcasing the adaptivity and stability of the controller in each degree of freedom. The simulation is integrated at a frequency of 1 kHz using fourth order Runge-Kutta. To calculate $\hat{f}$ for (3.68), the relation given in (3.40) is used to find $\hat{f} = m(\dot{V}_z + g)/R_{33}$. The first order sliding mode differentiator as described in [16] is used to calculate $\dot{V}_z$ for this equation. The same sliding mode differentiator in [16], now third order, is also used to calculate the values $\dot{V}_x, \dot{V}_y, \ddot{V}_x, \ddot{V}_y$ as required by the controller.

Estimates of m and c_T are needed throughout the controller. As described above, the controller also uses these estimated values to zero the rotor speeds about the hover mode mg/c_T. Further, estimates of the moment arms d_{1-4} are needed for (3.69), which are easily measured. For the simulation experiment in this section, approximate values on the $\{m, c_T, d_{1-4}\}$ measurements are used, to show the robust and adaptive nature of the controller. The hyperparameters used for the controller in the simulation are given in Table 3.3.

Table 3.3: Hyperparameters used for the simulated quadcopter controller.

Hyperparameter	**Value**
d_{1-4}	0.03 m
c_T	3×10^{-10}
m	0.025 kg
D_1	0
η_1	2.2×10^7
λ_1	4
ϕ_1	0.006
F_1	$\text{diag}((10,) \times n_{H,1})$
G_1	$\text{diag}((10,) \times 1)$
H_1	200000
$n_{H,1}$	10
D_2, D_3	0
η_2, η_3	150
λ_2, λ_3	1
ϕ_2, ϕ_3	0.006
F_{23}	$\text{diag}((10,) \times n_{H,23})$
G_{23}	$\text{diag}((10,) \times 7)$
H_{23}	10000
$n_{H,23}$	15
D_4	0
η_4	5×10^5
λ_4	3
ϕ_4	0.001
F_4	$\text{diag}((10,) \times n_{H,4})$
G_4	$\text{diag}((10,) \times 7)$
H_4	1000
$n_{H,4}$	15

The quadcopter is initialized at rest at the position $r(t=0) = [-0.002, 0.002, 0.05]^T$ m, with an attitude of $\Phi(t=0) = [0.001, -0.001, 0.001]^T$ rad. The initial values of the coefficient matrices are chosen as $\hat{M}_1(t=0) = [1 \times 10^8]$, $\hat{M}_{23}(t=0) = \text{diag}(1000, 1000)$, $\hat{M}_4(t=0) = [1 \times 10^7]$, from approximations of the coefficients given in Equations (3.42), (3.51), (3.61), and (3.62), respectively. All NN weights are initialized by uniform sampling on the interval $[-0.1, 0.1)$. The state vectors input to the NNs in each of the subsystem controllers are $x_1 = [1]$, $x_{23} = [\Phi^T, \omega^T, 1]^T$, and $x_4 = [\Phi^T, \omega^T, 1]^T$. Note that these state vectors differ from the nominal state vectors proposed above – in the simulation tests conducted, it was found that values such as $\hat{f}$ and its derivatives change very slowly, and similar results in the controller can be achieved using simplified state vectors as input to the NNs. A 1 is appended to the state vectors input to each NN and to each vector of activations $\sigma(\hat{V}^T x)$ to account for bias terms in each NN. For tuning this controller, each subsystem was tuned individually for stability and desirable performance. For instance, the r_z subsystem controller requires higher learning gains than the other subsystems, and this controller was tuned first for stability in a hover mode. Next, the yaw controller was tuned for directional control, with the r_x and r_y controller being tuned last for full control of the quadcopter. Also note that $D_1 = D_2 = D_3 = D_4 = 0$ for all control subsystems, since the D term is used as a formality in the proof of controller stability. Tuning η alone in each controller renders the same effect, as $(D + \eta)$ is the effective gain in each controller. The trajectory-tracking simulation is run for 200 seconds, for one full figure eight. The 3D plot of position over time is shown in Figure 3.6. The error

over time of each control variable is shown in Figure 3.7, with the time history of the learned parameters in the controller shown in Figure 3.8.

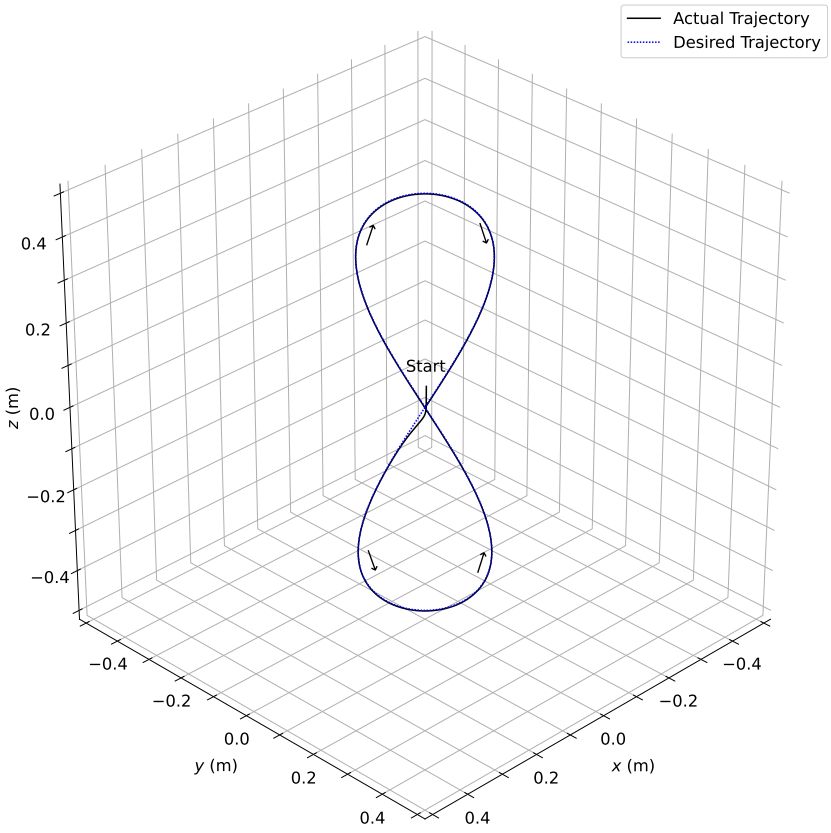

Figure 3.6: Quadcopter trajectory over time in 3D space.

In Figure 3.6, the quadcopter is seen to initially drop. This is due to a slight underestimation in the hover mode based on the assumed values of m and c_T in the controller. Once the gain $\hat{M}_1$ is adapted effectively (near $t = 4$ s in Figure 3.8), the r_z subsystem controller is able to achieve sliding mode. The learning gain H_1 required a large value for this reason, such that the quadcopter does not drop below some undesirable value in r_z. In Figure 3.8, the updates for the r_x, r_y and

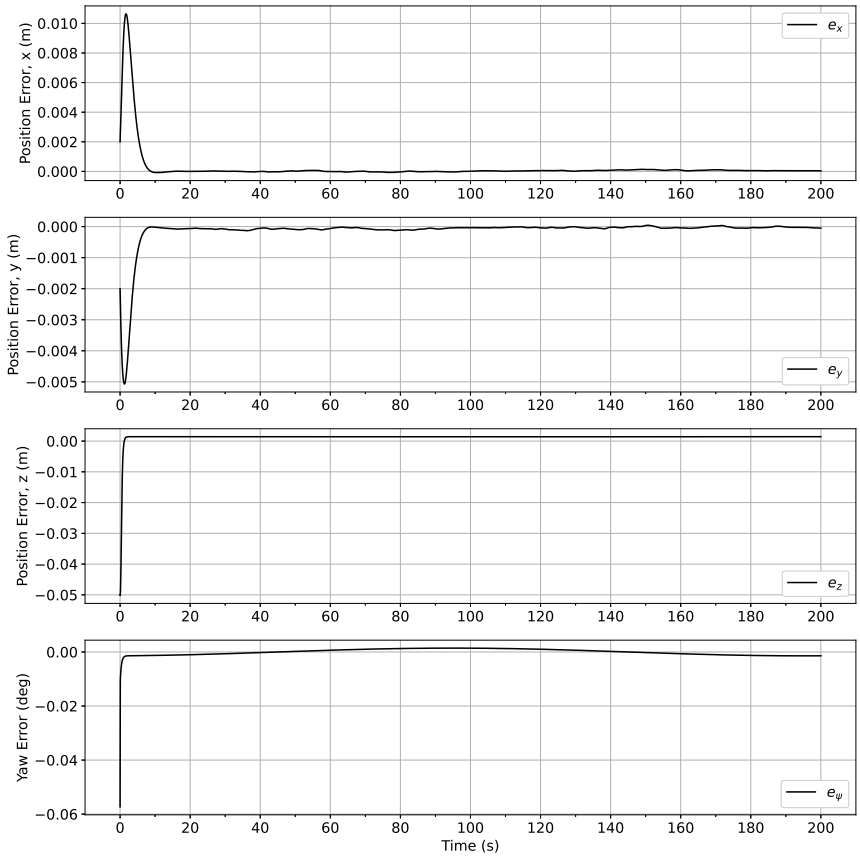

Figure 3.7: Error over time for each of the control variables in the quadcopter controller.

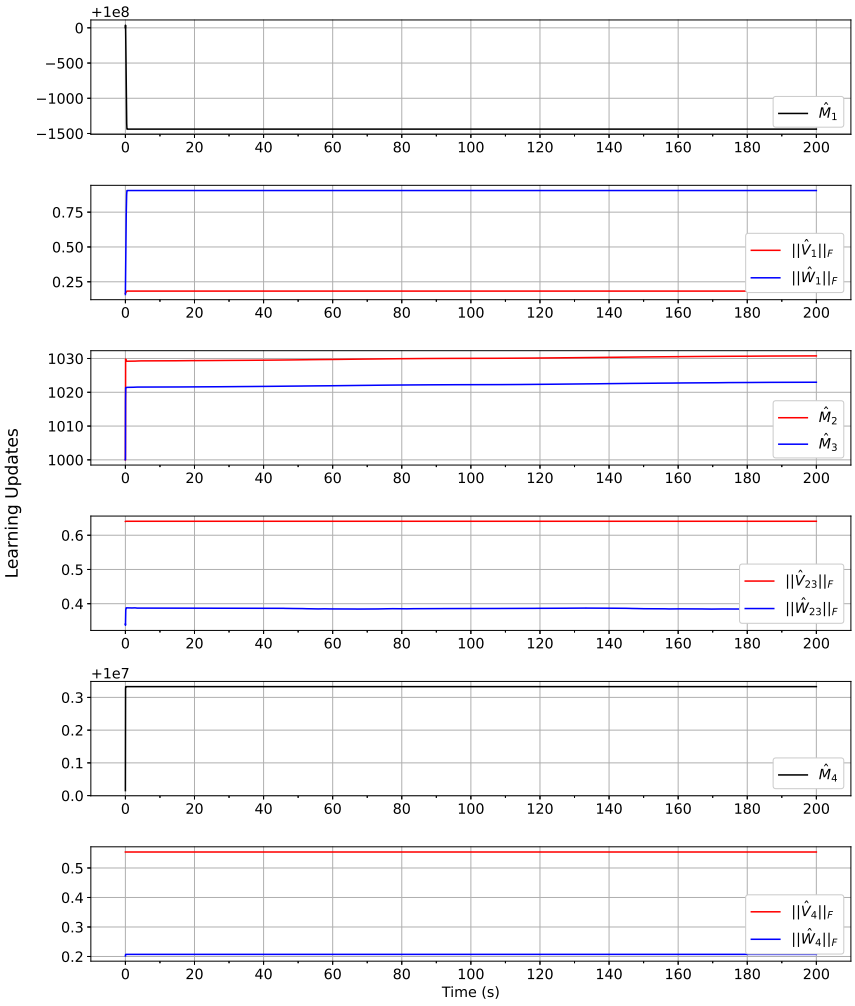

Figure 3.8: Learning parameter estimates over time for the quadcopter controller.

ψ controllers converge within the first few seconds of the simulation. This is due to the high gains and update rates required for these control subsystems, since any unstable flight early in the simulation would crash the quadcopter. With level flight established in the r_x, r_y control subsystem, the r_z controller can adapt to the correct thrust needed. In Figure 3.7, the bounded error in e_x and e_y is due to aerodynamic disturbances and noise in the numerical differentiatior. Note that, in controllers such as [78], numerical differentiation is not needed, as knowledge of the quadcopter system model is used to relate higher-order derivatives to the system states. However, the general controller in the beginning of this chapter is abstracted to any system under the dynamics form of (3.1), and this system-agnostic approach requires numerical differentiation when those system states are not measured or known. This can be seen as a sort of trade-off between *a priori* modeling and the online learning of system models as in this chapter. Further, the simulated quadcopter controller effectively rejects the 5 knot wind disturbance and adapts to the uncertainties in assumed parameter values. Based on the simulation results, the developed controller is able to accurately and desirably control the quadcopter system in the presence of parameter uncertainty and external disturbances.

Chapter 4. Conclusions, Discussion, and Future Work

4.1 Summary and Conclusions

Part I establishes the critical research progress made in incorporating learning-based AI/ML elements stably inside online controllers. The primary idea of Part I is that AI/ML elements can be cleanly reconciled with the analysis and theories of adaptive control – since the goals of adaptive control and the function approximation of AI/ML are often aligned, the use of adaptive control theory to evolve these AI/ML elements online provides rigorous convergence and stability guarantees not present in conventional AI/ML optimization. Further, using these AI/ML elements to learn important features directly in the control law can render more accurate and versatile controllers that require less *a priori* system modeling or identification than their conventional counterparts.

Namely, Chapter 2 describes incorporating the most commonly used tool in modern AI/ML, the feedforward neural network, inside of an adaptive control law. This chapter assumes a basic robot model and uses a nonlinear sliding mode control (SMC) term to guarantee learning stability and controller performance, especially during the initial stages while the learning parameters converge. This nonlinear control term is required due to the assumption of limited *a priori* knowledge about the system and the problem of initialization of neural networks with

no pretraining. The learning error is categorized as an internal "disturbance" and rejected by the nonlinear SMC term, with stability proven via the Lyapunov analysis given in the chapter. The developed control law uses both a neural network and direct parameter estimation to learn a model suitable for control fully online. This controller is compared to a conventional model-based SMC with no learning elements, in two simulated robot arm experiments. In the experiments, the robot manipulator experiences different loads on the end effector, both inside and outside of the designed load margin of the conventional SMC controller. The proposed controller is shown to successfully adapt to the end effector loads, even when conventional model-based SMC fails. Additionally, a novel update rule and projection is developed for estimation of a positive definite coefficient matrix, which helps simplify the learning problem for the neural network term. These experiments validate the adaptive nature of the developed controller, and this work shows how AI/ML techniques like neural networks can be used to extend the performance and versatility of conventional controllers such as SMC.

To maximize the impact and application of the research progress for this dissertation, Chapter 3 describes extending the general controller presented in Chapter 2 to two notable aerospace control problems: rigid-body spacecraft attitude control and full quadcopter control. This is done to showcase different modifications required to render desired performance from the general controller in specific control scenarios. In spacecraft attitude control, the parameterization of attitude orientation using quaternions requires some modification to variables used in the controller, such as the sliding surface. The spacecraft attitude con-

troller is verified in a sky-scanning trajectory tracking simulation under various perturbations, with the controller being particularly desirable for complicated spacecraft structures. The perturbations simulated are modeled to reflect the perturbation torques common to spacecraft operations, such as solar radiation pressure, gravity gradient, or docking/undocking with other spacecraft. In quadcopter control, varying-order control subsystems are used alongside virtual and intermediate control inputs to guarantee trajectory tracking in position and yaw. The developed controller is shown to be robust to internal modeling estimations and aerodynamic perturbations such as wind. This chapter verifies that the developed AI/ML-based online controllers can safely and stably control relevant aerospace systems, which was one of the original goals of this dissertation.

4.2 Discussion

As mentioned Chapter 1, ongoing challenges in online adaptation of AI/ML elements in controllers includes initialization, computational cost, catastrophic forgetting, and exploration without jeopardizing the control goal. The controllers described throughout Part I are initialized with limited-to-no system modeling or identification, so initialization becomes a pertinent issue. This is overcome by the nonlinear SMC term described above, thereby guaranteeing controller convergence assuming properly tuned gains. The computational cost of the developed controllers is relatively small when compared to the massive neural networks used in modern data science, often with millions of trainable parameters. However, in control of aerospace and robotic systems, computation must be minimized for

fast online control. The general controller of Chapter 2 namely has four primary calculations: the matrix-vector operations of the neural network term, the matrix-vector operations of the other control law terms, the matrix-vector operations of the parameter update rules, and the integration and update of the parameters. For this reason, few neural network hidden neurons were used in simulation. It was found that increasing the hidden layer neurons did not result in much improved control accuracy for the increase in computation. This observation is likely very relevant to the control of real-world systems, where acute real-time control in more important than marginal increases in control calculation fidelity.

An important facet of joining AI/ML and control, noticed during the implementation and testing of the developed controllers, is the notion of "learning." The neural networks in the controllers developed in Part I are evolved at each timestep using a rule similar to mean-squared-error-minimizing backpropagation, with various sliding variables replacing conventional prediction error. This is very different from typical neural network training common to modern data science, which usually includes backpropagation with a form of batch gradient descent. In typical neural network training, the network is validated for performance across the entire space of possible (or known) inputs. In the adaptive control formulation here, since the networks are only evolved using the most recent data point at each timestep, catastrophic forgetting becomes a topic of importance. However, the goal of the neural network training (or *evolution*, as it is commonly referenced in Part I) in the developed controllers is decreasing the relevant sliding variable to zero at the current timestep, thereby decreasing control error to zero. This is

very different from a data science point of view, where a machine learning engineer may be disconcerted to find that the neural networks trained online are used more as an "instrument" for control and much less for function approximation in the data-based modeling sense. In adaptive control, the notion of "learning" can be considered as converging on parameters that achieve the desired control goal. In AI/ML, learning is likely more concerned with the exploration and achievement of set goals without direct instruction. More discussion on some connections between adaptive control and machine learning can be found in the works of [10] and [79]. Lastly, the AI/ML elements in the developed controllers are designed to drive control performance. Exploration was not incorporated in the control goal of this research.

4.3 Future Work

There are many avenues for future development of the AI/ML-based online controllers described in Part I. Notably, the control formulations and update laws are formulated in continuous time – it is likely a discrete time formulation may have benefit to both controls engineers and AI/ML engineers; as real-world actuators, simulations, and epoch-based conventional AI/ML training all operate on discrete-time intervals. Further, as described in the stability proofs for the general controllers in Chapters 2 and 3, the learning stability and control convergence is guaranteed for a gain on the nonlinear SMC term greater than the combined internal learning "disturbance" and external disturbance. Upon initialization, since the neural network is not pretrained, this internal learning disturbance is high.

Thus, the gain on the nonlinear term – which also affects final control accuracy through the chatter attenuation via boundary layer – requires a high term initially. During learning, however, this disturbance will decrease, requiring a lower discontinuous gain for guaranteed stability (and thus, lower boundary layer and increased control accuracy). An interesting route of future work is an adaptation rule for this discontinuous gain to accommodate the various stages of the controller, from initialization to learning convergence to unseen disturbances.

Further, the AI/ML tool used in the controllers developed in Part I is the feedforward neural network, with only two layers of trainable parameters. Modern AI/ML implementations use much more complicated architectures of neural networks, including recurrent, convolution, attention, or residual layers, depending on the type of data and application. Even the feedforward neural networks use many more layers, thereby increasing the prediction power and nonlinearity for a given amount of neurons. The controllers developed in Part I only required a basic nonlinear function approximator, and thus only used a basic neural network architecture. While it is not clear if more advanced architectures would result in greater control accuracy or robustness, it is certainly of merit to study how to stabilize the evolution of these neural network architectures and data processing systems. The most immediate improvement to the update laws presented in Part I is likely a recursive relation for a deep neural network with an arbitrary number of layers.

Lastly, the extended controller described in Chapter 2 that estimates a positive definite coefficient matrix was found to experience reduced chattering

than the controller that estimates a diagonal coefficient matrix. The adaptive controllers compared in Chapter 2 were tuned to exhibit adaptivity over conventional model-based SMC. It is likely that a different set of hyperparameters for the M-projection controller could exhibit further desirable behavior, either in reduced chatter or increased control accuracy.

Part II: Control for Learning

Stable Online Adaptation of AI/ML Elements Using Control-Theoretic Techniques

Chapter 5. Online Transfer Learning Using Super-Twisting Control

This chapter presents research progress in using techniques from control theory to allow modern AI/ML tools to stably improve prediction performance online, while deployed. The mathematical rigor and associated stability guarantees of control-theoretic techniques give desirable convergence properties and error bounds to the AI/ML engineer, which is not generally possible in typical AI/ML training with gradient-based backpropagation. This chapter specifically considers an arbitrary-depth feedforward deep neural network (DNN), pretrained on a training dataset, deployed online under domain shift. The DNN is analyzed as a continuous-time dynamical system to be controlled, and the output of the penultimate DNN layer is assumed to be a functional basis for the last-layer prediction. Last-layer parameter update laws are developed that guarantee prediction performance online under domain shift, with algorithmic improvements and relevant proofs presented to maximize the effectiveness of the developed update laws. Notably, spectrally normalized training is shown to control the upper bound of online prediction error. This chapter is adapted from the work "Control-Theoretic Techniques for Online Adaptation of Deep Neural Networks in Dynamical Systems" by Jacob G. Elkins, Avimanyu Sahoo, and Farbod Fahimi, currently under re-

view for publication in the IEEE Transactions on Neural Networks and Learning Systems [12].

5.1 Introduction

Deep neural networks (DNNs), trained with gradient-based optimization via backpropagation, are currently the primary tool in modern artificial intelligence, machine learning, and data science. In many applications, DNNs are trained offline, through supervised learning or reinforcement learning, and deployed online for inference. However, training DNNs with standard backpropagation and gradient-based optimization gives no intrinsic performance guarantees or bounds on the DNN, which is essential for applications such as controls. Additionally, many offline-training and online-inference problems, such as sim2real transfer of reinforcement learning policies, experience domain shift from the training distribution to the real-world distribution. To address these stability and transfer learning issues, this chapter uses techniques from control theory to update DNN parameters online. The fully-connected feedforward DNN is formulated as a continuous-time dynamical system, and novel last-layer update laws are developed that guarantee desirable error convergence under various conditions on the time derivative of the DNN input vector. It is further shown that training the DNN under spectral normalization controls the upper bound of the error trajectories of the online DNN predictions, which is desirable when numerically differentiated quantities or noisy state measurements are input to the DNN. The proposed online DNN adaptation laws are validated in simulation to learn the

dynamics of the Van der Pol system under domain shift, where parameters are varied in online inference from the training dataset. The simulations demonstrate the effectiveness of using control-theoretic techniques to derive performance improvements and guarantees in DNN-based learning systems. A model reference adaptive control example is also provided for control of a robotic arm, where the developed last-layer update laws allow the DNN to effectively compensate for unmodeled dynamics on the simulated robotic arm.

5.2 Background and Literature Review

DNNs are biologically-inspired general nonlinear function approximators that learn from processing data. DNNs are currently the primary tool in modern artificial intelligence, machine learning, and data science; driving popular breakthroughs in reinforcement learning [8], natural language processing [80], and content generation [81]. DNNs are popular and successful due to their generality, ability to learn intricacies from data, and ease of implementation for parallel computation. The modeling and function approximation capability of DNNs has recently exploded, largely due to advancements in computer hardware and increased data generation.

Most commonly, DNNs are trained on an offline dataset via gradient-based optimization and backpropagation [82, 83]. Often, these DNNs are then deployed online for inference, with the DNN model's parameters static, only optimized during the pre-deployment training. If there is a change in the underlying process that generates the DNN training data, known as *concept drift* or *domain shift*,

DNN performance can suffer [84, 85]. It is desirable for DNN-based models to further improve their performance over time during deployment in the real-world, learning from the novel data being processed by the DNN. However, efficiently updating DNN model parameters from novel signals to guarantee improved performance is not straightforward. Retraining the entire DNN model on only the novel data can introduce catastrophic forgetting, with the model parameters over-optimizing to the recent inputs. However, retraining the entire DNN model with the fully updated dataset can become computationally expensive and inefficient at each datapoint [86]. Relevant problem areas in deep learning include transfer learning, domain adaptation, and domain generalization [87–90]. In many DNN applications, data is processed by the deployed DNN model sequentially in time, especially when the DNN is approximating a dynamical quantity or system. Common examples of online DNN deployment include forecast models [91] and policies trained with reinforcement learning [92]. In these examples, there are no intrinsic properties on the DNN trained with gradient-based backpropagation, such as boundedness of DNN outputs or guarantees of convergence.

The motivating example for this chapter is the sim2real transfer of reinforcement learning (RL) policies, particularly when policies are used as controllers [92, 93]. RL-based control policies are typically trained using a simulation of the real-world control problem, where the simulation is constructed to model the real-world physics as accurately as possible. Once desirable performance in simulation is achieved, the policy is then deployed onto the real-world control problem [94]. However, the real-world is highly nonlinear, complex, and difficult to model; and

obtaining dynamic models for use in simulation can be expensive [21]. A discrepancy between the simulated problem and the real-world problem represents a shift in the training data distribution and the online data distribution; and standard feedforward DNNs trained with conventional RL are not designed to adequately adapt to this distribution shift, resulting in degraded policy performance in the real-world. The progress in this chapter is further inspired by the controllers developed in [74], where a spacecraft attitude controller derived from pure RL has no intrinsic stability guarantees, limiting its real-world application and effectiveness. Common methods for solving the sim2real "reality gap" include domain randomization [95–97], enforcing robustness via adversarial training [98, 99], and meta-learning [100, 101].

Considering the important applications of DNNs to approximating dynamical systems, the study of how to train and implement DNNs to maximize online performance is of merit. In this chapter, techniques from control theory are used to evolve offline-trained DNNs online to improve performance when deployed under domain shift. The mathematical rigor of control theory can be used to establish desirable properties on DNN outputs, such as performance bounds and convergence guarantees. Conversely, neural networks have been studied extensively for use in control theory, beginning with notable works such as [102] and [32]. In most neural-network-based controllers, including Part I of this dissertation, the neural network is simply used as an online adaptation instrument to guarantee a control objective (such as tracking error convergence) [33–36, 38, 39, 46]. However, this dissertation includes these works as "learning for control," whereas

this chapter attempts to use "control for learning," utilizing the rigor and proven convergence properties of control theory to increase learning performance itself.

When a DNN of arbitrary depth is deployed to control or predict a dynamical system, the DNN itself can be modeled and analyzed as a dynamical system. Assuming the training distribution is reasonably close to the target (or online) distribution, the output of the next-to-last layer forms a basis for function approximation of the last layer. Similar to the distribution shift and transfer learning approaches which retrain the last layer while freezing upper layers in the DNN [103–106], the last layer of the DNN is evolved to learn online during operation in a provably-stable manner, shown in the block diagram in Figure 5.4. Fine-tuning DNN models by retraining lower layers is common in image processing and computer vision, such as in the works of [103–106] previously mentioned. Deep model reference adaptive control (MRAC) in [107] uses a similar offline DNN pretraining step with an online update rule based on concurrent learning, though the update rules and controller are specific to the linear MRAC case. The work in [46] derives control-based update laws for both inner and outer layer weights for an arbitrary-depth DNN in an assumed system. Both [46] and [107] utilize the projection operator to stabilize the DNN weight updates for the trajectory tracking error convergence control goal. The work in [44] uses a Lyapunov-based update law on the DNN outer layer, which is very similar to mean-squared-error-minimizing backpropagation. Further, DNN learning stability in [44] is guaranteed by a system control law. This chapter also takes inspiration from [26], which combines offline DNN learning with an online adaptation law. In [26], the authors use

a DNN to compensate for the aerodynamic disturbances in quadcopter control, where the custom-designed adversarial learning algorithm optimizes an invariant basis set for a Kalman-based online adaptation law. The works of [26] and [25] both use spectral normalization to bound the Lipschitz constant on the DNN for incorporation in a control law. This chapter aims to combine the ideas from these data science and control-based works to:

(1) Improve *learning* itself online, *i.e.*, decreasing DNN approximation error over time, specifically under domain shift. This is contrasted to decreasing trajectory tracking error of a control system by calculating a system input via some DNN output, as in the works listed above. While the proposed DNN update law can certainly be used inside an outer-loop controller (as shown in Section 5.8 below), this chapter aims to generalize to online DNN learning.

(2) Stabilize the DNN outputs intrinsically by directly updating the DNN parameters.

(3) Show how machine learning developments, such as spectral normalization, can give desirable control of prediction error convergence bounds in the proposed control-based online DNN update.

5.3 Contributions

The novel update rule described in this chapter is based on the super-twisting algorithm (STA) [16, 108, 109]. The STA is used in control, observation, and differentiation, when finite-time asymptotic convergence is desired via a continuous control law [16]. Using the STA, this chapter further shows that other

deep learning techniques can be combined with the developed online adaptation law to improve performance, such as spectral normalization during training [110–112]. When the time derivative of the DNN input vector is noisy (from measurement or numerical differentiation), it is shown that spectrally normalizing the DNN during training defines the upper bound of the convergence error. The primary contributions of this chapter are as follows:

(1) A novel online last-layer update law is developed for a pretrained DNN when implemented onto a dynamical system, provably driving prediction error to (or, depending on implementation, upper-bounding near) zero. This is particularly desirable under domain shift between the training data and online data distributions.

(2) Spectral normalization is utilized to stabilize DNN training and to control the upper bound on the prediction error, which improves online DNN prediction performance by lowering the ultimate upper bound on the prediction error.

5.4 Motivating Example

This section presents a concise and simple example to motivate the developments described in this chapter. Consider the unforced Van der Pol equation [113], with dynamics written as

$$\begin{aligned} \dot{z} &= \epsilon \left(z - \frac{1}{3} z^3 - \theta \right) \\ \dot{\theta} &= z, \end{aligned} \tag{5.1}$$

where ϵ is a constant parameter. The goal is to use a DNN to learn the unknown dynamics of z, assuming the dynamics of θ are known, and z, θ, $\dot{z}$ can be measured. Training a DNN on a feature dataset $\mathcal{X} = \{z, \theta\}$ and a label dataset of $\mathcal{Y} = \{\dot{z}\}$, generated by the nominal system of Equation (5.1) with $\epsilon = 1$, $\dot{z}$ can be approximated by a DNN to arbitrary accuracy, shown in Figure 5.1.

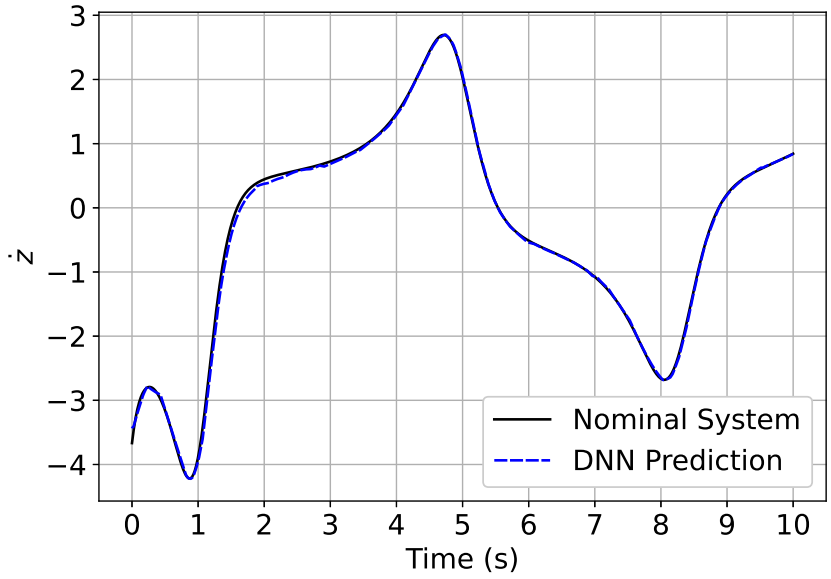

Figure 5.1: DNN predictions on the nominal system ($\epsilon = 1$).

However, assuming a "real" system of Equation (5.1) with $\epsilon = 1.5$, the DNN experiences domain shift, with the test/online data (real system) generated

from a different distribution than the training data (nominal system). Without any online learning or retraining, the DNN performance suffers, as shown in Figure 5.2.

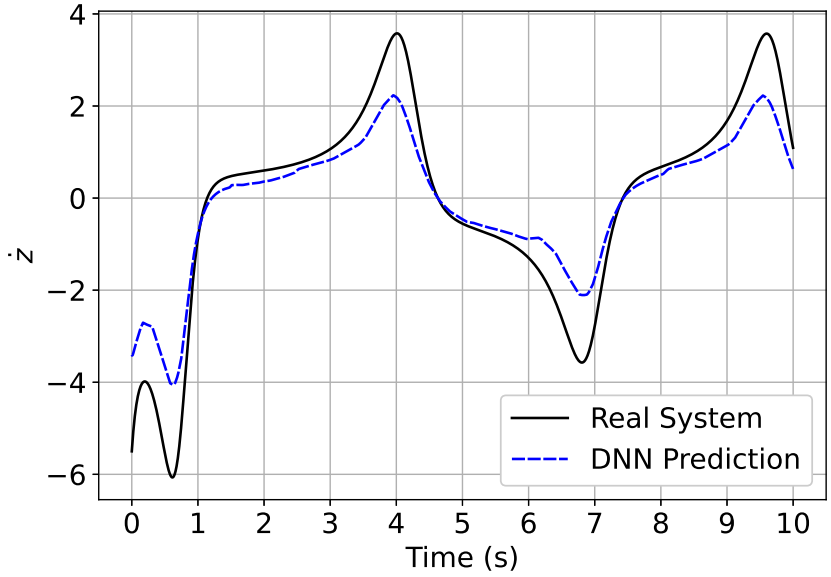

Figure 5.2: DNN predictions on the real system ($\epsilon = 1.5$).

Since the DNN is trained to approximate the dynamical system in (5.1), the DNN itself can be considered and analyzed as a dynamical system. Considering the DNN as a dynamical system to control, control theory can be applied to update the DNN parameters online. This is done to achieve desirable performance on the real domain-shifted system without full retraining, shown in Figure 5.3.

The developed online update rule in this chapter takes advantage of the dynamical nature of the DNN predicting online to achieve desirable performance under domain shift. As described above, this is a common problem in transfer learning and reinforcement learning, notably in sim2real transfer of control policies [92].

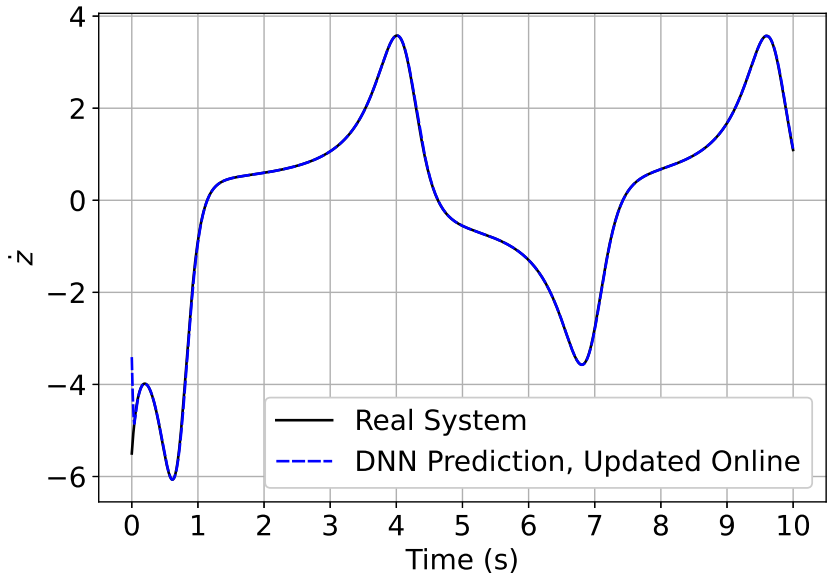

Figure 5.3: Online-adapted DNN predictions on the real system ($\epsilon = 1.5$).

The methodology developed in this chapter is summarized in Figure 5.4, for a sim2real control example.

5.5 Notation and Preliminaries

This section describes the notation used throughout this chapter, the DNN architecture considered in this chapter, and the general theory and assumptions of the control algorithm used.

5.5.1 Notation

The notation of this chapter is again specific to this chapter. The set of all real numbers is denoted as $\mathbb{R}$, and the set of real vectors with n components

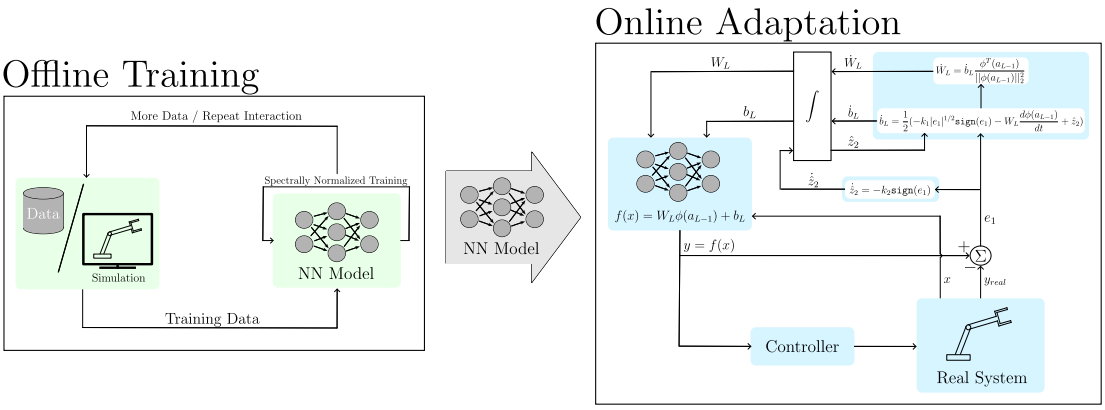

Figure 5.4: Block diagram of the proposed online learning method, for a sim2real control example. The neural network model is first trained offline using conventional supervised or reinforcement learning under spectral normalization. The learned model, once deployed, is then updated online using the adaptation laws in Equations (5.10), (5.11), and (5.12).

is denoted as $\mathbb{R}^n$. The set of nonnegative real numbers is denoted as $\mathbb{R}_{\geq 0}$, and the set of real positive numbers is denoted as $\mathbb{R}^+$. A general function of time $y = f(x,t)$, whose m time derivatives are continuous and differentiable, is denoted as $f \in C^m(T \times X, Y)$, where $t \in T$ is the time domain, $x \in X$ is the input domain, and $y \in Y$ is the output range. The first derivative with respect to time for a variable x is denoted as $\frac{dx}{dt} = \dot{x}$ for shorthand. $||\cdot||_2$ denotes the 2-norm (or Euclidean norm) of a vector. The operators $\lambda_{min}(Q)$ and $\lambda_{max}(Q)$ return the minimum and maximum eigenvalues of a matrix Q, respectively. Any other relevant and uncommon notations will be defined throughout the chapter.

5.5.2 Deep Neural Network Preliminaries

This chapter considers a fully-connected, feedforward DNN $f(x) : \mathbb{R}^n \to \mathbb{R}^m$, with L layers:

$$f(x) = W_L \phi(a_{L-1}) + b_L, \tag{5.2}$$

where a_{L-1} is the output of the next-to-last layer, written as

$$a_{L-1} = W_{L-1}\phi(W_{L-2}\phi(\cdots\phi(W_1 x + b_1)\cdots) + b_{L-2}) + b_{L-1}, \tag{5.3}$$

where W_i and b_i is the weight and bias of the i^{th} layer, respectively; $\phi(\cdot)$ is a bounded nonlinear activation function, and $x \in \mathbb{R}^n$ is the neural network input. The DNN is assumed to be trained offline to optimize a feedback signal (such as minimizing error or maximizing reward) to be implemented for online inference, processing data and receiving error signals sequentially in time.

5.5.3 Control Preliminaries

As discussed throughout this dissertation, in model-based control of real-world systems, it is difficult to accurately model the real-world system being controlled. Control engineers have long studied how to design closed-loop controllers that render desirable performance under model uncertainty and discrepancy [16, 113]. One such robust control methodology is *sliding mode control*, which aims to drive the nonlinear system to a desired manifold through the use of a discontinuous control signal [17]. The controller is typically designed for finite-time convergence of the sliding manifold to zero, which in turn is formulated for exponential convergence of error to zero.

In conventional sliding mode control, the discontinuous control signal is undesirable, as it can introduce chatter and harm actuators in real-world systems. The *super-twisting algorithm* (STA), introduced in [108], is a sliding mode control algorithm that uses a continuous control signal to guarantee finite-time convergence of the sliding variable, among other important qualities (cf. [16]). This chapter follows the STA formulation and applies the Lyapunov stability proof given in [109].

Consider the STA system in state variable form

$$\begin{aligned} \dot{x}_1 &= -k_1 |x_1|^{1/2} \texttt{sign}(x_1) + x_2 + p_1(x_1, x_2, t) \\ \dot{x}_2 &= -k_2 \texttt{sign}(x_1) + p_2(x_1, x_2, t), \end{aligned} \tag{5.4}$$

where $x_1, x_2 \in \mathbb{R}$ are state variables, $k_1, k_2 \in \mathbb{R}^+$ are constant design gains, and $p_1(x_1, x_2, t)$, $p_2(x_1, x_2, t) : \mathbb{R}^2 \times \mathbb{R}_{\geq 0} \to \mathbb{R}$ are system perturbations. Note that x_1, x_2 here are unrelated to the DNN input vector, x. The right-hand side of Equation (5.4) is discontinuous, so the solution to the differential inclusion of (5.4) is understood in the sense of Filippov [114]. The STA is widely used in control, observation, and robust numerical differentiation [16, 115].

Notably, the STA in (5.4) is robust to the perturbation defined by

$$\begin{aligned} p_1(x_1, x_2, t) &= 0 \\ |p_2(x_1, x_2, t)| &\leq D, \end{aligned} \tag{5.5}$$

where $D \in \mathbb{R}^+$ is any constant, provided that the gains k_1, k_2 in (5.4) are selected appropriately [108, 109, 116–118]. More precisely, (5.4) under the perturbation defined in (5.5) establishes the origin $\{x_1 = 0, x_2 = 0\}$ as a global finite-time stable equilibrium point. This was proven geometrically in [118], by the homogeneity property of the controllers in [117] and [119], and by Lyapunov analysis in [109]. The Lyapunov analysis in [109] also establishes that, assuming the STA in (5.4) is robust to perturbations of the form defined in (5.5), the STA is also robust to perturbations of the form

$$\begin{aligned} |p_1(x_1, x_2, t)| &\leq \delta_1 + \delta_2 \sqrt{|x_1| + x_2^2} \\ |p_2(x_1, x_2, t)| &\leq D, \end{aligned} \tag{5.6}$$

where $\delta_1, \delta_2 \in \mathbb{R}_{\geq 0}$ are constants. The authors in [109] further show that if $\delta_1 = 0$, $p_1(x_1, x_2, t)$ will vanish at the origin, and the STA in (5.4) will still converge to the origin in finite time. However, for perturbations that do not vanish at the origin in the x_1 channel (that is, $\delta_1 \neq 0$ such that $p_1(0, 0, t) \neq 0$), the state trajectories will not generally converge to the origin, but will be globally ultimately bounded [113]. This result is important, as in the DNN training development, the perturbation in the x_1 channel manifests from noise in the time derivative of the DNN input vector, from either numerical differentiation or measurement noise.

5.6 Online DNN Updates Using Super-Twisting Control

Assume an arbitrary function $y \in C^2(T \times X, Y)$ to be approximated offline by an arbitrary-depth DNN, where $t \in T = \mathbb{R}_{\geq 0}$, $x \in X = \mathbb{R}^n$, and $y \in Y = \mathbb{R}^m$. The "nominal" system can be written in state space form as

$$\begin{aligned} \dot{z}_1 &= z_2 \\ \dot{z}_2 &= \ddot{y} \\ y &= z_1, \end{aligned} \tag{5.7}$$

where $z_1, z_2 \in \mathbb{R}^m$ are state variables. The DNN is trained offline on data generated from the nominal system to some arbitrary approximation accuracy.

The trained DNN is assumed to be implemented online to estimate the "real" system, y', which is a domain-shifted process of the nominal system, y, written as

$$\begin{aligned} \dot{z}'_1 &= z'_2 \\ \dot{z}'_2 &= \ddot{y}' \\ y' &= z'_1, \end{aligned} \tag{5.8}$$

where $z'_1, z'_2 \in \mathbb{R}^m$ are real-system state variables. The goal of this chapter is to use control theory to find a provably-stable update law such that the DNN trained on the nominal system can perform effectively on the real system using feedback

from an error signal. As stated above, this problem corresponds to domain shift and transfer learning problems common in using DNNs for reinforcement learning and forecasting, to name a few. In general, this problem is applicable to any DNN which is implemented to approximate a dynamical system online.

One possible method for the DNN-based model to learn online includes retraining the entire model on both the nominal and real system data at certain intervals during deployment. This method becomes computationally inefficient and expensive as the training dataset grows, and typical backpropagation with gradient-based optimization does not intrinsically provide stability guarantees. Further, only retraining the model on the online real-system data acquired can induce catastrophic forgetting, with the DNN parameters over-optimizing to the recent data [86]. Taking inspiration from [26], only the parameters of the last layer of the DNN are updated, which is computationally efficient and preserves the overall learned feature representation of the training data generated by the nominal system. In this way, the output of the next-to-last layer acts as a basis for function approximation of the last layer of the DNN. In [26], a custom adversarial learning algorithm is designed to maximize the independence of the learned basis output. This chapter simply considers a DNN pretrained with conventional gradient-based optimization, due to its prevalence in modern AI/ML. It is thus assumed that the output of the next-to-last-layer, $\phi(a_{L-1})$, which can be considered as the learned representation of the important features of both y and y', is a suitable basis for approximation of the DNN's output layer.

To implement a controller for updating the last layer of the DNN online, (5.2) is differentiated with respect to time to get

$$\dot{\hat{y}} = \frac{df(x)}{dt} = \frac{\partial f}{\partial t} + \frac{\partial f}{\partial x}\frac{\partial x}{\partial t} = \dot{W}_L \phi(a_{L-1}) + \dot{b}_L + \Gamma, \tag{5.9}$$

where $\hat{y} \in \mathbb{R}^m$ is the DNN output and $\Gamma = \Gamma(\dot{x}) = W_L \frac{d\phi(a_{L-1})}{dt}$ for notational simplicity. The control problem is to derive update laws $\dot{W}_L$ and $\dot{b}_L$ to drive $e_1 = \hat{y} - y' \to 0$ as $t \to \infty$.

Note that the activation derivative term $\frac{d\phi(a_{L-1})}{dt}$ term in Γ is easily calculated using the chain rule, similar to backpropagation of error. The term Γ is only a function of the time derivative of the DNN input vector, $\dot{x}$, since the weights and biases of the DNN are known and unchanging with respect to time in this formulation. Thus, from the $\frac{\partial f}{\partial x}\frac{\partial x}{\partial t}$ term in the chain rule, the online last-layer update can be said to take advantage of the knowledge of the DNN architecture and the knowledge of $\dot{x}$. This chapter discusses two cases below: (1) when $\dot{x}$ is known and (2) when $\dot{x}$ is estimated or noisy.

5.6.1 Case I: Known $\dot{x}$

The online update laws for W_L, the output layer weight, and b_L, the output layer bias, are given as

$$\dot{b}_L = \frac{1}{2}(-k_1|e_1|^{1/2}\texttt{sign}(e_1) - \Gamma + \hat{z}_2) \tag{5.10}$$

$$\dot{W}_L = \dot{b}_L \frac{\phi^T(a_{L-1})}{||\phi(a_{L-1})||_2^2}, \tag{5.11}$$

where $\hat{z}_2$ is an augmented integral control term, evolved as

$$\dot{\hat{z}}_2 = -k_2 \texttt{sign}(e_1) \tag{5.12}$$

and $k_1, k_2 \in \mathbb{R}^+$ are again constant design gains. Note that in (5.10), $\Gamma = \Gamma(\dot{x})$ is used, since $\dot{x}$ is known in this case.

Theorem 5.6.1. *Suppose the approximation target $y' \in C^2(T \times X, Y)$, and its second time derivative is bounded such that $|\ddot{y}'| \leq D_y$. Suppose the activation derivative term $\Gamma = W_L \frac{d\phi(a_{L-1})}{dt}$ is known. Further, suppose y' is approximated by the DNN in* (5.2)*, which is trained offline on data generated from y, and the NN input x is continuously differentiable with respect to time. Then, for every $D_y > 0$, there exist design gains k_1, k_2 such that the last-layer update rules in* (5.10)*,* (5.11)*, and* (5.12) *cause $e_1 = 0$ to be a robustly, globally, finite-time stable equilibrium point, such that $e_1 \to 0$ at some finite time $t < t'(e_1, e_2)$.*

Proof. Substituting the update laws in Equations (5.10)-(5.12) into Equation (5.9) gives the system

$$\begin{aligned} \dot{\hat{z}}_1 &= -k_1 |e_1|^{1/2} \texttt{sign}(e_1) + \hat{z}_2 \\ \dot{\hat{z}}_2 &= -k_2 \texttt{sign}(e_1) \\ \hat{y} &= \hat{z}_1. \end{aligned} \tag{5.13}$$

Subtracting the system in (5.13) by the real system given in (5.8) gives the DNN error dynamics under the proposed online STA update given in (5.10)-(5.12):

$$\begin{aligned} \dot{e}_1 &= -k_1|e_1|^{1/2}\texttt{sign}(e_1) + e_2 \\ \dot{e}_2 &= -k_2\texttt{sign}(e_1) - \ddot{y}', \end{aligned} \tag{5.14}$$

where $e_2 = \hat{z}_2 - z_2'$. Since it is assumed the real system signal is bounded such that $|\ddot{y}'| \leq D_y$, each component of the system given in (5.14) is equivalent to the robust scalar STA system in (5.4) under the perturbation given in (5.5), with $p_1(x_1, x_2, t) = 0$ and $|p_2(x_1, x_2, t)| = |\ddot{y}'| \leq D_y$. The following analysis holds for each component of the system in (5.14).

The derivative of the vector $\zeta = [\zeta_1, \zeta_2]^T = [|e_1|^{1/2}\texttt{sign}(e_1), e_2]$ can be written as

$$\dot{\zeta} = \frac{1}{|\zeta_1|}A\zeta, \tag{5.15}$$

where $|\zeta_1| = |e_1|^{1/2}$ and

$$A = \begin{bmatrix} -\frac{1}{2}k_1 & \frac{1}{2} \\ -k_2 & 0 \end{bmatrix}. \tag{5.16}$$

For the system in (5.14), the authors of [109] consider the strict Lyapunov function candidate

$$V(e) = \zeta^T P \zeta, \tag{5.17}$$

where $e = [e_1, e_2]^T$ and $P = P^T$ is positive definite, related to matrix A via the Algebraic Lyapunov Equation (ALE)

$$A^T P + PA = -Q_R. \tag{5.18}$$

Assuming appropriate selection of gains k_1, k_2 ([109], Algorithm 1), the time derivative of (5.17) along the system trajectories in (5.14) thus satisfies the relation

$$\dot{V} \leq -|e_1|^{1/2} \zeta^T Q_R \zeta \tag{5.19}$$

almost everywhere, where $Q_R = Q_R^T$ is positive definite. However, the Lyapunov function in (5.17) is not locally Lipschitz continuous, as assumed in classical Lyapunov theory [114, 120]. By Zubov's theorem, which permits continuously differentiable (and not necessarily locally Lipschitz continuous) Lypaunov functions [113], the Lyapunov function in (5.17) decreases to zero monotonically in time [109].

Further, the time of error convergence to the origin is finite, upper bounded by

$$t' = \frac{2}{\bar{\sigma}} V^{1/2}(e(t=0)), \tag{5.20}$$

where

$$\bar{\sigma} = \frac{\lambda_{\min}^{1/2}(P)\lambda_{\min}(Q_R)}{\lambda_{\max}(P)} \tag{5.21}$$

and the notation $\lambda_{\min}(A)$ and $\lambda_{\max}(A)$ denotes the minimum and maximum eigenvalue on the spectrum of some matrix A, respectively. For details on the time convergence of the STA algorithm and gain selection of k_1, k_2, the reader is referred to [109], Theorem 2.

□

5.6.2 Case II: Unknown or Estimated $\dot{x}$

In some applications, the activation derivative term $\frac{d\phi(a_{L-1})}{dt}$ term is noisy and/or estimated. This can happen in some model reference adaptive control cases, specifically when the time derivative of the DNN input vector, $\dot{x}$, must be numerically differentiated for (or is calculated using noisy measurements, such as from real-world sensors). This can be seen by calculating $\frac{d\phi(a_{L-1})}{dt}$ for the DNN given in (5.2):

$$\frac{d\phi(a_{L-1})}{dt} = \phi'(a_{L-1}) \odot W_{L-1}(\phi'(a_{L-2}) \odot W_{L-2}(\cdots \phi'(a_1) \odot W_1 \dot{x})), \tag{5.22}$$

where $\odot$ denotes the Hadamard product, $a_i = W_i\phi(a_{i-1}) + b_i$ is the output of the i^{th} DNN layer, and $\phi'(z) = d\phi/dz$ denotes the activation function derivative. In (5.22), the only potentially unknown term is $\dot{x}$, since the weights and biases of the DNN are known. Using (5.22), $\Gamma = W_L \frac{d\phi(a_{L-1})}{dt}$ can be expanded as

$$\Gamma = W_L(\phi'(a_{L-1}) \odot W_{L-1}(\phi'(a_{L-2}) \odot W_{L-2}(\cdots \phi'(a_1) \odot W_1 \dot{x}))). \tag{5.23}$$

Denoting the estimate of Γ as $\hat{\Gamma} = \Gamma(\dot{\hat{x}})$ for shorthand, $\hat{\Gamma}$ can similarly be written as

$$\hat{\Gamma} = W_L(\phi'(a_{L-1}) \odot W_{L-1}(\phi'(a_{L-2}) \odot W_{L-2}(\cdots \phi'(a_1) \odot W_1 \dot{\hat{x}}))), \tag{5.24}$$

where $\dot{\hat{x}}$ denotes the estimate of $\dot{x}$. The STA online update rules in (5.10)-(5.12) are thus modified to use the estimate $\hat{\Gamma}$:

$$\dot{b}_L = \frac{1}{2}(-k_1|e_1|^{1/2}\texttt{sign}(e_1) - \hat{\Gamma} + \hat{z}_2) \tag{5.25}$$

$$\dot{W}_L = \dot{b}_L \frac{\phi^T(a_{L-1})}{||\phi(a_{L-1})||_2^2} \tag{5.26}$$

$$\dot{\hat{z}}_2 = -k_2\texttt{sign}(e_1), \tag{5.27}$$

where $k_1, k_2 \in \mathbb{R}^+$ are again constant design gains. In this case, the estimate of $\hat{\Gamma}$ will introduce a perturbation term in the x_1 channel of the system in (5.4), which will cause the error system trajectories to not converge to zero but to be ultimately bounded.

Since it is desirable to decrease the error trajectory bound, this section further explores how to quantify and control the bounds on Γ and $\hat{\Gamma}$. The works [26] and [25] have shown that training DNNs under spectral normalization (SN) can be used to derive convenient stability guarantees when using DNNs in dynam-

ical systems. Namely, spectral normalization controls the Lipschitz constant of a DNN, which defines the bound on the DNN output from a bounded input [110]. The authors of [26], [25], and [110] each show that SN training can improve out-of-domain generalization on DNN predictions and stabilizes training, especially in sensitive network structures such as Generative Adversarial Networks [112].

Definition 5.6.2. A real function g is Lipschitz continuous if there exists a constant $\gamma_{Lip} \in \mathbb{R}^+$ such that

$$\frac{||g(w_1) - g(w_2)||_2}{||w_1 - w_2||_2} \leq \gamma_{Lip} \tag{5.28}$$

for any w_1, w_2 in the domain of g. The smallest constant γ_{Lip} satisfying (5.28), $||g||_{Lip}$, is called the Lipschitz constant, which provides a convenient bound for relating function output given a bounded input.

Following the analysis in [25] and [112], the Lipschitz constant of a function g is the maximum singular value of its gradient in the domain, written as $||g||_{Lip} = \texttt{sup}_\xi \sigma(\nabla g(\xi))$, where $\sigma(\cdot)$ denotes the maximum singular value operator. The DNN in (5.2) is a composition of functions, recursively $h_i(\xi) = W_i\xi_{i-1} + b_i$ and $\xi_{i-1} = \phi(a_{i-1})$ for the i^{th} layer. Further, the Lipschitz constant for a composition of functions $g_L \circ g_{L-1} \circ \cdots \circ g_1$ is bounded by the inequality

$$||g_L \circ g_{L-1} \circ \cdots \circ g_1||_{Lip} \leq ||g_L||_{Lip}||g_{L-1}||_{Lip} \cdots ||g_1||_{Lip}. \tag{5.29}$$

Since $||g||_{Lip} = \texttt{sup}_\xi \sigma(\nabla g(\xi))$, the Lipschitz constant of the i^{th} DNN layer is $||h_i||_{Lip} = \texttt{sup}_{\xi_{i-1}} \sigma(\nabla(W_i\xi_{i-1} + b_i)) = \texttt{sup}_{\xi_{i-1}} \sigma(W_i) = \sigma(W_i)$. That is, the Lips-

chitz constant of the i^{th} DNN layer is the maximum singular value of the weight matrix W_i. This can be calculated by $\sigma(W_i) = \texttt{max}_j(\sqrt{\lambda_j(W_i^T W_i)})$, where $\lambda(\cdot)$ is the eigenvalue operator. Using the inequality in (5.29), the Lipschitz constant for the DNN in (5.2) can be bounded by

$$||f(x)||_{Lip} \leq \sigma(W_L)||\phi||_{Lip}\sigma(W_{L-1})||\phi||_{Lip} \cdots \sigma(W_1), \tag{5.30}$$

where the Lipschitz constant of the activation functions $||\phi||_{Lip}$ can be easily found based on the activation function used in the DNN. The ReLU activation function is defined as $\phi(a_i) = \texttt{max}(0, a_i)$. Its derivative can be written as

$$\phi'(a_i) = \begin{cases} 1 & \text{if } a_i \geq 0, \\ 0 & \text{otherwise,} \end{cases} \tag{5.31}$$

which shows a Lipschitz constant $||\phi||_{Lip} = 1$. The Lipschitz constant of the DNN in (5.2) with ReLU activations is thus bounded by

$$||f(x)||_{Lip} \leq \prod_{i=1}^{L} \sigma(W_i), \tag{5.32}$$

which follows directly from (5.30). To control $||f(x)||_{Lip}$ during training, Algorithm 1 is implemented, where γ is the desired upper bound on the Lipschitz constant of the DNN in Equation (5.2) with ReLU activations, such that $||f(x)||_{Lip} \leq \gamma$ [25].

Algorithm 1 Spectrally normalized ReLU DNN training.

for epoch **in** range(max_epochs)

optimize NN parameters W_i, b_i **for** $i = 1, 2, \cdots, L$

for $i = 1, 2, \cdots, L$:

$\sigma_i = \sigma(W_i)$

if $\sigma_i > \gamma^{1/L}$:

$W_i \leftarrow \frac{W_i}{\sigma_i}\gamma^{1/L}$

else:

continue

Note that in Algorithm 1, the normalizing weight update $W_i \leftarrow \frac{W_i}{\sigma_i}\gamma^{1/L}$ upper bounds the Lipschitz constant of the i^{th} DNN layer h_i by $||h_i||_{Lip} \leq \gamma^{1/L}$.

Lemma 5.6.3. *Consider the DNN given in* (5.2)*, with activation functions* $\phi(\cdot)$ *defined as the rectified linear unit (ReLU) function, trained with spectral normalization via Algorithm 1. If the Lipschitz constant of the DNN is upper bounded by* $||f(x)||_{Lip} \leq \gamma$*, then the Lipschitz constant of* Γ *is also upper bounded by* $||\Gamma||_{Lip} \leq \gamma$.

Proof. From (5.23), it can be seen that Γ is a recursion of linear operators, similar to the analysis above for the ReLU DNN itself. Equation (5.23) can be rewritten as a recursive composition of functions as

$$\Gamma = h'_L \circ z'_{L-1} \circ h'_{L-1} \circ \cdots \circ z'_1 \circ h'_1, \tag{5.33}$$

where $h'_i = W_i z'_{i-1}$ and $z'_{i-1} = \phi'(a_{i-1}) \odot h'_{i-1}$. The Lipschitz constant of a linear map is its maximum singular value of its gradient, giving $||h'_i||_{Lip} = \sigma(W_i)$. The function z'_{i-1} can be rewritten as the linear map $z'_{i-1} = \texttt{diag}(\phi'(a_{i-1}))h'_{i-1}$ such that $||z'_{i-1}||_{Lip} = \sigma(\texttt{diag}(\phi'(a_{i-1})))$. The maximum singular value of a diagonal matrix is equivalent to the infinity norm of its diagonal. With the ReLU derivative given in (5.31), $||z'_{i-1}||_{Lip} = ||\phi'(a_{i-1})||_\infty = 1$. From the inequality in (5.29), the Lipschitz constant of Γ is thus upper bounded by

$$||\Gamma||_{Lip} \leq \sigma(W_i)\sigma(W_{i-1}) \cdots \sigma(W_1) = \prod_{i=1}^{L} \sigma(W_i). \tag{5.34}$$

Using Algorithm 1 to train the DNN controls the singular values of the DNN weights, such that $\prod_{i=1}^{L} \sigma(W_i) = \gamma$. Thus, the Lipschitz constant of Γ is upper bounded by

$$||\Gamma||_{Lip} \leq \gamma. \tag{5.35}$$

□

Theorem 5.6.4. *Suppose the approximation target $y' \in C^2(T \times X, Y)$, and its second time derivative is bounded such that $|\ddot{y}'| \leq D_y$. Suppose the activation derivative term $\Gamma = W_L \frac{d\phi(a_{L-1})}{dt}$ is estimated by $\hat{\Gamma}$, with $\dot{x}$ and its approximation $\dot{\hat{x}}$ both bounded. Further, suppose y' is approximated by the DNN in* (5.2) *with ReLU activation functions, which is trained offline on data generated from y via Algorithm 1, and the NN input x is continuously differentiable with respect to time. Then, for every $D_y > 0$, there exist design gains k_1, k_2 such that the*

last-layer update rules in (5.25), (5.26), *and* (5.27) *cause the error trajectories* e_1, e_2 *to be globally ultimately bounded. The upper bound of each component of the combined error* $||\zeta||_2 = \sqrt{|e_1| + e_2^2}$ *is defined by*

$$\bar{e} = \sqrt{\frac{\lambda_{max}(P)}{\lambda_{min}(P)}} \frac{\eta\gamma||\dot{x} - \dot{\hat{x}}||_2}{(1-\kappa)\lambda_{min}(Q_R)}. \tag{5.36}$$

Proof. Substituting the update laws in Equations (5.25)-(5.27) into Equation (5.9) renders the system

$$\begin{aligned} \dot{\hat{z}}_1 &= -k_1|e_1|^{1/2}\texttt{sign}(e_1) + \hat{z}_2 + \Gamma - \hat{\Gamma} \\ \dot{\hat{z}}_2 &= -k_2\texttt{sign}(e_1) \\ \hat{y} &= \hat{z}_1. \end{aligned} \tag{5.37}$$

Subtracting the system in (5.37) by the real system given in (5.8), the DNN error dynamics under the modified online STA update rules in (5.25)-(5.27) are written as

$$\begin{aligned} \dot{e}_1 &= -k_1|e_1|^{1/2}\texttt{sign}(e_1) + e_2 + \Gamma - \hat{\Gamma} \\ \dot{e}_2 &= -k_2\texttt{sign}(e_1) - \ddot{y}'. \end{aligned} \tag{5.38}$$

Since it is assumed $|\ddot{y}'| \leq D_y$, each component of the system given in (5.14) is equivalent to the scalar STA system in (5.4) with perturbation terms $p_1(x_1, x_2, t) = \Gamma - \hat{\Gamma}$ and $p_2(x_1, x_2, t) = -\ddot{y}'$. The perturbation in the e_2 channel, p_2, is assumed

to be upper-bounded by $|p_2(x_1, x_2, t)| = |\ddot{y}'| \leq D_y$. The following analysis holds for each component of the system in (5.38).

Investigating the perturbation in the e_1 channel, $p_1(x_1, x_2, t) = \Gamma - \hat{\Gamma}$, the relation

$$\frac{||\Gamma - \hat{\Gamma}||_2}{||\dot{x} - \dot{\hat{x}}||_2} \leq ||\Gamma||_{Lip} \tag{5.39}$$

can be written directly from the definition of Lipschitz continuity in (5.28). Using Lemma 5.6.3 and rearranging, $p_1(x_1, x_2, t)$ is upper bounded by

$$||p_1(x_1, x_2, t)||_2 \leq \gamma ||\dot{x} - \dot{\hat{x}}||_2. \tag{5.40}$$

That is, the disturbance in the e_1 channel is bounded by the desired Lipschitz bound on the DNN times the error in the time derivative of the DNN input vector. Since it is assumed that $\dot{x}$ and its approximation $\dot{\hat{x}}$ are both bounded, the quantity $||\dot{x} - \dot{\hat{x}}||_2$ is also bounded.

Further, since $|p_1 x_1, x_2, t)|_i \leq ||p_1(x_1, x_2, t)||_\infty \leq ||p_1(x_1, x_2, t)||_2$ for $i = 1, 2, ..., m$; the perturbation on each component of the error system in (5.38) is bounded by

$$\begin{aligned} |p_1(x_1, x_2, t)| &\leq \gamma ||\dot{x} - \dot{\hat{x}}||_2 \\ |p_2(x_1, x_2, t)| &\leq D_y. \end{aligned} \tag{5.41}$$

Comparing the perturbation bounds in (5.41) to the form in (5.6), it can be seen that $\delta_1 = \gamma||\dot{x} - \dot{\hat{x}}||_2$ and $\delta_2 = 0$. Assuming the gains k_1, k_2 are selected according to the stability conditions in Theorem 5.6.1 (*i.e.*, [109], Algorithm 1) and P, Q_R satisfy (5.18), the time derivative of the Lyapunov candidate $V(e) = \zeta^T P \zeta$ along the trajectories of the error system in (5.38) can be written as

$$\dot{V} \leq -\frac{1}{|e_1|^{1/2}} \left(\zeta^T Q_R \zeta - \begin{bmatrix} p_1(x_1, x_2, t) & 0 \end{bmatrix} P \zeta \right). \tag{5.42}$$

Substituting the perturbation inequality from (5.41) into (5.42) and simplifying the quadratic term gives

$$\dot{V} \leq -\frac{1}{|e_1|^{1/2}} (\lambda_{\min}(Q_R)||\zeta||_2^2 - \eta\gamma||\dot{x} - \dot{\hat{x}}||_2||\zeta||_2) \tag{5.43}$$

which, using the fact $|e_1|^{1/2} \leq ||\zeta||_2$, can be simplified to

$$\dot{V} \leq -\lambda_{\min}(Q_R)||\zeta||_2 + \eta\gamma||\dot{x} - \dot{\hat{x}}||_2. \tag{5.44}$$

The Lyapunov function $V(e) = \zeta^T P \zeta$ is upper bounded by $V \leq \lambda_{\max}(P)||\zeta||_2^2$, which can be rearranged to $\frac{V^{1/2}}{\lambda_{\max}^{1/2}(P)} \leq ||\zeta||_2$. Substituting this relation and $||\zeta||_2 = \kappa||\zeta||_2 + (1-\kappa)||\zeta||_2$ for a constant $\kappa \in (0, 1)$ into (5.44) gives the simplified inequality

$$\dot{V} \leq -\frac{\kappa\lambda_{\min}(Q_R)}{\lambda_{\max}^{1/2}(P)} V^{1/2} \tag{5.45}$$

in the region where $||\zeta||_2 \geq \mu$, with

$$\mu = \frac{\eta\gamma||\dot{x} - \dot{\hat{x}}||_2}{(1-\kappa)\lambda_{\min}(Q_R)}. \tag{5.46}$$

Thus, the error trajectories of each component of (5.38) enter the set $\Omega_\mu = \{e \in \mathbb{R}^2 \mid V(e) \leq \lambda_{\max}(P)\mu^2\}$ at some finite time $t' > 0$ and remain in the set Ω_μ $\forall t > t'$ [109]. This convergence time t' can be estimated by Bihari's inequality as

$$t' = \frac{2\lambda_{\max}^{1/2}(P)}{\kappa\lambda_{\min}(Q_R)} \left[V^{1/2}(e(t=0)) - \lambda_{\max}^{1/2}(P)\mu\right]. \tag{5.47}$$

Finally, using the bounds on the Lyapunov function $\lambda_{\min}(P)||\zeta||_2^2 \leq V \leq \lambda_{\max}(P)\mu^2$, the upper bound of the trajectories of $||\zeta||_2 \leq \bar{e}$ are found to be

$$\bar{e} = \sqrt{\frac{\lambda_{\max}(P)}{\lambda_{\min}(P)}} \frac{\eta\gamma||\dot{x} - \dot{\hat{x}}||_2}{(1-\kappa)\lambda_{\min}(Q_R)}, \tag{5.48}$$

where P and Q are defined from the ALE in (5.18), and $\eta = \sqrt{1 + p_{12}^2}$, where p_{12} is the $(1,2)^{\text{th}}$ component of P [109]. Thus, in the presence of noise in $\dot{x}$ via measurements or numerical differentiation, training the DNN with spectral normalization can give the control designer quantifiable upper bounds on trajectory convergence for the proposed online last-layer update rules in (5.25), (5.26), and (5.27). For further details on the error bounds and time convergence of the STA algorithm in this condition, the reader is referred to [109], Theorem 3.

□

5.7 Simulation Examples

Case I: Consider the Van der Pol system in (5.1), with the training (or "nominal") system value $\epsilon = 1$, assuming the dynamics $\dot{\theta} = z$ are known. The DNN in (5.2) with 3 layers of 8, 16, 8 neurons, respectively, is trained with ReLU activation functions and spectral normalization via Algorithm 1, with $\gamma = 32$. The training dataset of features $\mathcal{X} = \{z, \theta\}$ and label dataset of $\mathcal{Y} = \{\dot{z}\}$ are generated by the nominal system for 30 seconds at time intervals of 0.01 seconds. The DNN is trained with the Adam optimizer for 20,000 epochs with a batch size of 8 [121]. The DNN is then implemented online onto the real system of $\epsilon = 1.5$, where the time derivative of the DNN input vector $\dot{x} = [\hat{y}, z]^T$ is used. Note that this input vector uses the DNN prediction $\hat{y}$, which approximates the learning target $\dot{z}$. The update rules (5.10), (5.11), and (5.12) are used, with $k_1 = 50$, $k_2 = 1$.

Case II.a: The Van der Pol system in (5.1) is again considered, with the training (or "nominal") system value $\epsilon = 1$, assuming the dynamics of θ are known but noisy. The DNN in (5.2) with 4 layers of 8, 16, 8 neurons, respectively, is trained with ReLU activation functions and spectral normalization via Algorithm 1, with $\gamma = 1$. The training dataset of features $\mathcal{X} = \{z, \theta\}$ and label dataset of $\mathcal{Y} = \{\dot{z}\}$ is generated by the nominal system for 30 seconds at time intervals of 0.01 seconds. The DNN is trained with the Adam optimizer for 20,000 epochs with a batch size of 8. The DNN is then implemented online onto the real system of $\epsilon = 1.5$, where the time derivative of the DNN input vector

$\dot{\hat{x}} = [\hat{y}, z + 10\sin(20\pi t)]^T$ is used. The update rules (5.25), (5.26), and (5.27) are used, with $k_1 = 50$, $k_2 = 1$.

Case II.b: Case II.b is the same as Case II.a above, but the DNN is not trained with spectral normalization. The DNN is implemented online to predict the real system of $\epsilon = 1.5$, where update rules (5.25), (5.26), and (5.27) are used, with $k_1 = 50$, $k_2 = 1$.

The DNN outputs over time for each case, compared to the real system, are shown in Figure 5.5. To compare the behavior of each case, the prediction error e_1 for each case is shown over time in Figure 5.6. To compare the error analysis in Theorem 5.6.4, $||\Gamma - \hat{\Gamma}||_2$ for Case II.a and Case II.b are plotted over time, along with the defined error bound in (5.35), in Figure 5.7.

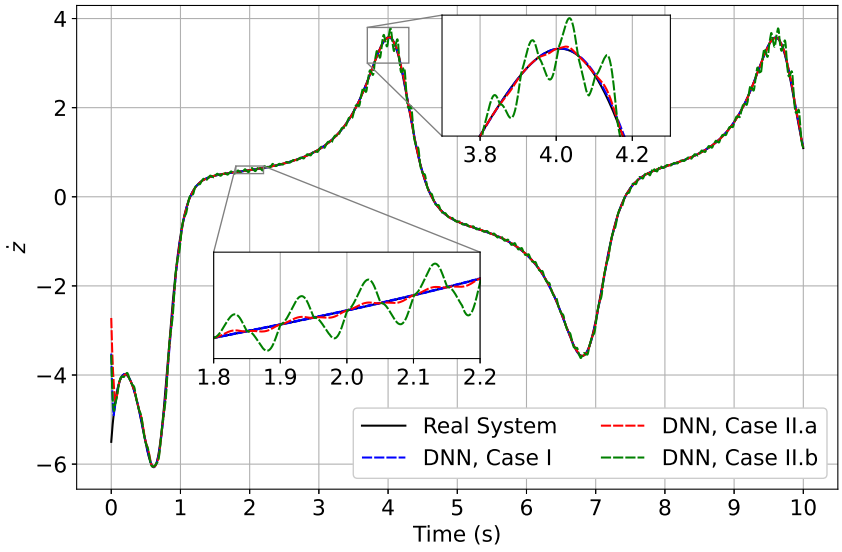

Figure 5.5: Online-adapted DNN predictions on the real system ($\epsilon = 1.5$) for each case.

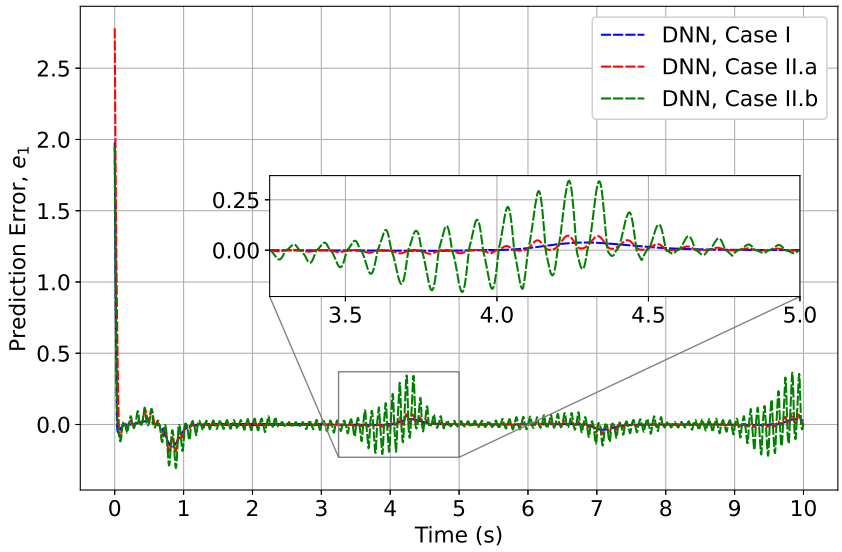

Figure 5.6: DNN prediction error on the real system ($\epsilon = 1.5$) for each case.

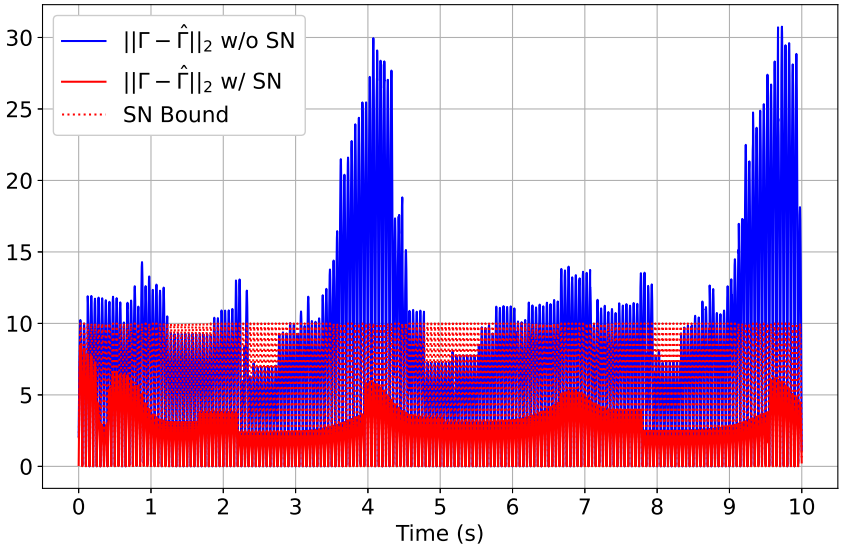

Figure 5.7: Perturbation analysis on the real system ($\epsilon = 1.5$) for the DNN trained with SN (Case II.a) and the DNN trained without SN (Case II.b). The red dotted line represents the upper bound from (5.34) at each timestep.

In Figure 5.5, each DNN successfully converges near the real system using the proposed online update rules, after being trained on a small nominal system dataset. In Figure 5.6, the perturbation in the e_1 channel from the disturbance in $\Gamma - \hat{\Gamma}$ now causes the e_1 trajectories in Case II.a and II.b to be ultimately bounded. However, when the DNN is trained with spectral normalization, the controlled Lipschitz constant upper bounds the disturbance $\Gamma - \hat{\Gamma}$, causing the upper bound of the e_1 trajectory to be smaller than the DNN trained without spectral normalization in Case II.b. In Figure 5.7, the upper bound on $\Gamma - \hat{\Gamma}$ from (5.34) is plotted over time, along with the $\Gamma - \hat{\Gamma}$ disturbance from both Case II.a (w/ SN) and Case II.b (w/o SN). It can be seen from this figure that the DNN trained without spectral normalization causes a larger disturbance in the e_1 channel, where the DNN trained with spectral normalization obeys the proven disturbance bound in (5.34).

5.8 Example: Sim2real Model Reference Adaptive Control

As mentioned above, the sim2real problem of control policies is a notable application of the online DNN learning methodology developed above. To show a possible application example, this section describes a model reference adaptive control (MRAC) problem of a robot arm. Consider the robot arm simulated in Chapter 2, where it is assumed that a nominal dynamic model of the robot is known. The goal of this section is to use a DNN to learn the model discrepancies between the nominal model and the real system, where the control policy will be deployed.

The dynamics of the nominal system are assumed to be of the form

$$F_{nom}(y, \dot{y}, \ddot{y}) = \tau - \hat{F}(y, \dot{y}, \ddot{y}), \tag{5.49}$$

where $F_{nom}(y, \dot{y}, \ddot{y}) = M_{nom}\ddot{y} + f_{nom}$ is the nominal part of the dynamics adequately explained by the known dynamic model, $\hat{F}(y, \dot{y}, \ddot{y}) = \hat{M}\ddot{y} + \hat{f}$ is the unmodeled part of the dynamics to be learned (due to parameter uncertainty in this example), and τ is the control input. Note that M and $f = C(q, \dot{q})\dot{q} + G(q)$ are from the reference robot model described in Appendix A.1.

The nominal dataset $\{x, \tau, F_{nom}\}$ is constructed via simulation of the known robot model performing a continuous slew maneuver for 10 seconds, at a frequency of 100 Hz. The state vector x given as input to the DNN is $x = \{q, \dot{q}, \ddot{q}\}$, where $q \in \mathbb{R}^3$ is the vector of robot joint angles. The control law used during dataset generation is $\tau = \tau_{nom} - Ks$, where $\tau_{nom} = M_{nom}\dot{s}_r + f_{nom}$ and $K = \text{diag}([2, 0.9, 0.5])$ is a design gain matrix. The filtered control error s is calculated as $s = \dot{e} - \lambda e$, with $\lambda = \text{diag}([4, 4, 4])$, and $\dot{s}_r = \dot{q} - s$ denoting a reference velocity error. A 3-layer spectrally-normalized DNN with 8, 16, and 8 neurons per layer is trained via the Adam optimizer on the feature dataset $\mathcal{X} = \{x\}$ for 5000 epochs with a desired SN constant of 1.2. The learning target is to minimize the mean squared error between $\hat{F}$, now parameterized by a DNN, and $\tau - F_{nom}$. The parameters assumed for the nominal robot model are given in Table 5.1.

To simulate implementing this controller in a sim2real scenario, the physical parameters of the simulated "real" robot to be controlled are varied from those used during the dataset generation. The parameters used for the real robot

Table 5.1: Nominal robot parameters used during dataset generation.

Parameter	Value
R_0	0.06 m
l_1	0.3 m
l_2	0.3 m
m_0	4 kg
m_1	0.8 kg
m_2	0.85 kg

model are given in Table 5.2. Note that both sets of parameters are designed in this example to simulate a real robotic system, where an imperfect but reasonable dynamics model is acquired via measurement and system identification.

Table 5.2: Real robot parameters used during simulation.

Parameter	Value
R_0	0.065 m
l_1	0.296 m
l_2	0.31 m
m_0	4 kg
m_1	0.815 kg
m_2	0.8 kg

The control law on the real robot now becomes

$$\tau = \tau_{nom} - Ks + \tau_{dist}, \tag{5.50}$$

where τ_{dist} is the current DNN approximation of $\hat{F}$, the combined disturbance due to parameter uncertainty. This control law simply uses the DNN to learn and directly compensate for the effects of parameter uncertainty in the real robot, where the dynamics of the real system ideally become equivalent to the dynamics of the nominal system. The update rules of the last DNN layer given in (5.25), (5.26), and (5.27) are performed online at every timestep. A super-twisting differentiator [16] is used to approximate $\dddot{q}$, which is needed for constructing $\dot{x}$ for calculation of $\hat{\Gamma}$ in (5.25). As mentioned above, this causes the error trajectories e_1 to not converge to zero but to be ultimately bounded near zero (dependent on differentiation accuracy).

The comparison of the nominal robot controlled by $\tau = \tau_{nom} - Ks$ (labeled "Nominal System"), the real robot controlled via (5.50) without online learning (labeled "Without Online Update, Real System"), and the real robot controlled via (5.50) with the last-layer update laws given in (5.25), (5.26), and (5.27) (labeled "With Online Update, Real System") is shown below in Figure 5.8.

Figure 5.8 shows that the model reference controller on the real system without online updates has control error, since the real system has differing dynamics from the model used to build and train the controller (*i.e.*, domain shift). However, for the model reference controller on the real system with the developed online update laws, the prediction error e_1 converges to a small error ball, allowing the model reference controller to return the system to dynamics almost identical to the nominal system. The prediction error here is due to the numerical differentiation used in the update. It is important to note that this controller is able to

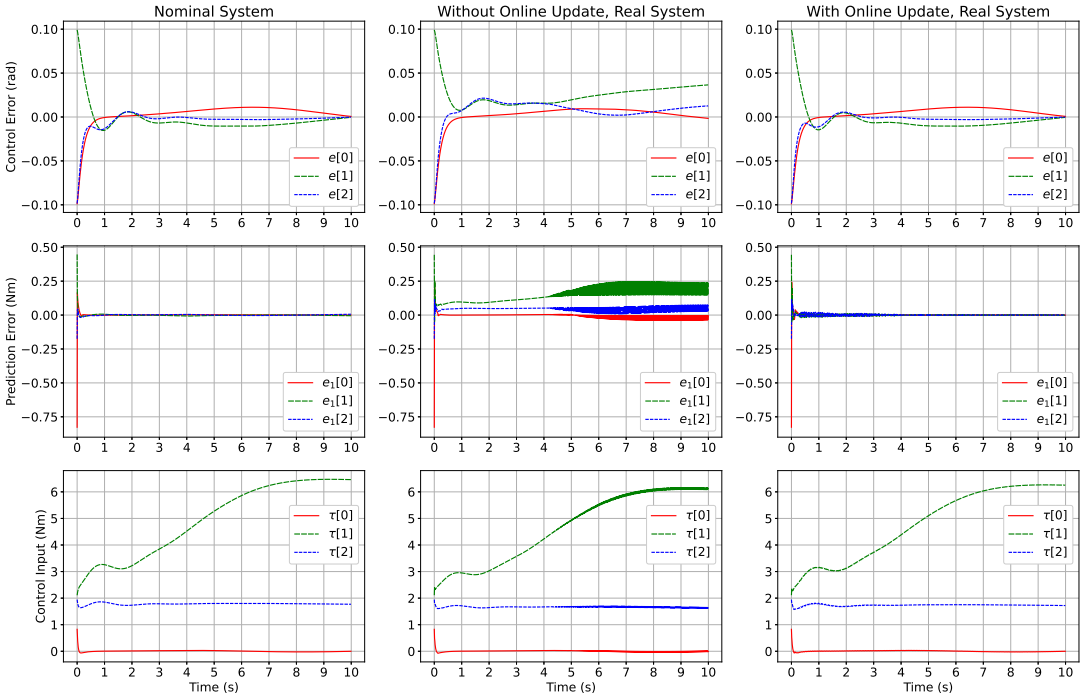

Figure 5.8: Comparison of the nominal system dynamics, the DNN-compensated dynamics without the online update rule, and the DNN-compensated dynamics with the developed online update rule.

learn to compensate the dynamics of the uncertain real-world system from only 10 seconds of simulation data due to the proposed online learning laws, essentially negating the sim2real problem. Further, this controller now has a quantification of what is unknown or unmodeled about the system, which could be beneficial information for operators or controllers that use uncertainty to adapt control gains or to drive safe exploration.

Chapter 6. Conclusions, Discussion, and Future Work

6.1 Summary and Conclusions

Part II describes the research progress made in using control theory itself to improve AI/ML performance online, when deployed. The central idea of Part II is that the mathematical rigor and goals of control theory are often aligned with many tasks in modern AI/ML, especially in aerospace and robotics; and these control-theoretic tactics can be used directly to improve AI/ML performance and stability. This is contrasted with Part I, which is primarily concerned with studying these AI/ML elements inside adaptive control laws to improve control performance, versatility, and robustness. Additionally, many tricks used to stabilize and improve training of AI/ML elements, such as normalization or regularization, likely have applications to controls and additional mathematical description via control theory.

Chapter 5 considers evolving deep neural networks (DNNs) online during inference, when the online distribution is shifted from the offline distribution used to train the DNN. This problem frequently arises when DNNs are used to approximate dynamical systems, which is of particular interest to transfer learning and reinforcement learning for forecasting, controls, and sim2real transfer. When using a DNN to approximate a dynamical system online, the DNN itself can be con-

sidered as a dynamical system to be controlled. The chapter describes using the super-twisting algorithm, a well-known sliding mode control algorithm, to evolve the last-layer parameters of the DNN in continuous time. Last-layer evolution has been shown in transfer learning to effectively and efficiently provide updates to DNN-based models from newly acquired data while limiting catastrophic forgetting, when retraining the entire model can be expensive and infeasible. Further, the developed online update laws in Chapter 5 render proven error trajectory convergence of the DNN outputs, which is desirable in many online DNN prediction scenarios, since conventional gradient-based backpropagation does not intrinsically provide any DNN performance guarantees or bounds. This can allow drop-in performance and stability guarantees on the DNN in controls scenarios, when some notion of input-output stability is required or necessary. Chapter 5 describes the application of the developed update laws in two possible cases: (1) when the time derivative of the DNN input vector is known and (2) when the time derivative of the DNN input vector is approximated or noisy. In the latter case, spectrally normalizing the DNN during training improves online DNN prediction performance, since the desired spectral normalization constant of the DNN upper-bounds the prediction error trajectory convergence. The methodology and relevant convergence proofs are validated for each described case via numerical simulation, with the DNN trained to approximate the dynamics of the Van der Pol oscillator online under domain shift. In each case, the developed last-layer update rules allow the DNN to perform effectively on the online domain-shifted system. Lastly, Chapter 5 gives an example of using the developed update laws

inside a model reference adaptive controller for a simulated sim2real robot control problem, where the controller effectively learns to compensate the dynamics of the uncertain real-world system from only 10 seconds of simulation data.

6.2 Discussion

Chapter 1 mentions that one of the primary theories of this dissertation is ***the data that a system encounters online, during operation, is the most important for learning.*** To this end, the developed online update laws of Chapter 5 effectively perform online learning on the contemporarily dominant AI/ML tool, the feedforward DNN. As described in Chapter 4, this notion of "learning" is likely foreign to a typical AI/ML engineer or data scientist, whom would opt for recording data over time and retraining the entire DNN at some prescribed interval during deployment. It is likely that retraining the entire DNN, when the online data distribution shift from the offline distribution is high or highly dynamic, would improve performance of last-layer online update laws similar to those described in Chapter 5. This is closely related to *concurrent learning* in adaptive control, which is involves recording data over time for updating parameters of an adaptive controller [122–124]. However, the developments in Chapter 5 aim to keep the application generalized to online learning in AI/ML instruments such as the deep neural network, although the results and theories of concurrent learning likely give similar performance for online learning and estimation of dynamical systems. The mentioned concurrent learning works specifically aim to match the neural network parameters to the "ideal" neural network parameters

to decrease prediction error. While the goal of decreasing neural network prediction error itself is aligned with the goal of Chapter 5, the concept of ideal neural network parameters is external to most modern data science, with models using as many parameters as necessary to increase model accuracy and decrease training time/compute required. The ideas described Chapter 5 can be further contrasted to similar works using full Lyapunov-based updates, such as [44, 66, 67], where stability in the neural network is extrinsically dependent on a control law itself (as in Chapters 2 and 3 of this dissertation).

The basic ideas of Part II came from deep learning research connecting backpropagation of errors, gradient descent on an objective function, and neuroevolution in the brain [125]. It is likely that our brains use much more efficient and effective neuroevolution methods than the calculus of gradient descent that currently drives deep learning data-based models. Thus, similar to the controllers in this dissertation driving errors to zero using a sliding manifold, this dissertation hypothesizes that some error representation or error-based update is more effective and efficient than first-order backpropagation with gradient descent, especially for the safety-critical applications of aerospace and robotics. Chapter 5 thus, in a general sense, presents the research progress made in using ideas from control theory to help accomplish the goal of going beyond backpropagation to achieve improved, more efficient neural network training.

6.3 Future Work

The most direct avenue of future work, expanding on the ideas presented in Part II of this dissertation, is the study of other control formulations to stably update neural network parameters online. In Chapter 5, the control-based online update laws are based on super-twisting control, which avoids the need for calculating the time derivative of the sliding variable. Other control methodologies, such as backstepping, could be of merit for study in the last-layer update scheme. These control methodologies may be used alongside other pertinent AI/ML techniques to derive improved performance, just as Chapter 5 shows spectrally normalizing the DNN layers controls the upper bound on prediction error in the developed super-twisting scheme. Additional future work includes transferring the ideas of Chapter 5 to other neural network architectures, such as recurrent neural networks or convolutional neural networks. It is likely that these architectures require different formulations on the dynamics and analysis to guarantee performance, which may require an alternative control formulation and derived update law (such as a generalized parameter tensor update law versus a parameter matrix update law).

The developments described in Chapter 5, as mentioned above, are inspired by the sim2real transfer of reinforcement learning policies. When sim2real control policies are used from pure reinforcement learning, such as in [74], there are no intrinsic stability guarantees. The stable DNN-based model reference adaptive control example in Chapter 5 gives promising results for sim2real transfer of DNNs

using online learning. Future work includes the study of stable sim2real transfer of full reinforcement learning based policies using the derived online update laws in Chapter 5 for the control of robotic arms, spacecraft, or aircraft. Not only would these stability guarantees allow drop-in use of online-adapted DNNs in controllers (such as the controller in the model reference adaptive control example of Chapter 5), reinforcement learning policies could adapt online to novel behavior not seen in simulation, which is currently a difficult and ongoing research problem. Online adaptation of learning-based policies is of paramount research importance for the intelligent systems of tomorrow that will work with and for humans in aerospace and robotics.

References

[1] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, Cambridge, MA, 2016.

[2] Alan Turing. Intelligent machinery (1948). *B. Jack Copeland*, pages 395–432, 2004.

[3] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, 5: 115–133, 1943.

[4] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386, 1958.

[5] Marvin Minsky and Seymour A. Papert. *Perceptrons*. MIT Press, Cambridge, MA, USA, 1969.

[6] John J Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 79(8):2554–2558, 1982.

[7] Paul J. Werbos. Beyond regression: New tools for prediction and analysis in the behavioral sciences. *PhD thesis, Committee on Applied Mathematics, Harvard University, Cambridge, MA*, 1974.

[8] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[9] Jean-Jacques Slotine and Weiping Li. *Applied Nonlinear Control*. Prentice-Hall, New Jersey, USA, 1991.

[10] Joseph E. Gaudio, Travis E. Gibson, Anuradha M. Annaswamy, Michael A. Bolender, and Eugene Lavretsky. Connections between adaptive control and optimization in machine learning. In *IEEE 58th Conference on Decision and Control (CDC)*, pages 4563–4568, Nice, France, 2019.

[11] Jacob G. Elkins and Farbod Fahimi. Online neural sliding mode control with guaranteed stability. *International Journal of Control*, pages 1–11, 2024. doi: 10.1080/00207179.2024.2332521.

[12] Jacob G. Elkins, Avimanyu Sahoo, and Farbod Fahimi. Control-theoretic techniques for online adaptation of deep neural networks in dynamical systems. *IEEE Transactions on Neural Networks and Learning Systems*, 2024. Under review.

[13] Sebastian Höfer, Kostas Bekris, Ankur Handa, Juan Camilo Gamboa, Melissa Mozifian, Florian Golemo, Chris Atkeson, Dieter Fox, Ken Goldberg, John Leonard, et al. Sim2real in robotics and automation: Applications and challenges. *IEEE Transactions on Automation Science and Engineering*, 18(2):398–400, 2021.

[14] Jean-Jacques Slotine and J. A. Coetsee. Adaptive sliding controller synthesis for non-linear systems. *International Journal of Control*, 43(6):1631–1651, 1986.

[15] Shankar Sastry and Marc Bodson. *Adaptive Control: Stability, Convergence, and Robustness*. Prentice-Hall, New Jersey, USA, 1994.

[16] Yuri Shtessel, Christopher Edwards, Leonid Fridman, and Arie Levant. *Sliding Mode Control and Observation*. Springer, New York, NY, 2014.

[17] Christopher Edwards and Sarah K. Spurgeon. *Sliding Mode Control: Theory and Applications*. CRC Press, London, 1998.

[18] Vadim I. Utkin. *Sliding Modes in Control and Optimization*. Springer, Berlin, Germany, 2013.

[19] Farbod Fahimi. *Autonomous Robots: Modeling, Path Planning, and Control*. Springer, New York, 2008.

[20] Nikolas Sacchi, Gian Paolo Incremona, and Antonella Ferrara. Neural network-based practical/ideal integral sliding mode control. *IEEE Control Systems Letters*, 6:3140–3145, 2022. doi: 10.1109/LCSYS.2022.3182814.

[21] Karel J. Keesman. *System Identification: An Introduction*. Springer, London, 2011.

[22] Pierangela Morga, Mauro Mancini, and Elisa Capello. Flexible spacecraft model and robust control techniques for attitude maneuvers. In *2022 American Control Conference (ACC)*, pages 1120–1126, Atlanta, GA, USA, 2022. doi: 10.23919/ACC53348.2022.9867280.

[23] Farbod Fahimi and Chris Van Kleeck. Alternative trajectory-tracking control approach for marine surface vessels with experimental verification. *Robotica*, 31(1):25–33, 2013. doi: 10.1017/S0263574712000070.

[24] Farbod Fahimi. Towards full formation control of an autonomous helicopters group. In *2007 IEEE Aerospace Conference*, Big Sky, MT, USA, 2007. doi: 10.1109/AERO.2007.352851.

[25] Guanya Shi, Xichen Shi, Michael O'Connell, Rose Yu, Kamyar Azizzadenesheli, Animashree Anandkumar, Yisong Yue, and Soon-Jo Chung. Neural lander: Stable drone landing control using learned dynamics. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 9784–9790, Montreal, QC, Canada, 2019. doi: 10.1109/ICRA.2019.8794351.

[26] Michael O'Connell, Guanya Shi, Xichen Shi, Kamyar Azizzadenesheli, Anima Anandkumar, Yisong Yue, and Soon-Jo Chung. Neural-fly enables rapid learning for agile flight in strong winds. *Science Robotics*, 7(66), 2022. doi: 10.1126/scirobotics.abm6597.

[27] Guanya Shi, Wolfgang Hönig, Xichen Shi, Yisong Yue, and Soon-Jo Chung. Neural-swarm2: Planning and control of heterogeneous multirotor swarms using learned interactions. *IEEE Transactions on Robotics*, 38(2):1063–1079, 2021.

[28] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873):583–589, 2021.

[29] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents. *arXiv preprint arXiv:2204.06125*, 2022.

[30] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989. ISSN 0893-6080. doi: 10.1016/0893-6080(89)90020-8.

[31] Andrew G. Barto, Richard S. Sutton, and Charles W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13(5):834–846, 1983. doi: 10.1109/TSMC.1983.6313077.

[32] Kumpati S. Narendra and Kannan Parthasarathy. Identification and control of dynamical systems using neural networks. *IEEE Transactions on Neural Networks*, 1(1):4–27, 1990. doi: 10.1109/72.80202.

[33] Robert M. Sanner and Jean-Jacques Slotine. Gaussian networks for direct adaptive control. *IEEE Transactions on Neural Networks*, 3(6):837–863, 1992. doi: 10.1109/72.165588.

[34] Fu-Chuang Chen and Hassan K. Khalil. Adaptive control of a class of nonlinear discrete-time systems using neural networks. *IEEE Transactions on Automatic Control*, 40(5):791–801, 1995. doi: 10.1109/9.384214.

[35] Frank L. Lewis, Aydin Yesildirek, and Kai Liu. Multilayer neural-net robot controller with guaranteed tracking performance. *IEEE Transactions on Neural Networks*, 7(2):388–399, 1996. doi: 10.1109/72.485674.

[36] Sarangapani Jagannathan and Frank L. Lewis. Discrete-time neural net controller for a class of nonlinear dynamical systems. *IEEE Transactions on Automatic Control*, 41(11):1693–1699, 1996. doi: 10.1109/9.544013.

[37] Indrani Kar and Laxmidhar Behera. Direct adaptive neural control scheme for discrete time affine nonlinear systems. In *2008 IEEE International Symposium on Intelligent Control*, pages 1097–1102, 2008. doi: 10.1109/ISIC.2008.4635966.

[38] Avimanyu Sahoo, Hao Xu, and Sarangapani Jagannathan. Neural network-based adaptive event-triggered control of affine nonlinear discrete time systems with unknown internal dynamics. In *2013 American Control Conference (ACC)*, pages 6418–6423, Washington, DC, USA, 2013. doi: 10.1109/ACC.2013.6580845.

[39] Avimanyu Sahoo, Hao Xu, and Sarangapani Jagannathan. Adaptive neural network-based event-triggered control of single-input single-output nonlinear discrete-time systems. *IEEE Transactions on Neural Networks and Learning Systems*, 27(1):151–164, 2016. doi: 10.1109/TNNLS.2015.2472290.

[40] Qudrat Khan Safeer Ullah and Adeel Mehmood. Neuro-adaptive fixed-time non-singular fast terminal sliding mode control design for a class of under-actuated nonlinear systems. *International Journal of Control*, 96(6): 1529–1542, 2023. doi: 10.1080/00207179.2022.2056514.

[41] Safeer Ullah, Qudrat Khan, Adeel Mehmood, Syed Abdul Mannan Kirmani, and Omar Mechali. Neuro-adaptive fast integral terminal sliding mode control design with variable gain robust exact differentiator for under-actuated quadcopter uav. *ISA Transactions*, 120:293–304, 2022.

[42] Hiroyasu Tsukamoto, Soon-Jo Chung, and Jean-Jacques Slotine. Learning-based adaptive control using contraction theory. In *2021 60th IEEE Conference on Decision and Control (CDC)*, pages 2533–2538, Austin, TX, USA, 2021. doi: 10.1109/CDC45484.2021.9683435.

[43] Spencer M Richards, Felix Berkenkamp, and Andreas Krause. The lyapunov neural network: Adaptive stability certification for safe learning of dynamical systems. In *Conference on Robot Learning (CoRL)*, pages 466–476, Zürich, Switzerland, 2018.

[44] Runhan Sun, Max L. Greene, Duc M. Le, Zachary I. Bell, Girish Chowdhary, and Warren E. Dixon. Lyapunov-based real-time and iterative adjustment of deep neural networks. *IEEE Control Systems Letters*, 6:193–198, 2022. doi: 10.1109/LCSYS.2021.3055454.

[45] Alejandro Guarneros-Sandoval, Mariana Ballesteros, Ivan Salgado, Julia Rodríguez-Santillán, and Isaac Chairez. Lyapunov stable learning laws for multilayer recurrent neural networks. *Neurocomputing*, 491:644–657, 2022. ISSN 0925-2312. doi: 10.1016/j.neucom.2021.12.041.

[46] Omkar Sudhir Patil, Duc M. Le, Max L. Greene, and Warren E. Dixon. Lyapunov-derived control and adaptive update laws for inner and outer

layer weights of a deep neural network. *IEEE Control Systems Letters*, 6: 1855–1860, 2022. doi: 10.1109/LCSYS.2021.3134914.

[47] Mou Chen, Peng Shi, and Cheng-Chew Lim. Adaptive neural fault-tolerant control of a 3-dof model helicopter system. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 46(2):260–270, 2016. doi: 10.1109/TSMC.2015.2426140.

[48] Juntao Fei and Hongfei Ding. Adaptive sliding mode control of dynamic system using rbf neural network. *Nonlinear Dynamics*, 70:1563–1573, 2012.

[49] Bing Xiao, Qinglei Hu, and Youmin Zhang. Adaptive sliding mode fault tolerant attitude tracking control for flexible spacecraft under actuator saturation. *IEEE Transactions on Control Systems Technology*, 20(6):1605–1612, 2012. doi: 10.1109/TCST.2011.2169796.

[50] Tong Yang, Ning Sun, and Yongchun Fang. Neuroadaptive control for complicated underactuated systems with simultaneous output and velocity constraints exerted on both actuated and unactuated states. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–11, 2021. doi: 10.1109/TNNLS.2021.3115960.

[51] Van Trong Dang, Dinh Bao Hung Nguyen, Thi Dieu Trinh Tran, Duc Thinh Le, and Tung Lam Nguyen. Model-free hierarchical control with fractional-order sliding surface for multisection web machines. *International Journal of Adaptive Control and Signal Processing*, 37(2):497–518, 2023. doi: 10.1002/acs.3534.

[52] Liangyong Wang, Tianyou Chai, and Lianfei Zhai. Neural-network-based terminal sliding-mode control of robotic manipulators including actuator dynamics. *IEEE Transactions on Industrial Electronics*, 56(9):3296–3304, 2009. doi: 10.1109/TIE.2008.2011350.

[53] J. Park and I. W. Sandberg. Universal approximation using radial-basis-function networks. *Neural Computation*, 3(2):246–257, 1991. doi: 10.1162/neco.1991.3.2.246.

[54] Juntao Fei and Cheng Lu. Adaptive sliding mode control of dynamic systems using double loop recurrent neural network structure. *IEEE Trans-*

actions on Neural Networks and Learning Systems, 29(4):1275–1286, 2018. doi: 10.1109/TNNLS.2017.2672998.

[55] Joseph P. LaSalle. Some extensions of liapunov's second method. *IRE Transactions on Circuit Theory*, 7(4):520–527, 1960. doi: 10.1109/TCT.1960.1086720.

[56] Frank L. Lewis, Chaouki T. Abdallah, and Darren W. Dawson. *Robot Manipulator Control: Theory and Practice*. Macmillan, New York, 2014.

[57] John J. Craig. *Introduction to Robotics: Mechanics and Control*. Pearson, New York, NY, USA, 2018.

[58] Rahul Moghe. *Adaptive Algorithms for Identification of Symmetric and Positive Definite Matrices*. Phd dissertation, The University of Texas at Austin, 2021.

[59] Rahul Moghe and Maruthi R. Akella. Projection scheme and adaptive control for symmetric matrices with eigenvalue bounds. *IEEE Transactions on Automatic Control*, 68(3):1738–1745, 2023. doi: 10.1109/TAC.2022.3153458.

[60] Jacob G. Elkins, Farbod Fahimi, and Rohan Sood. Stable online learning-based adaptive control of spacecraft and quadcopters. In *2024 IEEE Aerospace Conference*, Big Sky, MT, USA, 2024. doi: 10.1109/AERO58975.2024.10521254.

[61] F. Landis Markley and John L. Crassidis. *Fundamentals of Spacecraft Attitude Determination and Control*. Springer, New York, NY, 2014.

[62] Hazim Shakhatreh, Ahmad H. Sawalmeh, Ala Al-Fuqaha, Zuochao Dou, Eyad Almaita, Issa Khalil, Noor Shamsiah Othman, Abdallah Khreishah, and Mohsen Guizani. Unmanned aerial vehicles (uavs): A survey on civil applications and key research challenges. *IEEE Access*, 7:48572–48634, 2019. doi: 10.1109/ACCESS.2019.2909530.

[63] Petros A. Ioannou and Jing Sun. *Robust Adaptive Control*. Dover Publications, Inc., Mineola, NY, USA, 2012.

[64] Karol Kurach, Mario Lučić, Xiaohua Zhai, Marcin Michalski, and Sylvain Gelly. A large-scale study on regularization and normalization in GANs. In *36th International Conference on Machine Learning*, pages 3581–3590, Long Beach, CA, USA, 2019.

[65] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. *ArXiv*, abs/1802.05957, 2018.

[66] Emily J. Griffis, Omkar Sudhir Patil, Zachary I. Bell, and Warren E. Dixon. Lyapunov-based long short-term memory (lb-lstm) neural network-based control. *IEEE Control Systems Letters*, 7:2976–2981, 2023. doi: 10.1109/LCSYS.2023.3291328.

[67] Wanjiku A. Makumi, Zachary I. Bell, and Warren E. Dixon. Approximate optimal indirect regulation of an unknown agent with a lyapunov-based deep neural network. *IEEE Control Systems Letters*, 7:2773–2778, 2023. doi: 10.1109/LCSYS.2023.3289474.

[68] Hiroyasu Tsukamoto, Soon-Jo Chung, and Jean-Jacques Slotine. Learning-based adaptive control using contraction theory. In *2021 60th IEEE Conference on Decision and Control (CDC)*, pages 2533–2538, Austin, TX, USA, 2021. doi: 10.1109/CDC45484.2021.9683435.

[69] Aamer I. Bhatti, Sarah K. Spurgeon, and Xiao-Yun Lu. A nonlinear sliding mode control design approach based on neural network modelling. *International Journal of Robust and Nonlinear Control*, 9(7):397–423, 1999. doi: 10.1002/(SICI)1099-1239(199906)9:7⟨397::AID-RNC412⟩3.0.CO;2-0.

[70] Edoardo Vacchini, Nikolas Sacchi, Gian Paolo Incremona, and Antonella Ferrara. Design of a deep neural network-based integral sliding mode control for nonlinear systems under fully unknown dynamics. *IEEE Control Systems Letters*, 7:1789–1794, 2023. doi: 10.1109/LCSYS.2023.3281288.

[71] Qinglei Hu. Robust adaptive sliding mode attitude maneuvering and vibration damping of three-axis-stabilized flexible spacecraft with actuator saturation limits. *Nonlinear Dynamics*, 55:301–321, 2009. doi: 10.1007/s11071-008-9363-1.

[72] Shweta Gupte, Paul Infant Teenu Mohandas, and James M. Conrad. A survey of quadrotor unmanned aerial vehicles. In *2012 IEEE Southeastcon*, pages 1–6, Orlando, FL, USA, 2012. doi: 10.1109/SECon.2012.6196930.

[73] Benke Gao, Yan-Jun Liu, and Lei Liu. Adaptive neural fault-tolerant control of a quadrotor uav via fast terminal sliding mode. *Aerospace Science and Technology*, 129:107818, 2022. doi: 10.1016/j.ast.2022.107818.

[74] Jacob G. Elkins, Rohan Sood, and Clemens Rumpf. Bridging reinforcement learning and online learning for spacecraft attitude control. *Journal of Aerospace Information Systems*, 19(1):62–69, 2022. doi: 10.2514/1.I010958.

[75] Ryan Mathewson and Farbod Fahimi. Nonlinear adaptive sliding mode control with application to quadcopters. *Nonlinear Engineering*, 12(1), 2023. doi: doi:10.1515/nleng-2022-0268.

[76] John L. Crassidis, Srinivas R. Vadali, and F. Landis Markley. Optimal variable-structure control tracking of spacecraft maneuvers. *Journal of Guidance, Control, and Dynamics*, 23(3):564–566, 2000. doi: 10.2514/2.4568.

[77] J. A. Lenda. Manned maneuvering unit: User's guide. Technical report, Martin Marietta Corp., Denver, CO, 1978.

[78] Alexander Poultney, Christopher Kennedy, Garrett Clayton, and Hashem Ashrafiuon. Robust tracking control of quadrotors based on differential flatness: Simulations and experiments. *IEEE/ASME Transactions on Mechatronics*, 23(3):1126–1137, 2018. doi: 10.1109/TMECH.2018.2820426.

[79] Guanya Shi. *Reliable Learning and Control in Dynamic Environments: Towards Unified Theory and Learned Robotic Agility*. Phd thesis, California Institute of Technology, 2023.

[80] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[81] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents. *arXiv preprint arXiv:2204.06125*, 1(2):3, 2022.

[82] Geoffrey E. Hinton David E. Rumelhart and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986. doi: 10.1038/323533a0.

[83] Paul J. Werbos. Backpropagation: past and future. In *IEEE 1988 International Conference on Neural Networks*, pages 343–353 vol.1, 1988. doi: 10.1109/ICNN.1988.23866.

[84] Antonio Torralba and Alexei A. Efros. Unbiased look at dataset bias. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1521–1528, 2011. doi: 10.1109/CVPR.2011.5995347.

[85] Joaquin Quiñonero-Candela, Masashi Sugiyama, Anton Schwaighofer, and Neil D. Lawrence. *Dataset shift in machine learning*. MIT Press, Cambridge, MA, 2009.

[86] Steven C.H. Hoi, Doyen Sahoo, Jing Lu, and Peilin Zhao. Online learning: A comprehensive survey. *Neurocomputing*, 459:249–289, 2021. ISSN 0925-2312. doi: 10.1016/j.neucom.2021.04.112.

[87] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. A comprehensive survey on transfer learning. *Proceedings of the IEEE*, 109(1):43–76, 2021. doi: 10.1109/JPROC.2020.3004555.

[88] Yishay Mansour, Mehryar Mohri, and Afshin Rostamizadeh. Domain adaptation: Learning bounds and algorithms. In *Conference on Learning Theory (COLT)*, Montreal, QC, Canada, 2009.

[89] Donghyun Kim, Kaihong Wang, Stan Sclaroff, and Kate Saenko. Domain adaptation: Learning bounds and algorithms. In *European Conference on Computer Vision*, pages 621–638, Tel Aviv, Israel, 2022.

[90] Ishaan Gulrajani and David Lopez-Paz. In search of lost domain generalization. *arXiv preprint arXiv:2007.01434*, 2020.

[91] Bryan Lim and Stefan Zohren. Time-series forecasting with deep learning: a survey. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 379(2194), 2021. doi: 10.1098/rsta.2020.0209. URL `https://royalsocietypublishing.org/doi/abs/10.1098/rsta.2020.0209`.

[92] Erica Salvato, Gianfranco Fenu, Eric Medvet, and Felice Andrea Pellegrino. Crossing the reality gap: A survey on sim-to-real transferability of robot controllers in reinforcement learning. *IEEE Access*, 9:153171–153187, 2021. doi: 10.1109/ACCESS.2021.3126658.

[93] Wenshuai Zhao, Jorge Peña Queralta, and Tomi Westerlund. Sim-to-real transfer in deep reinforcement learning for robotics: a survey. In *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 737–744, 2020. doi: 10.1109/SSCI47803.2020.9308468.

[94] Michel Breyer, Fadri Furrer, Tonci Novkovic, Roland Siegwart, and Juan Nieto. Flexible robotic grasping with sim-to-real transfer based reinforcement learning. *arXiv preprint arXiv:1803.04996*, 2018.

[95] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 23–30, Vancouver, BC, Canada, 2017. doi: 10.1109/IROS.2017.8202133.

[96] Stephen James, Andrew J. Davison, and Edward Johns. Transferring end-to-end visuomotor control from simulation to real world for a multi-stage task. In *2017 Conference on Robot Learning (CoRL)*, pages 334–343, Mountain View, CA, USA, 2017.

[97] Jie Tan, Tingnan Zhang, Erwin Coumans, Atil Iscen, Yunfei Bai, Danijar Hafner, Steven Bohez, and Vincent Vanhoucke. Sim-to-real: Learning agile locomotion for quadruped robots. In *Proceedings of Robotics: Science and Systems*, pages 10–20, Pittsburgh, PA, USA, 2018.

[98] Lerrel Pinto, James Davidson, Rahul Sukthankar, and Abhinav Gupta. Robust adversarial reinforcement learning. In *International Conference on Machine Learning*, pages 2817–2826, Sydney, Australia, 2017. PMLR.

[99] Xinlei Pan, Daniel Seita, Yang Gao, and John Canny. Risk averse robust adversarial reinforcement learning. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8522–8528, Montreal, QC, Canada, 2019.

[100] Jane X. Wang, Zeb Kurth-Nelson, Dhruva Tirumala, Hubert Soyer, Joel Z. Leibo, Remi Munos, Charles Blundell, Dharshan Kumaran, and Matt Botvinick. Learning to reinforcement learn. *arXiv preprint arXiv:1611.05763*, 2016.

[101] Karol Arndt, Murtaza Hazara, Ali Ghadirzadeh, and Ville Kyrki. Meta reinforcement learning for sim-to-real domain adaptation. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2725–2731, 2020. doi: 10.1109/ICRA40945.2020.9196540.

[102] Paul J. Werbos. Neural networks for control and system identification. In *28th IEEE Conference on Decision and Control*, volume 1, pages 260–265, Tampa, FL, USA, 1989. doi: 10.1109/CDC.1989.70114.

[103] Tyler LaBonte, Vidya Muthukumar, and Abhishek Kumar. Towards last-layer retraining for group robustness with fewer annotations. In *Advances in Neural Information Processing Systems (NeurIPS)*, New Orleans, LA, USA, 2023.

[104] Yunhui Guo, Honghui Shi, Abhishek Kumar, Kristen Grauman, Tajana Rosing, and Rogerio Feris. Spottune: Transfer learning through adaptive fine-tuning. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4800–4809, Long Beach, CA, USA, 2019. doi: 10.1109/CVPR.2019.00494.

[105] Nima Tajbakhsh, Jae Y. Shin, Suryakanth R. Gurudu, R. Todd Hurst, Christopher B. Kendall, Michael B. Gotway, and Jianming Liang. Convolutional neural networks for medical image analysis: Full training or fine tuning? *IEEE Transactions on Medical Imaging*, 35(5):1299–1312, 2016. doi: 10.1109/TMI.2016.2535302.

[106] Hossein Azizpour, Ali Sharif Razavian, Josephine Sullivan, Atsuto Maki, and Stefan Carlsson. Factors of transferability for a generic convnet representation. *IEEE Transactions on Pattern Analysis and Machine Intel-*

ligence, 38(09):1790–1802, 2016. ISSN 1939-3539. doi: 10.1109/TPAMI.2015.2500224.

[107] Girish Joshi, Jasvir Virdi, and Girish Chowdhary. Asynchronous deep model reference adaptive control. In *Conference on Robot Learning (CoRL)*, pages 984–1000, London, England, 2021.

[108] Arie Levant. Sliding order and sliding accuracy in sliding mode control. *International Journal of Control*, 58(6):1247–1263, 1993. doi: 10.1080/00207179308923053.

[109] Jaime A. Moreno and Marisol Osorio. Strict lyapunov functions for the super-twisting algorithm. *IEEE Transactions on Automatic Control*, 57(4):1035–1040, 2012. doi: 10.1109/TAC.2012.2186179.

[110] Peter L. Bartlett, Dylan J. Foster, and Matus J. Telgarsky. Spectrally-normalized margin bounds for neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 30, Long Beach, CA, USA, 2017.

[111] Yuichi Yoshida and Takeru Miyato. Spectral norm regularization for improving the generalizability of deep learning. *arXiv preprint arXiv:1705.10941*, 2017.

[112] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. In *International Conference on Learning Representations*, Vancouver, BC, Canada, 2018.

[113] Hassan K. Khalil. *Nonlinear Systems*. Prentice-Hall, New Jersey, USA, 2002.

[114] Aleksei F. Filippov. *Differential Equations with Discontinuous Right-Hand Side*. Springer Dordrecht, The Netherlands, 1988.

[115] Arie Levant. Robust exact differentiation via sliding mode technique. *Automatica*, 34(3):379–384, 1998. ISSN 0005-1098. doi: 10.1016/S0005-1098(97)00209-4.

[116] Leonid Fridman and Arie Levant. Higher order sliding modes. *Sliding mode control in engineering*, 11:53–102, 2002.

[117] Arie Levant. Homogeneity approach to high-order sliding mode design. *Automatica*, 41(5):823–830, 2005. ISSN 0005-1098. doi: 10.1016/j.automatica.2004.11.029.

[118] Arie Levant. Principles of 2-sliding mode design. *Automatica*, 43(4):576–586, 2007. ISSN 0005-1098. doi: 10.1016/j.automatica.2006.10.008.

[119] Yury Orlov. Finite time stability and robust control synthesis of uncertain switched systems. *SIAM Journal on Control and Optimization*, 43(4):1253–1271, 2004. doi: 10.1137/S0363012903425593.

[120] Andrea Bacciotti and Lionel Rosier. *Liapunov Functions and Stability in Control Theory*. Springer-Verlag Berlin, Heidelberg, Germany, 2005.

[121] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[122] Girish Chowdhary and Eric Johnson. Concurrent learning for convergence in adaptive control without persistency of excitation. In *49th IEEE Conference on Decision and Control (CDC)*, pages 3674–3679, Atlanta, GA, USA, 2010. doi: 10.1109/CDC.2010.5717148.

[123] Girish Chowdhary, Tansel Yucelen, Maximillian Mühlegg, and Eric N. Johnson. Concurrent learning adaptive control of linear systems with exponentially convergent bounds. *International Journal of Adaptive Control and Signal Processing*, 27(4):280–301, 2013. doi: 10.1002/acs.2297.

[124] Anup Parikh, Rushikesh Kamalapurkar, and Warren E. Dixon. Integral concurrent learning: Adaptive control with parameter convergence using finite excitation. *International Journal of Adaptive Control and Signal Processing*, 33(12):1775–1787, 2019. doi: 10.1002/acs.2945.

[125] Yoshua Bengio, Dong-Hyun Lee, Jorg Bornschein, Thomas Mesnard, and Zhouhan Lin. Towards biologically plausible deep learning. *arXiv preprint arXiv:1502.04156*, 2015.

Appendix A: Simulation Details for Chapter 2

A.1 Dynamic Model

Langrangian mechanics were used to derive the dynamic model for the RR-planar robot on a revolute base simulated in Section 2.7. Each link of the robot arm is assumed to be $l_1 = l_2 = 0.2$ m in length, with $m_1 = m_2 = 0.3$ kg point masses at the distal ends of each link. The radius of the base is assumed to be $R_0 = 0.05$ m, with a mass of $m_0 = 0.5$ kg. These values were assumed to model a Robotis OpenManipulator robotic arm. Robot dynamics can generally be written in the form

$$M(q)\ddot{q} + C(q,\dot{q})\dot{q} + G(q) + d(t) = \tau, \tag{A.1}$$

where $q = [q_0, q_1, q_2]^T$ is the vector of joint angles of the robot [57]. In each of the following equations, g is the acceleration due to gravity, l_1 is the length of the arm from joint 1 to joint 2, l_2 is the length of the arm from joint 2 to the end effector, m_i is the mass of the i^{th} joint, and R_0 is the radius of the revolute base.

The mass moment of inertia matrix, $M(q)$, for this robot is given as

$$M(q) = \begin{bmatrix} M_{0,0} & M_{0,1} & M_{0,2} \\ M_{1,0} & M_{1,1} & M_{1,2} \\ M_{2,0} & M_{2,1} & M_{2,2} \end{bmatrix}, \tag{A.2}$$

where

$$M_{0,0} = \frac{1}{2}m_0 R_0^2 + (m_1 + m_2)l_1^2 \cos^2(q_1) + 2m_2 l_1 l_2 \cos(q_1 + q_2)\cos(q_1)$$
$$- 2m_2 l_2^2 \sin(q_1)\sin(q_2)\cos(q_1 + q_2) + m_2 l_2^2(\cos^2(q_1) + \cos^2(q_2)) - m_2 l_2^2 \quad \text{(A.3)}$$

$$M_{0,1} = M_{1,0} = 0 \quad \text{(A.4)}$$

$$M_{1,1} = (m_1 + m_2)l_1^2 + m_2 l_2^2 + 2m_2 l_1 l_2 \cos(q_2) \quad \text{(A.5)}$$

$$M_{0,2} = M_{2,0} = 0 \quad \text{(A.6)}$$

$$M_{1,2} = M_{2,1} = m_2 l_2^2 + m_2 l_1 l_2 \cos(q_2) \quad \text{(A.7)}$$

$$M_{2,2} = m_2 l_2^2. \quad \text{(A.8)}$$

The Coriolis matrix, $C(q, \dot{q})$, for this robot is given as

$$C(q, \dot{q}) = \begin{bmatrix} C_{0,0} & C_{0,1} & C_{0,2} \\ C_{1,0} & C_{1,1} & C_{1,2} \\ C_{2,0} & C_{2,1} & C_{2,2} \end{bmatrix}, \quad \text{(A.9)}$$

where

$$C_{0,0} = -\frac{1}{2}(m_1+m_2)l_1^2 \sin(2q_1)\dot{q}_1 - m_2 l_1 l_2 \sin(2q_1+q_2)\dot{q}_1 - \frac{1}{2}m_2 l_1 l_2 \sin(2q_1+q_2)\dot{q}_2$$
$$- \frac{1}{2}m_2 l_1 l_2 \sin(q_2)\dot{q}_2 - \frac{1}{2}m_2 l_2^2 \sin(2q_1+2q_2)\dot{q}_1 - \frac{1}{2}m_2 l_2^2 \sin(2q_1+2q_2)\dot{q}_2 \quad \text{(A.10)}$$

$$C_{0,1} = \left(-\frac{1}{2}(m_1+m_2)l_1^2 \sin(2q_1) - m_2 l_1 l_2 \sin(2q_1+q_2) \right.$$
$$\left. - \frac{1}{2}m_2 l_2^2 \sin(2q_1+2q_2) \right) \dot{q}_0 \quad \text{(A.11)}$$

$$C_{0,2} = \left(-\frac{1}{2}m_2 l_1 l_2 \sin(2q_1+q_2) - \frac{1}{2}m_2 l_1 l_2 \sin(q_2) - \frac{1}{2}m_2 l_2^2 \sin(2q_1+2q_2) \right) \dot{q}_0 \quad \text{(A.12)}$$

$$C_{1,0} = \left(\frac{1}{2}(m_1+m_2)l_1^2 \sin(2q_1) + m_2 l_1 l_2 \sin(2q_1+q_2) \right.$$
$$\left. + \frac{1}{2}m_2 l_2^2 \sin(2q_1+2q_2) \right) \dot{q}_0 \quad \text{(A.13)}$$

$$C_{1,1} = -m_2 l_1 l_2 \sin(q_2)\dot{q}_2 \quad \text{(A.14)}$$

$$C_{1,2} = -m_2 l_1 l_2 \sin(q_2)\dot{q}_1 - \frac{3}{2}m_2 l_1 l_2 \sin(q_2)\dot{q}_2 \quad \text{(A.15)}$$

$$C_{2,0} = \left(\frac{1}{2} m_2 l_1 l_2 \sin\left(2q_1 + q_2\right) + \frac{1}{2} m_2 l_1 l_2 \sin\left(q_2\right) + \frac{1}{2} m_2 l_2^2 \sin\left(2q_1 + 2q_2\right) \right) \dot{q}_0 \tag{A.16}$$

$$C_{2,1} = m_2 l_1 l_2 \sin\left(q_2\right) \dot{q}_1 + \frac{1}{2} m_2 l_1 l_2 \sin\left(q_2\right) \dot{q}_2 \tag{A.17}$$

$$C_{2,2} = 0. \tag{A.18}$$

Lastly, the gravity vector, $G(q)$, for this robot can be written as

$$G(q) = \begin{bmatrix} G_0 \\ G_1 \\ G_2 \end{bmatrix}, \tag{A.19}$$

where

$$G_0 = 0 \tag{A.20}$$

$$G_1 = (m_1 + m_2) g l_1 \cos\left(q_1\right) + m_2 g l_2 \cos\left(q_1 + q_2\right) \tag{A.21}$$

$$G_2 = m_2 g l_2 \cos\left(q_1 + q_2\right). \tag{A.22}$$

Given a torque input τ and an external disturbance $d(t)$, we can solve (A.1) for the angular acceleration of the joints, $\ddot{q}$:

$$\ddot{q} = M^{-1}(q)\left(\tau - C(q,\dot{q})\dot{q} - G(q) - d(t)\right). \tag{A.23}$$

The relation in (A.23) is integrated twice at each timestep for $\dot{q}$ and q, given the initial position and velocity of each joint.

To simulate load on the end effector, the mass of link 2, m_2, is changed to $m_2 \leftarrow m_2 + m_{load}$, where m_{load} is the mass picked up by the end effector.

A.2 Conventional Model-Based Sliding Mode Controller

The conventional model-based sliding mode controller used in the simulation experiments in Chapter 2 is implemented as in Section 5.5 of [19]. Defining joint error $\tilde{q} = q - q_d$, the sliding variable $s = \dot{\tilde{q}} - \Lambda\tilde{q}$, and a reference sliding variable $s_r = \dot{q}_d - \Lambda\tilde{q}$; the control law applied to the system is written as

$$u = \hat{M}\dot{s}_r + \hat{C}s_r + \hat{G} - K\texttt{sat}(s/\phi), \tag{A.24}$$

where Λ is a positive diagonal design matrix for desired error behavior, ϕ is the boundary layer for chatter attenuation, $\hat{M}$, $\hat{C}$, and $\hat{G}$ are the system model terms calculated using nominal system parameters, K is a diagonal gain matrix whose selection is to be described, and $\texttt{sat}(s/\phi)$ is the saturation function given in (2.9).

The nominal system model terms $\hat{M}$, $\hat{C}$, and $\hat{G}$ are calculated using (A.2)-(A.22) with the nominal (real, with end effector unloaded) system parameters $\hat{m}_0 = 0.5$ kg, $\hat{m}_1 = 0.3$ kg, $\hat{m}_2 = 0.3$ kg, $\hat{R}_0 = 0.05$ m, $\hat{l}_1 = 0.2$ m, and $\hat{l}_2 = 0.2$ m.

The gain matrix K must be selected to guarantee robust control under model uncertainty. The gain matrix, which guarantees controller stability, is calculated as

$$K = \texttt{diag}(|\tilde{M}\dot{s}_r + \tilde{C}s_r + \tilde{G}| + \eta), \tag{A.25}$$

where η is a design gain vector, $\tilde{M} = \hat{M} - M$, $\tilde{C} = \hat{C} - C$, and $\tilde{G} = \hat{G} - G$ are the uncertainties in the system model; and M, C, G are the system model terms under "worst case" values of the system parameters. The terms M, C, G are calculated using (A.2)-(A.22) assuming $m_2 = \hat{m}_2 + \Delta m_2$, where $\Delta m_2 = 0.2$ kg is the designed load margin. The rest of the system parameters used for the "worst-case" calculation are the same as the nominal case above. The desired error behavior is selected as $\Lambda = \texttt{diag}(1, 1, 1)$ to be consistent in comparing the two controllers in Section 2.7. The column gain is set as $\eta = [1, 1, 1]^T$, with a boundary layer value $\phi = [0.05, 0.05, 0.05]^T$ rad. Note that the derivation of this controller assumes that the C matrix used for the system model is chosen such that $\dot{M} - 2C = 0$, which can be verified from the system model given in (A.2)-(A.22). Another important note is that this controller can require much higher integration fidelity for desirable performance in simulation, due to numerical integration failing to resolve discontinuities in the $\texttt{sign}(\cdot)$ function. Further details on this controller can be found in [19].

Appendix B: Stability Analysis of the General Controller for Chapter 3

Proof of Theorem 3.4.1. Differentiating the Lyapunov candidate in (3.10) with respect to time gives

$$\dot{L} \;=\; s_{\Delta}^{T} M \dot{s} \;+\; \mathrm{Tr}\left(\tilde{M}^{T} H^{-1} \dot{\tilde{M}}\right) \;+\; \mathrm{Tr}\left(\tilde{W}^{T} F^{-1} \dot{\tilde{W}}\right) \;+\; \mathrm{Tr}\left(\tilde{V}^{T} G^{-1} \dot{\tilde{V}}\right), \quad \text{(B.26)}$$

where $\dot{s}_{\Delta} = \dot{s}$ since ϕ is a constant and s_{Δ} is only nonzero when $s > \phi$ (outside of the boundary layer). Differentiating (3.2) with respect to time and using the notation simplification in (3.4), $\dot{s}$ can be written as

$$\dot{s} = y_{r}^{(n)} - y^{(n)}, \quad \text{(B.27)}$$

where the system in (3.1) can be rewritten in an affine form as

$$y^{(n)} = M^{-1}\left(u(t) - f(x) - d(t)\right). \quad \text{(B.28)}$$

Substituting (3.3), (B.27), and (B.28) into (B.26) and simplifying gives

$$\dot{L} = s_\Delta^T M y_r^{(n)} - s_\Delta^T \hat{M} y_r^{(n)} + \mathrm{Tr}\left(\tilde{M}^T H^{-1} \dot{\tilde{M}}\right) + s_\Delta^T \left(f - \hat{f}\right)$$
$$+ \mathrm{Tr}\left(\tilde{W}^T F^{-1} \dot{\tilde{W}}\right) + \mathrm{Tr}\left(\tilde{V}^T G^{-1} \dot{\tilde{V}}\right) - s_\Delta^T (D+\eta)\mathtt{sat}(s/\phi) + s_\Delta^T d. \quad \text{(B.29)}$$

Considering the "ideal" neural network as $f = f(x) = W^T\sigma(V^T x) + \epsilon(x)$ (compared to (3.6)), the fourth term of (B.29) can be written as

$$f - \hat{f} = W^T\sigma(V^T x) - \hat{W}^T\sigma(\hat{V}^T x) + \epsilon\,(x)\,, \quad \text{(B.30)}$$

where $\epsilon(x)$ is an approximation error due to the finiteness of the neural network approximator. Adding and subtracting $W^T\sigma(\hat{V}^T x)$ to (B.30) and simplifying gives

$$f - \hat{f} = W^T(\sigma(V^T x) - \sigma(\hat{V}^T x)) + (W^T - \hat{W}^T)\sigma(\hat{V}^T x) + \epsilon\,(x)\,. \quad \text{(B.31)}$$

Using the shorthand notations $\tilde{\sigma} = \sigma(V^T x) - \sigma(\hat{V}^T x)$ and $\hat{\sigma} = \sigma(\hat{V}^T x)$, (B.31) can be rewritten as

$$f - \hat{f} = W^T\tilde{\sigma} + \tilde{W}^T\hat{\sigma} + \epsilon\,(x)\,. \quad \text{(B.32)}$$

Adding and subtracting $\hat{W}^T\tilde{\sigma}$ to (B.32) and simplifying finally gives

$$f - \hat{f} = \tilde{W}^T\tilde{\sigma} + \hat{W}^T\tilde{\sigma} + \tilde{W}^T\hat{\sigma} + \epsilon\,(x)\,. \quad \text{(B.33)}$$

Following [35], the Taylor series expansion for the hidden layer activation function $\sigma(\cdot)$ about a $\hat{V}^T$ for a given input x is written as

$$\sigma(V^T x) = \sigma(\hat{V}^T x) + \sigma'(\hat{V}^T x)\tilde{V}^T x + \mathcal{O}\left((\tilde{V}^T x)^2\right), \tag{B.34}$$

where $\hat{\sigma}'$ denotes the hidden layer activation derivative and $\mathcal{O}\left((\tilde{V}^T x)^2\right)$ denotes higher-order terms. Subtracting over the $\sigma(\hat{V}^T x)$ in (B.34) and using the shorthand notation gives

$$\tilde{\sigma} = \hat{\sigma}'\tilde{V}^T x + \mathcal{O}\left((\tilde{V}^T x)^2\right). \tag{B.35}$$

Substituting (B.35) into the first and second terms of (B.33) now gives

$$f - \hat{f} = \hat{W}^T\hat{\sigma}'\tilde{V}^T x + \tilde{W}^T\hat{\sigma} + d_a(t), \tag{B.36}$$

where $d_a(t)$ is the internal "disturbance" due to neural network approximation error and higher-order Taylor series terms, written as

$$d_a(t) = \tilde{W}^T\hat{\sigma}'\tilde{V}^T x + W^T\mathcal{O}\left((\tilde{V}^T x)^2\right) + \epsilon(x) \tag{B.37}$$

since $\tilde{W}^T + \hat{W}^T = W^T$. Substituting (B.36) back into the Lyapunov derivative in (B.29) gives

$$\dot{L} = s_\Delta^T M y_r^{(n)} - s_\Delta^T \hat{M} y_r^{(n)} + \mathrm{Tr}\left(\tilde{M}^T H^{-1} \dot{\tilde{M}}\right) + s_\Delta^T \hat{W}^T \hat{\sigma}' \tilde{V}^T x + s_\Delta^T \tilde{W}^T \hat{\sigma}$$
$$+ \mathrm{Tr}\left(\tilde{W}^T F^{-1} \dot{\tilde{W}}\right) + \mathrm{Tr}\left(\tilde{V}^T G^{-1} \dot{\tilde{V}}\right) - s_\Delta^T \hat{M}(D+\eta)\mathtt{sat}(s/\phi) + s_\Delta^T (d_a + d). \tag{B.38}$$

In (B.38), the "disturbance" due to learning, d_a, acts on the system just as the external disturbance, d, does. Thus, as discussed in Chapter 2, a disturbance-rejecting robustifying control term can stabilize the system in the early stages of learning when d_a is large, which is especially important in aerospace control (such as quadcopters).

In general, for column vectors $\vec{\alpha}$ and $\vec{\beta}$, the inner product can be written as the matrix trace of the outer product, $\vec{\alpha}^T \vec{\beta} = \mathrm{Tr}\,(\vec{\beta}\vec{\alpha}^T)$. Using this matrix trace property, relevant terms in (B.38) can be rewritten as

$$s_\Delta^T M y_r^{(n)} = \mathrm{Tr}\,(M y_r^{(n)} s_\Delta^T) \tag{B.39}$$

$$s_\Delta^T \hat{M} y_r^{(n)} = \mathrm{Tr}\,(\hat{M} y_r^{(n)} s_\Delta^T) \tag{B.40}$$

$$s_\Delta^T \hat{W}^T \hat{\sigma}' \tilde{V}^T x = \mathrm{Tr}\,(\tilde{V}^T x s_\Delta^T \hat{W}^T \hat{\sigma}') \tag{B.41}$$

$$s_\Delta^T \tilde{W}^T \hat{\sigma} = \mathrm{Tr}\,(\tilde{W}^T \hat{\sigma} s_\Delta^T). \tag{B.42}$$

Using the relations in (B.39)-(B.42) and the additive trace property $\mathrm{Tr}\,(A) + \mathrm{Tr}\,(B) = \mathrm{Tr}\,(A+B)$, (B.38) can be simplified to

$$\begin{aligned}\dot{L} = &\mathrm{Tr}\left(\tilde{M}\left(y_r^{(n)} s_\Delta^T + H^{-1}\dot{\tilde{M}}\right)\right) + \mathrm{Tr}\left(\tilde{W}^T\left(\hat{\sigma} s_\Delta^T + F^{-1}\dot{\tilde{W}}\right)\right) \\ &+ \mathrm{Tr}\left(\tilde{V}^T\left(x s_\Delta^T \hat{W}^T \hat{\sigma}' + G^{-1}\dot{\tilde{V}}\right)\right) - s_\Delta^T (D+\eta)\mathtt{sat}(s/\phi) + s_\Delta^T (d_a + d). \quad \text{(B.43)}\end{aligned}$$

Note that $\tilde{M} = M - \hat{M}$ was used to combine the first two terms of (B.38). Since $\dot{\tilde{M}} = -\dot{\hat{M}}$, $\dot{\tilde{V}} = -\dot{\hat{V}}$, and $\dot{\tilde{W}} = -\dot{\hat{W}}$, (B.43) can be further simplified to

$$\begin{aligned}\dot{L} = &\mathrm{Tr}\left(\tilde{M}\left(y_r^{(n)} s_\Delta^T - H^{-1}\dot{\hat{M}}\right)\right) + \mathrm{Tr}\left(\tilde{W}^T\left(\hat{\sigma} s_\Delta^T - F^{-1}\dot{\hat{W}}\right)\right) \\ &+ \mathrm{Tr}\left(\tilde{V}^T\left(x s_\Delta^T \hat{W}^T \hat{\sigma}' - G^{-1}\dot{\hat{V}}\right)\right) - s_\Delta^T (D+\eta)\mathtt{sat}(s/\phi) + s_\Delta^T (d_a + d). \quad \text{(B.44)}\end{aligned}$$

Substituting the update rules (3.7), (3.8), and (3.9) into (B.44), (B.44) finally reduces to

$$\dot{L} = s_\Delta^T \left(-(D+\eta)\mathtt{sat}(s/\phi) + \delta(t)\right), \quad \text{(B.45)}$$

where $\delta(t) = d_a(t) + d(t)$ is the total disturbance on the system. Assuming that D is the upper bound of the infinite norm of $\delta(t)$ for $\forall t \geq 0 : ||\delta(t)||_\infty \leq D$,

$$\dot{L} \leq -\eta ||s_\Delta||_1 \quad \text{(B.46)}$$

for all s outside the boundary layer. Since $L > 0$, selecting a positive gain $\eta > 0$ forces $\dot{L} \leq 0$ such that s_Δ approaches zero as $t \to \infty$ and $\tilde{M}$, $\tilde{V}$, and $\tilde{W}$ are bounded in time [9, 56]. Further, since $\tilde{M}$, $\tilde{V}$, and $\tilde{W}$ are bounded in time, the estimates $\hat{M}$, $\hat{V}$, and $\hat{W}$ are bounded in time [55]. □