Neural Networks as Add-on Modules for Improved Performance of Robot Control Systems

by

Siqi Zhou

A thesis submitted in conformity with the requirements
for the degree of Doctor of Philosophy

Graduate Department of Aerospace Science and Engineering
University of Toronto

Neural Networks as Add-on Modules for Improved Performance of Robot Control Systems

Siqi Zhou
Doctor of Philosophy
Graduate Department of Aerospace Science and Engineering
University of Toronto
2022

Abstract

Robots are envisioned to become reliable companions in domains ranging from advanced industrial applications to our daily lives. In the literature, well-established control techniques provide the foundation for designing high-performance robot systems with desired theoretical guarantees. However, these control techniques often rely on a dynamics model of the robot, and any inaccuracies in the model can result in suboptimal performance or even unsafe actions. These limitations motivate the incorporation of machine learning into the traditional robot decision-making stack to identify and thereby compensate for model uncertainties. In this dissertation, learning-based control approaches are developed to enhance the performance of black-box robot systems. While different machine learning models may be similarly applied to enhance the performance of a robot control system, in this thesis, we focus on neural-network-based approaches and analyze the properties that the neural network modules must satisfy for safe and data-efficient deployment.

This dissertation contains five contributions. In the first contribution, we introduce a neural network inverse dynamics learning approach for improving the performance of black-box robot control systems in arbitrary trajectory tracking tasks (i.e., impromptu trajectory tracking). In the second contribution, we extend the inverse dynamics approach to non-minimum phase robot systems (i.e., robot systems with unstable inverse dynamics). In the third contribution, we propose an active training trajectory generation framework for systematically training a neural network inverse

dynamics model. In the fourth contribution, we outline a data-efficient online learning scheme that allows us to transfer the inverse dynamics model trained on one robot to enhance the tracking performance of another robot. In the last contribution, we propose a learning-based model reference adaptive controller where a Lipschitz network is updated online to bridge the model-reality gap in uncertain robot systems and the system's stability is guaranteed by exploiting the architectural design of the network. Through theoretical analysis and physical experiments using quadrotors, we demonstrate that, by combining control theory and the expressiveness of neural networks, we can design safe and efficient robot learning algorithms to achieve high performance despite the uncertainties present in the environment.

Acknowledgments

I would like to thank my supervisor, Prof. Angela Schoellig, for taking me onto and guiding me through this exciting and memorable journey. She has always been inspirational, encouraging, and supportive. It was a tremendous honour and pleasure to have the opportunity to work with her.

I also wish to thank my Doctoral Examination Committee, Prof. Tim Barfoot and Prof. Jonathan Kelly, for providing me with invaluable feedback throughout my doctoral research and Prof. Aude Billard and Prof. Florian Shkurti for being external examiners in my Final Oral Examination.

During my PhD, I was grateful for having the opportunities to collaborate with amazing researchers at the Dynamic Systems Lab of the University of Toronto Institute for Aerospace Studies (UTIAS). I wish to thank my close collaborators and co-authors, Lukas Brunke, Xuchan Bao, Xintong Du, Melissa Greeff, Adam Hall, Mohamed Helwa, Karime Pereida, Jacopo Panerati, Michael Sorocky, Andriey Sarabakha, Jingxing Qian, Justin Yuan, Zining Zhu, and Wenda Zhao. I learned a lot from them through the collaborative projects, and the experiences were enjoyable and rewarding.

I would also like to thank Chris McKinnon, Mario Vukosavljev, and Thomas Bamford, who shared their wisdom as senior students and Adam Heins, Julian Foerster, Dave Kooijman, Carlos Luis, Sepehr Samavi, Ke Dong, Kennan Burnett, and Bhavit Patel, who have been great colleagues during my PhD. I would also like to express my gratitude for the support from the Vector Institute and the University of Toronto Robotics Institute communities.

I would like to thank my parents and family, who have always been the backbone and backed up my decisions in every adventure I took. I would not have gone this far without their continuous encouragement and endless support. Moreover, I would like to thank my friends, especially Olive Sun, Joanna Yu, Lian Liu, Yinan Xu, Ashley Ding, Rui Cai, and Linghong Li, for always being there whenever I needed them.

Contents

List of Figures

Notation

General Notation

$\mathbb{R}$	the set of real numbers
$\mathbb{Z}$	the set of integers
$\in$	an element of
$\lvert \cdot \rvert$	absolute value of a scalar
$\lvert\lvert \cdot \rvert\rvert$	norm of a vector
$(\cdot)^T$	transpose of a vector or a matrix
$\frac{\partial}{\partial (\cdot)}$	partial derivative
$\dot{(\cdot)}, \ddot{(\cdot)}$	the first and the second derivative with respect to time
∇	gradient operator
$\mathbb{E}$	expectation of a random variable
$\mathbb{V}$	variance of a random variable
$\mathbb{H}$	entropy of a random variable
min	minimization of a function
arg min	a value that minimizes a function
$(\cdot)^*$	an optimal value

Robot Control System

k	discrete-time index
x	system state
u	system input
y	system output
r	system (vector) relative degree
n	system order
(A, B, C)	linear system matrices
(f, g, h)	functions defining a control-affine nonlinear system
$(\mathcal{A}, \mathcal{B})$	matrices defining the input-output map of a linear system
$(\mathcal{F}, \mathcal{G})$	functions defining the input-output map of a nonlinear system

Neural Network

F	nonlinear map represented by a neural network
ξ	network input vector
ζ	hidden layer output vector
γ	network output vector
(W, b)	network weight matrices and bias vectors
θ	augmented vector of network parameters
$\mathcal{D}$	dataset

Subscripts

d	desired signal
a	variables associated with an actual robot control system
t	variables associated with a target robot control system
s	variables associated with a source robot control system
m	variables associated with a reference model

Chapter 1

Introduction

Robots are envisioned to become reliable companions in domains ranging from advanced industrial applications to our daily lives. Well-established control techniques such as optimal control [1] and model predictive control (MPC) [2] provide the foundation for designing high-performance robot control systems. However, these control techniques often rely on a dynamics model of the robot, and modeling accuracy is consequently a limiting factor affecting the robot's performance in experiments. While traditional control techniques have successfully been applied to robots in known and static environments, robots are expected to perform increasingly complex tasks in unstructured and dynamic environments (e.g., autonomous driving and warehouse management). These challenges motivate advanced decision-making approaches that allow robots to continuously learn and adapt after deployment to maintain safety and high performance over their lifespan [3].

In parallel with the advances in control theory, machine learning has had several recent breakthroughs in constructing complex models from data (e.g., in speech recognition [4], image classification [5], and object detection [6]). There are also numerous empirical successes of leveraging data-driven approaches for robot decision-making problems. Some examples include end-to-end visuomotor policy learning for acquiring manipulation skills based on raw image inputs [7], imitation learning for realizing autonomous helicopter acrobatic maneuvers [8] or quadruped locomotion in tough terrain [9], and simulation-to-reality (sim2real) transfer learning for dexterous manipulations [10]. Data-driven approaches typically utilize expressive learning models to allow robots to acquire complex skills with minimal human input. While being flexible, several open challenges hinder practical deployment of these algorithms in real-world applications. Commonly mentioned challenges include the interpretability of the learned components (e.g., policy, dynamics model, or reward function),

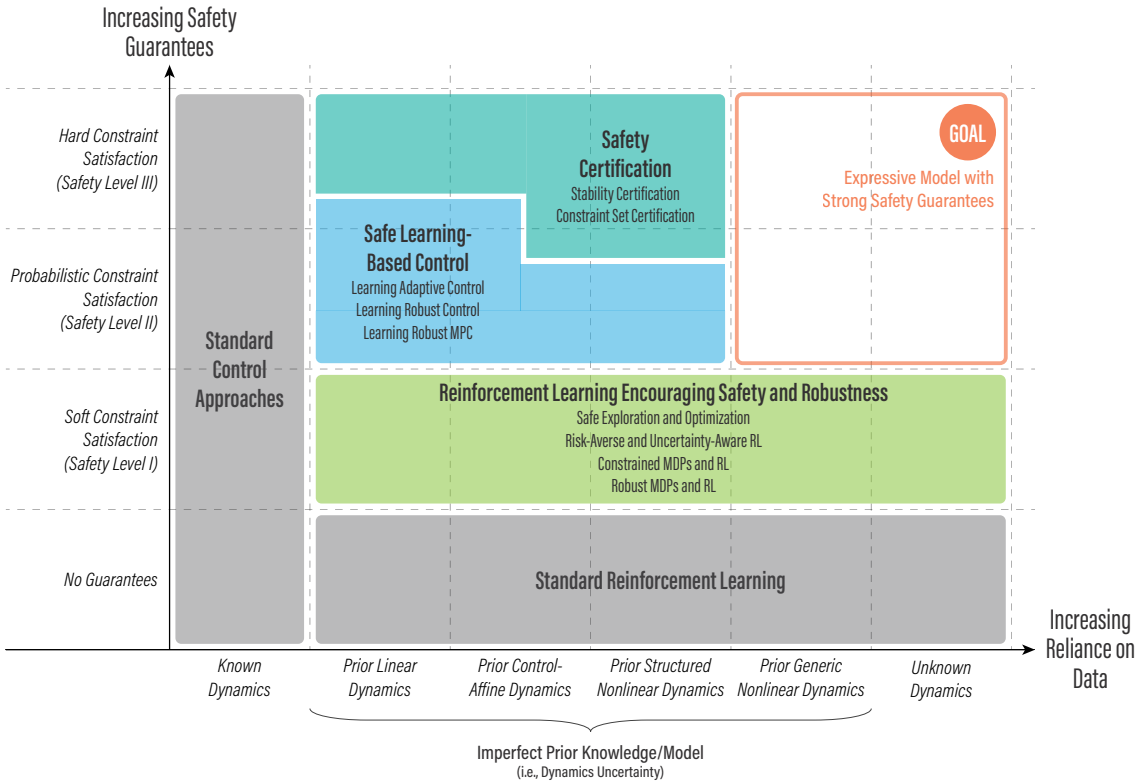


Figure 1.1: An overview of recent advances in safe decision making for robotics (from [3]). Both classical and recent approaches are plotted based on two features. The horizontal axis shows the extent of the reliance on data of the approach, and the vertical axis reflects the level of safety guarantees provided by the approach. Traditional control techniques typically exploit well-characterized system dynamics or structures to provide safety guarantees. In comparison, standard reinforcement learning approaches are applicable to a broader class of uncertain systems; however, it is often challenging to provide safety guarantees. Recently, we have observed an increasing number of works, including safe learning-based control, safe reinforcement learning and safety certification approaches, that combine the advantage of control and learning to address the problem of safe robot decision-making under uncertainties. The ultimate goal is to incorporate expressive models for closed-loop control such that the robot performance is improved without jeopardizing safety.

efficiently learning from limited samples, generalization to continuous and possibly high-dimensional state and action spaces, and most importantly, providing provable safety guarantees [3, 11].

Over the past five years, we have observed an increasing number of works combining the flexibility of machine learning and the domain knowledge from control theory to design autonomous systems with improved performance while providing desired theoretical guarantees [3] (Fig. 1.1). For instance, learning-based MPC leverages statistical learning for inferring unknown dynamics and the MPC framework for con-

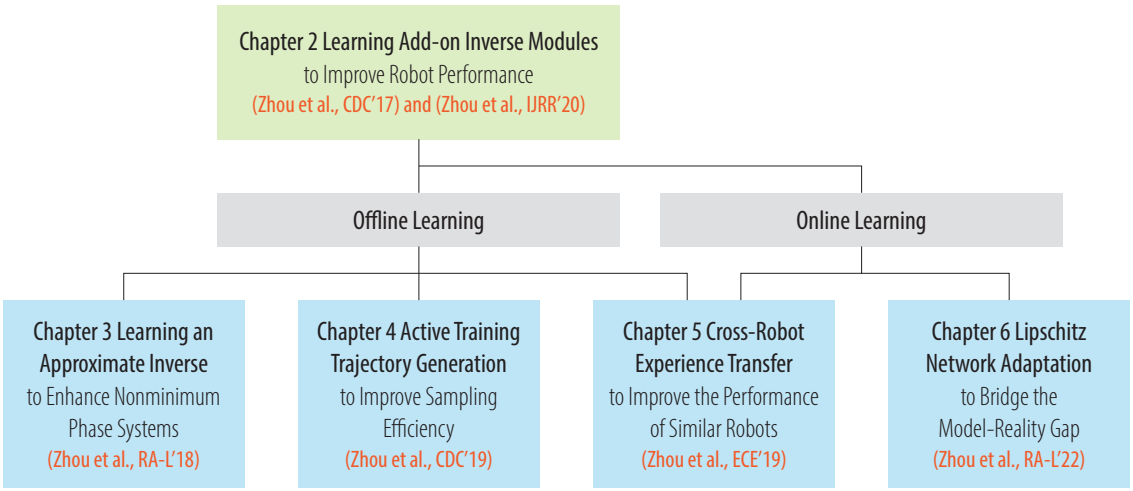


Figure 1.2: An overview of the contributions included in this dissertation.

straint satisfaction [12, 13], certified reinforcement learning integrates control theory (e.g., Hamilton-Jacobi reachability analysis [14] and robust control theory [15]) with reinforcement learning algorithms for learning complex policies with safety guarantees, and our work [16] exploits the expressiveness of neural networks to approximate the inverse dynamics of a closed-loop system to reliably enhance the performance of robots in arbitrary trajectory tracking tasks.

The successful examples in the literature show a great promise of integrating machine learning techniques and control theory for real-world robot autonomy [3]. This thesis investigates the incorporation of machine learning techniques, especially neural networks, to improve the performance of black-box robot control systems (i.e., robot control systems whose exact dynamics are not known). As compared to techniques such as Gaussian processes (GPs), neural networks have the advantage of having fixed computation cost and memory for test-time inference, making them desirable for robotic applications where computing resources are limited. Numerous works exploit this advantage of neural networks for real-time robot control (e.g., [17] and [18]). While these approaches have been successfully tested in experiments, it remains challenging to provide theoretical guarantees for neural controllers applied to robot systems subject to unknown and possibly time-varying disturbances.

This thesis presents approaches that safely and efficiently exploit the modeling capability of neural networks to improve the performance of uncertain robot systems. While other learning models such as GPs may be similarly applied to improve the performance of robot control systems, we focus on neural-network-based approaches and derive guidelines that allow neural networks to be safely and efficiently deployed

in closed-loop control applications. The work presented in this dissertation falls under the safe learning-based control category (blue region) in Fig. 1.1. The novel contributions presented in this thesis are as follows (see also Fig. 1.2):

1. Learning Add-on Inverse Modules to Improve Robot Performance (Chapter 2): We propose a novel neural network add-on inverse dynamics learning framework. This approach can be applied to efficiently enhance the performance of uncertain robot systems in tracking arbitrary feasible trajectories. We demonstrate our approach in quadrotor impromptu tracking experiments, where a quadrotor is tasked to track arbitrary hand-drawn trajectories accurately in single attempts.

2. Learning an Approximate Inverse to Enhance Non-minimum Phase Systems (Chapter 3): The neural network inverse dynamics learning approach presented in Chapter 2 requires the uncertain robot system to be minimum phase (i.e., has stable forward and inverse dynamics). This chapter introduces a stable, approximate inverse dynamics learning approach that extends to non-minimum phase robot systems (i.e., robot systems with unstable inverse dynamics). Through theoretical discussions, simulations, and experiments on two different platforms, we show the stability of the proposed approximate inverse dynamics learning approach and its effectiveness for high-accuracy, impromptu tracking. To the best of our knowledge, this work is the first work that learns stable approximate inverse dynamics models for non-minimum phase systems.

3. Active Training Trajectory Generation to Improve Sampling Efficiency (Chapter 4): The quality of training data directly determines the performance of a learning-based control system. In this chapter, we introduce an episode algorithm that integrates a spline trajectory optimization approach with active learning for efficiently training a neural network inverse dynamics module. We show that, as compared to ad hoc, intuitive training approaches, the proposed active training trajectory generation approach leads to improved data efficiency and performance.

4. Cross-Robot Experience Transfer to Improve the Performance of Similar Robots (Chapter 5): Inspired by the transfer learning literature, in this chapter, we present a novel online learning framework that enables the inverse dynamics model learned for a source robot to be transferred to a target robot and show a preliminary study on the role of system similarity in the cross-robot transfer problem. In quadrotor experiments, we illustrate that, by leveraging the

experience from the source robot, the proposed transfer learning approach allows the target robot to achieve high-accuracy trajectory tracking on arbitrary trajectories with minimal data recollection and retraining.

5. Lipschitz Network Adaptation to Bridge the Model-Reality Gap (Chapter 6): In the last part of this dissertation, we derive a neural network adaptation technique that allows an uncertain robot system to behave like a predefined model and thereby bridge the model-reality gap. The stability of the adapted system is guaranteed by exploiting the Lipschitz property of a special type of neural network, the Lipschitz network [19]. We demonstrate the approach in flying inverted pendulum experiments, where an uncertain, off-the-shelf quadrotor is challenged to balance an inverted pendulum while hovering at fixed points or tracking circular trajectories. Our work is the first that demonstrates the capability of Lipschitz networks, both theoretically and experimentally, for the closed-loop control of uncertain systems.

The publications pertinent to this dissertation are listed below in chronological order. Chapter 2 corresponds to [1] and [5], Chapter 3 through Chapter 5 correspond to [2] to [5], respectively, and Chapter 6 corresponds to [6].

[1] Siqi Zhou, Mohamed K. Helwa, and Angela P. Schoellig, "Design of deep neural networks as add-on blocks for improving impromptu trajectory tracking," in *Proceedings of the IEEE Conference on Decision and Control (CDC)*, 2017, pp. 5201-5207.

[2] Siqi Zhou, Mohamed K. Helwa, and Angela P. Schoellig, "An inversion-based learning approach for improving impromptu trajectory tracking of robots with non-minimum phase dynamics," *IEEE Robotics and Automation Letters (RA-L)*, vol. 3, no. 3, pp. 1663-1670, 2018.

[3] Siqi Zhou and Angela P. Schoellig, "Active training trajectory generation for inverse dynamics model learning with deep neural networks," in *Proceedings of the IEEE Conference on Decision and Control (CDC)*, 2019, pp. 1784-1790.

[4] Siqi Zhou, Andriy Sarabakha, Erdal Kayacan, Mohamed K. Helwa, and Angela P. Schoellig, "Knowledge transfer between robots with similar dynamics for high-accuracy impromptu trajectory tracking," in *Proceedings of the European Control Conference (ECC)*, 2019, pp. 1-8.

[5] Siqi Zhou, Mohamed K. Helwa, and Angela P. Schoellig, "Deep neural networks as add-on modules for enhancing robot performance in impromptu trajectory tracking," *International Journal of Robotics Research (IJRR)*, vol. 39, no. 12, pp. 1397-1418, 2020.

[6] Siqi Zhou, Karime Pereida, Wenda Zhao, and Angela P. Schoellig, "Bridging the model-reality gap with Lipschitz network adaptation," *IEEE Robotics and Automation Letters (RA-L)*, vol. 7, no. 1, pp. 642-649, 2022.

In addition to the publications above, there are ten contributions I have worked on that are related to the discussions of this dissertation (* denotes equal contribution); I am a main contributor in [7]-[9], [12], [14], and [15]:

[7] Siqi Zhou and Angela P. Schoellig, "An analysis of the expressiveness of deep neural network architectures based on their Lipschitz constant", *arXiv preprint arXiv:1912.11511*, 2019.

[8] Michael J. Sorocky*, Siqi Zhou* and Angela P. Schoellig, "Experience selection using dynamics similarity for efficient multi-source transfer learning between robots," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2020, pp. 2739-2745.

[9] Michael J. Sorocky, Siqi Zhou, and Angela P. Schoellig, "To share or not to share? performance guarantees and the asymmetric nature of cross-robot experience transfer," *IEEE Control Systems Letters (L-CSS)*, vol. 5, no. 3, pp. 923-928, 2021.

[10] Lukas Brunke, Siqi Zhou, and Angela P. Schoellig, "RLO-MPC: Robust learning-based output feedback MPC for improving the performance of uncertain systems in iterative tasks," in *Proceedings of the IEEE Conference on Decision and Control (CDC)*, 2021, pp. 2183-2190.

[11] Jacopo Panerati, Hehui Zheng, Siqi Zhou, James Xu, Amanda Prorok, and Angela P. Schoellig, "Learning to fly—a gym environment with PyBullet physics for reinforcement learning of multi-agent quadcopter control," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021, pp. 7512-7519.

[12] Lukas Brunke*, Melissa Greeff*, Adam W. Hall*, Zhaocong Yuan*, Siqi Zhou*, Jacopo Panerati, and Angela P. Schoellig, "Safe learning in robotics: From

learning-based control to safe reinforcement learning," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 5, no. 1, pp. 411-444, 2022.

[13] Melissa Greeff, Siqi Zhou, and Angela P. Schoellig, "Fly out the window: Exploiting discrete-time flatness for fast vision-based multirotor flight," *IEEE Robotics and Automation Letters (RA-L)*, vol. 7, no. 2, pp. 5023-5030, 2022.

[14] Lukas Brunke*, Siqi Zhou*, and Angela P. Schoellig, "Barrier Bayesian linear regression: Online learning control barrier conditions for safety-critical control under model uncertainty," in *Proceedings of the Annual Learning for Dynamics and Control Conference (L4DC)*, 2022, accepted.

[15] Zhaocong Yuan, Adam W. Hall, Siqi Zhou, Lukas Brunke, Melissa Greeff, Jacopo Panerati, and Angela P. Schoellig, "safe-control-gym: A unified benchmark suite for safe learning-based control and reinforcement learning," *IEEE Robotics and Automation Letters (RA-L)*, 2022, accepted.

[16] Lukas Brunke, Siqi Zhou, and Angela P. Schoellig, "Robust predictive output-feedback safety filter for uncertain nonlinear control systems," in *Proceedings of the IEEE Conference on Decision and Control (CDC)*, 2022, accepted.

Chapter 2

Learning Add-on Inverse Modules to Improve Robot Performance

2.1 Introduction

As continued advancements in algorithms, actuation, and sensor technology push robots into more complex environments, increasingly sophisticated methods for controlling robot motion are needed. In particular, controllers that are capable of high-accuracy trajectory tracking are becoming increasingly important in robot applications where safety and/or efficiency are essential. For example: in search and rescue, where robots must operate in close proximity to people [20]; in advanced manufacturing, where robot arms must efficiently follow pre-designed trajectories to perform complex manipulation tasks [21]; or in industrial inspection, where unmanned aerial vehicles fly in close proximity to facilities to enable visual inspection [22].

The trajectory tracking problem has been extensively studied in the control literature. Among various techniques, the proportional-integral-derivative (PID) controller is often used in trajectory tracking applications. However, tuning PID parameters is typically time-consuming, and the performance of a PID controller can be conservative [23]. Moreover, control theory shows that a standard PID control architecture cannot achieve exact tracking for arbitrary trajectories [24].

In addition to the PID controller, model-based techniques such as Model Predictive Control (MPC) have been studied for finding optimal control commands that lead to accurate and agile robot motions [25]. Moreover, inversion-based feedforward approaches have been widely applied to achieve high-accuracy tracking [26, 27]. However, one general limitation of model-based approaches is the reliance on a suffi-

ciently accurate dynamic model of the system, which is difficult to obtain in practice. Adaptive control [28] and robust control [29] strategies have been used to address uncertainties in system parameters. Yet while these approaches typically guarantee stability, they do not take past experience into account, and the same errors are repeated from trial to trial if the same reference is given.

As robot dynamics and operating environments become ever more complex, researchers are increasingly turning to learning-based approaches to address the resulting model uncertainties. These learning-based approaches have been successfully applied to manipulators [30], bipedal robots [31], autonomous cars [32], and unmanned aerial vehicles [33, 34], to name a few. A common learning-based approach that yields high-accuracy tracking is iterative learning control (ILC). In ILC, the tracking performance is improved by adjusting control inputs or reference signals in repeated trials [35, 36, 37]. In addition to ILC, reinforcement learning (RL)-based approaches have also been proposed to iteratively optimize the tracking performance [38, 39, 40].

Apart from iterative approaches, there are various works on improving the tracking performance of classical model-based controllers by learning the uncertain or unknown system dynamics using techniques such as Gaussian processes (GPs) [41, 42], neural networks (NNs) [43, 44], and locally weighted projection regression (LWPR) [47]. These learning techniques have also been applied to improve the tracking performance by approximating inverse dynamic models in inversion-based feedforward approaches [46, 47]. The survey paper [111] provides a more detailed review of machine learning techniques and their applications in robot control. We note that various nonlinear function approximation techniques could be potentially applied in our framework. We focus on DNNs due to their capability to model complex nonlinear functions while having relatively fast real-time inference. Probabilistic learning approaches such as GPs have the advantage of providing uncertainty bounds that can be leveraged for guaranteeing closed-loop stability (e.g., [42]); however, their computation time typically scales quadratically or cubically with the number of data points [111]. LWPRs can also be used for nonlinear model learning. Their computational complexity scales linearly with the number of data points, but they are local function approximators that do not generalize to the entire state space of a robot system [111]. DNNs, on the other hand, are universal function approximators [73]; but, due to their black-box nature, they usually require more training data to achieve a good performance. In this work, we focus on an offline learning setting and assume that a sufficient amount of data is available for training. We further explore efficient training data collection approaches in Chapter 4.

In this chapter, we consider the impromptu tracking problem. That is, we aim to achieve high-accuracy tracking of arbitrary, feasible trajectories from the first attempt. Motivated by the success of the learning-based control approaches for robot control, we present a deep neural network (DNN)-based approach for enhancing the impromptu tracking control performance of black-box systems. This work is motivated by our previous work [48], in which a DNN add-on module was used to improve the performance of quadrotors in tracking arbitrary, hand-drawn trajectories. The proposed DNN-enhancement architecture is illustrated in Fig. 2.1. During the training phase, the input, output, and state of the baseline system are recorded for training a DNN module. Then, during the testing phase, the DNN module is pre-cascaded to the baseline system to adapt the reference signals to establish an identity map from the desired output to the actual output. In [48], experiments on 30 arbitrary, hand-drawn trajectories show that the DNN-enhancement control architecture effectively reduces the tracking error of the quadrotor vehicle by 43% on average as compared to the baseline controller. As compared with the other learning-based tracking control approaches, the proposed DNN-approach has the following advantages:

- Unlike the iterative learning methods (ILC approaches and some RL-based approaches such as [40]), the proposed DNN approach can be directly used for tracking arbitrary, feasible trajectories without further adaptations during the testing phase, and consequently, it satisfies the impromptu tracking requirement.

- Compared to more common approaches (such as forward or inverse dynamic learning) where the learning component typically resides in the main control loop, we use the DNN module as an add-on block that is placed outside of the closed-loop system to improve the tracking performance. This add-on approach enables black-box control systems to be improved retrospectively.

- As will be discussed in the following section, the proposed approach is less prone to instability than other inverse-based approaches because the DNN loop can be run at a lower rate than the baseline control loop [48]. Moreover, the proposed architecture can potentially lead to better learning-enhanced performance as the closed-loop system has a more repeatable behaviour [49].

While experiments from [48] have shown that the DNN approach shown in Fig. 2.1 is effective for quadrotors, the DNN module was designed by trial-and-error, and guidelines for systematically applying the approach to other robotic platforms were

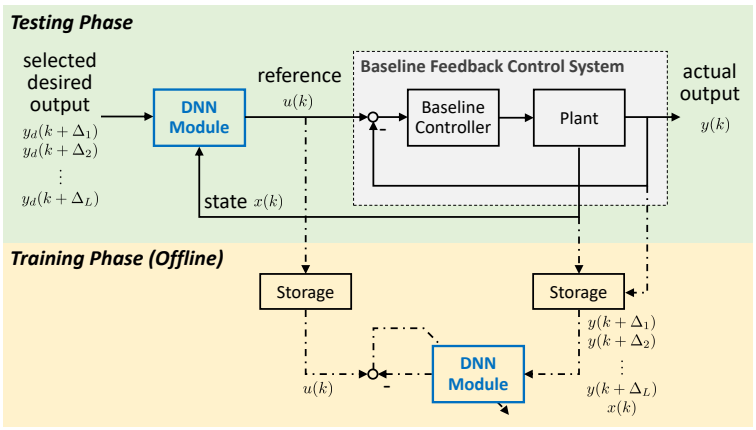


Figure 2.1: The DNN-enhancement control architecture: During the training phase (shaded yellow region), a baseline system is treated as a black box, and the reference u, output y, and state x are recorded for training a DNN module. During the testing phase (shaded green region), the DNN module is pre-cascaded to the baseline system and adjusts the reference $u(k)$ based on the current state $x(k)$ and a set of selected future desired output $y_d(k+\Delta_i)$ to enhance the tracking performance of the baseline system, where $k \in \mathbb{Z}_{\geq 0}$ is the discrete-time index and $\Delta_i \in \mathbb{Z}_{>0}$.

not given. In [50], we presented preliminary theoretical results on the DNN-based approach based on a single-input-single-output (SISO) system formulation. In this chapter, we provide a comprehensive theoretical study of the DNN-based approach and support it with both simulations and extensive experimental results. In particular, based on a multi-input-multi-output (MIMO) system formulation, our study in this chapter includes: (1) characterizing the underlying function represented by the DNN module, (2) identifying a necessary condition for the DNN approach to be effective, (3) deriving a condition that allows for further improving the DNN training data efficiency, and (4) analyzing the stability of the overall DNN-enhanced system given the presence of modeling errors in the DNN module. These theoretical insights are illustrated in simulation and verified with extensive quadrotor experiments.

We also note that, in the first work [48], a DNN is chosen as the learning technique to construct the add-on block. This design decision was motivated by the fact that the amount of memory and computational cost of the forward pass of the DNN is fixed as more data is collected. In contrast to nonlinear regression methods such as GPs, the relatively fixed memory and computational cost allow the DNN model to be implemented on robot platforms where onboard computational resources are limited [48]. Following [48], we use DNNs as the learning technique in our work; however, the presented theoretical insights can be potentially generalized to other nonlinear regression techniques.

2.2 Related Work

The DNN module in the proposed control architecture (Fig. 2.1) aims to establish an identity map from the desired output to the actual output [48]. In the literature of NN-based control, common approaches that have a similar objective include *direct inverse control*, *feedback-error learning control*, and *adaptive inverse control*.

In direct inverse control, an NN is trained to approximate the inverse dynamics of the open-loop plant, and is pre-cascaded to the plant as the controller to achieve exact tracking [51, 52]. Early literature such as [53, 54] compared different approaches for training the NN inverse model and discussed details concerning practical implementation. For example, [53] pointed out that an NN directly trained with the reversed input-output data from the open-loop plant is not 'goal-directed' — the training objective of minimizing the regression error of the model output does not directly reflect the control objective of minimizing the tracking error of the system. To address these concerns, training schemes such as the distal teacher [53] have been proposed. However, apart from these discussions, a fundamental drawback of the direct inverse control approach is the lack of robustness against disturbances in the system. This drawback is attributed to the fact that the NN inverse model is often used as the only controller of the system.

To address the issues with direct inverse control, [54] proposes a feedback-error learning scheme. This approach employs a feedback control loop, where the input command to the plant is the sum of the signal from the feedback controller and feedforward signal from an NN-based inverse model. In contrast with typical direct inverse control, the error signal for training the NN is the output of the feedback controller instead of the typical regression errors based on the plant input-output data. Although practical considerations such as the 'goal-directness' issue and robustness issue are addressed in the feedback error learning approach, the training of the NN requires a plant in the loop, which may not be desired in the early training phase.

Another inversion-based approach for trajectory tracking problems is adaptive inverse control, in which the parameters of an NN controller are updated online to realize tracking functionalities [55, 56, 57]. A limitation of the adaptive-NN approach is that an appropriate initialization for the NN parameters are typically needed for convergence [55].

Overall, despite the similarity in the control objective, there are fundamental differences between the proposed DNN control architecture in Fig. 2.1 and the common NN-based inverse control architectures. One of the differences is that the proposed

DNN control architecture modifies the reference of a stabilized closed-loop system, while the common NN-based inverse control approaches directly modify the input to the open-loop plant. From a practical perspective, this difference has two potential benefits: *(i)* By introducing the DNN as an outer loop that runs at a lower rate as compared to the baseline system, the overall approach is less prone to stability issues [48]. *(ii)* Since the closed-loop system partially compensates non-repeated disturbances, the response of the closed-loop system is more repeatable than that of the open-loop plant [49]. Thus, learning to adapt the reference of a closed-loop system can be potentially more effective for achieving good tracking performance. In contrast to adaptive inverse control, in which high-accuracy tracking control and stability of the plant are simultaneously achieved by the designed NN parameter update laws, the proposed DNN approach achieves stabilization through the design of the baseline controller, and tracking performance is enhanced separately by the pre-cascaded DNN module. This approach of decoupling the stabilization and tracking performance enhancement problems can greatly simplify the DNN design and training in practical applications. We furthermore investigate the effectiveness of the proposed DNN approach for the problem of impromptu tracking, and verify this experimentally by testing whether quadrotors are able to accurately fly arbitrary, hand-drawn trajectories from the first attempt. Although the NN-based inverse control approaches in the literature provide theoretical foundations for designing high-accuracy tracking controllers, their ability to track arbitrary, feasible trajectories has not been thoroughly demonstrated in experiments.

2.3 Problem Formulation

Our objective is to enhance black-box control systems to achieve high-accuracy, impromptu tracking. In [48], with quadrotors as the test platform, a DNN-enhancement control architecture (Fig. 2.1) was proposed to establish an identity map from the desired output y_d to the actual output y. In this chapter, we aim to provide a platform-independent formulation of the proposed DNN-enhancement control architecture [48]. This formulation includes the following objectives:

(O1) identifying the underlying function that should be represented by the DNN module in order to establish an identity map from y_d to y;

(O2) identifying necessary conditions for the approach to be effective;

(O3) deriving guidelines for systematically selecting the inputs and outputs of the

DNN module;

(O4) analyzing the stability of the DNN-enhanced system in the presence of regression errors; and

(O5) characterizing a condition that allows for further improving data efficiency of the DNN training.

In the following discussion, we first consider linear time invariant (LTI) multi-input-multi-output (MIMO) baseline systems represented by the following state space model

$$\begin{aligned} x(k+1) &= Ax(k) + Bu(k), \\ y(k) &= Cx(k), \end{aligned} \tag{2.1}$$

where $k \in \mathbb{Z}_{\geq 0}$ denotes the discrete-time index, $x \in \mathbb{R}^n$ is the system state, $u \in \mathbb{R}^m$ is the reference signal sent to the baseline system, $y \in \mathbb{R}^m$ is the system output, and A, B, and C are constant matrices of appropriate dimensions. After presenting the insights from the linear system formulation, we then extend the discussion to nonlinear MIMO baseline systems represented by

$$\begin{aligned} x(k+1) &= f(x(k)) + g(x(k))u(k), \\ y(k) &= h(x(k)), \end{aligned} \tag{2.2}$$

where $f(\cdot)$, $g(\cdot)$, and $h(\cdot)$ are matrices of smooth functions with appropriate dimensions. Note that, in the discussion of this work, we focus on square MIMO systems having the same number of inputs and outputs. This is not a restrictive formulation for tracking applications, since systems (2.1) and (2.2) typically represent baseline closed-loop systems, and each output in y has a corresponding reference input in u. Also note that we consider discrete-time closed-loop baseline systems. The sampling time of the system is defined by the underlying controller that is assumed to be given.

In deriving the theoretical insights, we make the following assumptions:

(A1) the baseline system is input-to-state stable [58]. For the nonlinear system (2.2), we additionally assume that the state can be bounded by

$$||x||_\infty \leq L_1||u||_\infty + L_2||x_0|| + L_3, \tag{2.3}$$

where $||\cdot||_\infty$ denotes the infinity norm, $x_0 \in \mathbb{R}^n$ is the initial state, and L_1, L_2, and L_3 are constant, positive scalars;

(A2) at any instant k, a preview of n future time steps of the desired trajectory (i.e., $\{y_d(k), y_d(k+1), ..., y_d(k+n)\}$) is available, where n is the system order;

(A3) the DNN module has a feedforward architecture and globally Lipschitz activation functions.

Note that assumptions (A1)-(A3) are not restrictive. Assumption (A1) on the stability of the baseline closed-loop system can be achieved by proper controller designs with well-developed control techniques even in the absence of a detailed or highly-accurate dynamic model of the system. We note that while we require the baseline control system to be stable, it does not necessarily have a satisfactory tracking performance. Our goal is to derive an add-on learning approach to enhance the tracking performance of the baseline control system on arbitrary feasible trajectories. The inequality (2.3) in Assumption (A1) holds for input-to-state stable linear systems; for nonlinear systems, this is an additional assumption that we use to provide a theoretical guarantee on stability of the overall control system. For assumption (A2), a preview of n steps of the desired trajectory is usually available in practice, and does not prevent combinations with online trajectory generation algorithms. For assumption (A3), although we use feedforward neural networks (FNNs) in our work, the proposed approach can be potentially adapted for use with other nonlinear regression techniques (e.g., GPs, recurrent neural networks). The globally Lipschitz condition in assumption (A3) holds for the commonly used activation functions such as the rectified linear unit (ReLU), sigmoid, and hyperbolic tangent.

2.4 Derivation and Theoretical Analysis

In this section we provide four theoretical insights to achieve the objectives (O1)-(O5) stated in Sec. 3.3. We begin our discussion with a background on the inversion of dynamic systems in Sec. 2.4.1. We then build on this conceptual overview in Sec. 2.4.2 to derive the underlying function to be modeled by the DNN module to establish an identity map between the desired and actual outputs, and identify conditions that are necessary for the proposed DNN approach to be effective. Building on the insight regarding the underlying function modeled by the DNN module, we provide guidelines for systematically selecting the inputs and outputs of the DNN module in Sec. 2.4.3, provide a proof of stability of the overall control system in the presence of DNN regression errors in Sec. 5.4.3, and derive a condition that allows us to further improve the data-efficiency of the DNN training in Sec. 2.4.5.

2.4.1 Background on System Inversion

Starting with the DNN-enhancement control architecture in Fig. 2.1, [48] initially designed a DNN module with $y_d(k)$ and $x(k)$ as input and $u(k)$ as output to enhance the tracking performance of the quadrotor baseline control system. The experiments of [48] show that the DNN module is able to enhance the tracking performance of the baseline system only after $y_d(k)$ in the DNN input is replaced by certain future desired outputs $\{y_d(k+\Delta_1), y_d(k+\Delta_2), ..., y_d(k+\Delta_L)\}$ with $\Delta_1, \Delta_2, ..., \Delta_L \in \mathbb{Z}_{>0}$ selected based on trial-and-error. As will be shown in Sec. 2.4.2, this experimental observation can be explained by associating the DNN module with the inverse dynamics of the baseline system.

In order to facilitate the following discussions, in this subsection, we state the formal definition of the *vector relative degree* [59], and discuss its connection to the system inverse. In the discussions below, we use $h \circ f$ to denote the composition of the functions h and f, and f^i to denote the i-th composition of the function f with $f^0\big(x(k)\big) = x(k)$ and $f^i\big(x(k)\big) = f \circ f^{i-1}\big(x(k)\big)$.

Definition 2.4.1 (Vector Relative Degree)**.** *The nonlinear MIMO system* (2.2) *has a vector relative degree* $(r_1, r_2, ..., r_m)$ *at an operating point* (x_0, u_0) *if*

(i) $\frac{\partial}{\partial u_j} h_i \circ f^p\big(f(x) + g(x)u\big) = 0$, $\forall i = \{1, 2, ..., m\}$, $\forall p = \{0, 2, ..., r_i - 2\}$, $\forall j = \{1, 2, ..., m\}$ *for every point* (x, u) *in some neighbourhood of* (x_0, u_0)*, where* u_j *is the* j*-th element of the input* u*, and* h_i *is the* i*-th element of the vector function* h*; and*

(ii) the decoupling matrix $\mathcal{G}(x, u) \in \mathbb{R}^{m \times m}$ *with elements* $[\mathcal{G}(x, u)]_{ij} = \frac{\partial}{\partial u_j} h_i \circ f^{r_i - 1}\big(f(x) + g(x)u\big)$ *has full rank at the operating point* (x_0, u_0)*.*

Note that, from the first condition *(i)* of Definition 2.4.1, if we focus on an output dimension y_i, the relative degree r_i can be interpreted as the number of sample delays between changing any of the inputs u_j, $j = 1, \cdots, m$, and changing the output y_i. Given that both *(i)* and *(ii)* of Definition 2.4.1 are satisfied, the relative degree r_i associates the value of an output y_i at time step $k + r_i$ with a non-zero input u applied at time step k. The decoupling matrix $\mathcal{G}(x, u)$ in the second condition *(ii)* of Definition 2.4.1 is the collection of the Jacobian of $y_i(k + r_i)$ with respect to the input u; the non-singularity condition requires that the outputs $y(k + r) = [y_1(k + r_1) \ \cdots \ y_m(k + r_m)]^T$ are influenced by the input $u(k)$ in non-repeated (linearly independent) ways.

Remark 2.4.1 (Vector Relative Degree for Linear Systems)**.** *As a special case of Definition 2.4.1, the linear MIMO system (2.1) has a vector relative degree* $(r_1, r_2, ..., r_m)$ *if*

(i) $C_i A^p B_j = 0$, $\forall i = \{1, 2, ..., m\}$, $\forall p = \{0, 2, ..., r_i - 2\}$, $\forall j = \{1, 2, ..., m\}$, *where* C_i *is the* i*-th row of the matrix* C *and* B_j *is the* j*-th column of the matrix* B*, and*

(ii) the decoupling matrix $\mathcal{B} \in \mathbb{R}^{m \times m}$ *with elements* $[\mathcal{B}]_{ij} = C_i A^{r_i - 1} B_j$ *has full rank.*

Example 2.4.1 (Relative Degree)**.** *Consider an LTI, SISO system with the system matrices* (A, B, C) *defined as follows:*

$$A = \begin{bmatrix} 0 & 1 \\ -0.15 & 0.8 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \quad \text{and} \quad C = \begin{bmatrix} 0.8 & 0 \end{bmatrix}. \tag{2.4}$$

The relative degree of the system is given by $(p+1)$ *with* p *being the smallest integer such that* $CA^pB \neq 0$*. To determine the relative degree of the system, we compute the value of* CA^pB *for* $p = \{0, 1, ...\}$ *until the product of the matrices becomes non-zero:*

p	CA^pB
0	*0.0*
1	*0.8*

For this example, the system relative degree is two. This implies that there is an inherent delay of two time steps between applying an input to the system and observing a corresponding change in the system output.

Note that, from Definition 2.4.1, for MIMO systems with a well-defined vector relative degree, one may relate the future output $y(k + r)$ to the current state $x(k)$ and input $u(k)$. Having a well-defined vector relative degree is a necessary condition for applying the proposed DNN-based enhancement approach. Although a generic nonlinear MIMO system may not necessarily have a full-rank decoupling matrix and thus a well-defined vector relative degree, this property holds for practical robot systems such as quadrotors and robot manipulators [59]. To the best of our knowledge, we are unaware of a robot system example where this condition does not hold. In the following subsections, based on the notion of vector relative degree, we formalize the DNN-based approach proposed in [48], develop theoretical insights for systematically designing the DNN module, and provide comments on its practical implementation.

2.4.2 Underlying Function Modeled by the DNN Module

In this subsection we show that, given the system representations in (2.1) and (2.2), an identity map from the desired output y_d to the actual output y is achieved if the DNN module learns the output equation of the inverse dynamics of the baseline system. Due to this association with inverse dynamics, a necessary condition for the proposed approach to be effective is that the baseline system has stable inverse dynamics. For simplicity, we will start our discussion with the linear system (2.1) and then extend the results to the nonlinear system (2.2). Note that although we start our discussion with known system models, we will later demonstrate that implementing the proposed DNN-enhancement approach requires only minimal knowledge about the baseline system (e.g., its order and relative degree). This required knowledge can typically be determined from simple dynamic models or step response experiments.

By applying the definition of the vector relative degree in Remark 2.4.1 to the linear system (2.1), we can relate the input u and the output y of the baseline system by

$$y_i(k+r_i) = C_i A^{r_i} x(k) + C_i A^{r_i-1} B u(k), \tag{2.5}$$

or in augmented form,

$$y(k+r) = \mathcal{A}x(k) + \mathcal{B}u(k), \tag{2.6}$$

where $y(k+r) = [y_1(k+r_1) \ \cdots \ y_m(k+r_m)]^T$, $\mathcal{A} = [(C_1A^{r_1})^T \ \cdots \ (C_mA^{r_m})^T]^T$, and $\mathcal{B}$ is the decoupling matrix of system (2.1).

Let $y_d(k+r) = [y_{1,d}(k+r_1) \cdots y_{m,d}(k+r_m)]^T$ be the desired output corresponding to $y(k+r)$. Since the decoupling matrix $\mathcal{B}$ has full rank by condition *(ii)* of the vector relative degree definition in Remark 2.4.1, it can be shown that if we choose the following control law

$$u(k) = \mathcal{B}^{-1}\left(-\mathcal{A}x(k) + y_d(k+r)\right), \tag{2.7}$$

then $y(k+r) = y_d(k+r)$, or exact tracking, is achieved. Thus, for the proposed DNN-enhancement control architecture in Fig. 2.1 and system (2.1), the DNN module should be trained to approximate (2.7) to establish an identity map between y_d and y. If we consider y_d as the input and u as the output, (2.7) is in fact the output equation of the inverse dynamics of system (2.1).

Note that the first condition *(i)* in Remark 2.4.1 implies that the relative degree r_i associated with the output dimension i is the smallest integer such that $C_iA^{r_i-1}B_j \neq 0$ for any input dimension j. As briefly noted in Sec. 2.4.1, the relative degree r_i is the

number of sample delays between applying an input u to the system and first seeing its effect in the particular output y_i. This inherent delay from input to output is a well-known fact for discrete-time linear systems. By training the DNN module to approximate (2.7), the inherent delay of the system is compensated by the preview of the future desired output $y_d(k+r)$. In practice, at a particular time k, a preview of r steps of the desired trajectory (where $r \leq n$) is not challenging to satisfy with online or offline trajectory generation algorithms; the non-causality in (2.7) is thus not an issue in practical applications.

We next generalize the previous discussion to nonlinear systems. By assuming the system (2.2) has a well-defined vector relative degree, and applying Definition 2.4.1, we can relate the input u and output y of the nonlinear MIMO system (2.2) by

$$y_i(k+r_i) = h_i \circ f^{r_i-1}\big(f(x(k)) + g(x(k))u(k)\big), \tag{2.8}$$

or in an augmented form

$$y(k+r) = h \circ f^{r-1}\big(f(x(k)) + g(x(k))u(k)\big), \tag{2.9}$$

where $h \circ f^{r-1}$ is a vector of composition functions with the i-th element being $h_i \circ f^{r_i-1}$. As discussed in [60, 59], by assuming $y(k+r)$ is affine in the input $u(k)$, the decoupling matrix $\mathcal{G}(x,u)$ is independent of u and (2.9) becomes

$$y(k+r) = \mathcal{F}\big(x(k)\big) + \mathcal{G}\big(x(k)\big)u(k), \tag{2.10}$$

where $\mathcal{F}\big(x(k)\big) = h \circ f^{r}\big(x(k)\big)$ is a composite function with the i-th element being $h_i \circ f^{r_i}$. This special case holds for nonlinear mechanical systems such as robot manipulators [59]. Since the decoupling matrix $\mathcal{G}$ has full rank by the second condition *(ii)* in Definition 2.4.1, exact tracking (i.e., $y(k+r) = y_d(k+r)$) can be achieved by choosing the control law

$$u(k) = \mathcal{G}^{-1}\big(x(k)\big)\big(-\mathcal{F}\big(x(k)\big) + y_d(k+r)\big) \tag{2.11}$$

for the affine case in (2.10), and it is reasonable to assume that

$$u(k) = F(x(k), y_d(k+r)) \tag{2.12}$$

for the general case in (2.9), where $F : \mathbb{R}^n \times \mathbb{R}^m \mapsto \mathbb{R}^m$ is a vector of nonlinear functions.

Based on the above results, we now present our insight on the underlying function modeled by the DNN module, and describe the conditions that are necessary for the learning-based approach to be effective:

Remark 2.4.2 (Underlying Function and Necessary Conditions)**.** *Consider the DNN-enhancement control architecture in Fig. 2.1. In order to establish an identity map between the desired output y_d and the actual output y, the DNN module should approximate the output equation of the baseline system's inverse dynamics. Due to the association with inverse dynamics, two necessary conditions for the learning approach to be effective are: (i) the baseline system has a well-defined (vector) relative degree; and (ii) the baseline system has stable zero dynamics.*

Example 2.4.2 (Ideal Control Law for Exact Tracking)**.** *Consider the SISO system in Example 2.4.1. The system has a relative degree of two and does not have any unstable zeros. For this example, we assume that the system matrices (A, B, C) are known and use the ideal control law (2.7) for exact tracking. We can compute the values of $\mathcal{A}$ and $\mathcal{B}$ in (2.7) as follows:*

$$\mathcal{A} = CA^r = [-0.12 \quad 0.64] \quad \textit{and} \quad \mathcal{B} = CA^{r-1}B = 0.8, \tag{2.13}$$

where the system relative degree r is two. Given $\mathcal{A}$ and $\mathcal{B}$, the ideal control law for this example is

$$u(k) = [0.15 \quad -0.8]\, x(k) + 1.25\, y_d(k+2). \tag{2.14}$$

We illustrate the control law in (2.14) for exact tracking using a simple numerical example. Suppose that the initial state of the system is $x_0 = [0,\ 0]^T$, and the desired output is $y_d(k) = \sin(k)$. The values of the desired output $y_d(k)$, the input $u(k)$ computed based on (2.14), and the system output $y(k)$ for the first few time steps are shown below:

k	$y_d(k)$	$u(k)$	$y(k)$	$y_d(k) - y(k)$
0	*0.00*	*1.14*	*0.00*	*0.00*
1	*0.84*	*-0.73*	*0.00*	*0.84*
2	*0.91*	*-0.92*	*0.91*	*0.00*
3	*0.14*	*-0.42*	*0.14*	*0.00*
4	*-0.76*	*0.47*	*-0.76*	*0.00*
5	*-0.96*	*0.92*	*-0.96*	*0.00*
⋮	⋮	⋮	⋮	⋮

Note that the system output $y(k)$ is zero for the first two time steps. This is expected as the relative degree (or, the inherent delay) of the system is two. For $k \geq 2$, we see that the input computed based on (2.14) leads to exact tracking. As we will discuss in Sec. 2.4.3, when the system dynamics are not known, we can train a DNN to approximate the ideal control law.

By inspecting the control laws in (2.7) and (2.12), it can be seen that the ideal control law that leads to exact tracking is dependent on the current $x(k)$ and the future desired output $y_d(k+r)$ for either the linear or nonlinear case, where r is the vector relative degree. In practice, when training the DNN module to approximate the control law for achieving exact tracking, we do not require a detailed dynamic model of the system. Instead, we need only identify the vector relative degree r of the baseline system. Experimentally, for the linear system (2.1) and the special case of the nonlinear system (2.2) where $y(k+r)$ is affine in $u(k)$, one can identify the vector relative degree of the baseline system through m step response experiments detailed as follows. In each of the m experiments, the system is initialized at an equilibrium point, and one element of the input, u_j, is activated. Without loss of generality, we assume the equilibrium is the origin. After the m experiments, one may determine the minimum number of time delays between the output y_i and the inputs u_j for all j; the minimum number of time delays for the output dimension y_i is the estimated relative degree r_i associated with the particular output dimension. After estimating the relative degree for each output dimension, it remains to check the non-singularity condition *(ii)* in Definition 2.4.1. From the m experiments, one may construct a matrix $\widetilde{D}$, where the j-th column of $\widetilde{D}$ is $[y_1(r_1) \cdots y_m(r_m)]^T$ from the j-th experiment. By inspecting (2.6) and (2.10), it can be shown that the non-singularity condition *(ii)* in Definition 2.4.1 can be examined from the rank of $\widetilde{D}$.

The stability of the zero dynamics of the linear system (2.1) is equivalent to the stability of the system's inverse dynamics, and is characterized by the zeros of the system transfer function. In practice, for linear systems, we may infer the stability of zero dynamics from characteristics of the system's step responses such as undershoot and zero crossings [61]. The zero dynamics of the nonlinear system (2.2) is the system's invariant dynamics when the input $u(k)$ is chosen such that $y(k) = 0$ for all k. For nonlinear systems, achieving stable zero dynamics is a necessary but not sufficient condition for achieving stable inverse dynamics [62]. Hence, a necessary condition for applying the proposed DNN-learning approach to either the linear system (2.1) or the nonlinear system (2.2) is that the baseline system has stable zero dynamics.

2.4.3 DNN Input Selection

In this subsection, we identify the necessary and sufficient inputs of the DNN module to compute the reference $u(k)$ of the baseline system (2.1) and (2.2) to achieve exact tracking. By designating the output of the DNN module as $\mathcal{O} = \{u(k)\}$, we can determine the appropriate DNN input $\mathcal{I}$ for either the linear or the nonlinear case based on the following insight.

Remark 2.4.3 (DNN Input Selection)**.** *In order to establish an identity map from y_d to y, the necessary and sufficient input of the DNN add-on module is $\mathcal{I} = \{x(k),\ y_d(k+r)\}$, where $y_d(k+r) = [y_{1,d}(k+r_1)\ \cdots\ y_{m,d}(k+r_m)]^T$ and $r = (r_1, ..., r_m)$ is the vector relative degree of the system.*

Remark 2.4.3 directly follows from the fact that the DNN should approximate the baseline system inverse to achieve unity mapping between y_d and y and from (2.7) and (2.12) of the system inverse.

Example 2.4.3 (DNN Input Selection)**.** *For the SISO system in Example 2.4.1, we have $r = 2$, and the input selection of the DNN add-on module for establishing an identity map from y_d to y is $\mathcal{I} = \{x(k),\ y_d(k+2)\}$.*

The implementation of Remark 2.4.3 requires knowledge or estimation of the full state of the system x. In many robotics applications, linearization techniques are used for the baseline system controller designs, and this often leads to decoupled linear dynamics. Some examples include ground vehicles in which the dynamics in the two-dimensional position space can be converted to decoupled integrators with the point-ahead linearization technique [63], and fully-actuated manipulators in which the dynamics in the joint space can be turned into decoupled double integrators with feedback linearization [64]. In cases where the full state of the system is not available, but where the closed-loop dynamics can be approximated as a decoupled MIMO linear system, we can derive an alternative DNN input selection.

In deriving the alternative input selection, we first equivalently represent system (2.1) by $Y(z) = H(z)U(z)$, where

$$H(z) = C(zI - A)^{-1}B, \tag{2.15}$$

and $U(z)$ and $Y(z)$ are the z-transform of the input and output of the baseline system, respectively. To show the main idea, we first consider the special case of a SISO linear system (i.e., $m = 1$). Without loss of generality, we assume that the SISO system is

represented by a discrete-time transfer function of the following form:

$$H(z) = \frac{Y(z)}{U(z)} = \frac{\beta_{n-r}z^{n-r} + \beta_{n-r-1}z^{n-r-1} + \cdots + \beta_0}{z^n + \alpha_{n-1}z^{n-1} + \cdots + \alpha_0}, \tag{2.16}$$

where α_i and β_i are scalar constants, and r and n are the relative degree and degree of the system respectively. By calculating the inverse system of (2.16) and applying inverse z-transformation, it can be shown that the reference $u(k)$ for achieving exact tracking is

$$\begin{aligned} u(k) = &\frac{1}{\beta_{n-r}}y_d(k+r) + \frac{\alpha_{n-1}}{\beta_{n-r}}y_d(k+r-1) + \cdots + \frac{\alpha_0}{\beta_{n-r}}y_d(k-n+r) \\ &- \frac{\beta_{n-r-1}}{\beta_{n-r}}u(k-1) - \frac{\beta_{n-r-2}}{\beta_{n-r}}u(k-2) - \cdots - \frac{\beta_0}{\beta_{n-r}}u(k-n+r). \end{aligned} \tag{2.17}$$

Based on (2.17), we can alternatively select the DNN input for a SISO linear baseline system to be $\mathcal{I} = \{y_d(k-n+r : k+r), u(k-n+r : k-1)\}$, where the column ':' abbreviates consecutive discrete-time indexes.

The transfer matrix $H(z)$ of a decoupled MIMO linear system (2.1) is a diagonal matrix; the dynamics between each input-output pair (u_i, y_i) can be considered separately, where $i \in \{1, ..., m\}$. As outlined in Sec. (2.4.2), one can execute m experiments to identify the relative degree r_i for each output y_i. Similar to the SISO scenario discussed above, in the case of the decoupled MIMO linear system, we can consider each reference dimension separately and train m networks with the input of each network being $\mathcal{I}_i = \{y_{d,i}(k-n+r_i : k+r_i), u_i(k-n+r_i : k-1)\}$ and output being $\mathcal{O} = \{u_i(k)\}$, where r_i is the relative degree corresponding to the i-th output dimension, and $y_{d,i}$ denotes the i-th desired output dimension.

Remark 2.4.4 (Alternative Input Selection for Decoupled MIMO Linear Systems)**.** *Based on the transfer function formulation, we can derive an alternative, sufficient input selection of the DNN module for a decoupled MIMO linear system. For this case, we propose using a DNN module with m independent networks – one for each of the baseline system reference dimensions. The input to the i-th network in the DNN module is* $\mathcal{I}_i = \{y_{d,i}(k-n+r_i : k+r_i), u_i(k-n+r_i : k-1)\}$*, where* r_i *is the relative degree corresponding to the output dimension* y_i*. Note that, in the special case where* $r_i = n$*, the reference* $u_i(k)$ *for exacting tracking does not depend on past references, and the input to the i-th network is* $\mathcal{I}_i = \{y_{d,i}(k-n+r_i : k+r_i)\}$*.*

Example 2.4.4 (Alternative DNN Input Selection)**.** *The order and the relative degree of the SISO system we considered in Example 2.4.1 are* $n = 2$ *and* $r = 2$*, respectively.*

An alternative input selection for the DNN add-on module for establishing an identity map from y_d to y is $\mathcal{I} = \{y_d(k),\ y_d(k+1),\ y_d(k+2)\}$.

In comparison with the input selection based on the state space representation (Remark 2.4.3), the implementation of the alternative input for the linear systems does not require the estimation of the full state $x(k)$ of the system and instead only requires the identification of the order of the system n, which can be determined from the laws of physics for common robot systems such as multi-link manipulators and quadrotors. Note, however, that this transfer function approach is derived for decoupled linear systems; the state space approach is applicable to more general cases. When the state of the system is available, applying the state space approach has additional advantages. One advantage of the state space approach is the current state feedback to the DNN module. This additional feedback from the baseline system can help compensate for the initial errors and disturbances along the trajectory. Moreover, the input selection based on the state space approach typically leads to a DNN with a lower input dimension than the transfer function approach. As an example, for a SISO linear system, the dimension of the DNN inputs derived from the transfer function and the state space approaches are $(2n - r + 1)$ and $(n + 1)$, respectively. This reduced DNN input dimension implies that the amount of data required to cover the operational space is potentially less, and thus the DNN training can be made more efficient by using the state space approach.

2.4.4 Stability

In this subsection, we restrict our discussion to minimum phase systems (i.e., systems with stable inverse dynamics), and prove the stability of the overall DNN-enhancement control system in the presence of DNN modeling errors:

$$||u(k) - \hat{u}(k)|| \neq 0, \tag{2.18}$$

where $u(k)$ corresponds to the exact inverse in (2.12) ((2.7) for the linear system case) and $\hat{u}(k)$ corresponds to the reference outputted by the DNN module trained based on the system input-output data. Note that, in the ideal case, where the DNN models the inverse dynamics exactly, the response from the desired output to the actual output is the identity map, and the overall system is input-to-state stable. However, in the presence of modeling errors, due to the state feedback connection to the DNN module (see Fig. 2.1), the stability of the overall system needs to be assessed. In this subsection, we show that under Assumptions (A1) and (A3), the DNN-enhanced

system with the proposed input selection as in Remark 2.4.3 is input-to-state stable if the regression error of the DNN module is sufficiently small.

By assumption (A3), the DNN module has a feedforward architecture, and the activation functions are globally Lipschitz; since the DNN is a composite of linear combinations of Lipschitz functions, the output of the DNN module, $\hat{u}$, is globally Lipschitz in its inputs, x and y_d. In particular, we can bound the output of the DNN module by

$$||\hat{u}||_\infty \leq L_4||x||_\infty + L_5||y_d||_\infty, \tag{2.19}$$

where L_4 and L_5 are positive, constant scalars associated with a Lipschitz constant of the network, which can be either estimated [65] or prescribed [19]. Moreover, since we consider a baseline system that is minimum phase (i.e., has a stable inverse dynamics), the reference $u(k)$ corresponding to the exact inverse in (2.12) is bounded (i.e., $||u||_\infty < \infty$). As a result of the global Lipschitz condition of the DNN module in (2.19) and the boundedness of u, an upper bound on the modeling error of the DNN module can be derived as follows:

$$||u - \hat{u}||_\infty \leq ||\hat{u}||_\infty + ||u||_\infty \leq L_4||x||_\infty + L_5||y_d||_\infty + L_6, \tag{2.20}$$

where $L_6 = ||u||_\infty$ is the bound on the exact inverse reference u of the minimum phase baseline system.

Theorem 2.4.1 (Stability of the DNN-Enhancement Approach)**.** *Consider the DNN-enhancement control architecture (Fig. 2.1) and the case where the baseline system is minimum phase. Under assumptions* (A1) *and* (A3) *, the overall DNN-enhanced system is input-to-state stable if* $L_1L_4 < 1$*, where* L_1 *and* L_4 *are constant scalars defined in* (2.3) *and* (2.20)*, respectively.*

Proof. By assumption (A1) , the baseline system is input-to-state stable, and with $\hat{u}$ as the system input, the state of the system is bounded by

$$||x||_\infty \leq L_1||\hat{u}||_\infty + L_2||x_0|| + L_3. \tag{2.21}$$

By combining the bound on the regression error in (2.20) and the bound on state in (2.21), the following is obtained:

$$||x||_\infty \leq L_1||u - \hat{u}||_\infty + L_1||u||_\infty + L_2||x_0|| + L_3 \tag{2.22}$$

$$\leq L_1L_4||x||_\infty + L_1L_5||y_d||_\infty + L_7, \tag{2.23}$$

where $L_7 = L_1 L_6 + L_1 ||u||_\infty + L_2 ||x_0|| + L_3$. Based on (2.23), if

$$L_4 < \frac{1}{L_1}, \tag{2.24}$$

is satisfied, then the state of the system is bounded by

$$||x||_\infty \leq \frac{L_1 L_5 ||y_d||_\infty + L_7}{1 - L_1 L_4}, \tag{2.25}$$

which is bounded by a constant for bounded input y_d. Thus, if $L_4 < \frac{1}{L_1}$, then the DNN-enhanced system is input-to-state stable. □

Note that, by examining (2.21) and (2.20), L_1 is a constant characterizing the maximum possible gain of the baseline system, while L_4 is a constant associated with the regression error of the DNN model. Hence, the condition in (2.24) implies that if the regression error of the DNN module is sufficiently small, then the overall DNN-enhancement control architecture is input-to-state stable. One can notice the similarity between condition (2.24) and the well-known small gain theorem in robust control [66].

In practice, one could estimate the gain of the system from data [67, 68, 69] and use (2.24) as a certifying condition for training the DNN inverse module. However, we note that, since we use the global Lipschitz property of the DNN to show stability, this condition could be conservative. The globally Lipschitz activation function assumption in Assumption (A3) and thus the condition in (2.20) are generally sufficient (but not necessary) conditions for closed-loop stability.

2.4.5 Difference Learning Scheme for Improving the Training Efficiency

In this subsection, we derive a condition that allows us to further improve the data-efficiency of the proposed DNN-enhancement approach. This discussion is motivated by the DNN design in [48], where the position terms in the DNN input and output are taken relative to the current desired and actual positions in order to simplify the training process. The basic idea of this difference learning scheme is that with the relative positions (instead of the absolute positions), the function modeled by the DNN becomes invariant under spatial translations, which reduces the amount of data needed to cover the operation space. Based on the theoretical formulations presented in the previous subsections, we derive in this section a necessary condition

for the effectiveness of the difference learning scheme. This necessary condition will be further illustrated with quadrotor experiments in Sec. 2.6.

In order to motivate this insight on the difference learning scheme, we first focus our discussion on a SISO linear system represented by the transfer function representation in (2.16). Recall that, for system (2.16), the control law for achieving exact tracking is described by (2.17), and the corresponding DNN input-output selection for learning the system inverse is $\mathcal{I} = \{y_d(k-n+r : k+r), u(k-n+r : k-1)\}$ and $\mathcal{O} = \{u(k)\}$, where ':' is used to abbreviate consecutive time indexes. With the difference learning scheme, we aim to train a DNN that depends only on a set of relative terms: $\Delta_{y_d}(k+p) := y_d(k+p) - y_d(k)$ for $p \in \{-n+r, ..., r\}$ and $\Delta_u(k+p) := u(k+p) - y_d(k)$ for $p \in \{-n+r, ..., 0\}$, where y_d is the desired output, u is the reference of the baseline system, k is the current time index, and p is a shift in the time index. In this work, we aim to enhance the tracking performance of square MIMO baseline systems, and we assume that there is a one-to-one correspondence between the reference u and the output y, and hence a one-to-one correspondence between the reference u and the desired output y_d. For a position tracking system as an example, the terms Δ_{y_d} and Δ_u can be intuitively interpreted as the relative position vectors from the current desired position $y_d(k)$ to a past/future desired position $y_d(k+p)$ and a past reference position $u(k+p)$.

Lemma 2.4.2 (Difference Learning for SISO Linear Systems)**.** *Consider a SISO linear baseline system (2.16) and the DNN-enhancement control architecture in Fig. 2.1. A difference learning scheme can be applied to introduce translational invariance in the inverse learning problem and thereby reduce the amount of data required for training the DNN module if and only if the baseline system has a unity DC gain.*

Proof. Starting from the control law in (2.17), it can be shown that by subtracting $y_d(k)$ on both sides of the equation, and adding and subtracting $\frac{1}{\beta_{n-r}} y_d(k)$, $\frac{1}{\beta_{n-r}} \sum_{i=0}^{n-r-1} \beta_i y_d(k)$ and $\frac{1}{\beta_{n-r}} \sum_{i=0}^{n-1} \alpha_i y_d(k)$ on the right-hand side, (2.17) can be written as

$$\begin{aligned}\Delta_u(k) = {} & \frac{1}{\beta_{n-r}}\Delta_{y_d}(k+r) + \frac{\alpha_{n-1}}{\beta_{n-r}}\Delta_{y_d}(k+r-1) + \cdots + \frac{\alpha_0}{\beta_{n-r}}\Delta_{y_d}(k-n+r) \\ & - \frac{\beta_{n-r-1}}{\beta_{n-r}}\Delta_u(k-1) - \frac{\beta_{n-r-2}}{\beta_{n-r}}\Delta_u(k-2) - \cdots - \frac{\beta_0}{\beta_{n-r}}\Delta_u(k-n+r) \\ & + \underbrace{\frac{1}{\beta_{n-r}}\left(1 - \sum_{i=0}^{n-r}\beta_i + \sum_{i=0}^{n-1}\alpha_i\right) y_d(k)}_{\triangleq s(y_d(k))}.\end{aligned} \tag{2.26}$$

In the above expression, the only non-relative time-dependent term is the last term $s(y_d(k)) = \frac{1}{\beta_{n-r}}\left(1 - \sum_{i=0}^{n-r}\beta_i + \sum_{i=0}^{n-1}\alpha_i\right) y_d(k)$ on the right-hand side. Thus, one may express the control law for achieving exact tracking in terms of the relative terms Δ_{y_d} and Δ_u (and hence apply the difference learning scheme) if and only if $s\big(y_d(k)\big) = 0$. For arbitrary $y_d(k)$, the condition $s\big(y_d(k)\big) = 0$ is equivalent to

$$\frac{\sum_{i=0}^{n-r}\beta_i}{1 + \sum_{i=0}^{n-1}\alpha_i} = 1. \tag{2.27}$$

For system (2.16), the condition in (2.27) is equivalent to the condition that system (2.16) has a unity DC gain, i.e., it achieves zero steady state errors for step reference inputs. □

In our work, we consider a baseline system with an underlying feedback controller (Fig. 2.1). In practice, tracking step reference inputs is a common requirement for controller designs, and this can be often achieved with well-established classical controller design techniques [70]. As we will demonstrate in Sec. 2.6.5, when the baseline system is able to track step reference inputs with sufficiently small errors, the difference learning scheme can significantly reduce the amount of data required for training the DNN module.

In the discussion below, we prove the same result for the MIMO state space formulation for the special case of a position/velocity-like system.

Definition 2.4.2 (Position/Velocity-Like System)**.** *System (2.1) is called position/velocity-like if it has the following properties: (i) the output of the system y is the first m elements of the state vector (i.e., $x_1, ..., x_m$); and (ii) for step reference inputs, the remaining elements of the state vector (i.e., $x_{m+1}, ..., x_n$) are zero at steady state.*

Examples of position/velocity-like systems include but is not limited to mechanical systems with a position-velocity state space (e.g., industrial manipulators). Similar to the SISO transfer function scenario, we identify a necessary condition that allows us to express the control law in (2.7) in relative terms $\Delta_x(k) = x(k) - [y_d(k)^T 0 \cdots 0]^T$, $\Delta_{y_d}(k+r) = y_d(k+r) - y_d(k)$, and $\Delta_u(k) = u(k) - y_d(k)$. In particular, we prove the following lemma.

Lemma 2.4.3 (Difference Learning for MIMO Linear Systems)**.** *Consider a position/velocity-like MIMO system and the DNN-enhancement control architecture in Fig. 2.1. A DNN design based on the state space approach (Remark 2.4.3) and the*

difference learning scheme is able to achieve exact tracking only if the baseline system has zero steady state errors for step reference inputs.

Proof. Suppose by the way of contradiction that the DNN-based approach achieves exact tracking for arbitrary feasible trajectories and the baseline system does not achieve zero steady state error for an arbitrary step reference input $u(k) = a$, where $a \in \mathbb{R}^m$ is a constant vector. Hence, we have

$$y_{ss} = Ku_{ss} = Ka, \tag{2.28}$$

where $K \in \mathbb{R}^{m\times m}$ is a constant non-zero matrix characterizing the DC gains of the system, and $K \neq I_m$ by assumption, where I_m denotes the identity matrix. Note that, when K is a zero matrix, the system has zero DC gain; it can be easily shown that the mapping $\{\Delta_x(k), \Delta_{y_d}(k+r)\} \to \{\Delta_u\}$ is one-to-many and cannot be represented by the DNN module [53]. Next, for the case where K is non-zero, by assumption, the DNN module is able to achieve exact tracking $y_{ss} = y_d(k)$ for an arbitrary step input $y_d(k) = b$, where $b \in \mathbb{R}^m$ is a constant vector. With the difference learning scheme, the inputs to the DNN module are Δ_x and Δ_{y_d} and the output is Δ_u. When exact tracking is achieved, $\Delta_x = 0$ and $\Delta_{y_d} = 0$, while $\Delta_u = c$, where $c \in \mathbb{R}^m$ is a constant corresponding to the bias of the DNN model (i.e., the output of the DNN model when the inputs are zero). At the steady state, the reference of the baseline system is $u_{ss} = b + c$. From (2.28), the system output at the steady state is $y_{ss} = K(b + c)$. Since exact tracking is achieved by assumption, $y_{ss} = b$ and

$$Kc = (I_m - K)b. \tag{2.29}$$

Since K is non-zero and $K \neq I_m$ by assumption, (2.29) implies that the bias of the DNN, c, is correlated with the step input vector b. For a typical feedforward DNN, the bias c is a fixed vector determined from the training algorithm. The dependency of the bias c on the system desired output $y_d(k) = b$ leads to a contradiction. Thus, a DNN module trained with the difference learning scheme cannot achieve exact tracking for a baseline system for which the steady state error for step reference inputs is not zero. □

Note that, in the above discussion, the input and output of the DNN module are taken relative to the current desired output $y_d(k)$. In practice, the input and output of the DNN module can be alternatively taken relative to the current actual output $y(k)$ to additionally compensate for initial tracking errors or disturbances.

Based on the above theoretical results for linear systems, we present the following important insight.

Remark 2.4.5 (Necessary Condition for Applying the Difference Learning Scheme)**.** *In order to reduce the amount of training data, a difference learning scheme can be applied to the input and output selection of the DNN module. However, as shown in the theoretical analysis for the linear system formulations, for the DNN approach with the difference learning scheme to be effective, the baseline system controller needs to be designed such that the system response has zero or sufficiently small steady-state errors for step reference inputs.*

The insight above is motivated from the linear system formulations. Since nonlinear systems can be approximated by a set of piecewise linear/affine systems with arbitrary accuracy [71], it is reasonable to expect that the necessary condition is also required for the nonlinear system (2.2). In Sec. 2.6, we verify this necessary condition for nonlinear systems with quadrotor experiments.

2.5 Simulation Results

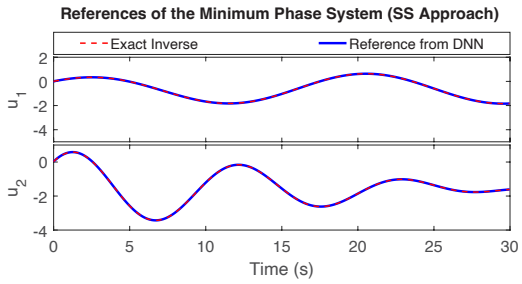
In this section, we illustrate Remarks 2.4.2-2.4.4 by considering two linear MIMO baseline closed-loop systems. The two systems have the same state equation:

$$x(k+1) = \begin{bmatrix} 0.2 & 1 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0 & 0.6 \end{bmatrix} x(k) + \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 0.5 \end{bmatrix} u(k). \tag{2.30}$$

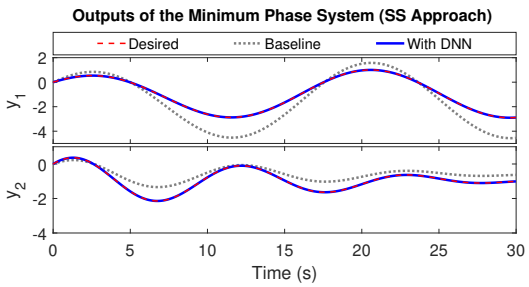
The output equations of the two systems are respectively defined in (2.31) and (2.32) below:

$$y(k) = \begin{bmatrix} 0.35 & 0.35 & 0 \\ 0 & 0 & 0.5 \end{bmatrix} x(k), \tag{2.31}$$

$$y(k) = \begin{bmatrix} -0.35 & 0.35 & 0 \\ 0 & 0 & 0.5 \end{bmatrix} x(k). \tag{2.32}$$



(a) References u of the minimum phase system (2.31) with the state space approach (Remark 2.4.3). The RMS modeling error of the DNN module is approximately 7.8×10^{-5}.



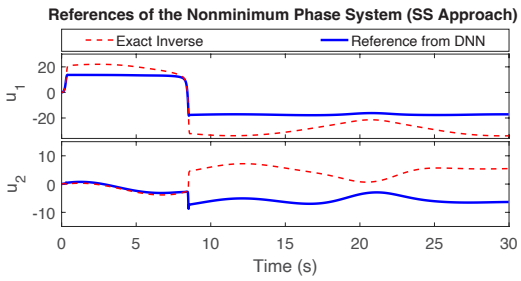
(b) Outputs y of the minimum phase system (2.31). The RMS tracking errors of the baseline system and the DNN-enhanced system are approximately 1.0 and 2.5×10^{-5}, respectively.

Figure 2.2: The references and outputs of the minimum phase closed-loop system (2.31) for a desired trajectory with $y_{d,1}(t) = \sin\left(\frac{4\pi}{33}t\right) + \cos\left(\frac{4\pi}{41}t\right) - 1$ and $y_{d,2}(t) = \sin\left(\frac{4\pi}{23}t\right) + \cos\left(\frac{4\pi}{21}t\right) - 1$. From **(a)**, the DNN module design based on Remark 2.4.3 is able to approximate the system's exact inverse equation (2.7) with high accuracy; from **(b)**, the reference computed by the DNN module is able to compensate for the errors in the baseline system response and approximately achieve exact tracking.

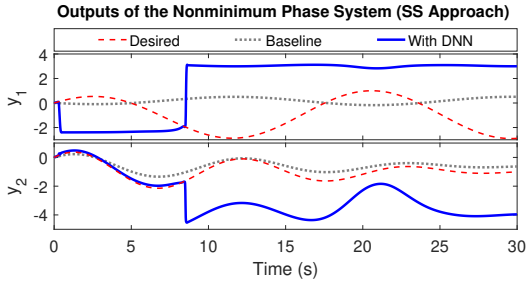
Note that the two systems we consider have identical dynamics and differ only in the output equations and hence the locations of zeros.[1] In particular, system (2.31) has a stable (minimum phase) zero at -0.8, while system (2.32) has an unstable (non-minimum phase) zero at 1.2. Both systems have three stable poles at $\{0.2, 0.5, 0.6\}$ and a vector relative degree of $(1, 1)$.

Upon introducing the DNN architecture and training in Sec. 2.5.1, we first follow the state space approach to select the input of the DNN module and show the necessity of stability of the baseline system zero dynamics (Remark 2.4.2 and Remark 2.4.3) in Sec. 2.5.2. After verifying the first two insights, we illustrate in Sec. 2.5.3 the efficacy of the alternative DNN input selection derived from the transfer function formulation (Remark 2.4.4). For this simulation study and with known system matrices, we can compute (2.7) and use it as the ground truth to assess the proposed approach.

[1] Both system (2.31) and system (2.32) are controllable and observable and are thus minimal state space realizations. The zeros of the MIMO systems are frequencies at which the system matrix of the MIMO systems or the equivalent transfer matrices $H(z)$ of the systems drop rank (see [72] for more details). The locations of zeros (and poles) of the systems can be conveniently verified with the Matlab command `pzmap`.



(a) References u of the nonminimum phase system (2.32) with the state space approach (Remark 2.4.3). The RMS modeling error of the DNN module is approximately 14.5.



(b) Outputs y of the nonminimum phase system (2.32). The RMS tracking errors of the baseline system and the DNN-enhanced system are approximately 2.0 and 4.6, respectively.

Figure 2.3: The reference and outputs of the nonminimum phase closed-loop system (2.32) for the desired trajectory shown in Fig. 2.2. Due to the inherent instability of the nonminimum phase system, the reference u for achieving exact tracking is unbounded [61]. From **(a)**, the DNN module consequently cannot effectively model the exact inverse of system (2.32); from **(b)**, when the necessary condition of achieving stable zero dynamics in Remark 2.4.2 is violated, the DNN inverse learning approach cannot be directly applied to enhance the tracking performance of the baseline system.

2.5.1 Simulation Setup

For comparison purposes, the DNN architecture and training trajectories are identical for all simulation cases presented in this section. Matlab's Neural Network Toolbox is used for implementing the DNN modules. The DNNs are a fully-connected feed-forward networks with two hidden layers; each hidden layer consists of 20 hyperbolic tangent activation units. The training trajectories are 25 sinusoidal trajectories with different combinations of amplitudes and frequencies; the amplitudes range between 1 and 5, and the frequencies range between 0.024 Hz and 1.25 Hz. Note that the architecture of the DNN modules (i.e., the number of hidden layers and the number of neurons) is chosen such that the DNNs have sufficient modeling complexity. In this study, we set aside a part of the training data as the validation set and use a standard validation procedure to ensure that the DNN modules do not overfit or underfit the training data [73]. We further test the sufficiency of the training data by running the DNN-enhanced system on untrained trajectories. In general, the DNN architecture and the training trajectories are not restricted to the particular choices we made, but one should validate the trained DNN module for its generalizability.

As shown in Fig. 2.1, the baseline systems we consider are feedback systems with reference input u and output y. Our goal is to use a DNN module to enhance the

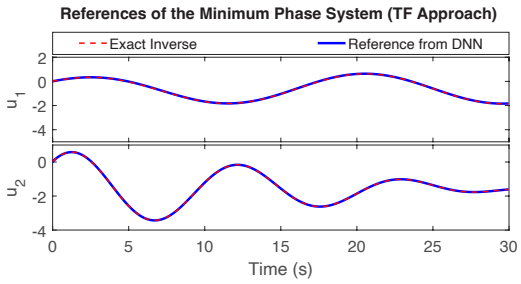
tracking performance of a baseline system by adjusting the reference input u sent to the baseline system. In the training phase, the responses of the baseline systems ($x(k)$, $y(k)$, $u(k)$) are recorded at 70 Hz for constructing the training datasets, which consist of labeled input-output pairs ($\mathcal{I}$, $\mathcal{O}$). The DNN input $\mathcal{I}$ and output $\mathcal{O}$ are defined for each simulation case as follows: In the first set of simulations, the state space approach (Remark 2.4.3) is examined. For both system (2.31) and system (2.32), the input and output of the DNN module are selected as $\mathcal{I}_{ss} = \{x(k), y_{d,1}(k+1), y_{d,2}(k+1)\}$ and $\mathcal{O} = \{u(k)\}$, where $y_{d,i}$ denotes the i-th element of y_d. In the second set of simulations, we focus on the minimum phase system (2.31). Based on the transfer function approach (Remark 2.4.4), the input and output of the DNN module are $\mathcal{I}_{tf} = \{y_{d,1}(k-2:k+1), y_{d,2}(k-2:k+1), u(k-2:k-1)\}$ and $\mathcal{O} = \{u(k)\}$, where ':' abbreviates consecutive discrete-time indexes. Note that, in the construction of the training dataset, the data pairs ($\mathcal{I}$, $\mathcal{O}$) are randomly sampled from the 25 training trajectories with balanced proportions to prevent the model from overfitting a particular frequency.

The Levenberg-Marquardt algorithm is used for training the weight and bias parameters of the DNN module. In the first set of simulations, the training objective is to minimize the mean squared error between the targets $\mathcal{O}$ and the DNN outputs. For the second set of simulations, we additionally include an L_2 regularization term in the training objective function to help the training algorithm eliminate any unnecessary dimensions in the DNN input $\mathcal{I}_{tf}$; the regularization constant is set to 0.005. In the training of each DNN module, 70% of the data is used for optimizing the model parameters and the rest is used for model validations. The generalizability of the DNN modules is further verified by testing the tracking performance of the overall DNN-enhanced system on test trajectories that differ from the training trajectories.

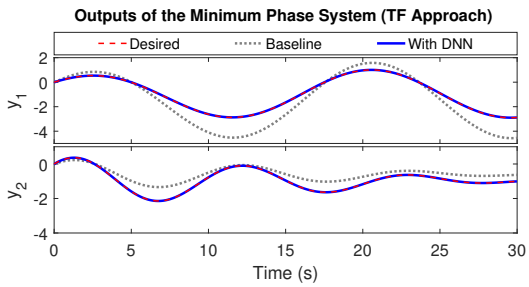
2.5.2 Simulation 1: Illustrations of Underlying Function and Necessary Condition

In this subsection, we illustrate Remark 2.4.2 and Remark 2.4.3 by using the state space approach and comparing the DNN-enhanced performance of the minimum phase system (2.31) and the nonminimum phase system (2.32). For this simulation illustration, the systems' performances are compared on a test trajectory that differs from those in training: $y_{d,1}(t) = \sin\left(\frac{4\pi}{33}t\right) + \cos\left(\frac{4\pi}{41}t\right) - 1$ and $y_{d,2}(t) = \sin\left(\frac{4\pi}{23}t\right) + \cos\left(\frac{4\pi}{21}t\right) - 1$.

The references and outputs of the DNN-enhanced tracking for system (2.31) and



(a) References u of the minimum phase system (2.31) with the transfer function approach (Remark 2.4.4). The RMS modeling error of the DNN module is approximately 1.2×10^{-2}.



(b) Outputs y of the minimum phase system (2.31). The RMS tracking errors of the baseline system and the DNN-enhanced system are approximately 1.0 and 4.2×10^{-3}, respectively.

Figure 2.4: The references and outputs of the minimum phase closed-loop system (2.31) for the desired trajectory shown in Fig. 2.2. From **(a)**, the DNN module design based on the transfer function approach (Remark 2.4.4) is an equivalent approximation of the exact inverse equation (2.7); from **(b)**, as with the state space approach (Fig. 2.2b), exact tracking is approximately achieved with the DNN module design based on the alternative transfer function formulation.

system (2.32) are shown in Fig. 2.2 and Fig. 2.3, respectively. It can be seen from Fig. 2.2a that, by selecting the DNN input as $\mathcal{I} = \{x(k), y_d(k+r)\}$, the DNN is able to effectively generalize the training data collected from the minimum phase system (2.31), and outputs references (blue solid line) that coincide with the reference computed based on the exact inverse in (2.7) (red dashed line). With (2.7) as the ground truth, the RMS modeling error of the DNN module is approximately 7.8×10^{-5}. From Fig. 2.2b, we see that the reference computed by the DNN module compensates for the magnitude errors in the baseline system response (grey dotted line), and leads to approximately exact tracking (blue solid line and red dashed line). On this particular test trajectory, the addition of the DNN module reduces the RMS tracking error from approximately 1.0 to approximately 2.5×10^{-5}. In this simulated setting, the performance of the proposed DNN approach is only limited by the modeling accuracies and numerical precisions. In contrast to the minimum phase case, the reference for achieving exact tracking is unbounded in the nonminimum phase system case (2.32) due to the inherent instabilities of the system inverse dynamics [61]. In the nonminimum phase case reflected in Fig. 2.3, though with the same architecture and training, the DNN module cannot effectively model the exact inverse in (2.7) (Fig. 2.3a) and leads to worse performance as compared to the baseline system (Fig. 2.3b).

2.5.3 Simulation 2: Illustrations of the Transfer Function Approach

In the previous subsection, we showed the effectiveness of the state space approach for designing the DNN module to enhance the tracking performance of the minimum phase system (2.31). In this subsection, we provide a brief discussion on a DNN design based on the equivalent transfer function formulation (Remark 2.4.4).

Fig. 2.4 shows the references and outputs of the system (2.31) with the DNN design based on Remark 2.4.4. From Fig. 2.4a, we can see that similar to the state space approach, the DNN module design based on the transfer function approach (blue solid line) is able to approximate the reference from the exact inverse equation (2.7) (red dashed line). For this particular test trajectory, the RMS modeling error of the DNN is approximately 1.2×10^{-2}. Consequently, as shown in Fig. 2.4b, the output of the DNN-enhanced system (blue solid line) also coincides with the desired trajectory (red dashed line). The RMS tracking error of the DNN-enhanced system is approximately 4.2×10^{-3}. This simulation example shows that the transfer function approach can be equivalently used to enhance the tracking performance of the minimum phase system (2.31) without relying on the knowledge or estimation of the full state as required by the state space approach.

2.6 Quadrotor Experiments

This section presents the results of quadrotor experiments designed to verify the theoretical insights derived in Sec. 2.4. In order to test the effectiveness of the DNN module design based on the provided theoretical insights, we adopt the fly-as-you-draw application setup from [48], where visitors are invited to draw desired trajectories on a mobile device, and the desired trajectories are tracked by a quadrotor vehicle. A demonstration video of the experiments is available at `http://tiny.cc/impromptuTracking`, and the hand drawings used for evaluating the proposed DNN-enhancement trajectory tracking approach are shown in Fig. 2.5.

In the following discussion, we first introduce the experimental setup, the control architecture, and the DNN architecture and training procedures in Sec. 5.6.1. In Sec. 2.6.2 we verify the proposed DNN input-output design and demonstrate the generalizability of the DNN module on the same 30 test trajectories. Upon verifying the proposed DNN input-output design, in Sec. 2.6.4 we show that the performance of the proposed approach can be pushed further by improving the representativeness of

the DNN training dataset. This section is concluded with illustrations of the improved training data efficiency of the difference-learning scheme in Sec. 2.6.5.

Note that, for the convenience of the discussion, we denote the desired trajectory with a subscript d, the references of the baseline system with a subscript r, and the measured states of the quadrotor with a subscript a.

2.6.1 Experiment Setup

The objective of the experiments is to design a control system such that the center of mass of a quadrotor vehicle $\mathbf{p}_a(k)$ tracks desired trajectories $\mathbf{p}_d(k)$ generated based on arbitrary hand-drawings with high accuracy from the first attempt. In the experiments, we use Parrot AR.Drone 2.0 as the testing platform and implement the control algorithm in the Robot Operating System (ROS) environment.

2.6.1.1 Desired Hand-drawn Test Trajectories

The desired trajectories to be tracked by the quadrotor are generated with the fly-as-you-draw application [48]. In particular, in order to generate the desired trajectories, we invite visitors to draw on a mobile device, which gives us sets of discrete points sampled at fixed time intervals along the hand-drawings. The distance between two consecutive points along a hand-drawing is proportional to the drawing speed. Given the set of sampled points from a hand-drawing, the desired position trajectory for the quadrotor is then interpolated using the sampling interval of the position controller. The speed along the desired trajectory is scaled based on a predefined maximum speed $v_{\max}$ and a predefined maximum acceleration $a_{\max}$, which are defined such that the generated trajectory is feasible for the quadrotor to track.

Given a desired trajectory generated from a hand-drawing, we use the root-mean-square (RMS) position tracking error as the measure for evaluating the tracking performance of the quadrotor

$$e_{\text{traj}} = \left(\frac{1}{N} \sum_{k=1}^{N} ||\mathbf{p}_d(k) - \mathbf{p}_a(k)||^2 \right)^{\frac{1}{2}}, \tag{2.33}$$

where N is the number of time steps for which the trajectory is defined.

2.6.1.2 Control Architecture

The quadrotor vehicle has 12 states: translational positions $\mathbf{p} = (x, y, z)$, translational velocities $\mathbf{v} = (\dot{x}, \dot{y}, \dot{z})$, attitudes $\boldsymbol{\theta} = (\phi, \theta, \psi)$, and rotational velocities $\boldsymbol{\omega} = (p, q, r)$. The baseline controller of the quadrotor vehicle consists of *(i)* an off-board position controller that receives the reference positions and velocities ($\mathbf{p}_r$ and $\mathbf{v}_r$) and outputs the desired roll angle, pitch angle, yaw rate and z-velocity commands ($\phi_{\text{cmd}}, \theta_{\text{cmd}}, r_{\text{cmd}},$ and $\dot{z}_{\text{cmd}}$) at 70 Hz; and *(ii)* an on-board attitude controller that adjusts motor thrusts based on the roll angle, pitch angle, yaw rate and z-velocity commands at 200 Hz. Of particular interest is the off-board position controller, which consists of a nonlinear transformation and PD control; from the internal model principle, it is known that this type of controller cannot be tuned to achieve perfect tracking for arbitrary desired reference frequencies [24]. We aim to enhance the baseline position controller with our proposed DNN add-on module, which models the output of the inverse dynamics of the baseline control system. In the experiments, we introduce a DNN module design based on Remark 2.4.3 to adjust the position reference $\mathbf{p}_r$ and the velocity reference $\mathbf{v}_r$ sent to the baseline controller, and compare the tracking performance of the DNN-enhanced controller against that of the baseline controller. As in previous work [48] and for the robustness of implementation against instability, the DNN-loop in the experiments runs at 7 Hz, which is 10 times slower than the baseline controller.

In the implementation of our baseline position controller and the DNN module, the states of the quadrotor are estimated based on a Vicon motion capture system running at 200 Hz. The onboard attitude controller of the ARDrone relies only on onboard sensing [74], and the onboard attitude estimation and control modules together are a black box for our baseline position controller and DNN module implementation.

2.6.1.3 Neural Network Architecture and Training

For comparison purposes, the DNN modules used in the experiments have the same architecture and training procedure as in [48]. The DNN modules are fully connected and have four hidden layers of 128 ReLU neurons, which are 1-Lipschitz functions; the Python TensorFlow library is used for implementing the DNN module.

Figure 2.5: Illustrations of 30 hand-drawn trajectories for testing the DNN-enhancement approach [48].

To construct training datasets, we record the response of the robot baseline system on one or multiple training trajectories. For the results in Sec. 2.6.2-2.6.3, in order to fairly compare the proposed approach with [48], the DNN modules are trained on a 400-second 3-dimensional sinusoidal trajectory similar to the trajectory used in [48]; for the results in Sec. 2.6.4-2.6.5, to explore the impact of training dataset choices on the performance of the DNN module, we further incorporated 30 arbitrary hand-drawn trajectories in the training process (more details are included in Sec. 2.6.4). For each training trajectory, the state, the input, and the output of the robot baseline system are recorded at a sampling rate of 7 Hz. The recorded data is then used to construct paired input-output datasets based on the DNN module input-output selection (2.34)-(2.35) or (2.36)-(2.37). Of all the training data collected from the baseline system, 90% is randomly selected for training and the remaining is used for validation. The generalizability of the DNN modules for enhancing the tracking performance on arbitrary trajectories is tested in closed-loop with the baseline system on 30 new hand-drawn trajectories.

The training loss function is the squared error between the DNN output and the labeled output in the training dataset. The Adam optimizer [75] is used for optimizing the weight parameters of the DNN. A dropout rate of 0.5 is used to improve the generalizability of the DNN to unseen inputs [76].

We note that, in our setup, the training of the DNN module is a standard supervised training problem, and the hyperparameters of the DNN module (e.g., the width and depth of the network) could be selected using a validation set. Based on our experience, the performance of the DNN module is not very sensitive to the hyperparameter selection, but achieving a satisfactory performance requires a sufficiently rich training dataset. We further explore this aspect in Sec.2.6.4 and Chapter 4.

2.6.2 Experiment 1: DNN Input-Output Design

Through experimental trial-and-error, [48] found that a DNN module with the following input and output can effectively improve the performance of the baseline system for tracking arbitrary hand-drawn trajectories:

$$\begin{aligned}\mathcal{I}_1 = \{&\mathbf{p}_d(k+4) - \mathbf{p}_a(k),\ \mathbf{p}_d(k+6) - \mathbf{p}_a(k),\ \mathbf{v}_a(k),\ \mathbf{v}_d(k+4),\\ &\mathbf{v}_d(k+6),\ \boldsymbol{\theta}_a(k),\ \boldsymbol{\theta}_d(k+4),\ \boldsymbol{\theta}_d(k+6),\ \boldsymbol{\omega}_a(k),\ \boldsymbol{\omega}_d(k+4),\\ &\boldsymbol{\omega}_d(k+6),\ \ddot{z}_a(k),\ \ddot{z}_d(k+4),\ \ddot{z}_d(k+6)\}\end{aligned} \tag{2.34}$$

$$\mathcal{O}_1 = \{\mathbf{p}_r(k) - \mathbf{p}_d(k),\ \mathbf{v}_r(k) - \mathbf{v}_d(k)\} \tag{2.35}$$

On the 30 hand-drawn trajectories shown in Fig. 2.5, the average RMS error reduction achieved by the DNN module is approximately 43%. In order to verify Remark 2.4.3, we repeat the 30 test trajectories from [48] with a DNN module design based on the proposed input-output selection and compare the improved tracking performance with that achieved in [48]. Note that we repeated the experiments in [48] on the quadrotors used for this work for comparability.

In order to apply our insights, we first performed simple step response experiments and identified the following properties of the baseline system:

(P1) the responses of the baseline system are approximately decoupled in the x-, y-, and z-direction;

(P2) the relative degrees of the baseline system in the x-, y-, and z- direction are 4, 4, and 3, respectively; and

(P3) zero steady state error for step reference inputs is approximately achieved in the three directions.

Given properties (P1)-(P3), we assume decoupled dynamics in the x-, y-, and z-direction and apply Remark 2.4.3 with the difference learning scheme to obtain the following input and output selection of the DNN module:

$$\begin{aligned}\mathcal{I}_2 = \{&x_d(k+4) - x_a(k),\ y_d(k+4) - y_a(k),\ z_d(k+3) - z_a(k),\ \dot{x}_d(k+3) - \dot{x}_a(k),\\ &\dot{y}_d(k+3) - \dot{y}_a(k),\ \dot{z}_d(k+2) - \dot{z}_a(k),\ \boldsymbol{\theta}_a(k),\ \boldsymbol{\omega}_a(k)\}\end{aligned} \tag{2.36}$$

$$\mathcal{O}_2 = \{\mathbf{p}_r(k) - \mathbf{p}_a(k),\ \mathbf{v}_r(k) - \mathbf{v}_a(k)\} \tag{2.37}$$

We note two differences between the DNN from [48] and the proposed DNN design based on Remark 2.4.3. The first is the DNN input selection. In comparison with the DNN from [48], which has 36 inputs ($\#\mathcal{I}_1 = 36$, where $\#$ denotes cardinality), the DNN design based on Remark 2.4.3 has only 12 inputs ($\#\mathcal{I}_2 = 12$). Based on the inverse-dynamics formulation, the input selection $\mathcal{I}_2$ represents the necessary and sufficient inputs that allow the DNN add-on module to achieve enhanced tracking performance. Another difference is in the application of the difference learning scheme for the two DNN designs. In particular, for the DNN design from [48], the position elements in the input $\mathcal{I}_1$ are taken relative to the actual output values (subscripted with a), and the position elements in the output $\mathcal{O}_1$ are taken relative to the desired output values (subscripted with d). For the proposed DNN design based

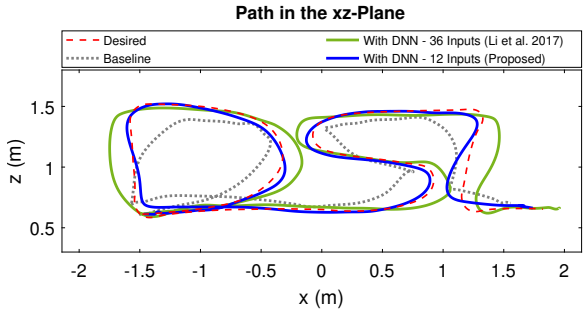


Figure 2.6: A comparison of the tracking performance enhancements between the DNN module from [48] and the DNN module design based on Remark 2.4.3 for a hand-drawn test trajectory (Traj. 24 in Fig. 2.5). On this test trajectory, the RMS tracking error of the baseline system is approximately 0.41 m. The RMS tracking errors of the baseline system enhanced by the DNN module from [48] and the proposed DNN module design based on Remark 2.4.3 are 0.23 m and 0.14 m respectively, which correspond to 45% and 67% error reductions.

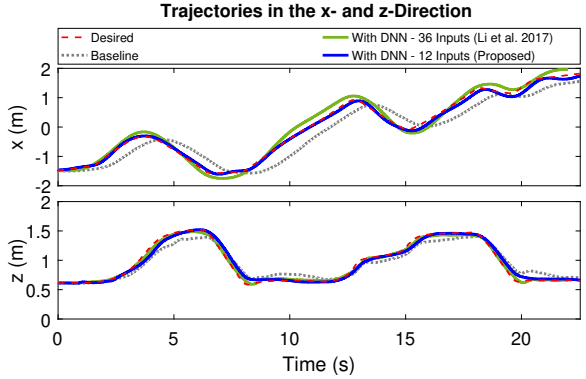


Figure 2.7: A comparison of the x- and z-position trajectories for a hand-drawn test trajectory (corresponding to Fig. 2.6). From the plots, the DNN module from [48] and the DNN module trained based on Remark 2.4.3 both tend to correct the delays and magnitude errors of the baseline system response. When compared to the DNN from [48], the proposed DNN design based on Remark 2.4.3 has two thirds fewer inputs while achieving better performance enhancements.

on Remark 2.4.3, the relative terms in the input $\mathcal{I}_2$ and output $\mathcal{O}_2$ are consistently taken with respect to the actual values (subscripted with a). Based on the theoretical discussions of Remark 2.4.5, we expect the consistency of the relative terms in the proposed design would further improve the capability of the DNN module in correcting for any deviations from the desired trajectories.

We first present the performance comparison between the DNN from [48] and the proposed DNN on one of the test trajectories (Figs. 2.6, 2.7, and 2.8). We then summarize the comparison between the two DNN designs on 30 hand-drawn test

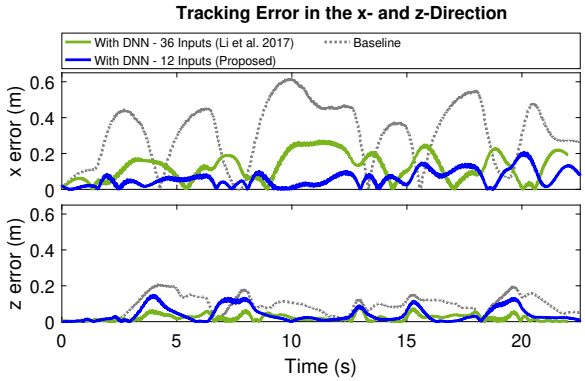


Figure 2.8: The tracking errors of the x- and z-position ($|x(k) - x_d(k)|$ and $|z(k) - z_d(k)|$) corresponding to Fig. 2.7. The DNN module from [48] and the DNN module trained based on Remark 2.4.3 both effectively reduce the peak tracking errors of the baseline system. The peak errors for the baseline system in the x- and z-direction are approximately 0.62 m and 0.21 m, respectively. For the DNN design from [48], the peak tracking errors in the x- and z-direction are reduced to approximately 0.27 m and 0.09 m, while for the proposed DNN, the peak tracking errors in the x- and z-direction are reduced to approximately 0.21 m and 0.15 m.

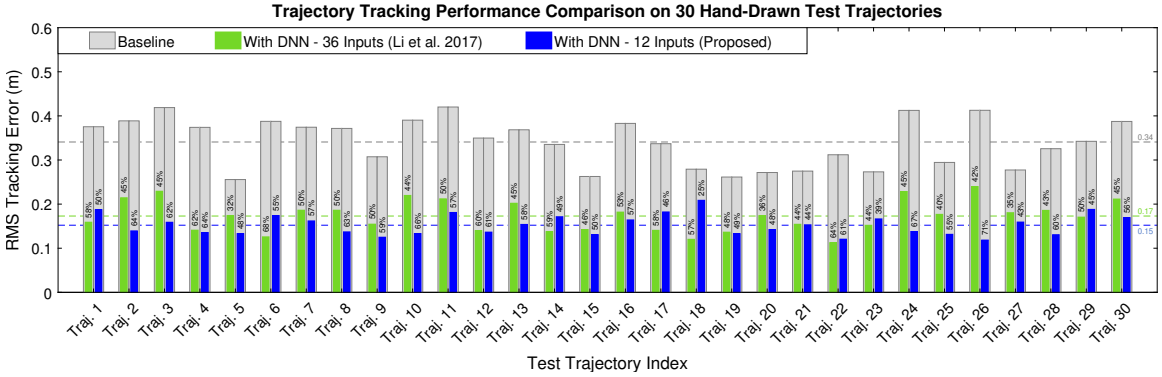


Figure 2.9: Comparisons of the tracking performance enhancements between the DNN module from [48] (with 36 inputs) and the proposed DNN module design based on Remark 2.4.3 (with 12 inputs). In the two sets of experiments, the percentage of the RMS tracking error reductions achieved by the DNN module are indicated above the corresponding bars; the mean RMS error over the 30 trajectories are indicated by the horizontal dashed lines. Despite having two thirds fewer inputs, the proposed DNN design based on Remark 2.4.3 yields a performance comparable to the DNN from [48]. On the 30 test trajectories, the average RMS error reduction is 49% for the DNN from [48] and 54% for the proposed DNN design based on Remark 2.4.3.

trajectories (Fig. 2.9). The test trajectories are generated based on the procedure described in Sec. 2.6.1.1. The maximum speed and maximum acceleration of the trajectories are $v_{\max} = 0.6$ m/s and $a_{\max} = 2.0$ m/s^2, respectively. Note that, for the experiments, the DNN modules are trained on a 400-second 3-dimensional sinusoidal

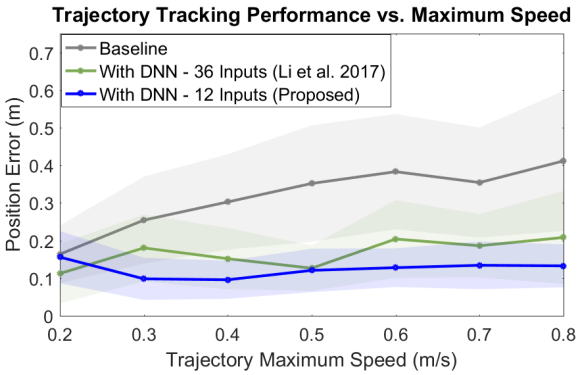


Figure 2.10: The tracking performance of *(i)* the baseline system, *(ii)* the system enhanced by the DNN module design from [48], and *(iii)* the system enhanced by the DNN module design based on Remark 2.4.3 as the trajectory speed increases. The solid lines and the shaded regions in the plot correspond to the mean and the standard deviation of the position error $\mathbf{p}_d - \mathbf{p}_a$ along the test trajectories. In contrast, the tracking error of the baseline system increases significantly with trajectory speed, while the tracking error of the system enhanced with the proposed DNN remains at a lower constant level. The average RMS tracking error over the presented trials is 0.35 m for the baseline system, 0.19 m for the system with the DNN module design from [48], and 0.14 m for the system with the DNN design based on Remark 2.4.3.

trajectory similar to that used in [48]. In order to establish a fair comparison, we use the same DNN architecture, training data, and training algorithm for the DNN design based on [48] and the DNN design based on Remark 2.4.3; the only difference between the two DNNs is the input-output selection.

From Fig. 2.6 and Fig. 2.7, it can be seen that both the DNN from [48] (green solid line) and the DNN design based on Remark 2.4.3 (blue solid line) are able to reduce the time delays and magnitude errors of the baseline system tracking response (grey dotted line) and lead to quadrotor tracking paths that are closer to the desired hand-drawing (red dashed line). On this test trajectory, the RMS tracking error reduction achieved by the DNN from [48] and the proposed DNN are 45% and 67%, respectively. The trajectory tracking error comparison depicted in Fig. 2.8 shows that the proposed DNN design based on Remark 2.4.3 achieves similar error reductions to the DNN design from [48] while having far fewer inputs.

Fig. 2.9 summarizes the performance comparison between the two DNN modules on the 30 hand-drawn test trajectories studied in [48] (see Fig. 2.5). The plot shows that the proposed DNN module design based on Remark 2.4.3 (blue bars) leads to similar tracking performance as the DNN module from [48] (green bars). On the 30 test trajectories, the mean RMS error of the baseline system enhanced by the DNN from [48] is approximately 0.17 m, and that of the baseline system enhanced by the

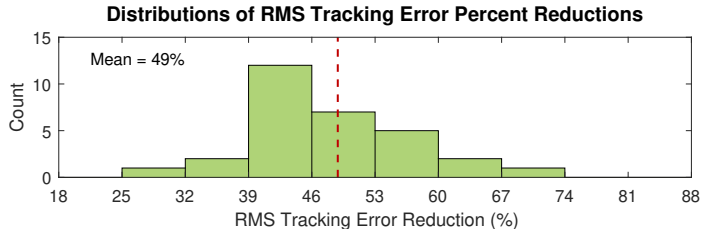
proposed DNN is approximately 0.15 m. The corresponding average tracking error reduction achieved by the DNN from [48] and that design based on Remark 2.4.3 are 49% and 54%, respectively.

From this set of experiments, we verify Remark 2.4.3 on the proposed DNN input selection. Although the input dimension is reduced by two thirds as compared with the DNN from [48], the DNN module design based on the derived theoretical insight can effectively enhance the quadrotor's baseline system performance. The comparison with the results from [48] further validates the generalizability of the proposed DNN for tracking arbitrary untrained trajectories impromptu.

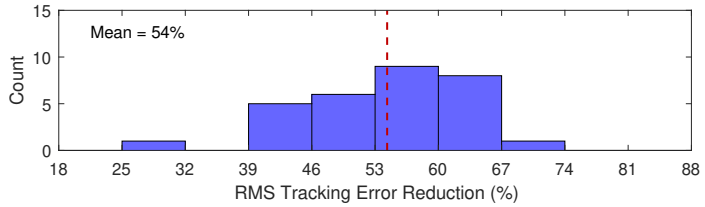
2.6.3 Experiment 2: Generalization to Different Trajectory Speeds

In this subsection, we examine the performance of the baseline system and the DNN-enhanced systems for different operating speeds. In particular, we use the trajectory shown in Fig. 2.6 as the test trajectory and scale the time-parameterized trajectory based on a set of specified maximum speeds $v_{\max} = \{0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8\}$ m/s. Fig. 2.10 summarizes the performance of *(i)* the baseline system, *(ii)* the system enhanced with the DNN module design from [48], and *(iii)* the system enhanced with the DNN module design based on our Remark 2.4.3. As can be seen from the plot, the tracking performance of the baseline system (grey) degrades quickly as the speed of the test trajectory increases. In contrast, the tracking errors of the systems with the DNN modules (green and blue) remain relatively at a lower constant level. The average RMS tracking error over the trials presented in Fig. 2.10 is 0.35 m for the baseline system, 0.19 m for the system enhanced with the DNN design from [48], and 0.14 m for the system enhanced with the DNN design based on Remark 2.4.3.

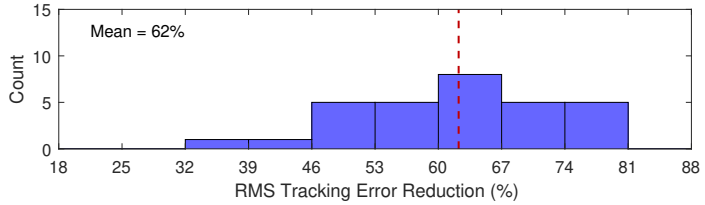
As we discussed in Sec. 2.4.2, the DNN module in our framework represents the inverse of the baseline system and, theoretically, establishes an identity map from the desired output y_d to the actual output of the system y. In the quadrotor experiments, we demonstrate the efficacy of the proposed DNN module approach for reducing the tracking error of the baseline system across multiple operating speeds. We note that the effectiveness of the DNN module generally relies on having training data that sufficiently covers the range of operating speeds of interest.



(a) DNN trained on 400-s sinusoidal trajectory [48]



(b) Proposed DNN trained on 400-s sinusoidal trajectory



(c) Proposed DNN trained on 400-s sinusoidal trajectory and 30 hand-drawn trajectories

Figure 2.11: A comparison of the tracking error reduction achieved by **(a)** the DNN used in [48] and trained on a 400-second sinusoidal trajectory, **(b)** the proposed DNN designed based on Remark 2.4.3 and trained on the 400-second sinusoidal trajectory, and **(c)** the proposed DNN designed based on Remark 2.4.3 and trained on the 400-second sinusoidal trajectory and 30 additional hand-drawn trajectories. Note that, for the last case **(c)**, the 30 additional hand-drawn trajectories used for training are different from the 30 test trajectories. The mean percent error reduction for each distribution is indicated by the vertical dashed line.

2.6.4 Experiment 3: DNN Training Dataset

In the previous set of experiments, the performance of the DNN from [48] and the proposed DNN are compared on the basis of training on 400-second sinusoidal trajectories. These trajectories have gradually increasing amplitudes but fixed frequencies in the x-, y-, and z-direction [48]. In this subsection, we show that the performance enhancement achieved by the proposed DNN design can be further improved with a richer training dataset. In particular, we compare two training datasets constructed from the baseline system responses to different training trajectories:

- Training Dataset 1 based on a 400-second sinusoidal training trajectory, and
- Training Dataset 2 based on the 400-second sinusoidal training trajectory from Training Dataset 1 and 30 additional hand-drawn trajectories.

Note that the 30 hand-drawn trajectories in Training Dataset 2 are different from the 30 trajectories (Fig. 2.5) for evaluating the performance of the DNNs. By adding hand-drawn trajectories to the DNN training, we expect to increase the similarity between the DNN inputs encountered at the training time and the test time. In particular, we expect the arbitrary training hand-drawn trajectories to capture a richer set of cases (e.g., sharp edges) that are nontrivial to define with analytical expressions. By having a more representative training dataset, we can then further reduce the generalization error of the DNN module for impromptu tracking performance enhancements.

Fig. 2.11 shows the performance comparison of three DNN-enhanced systems on the 30 test hand-drawn trajectories (Fig. 2.5). From the previous subsection, we show that, on average, the DNN with the proposed inputs (middle histogram in Fig. 2.11) leads to better performance as compared with the DNN from [48] (top histogram in Fig. 2.11). When comparing the proposed DNN trained with Training Dataset 1 (middle histogram in Fig. 2.11) and Training Dataset 2 (bottom histogram in Fig. 2.11), we see that the inclusion of the additional hand-drawn trajectories in training further improves the performance of the DNN-enhanced system in tracking arbitrary hand-drawn trajectories. Overall, the proposed DNN trained with Training Dataset 2 increases the average RMS tracking error reduction by 8% as compared with the proposed DNN trained with Training Dataset 1.

We note that, as similarly discussed in [49], the learning performance of the DNN approach is limited by the non-repeatable or stochastic error in the baseline system. More explicitly, the non-repeatable or stochastic error corresponds to the variations we see in the baseline system output when an identical reference is given to the baseline system multiple times. In our experimental setup, one primary source of the stochastic error is the noise in the onboard IMU- and camera-based attitude estimation and control, which we do not have direct access to. Other sources of the stochastic error also include the process noise present in the quadrotor system. In our experiments, the proposed DNN module trained with Training Dataset 2 reduces the average tracking error of the quadrotor on the 30 hand-drawn trajectories to approximately 0.07 m to 0.15 m. This performance is comparable to the standard deviation of the position error of the quadrotor at hover, which is an estimate of the inherent noise in the system and serves as a lower bound on the achievable tracking

accuracy. We expect our DNN to achieve lower tracking errors if the response of the baseline system could be made more repeatable.

2.6.5 Experiment 4: Difference Learning

In Sec. 2.4.5, we theoretically showed that, in order to apply the difference learning scheme to improve the data efficiency of the DNN training, the baseline system needs to achieve zero steady state error for step reference inputs. In this subsection, we first illustrate the necessity of the condition by applying the difference learning scheme to DNN modules to enhance *(i)* the original baseline system where zero steady state error for step reference inputs is achieved, and *(ii)* a modified baseline system where the necessary condition is not achieved. In the experiment, the modified baseline system is obtained by multiplying the reference signals z_r sent to the original baseline system by a factor of 0.5. The baseline and DNN-enhanced tracking performance for the two systems are shown in Fig. 2.12. The plots show that for the original baseline system (bottom panel), where zero steady state error for step reference inputs is achieved, the DNN with the difference learning scheme is able to effectively enhance the tracking performance of the baseline system. However, as expected from Remark 2.4.5, for the modified baseline system (top panel), where the zero steady state error condition is not satisfied, the DNN trained with the difference learning scheme only partially compensates for the magnitude error and the bias of the modified baseline system.

In order to evaluate the effectiveness of the proposed difference learning scheme for improving the training data efficiency, we next compare a DNN module trained with and a DNN trained without the difference learning scheme. Fig. 2.13 shows a comparison of the DNN modules trained with (blue) and without (red) the difference learning scheme for enhancing the tracking performance of the quadrotor baseline system where zero steady state error for step reference inputs is achieved. In the plot, the RMS tracking errors of the DNN-enhanced systems are compared as the amount of training data varies. Note that, in order to prevent overfitting, the training datasets are randomly sampled from a large training dataset (Training Dataset 2). Here, we use Traj. 24 (Fig. 2.5) as the test trajectory for evaluating the performance of the DNN-enhanced systems. Fig. 2.13 shows that, for the DNN without the difference learning scheme (red), the RMS tracking increases quickly as the amount of training data reduces. In contrast, the performance of the DNN trained with the difference learning scheme (blue) drops more gradually as the amount of training data decreases. The DNN trained with the difference learning scheme reaches the best performance of the

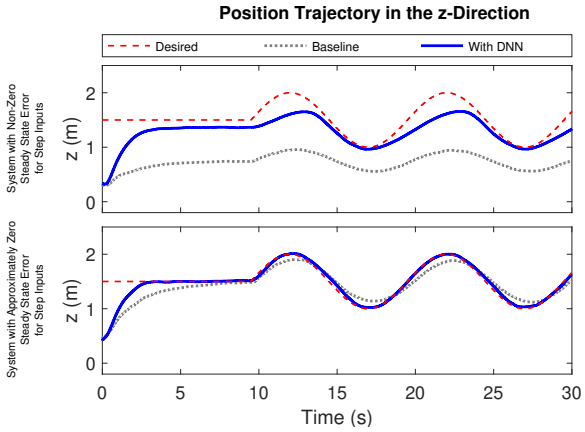


Figure 2.12: A comparison of the difference learning scheme as applied on: *(i)* a baseline system for which zero steady state error for step reference inputs is not achieved (top); and *(ii)* a baseline system for which zero steady state error for step reference inputs is achieved (bottom). When the necessary condition of having a baseline system that achieves zero steady state error for step reference inputs is not satisfied (see Remark 2.4.5), the DNN trained with the difference learning scheme cannot effectively compensate for the errors of the baseline system response.

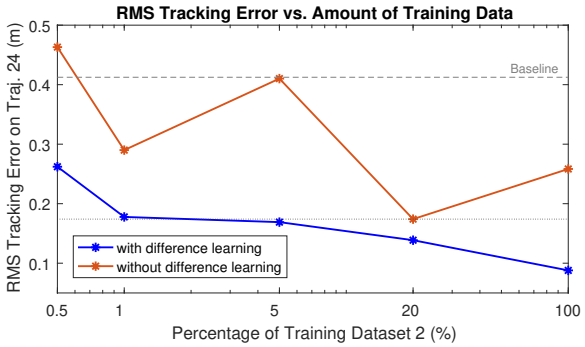


Figure 2.13: A comparison of the RMS tracking error versus the amount of data for training the DNNs with (blue) and without (red) the difference learning scheme. The horizontal axis shows the proportion of randomly selected data from Training Dataset 2 described in Sec. 2.6.4; the vertical axis shows the RMS error on Traj. 24 with the DNN-enhanced system (see Fig. 2.5). The plot shows that the DNN trained with the difference learning scheme is able to reach the best observed performance of the DNN trained without the difference learning scheme (indicated by the grey dotted line) with approximately 15 times less training data. Note that the RMS tracking error corresponding to the baseline system is shown as a grey dashed line for reference.

DNN without the difference learning scheme (grey dotted line), with approximately 15 times less data.

2.7 Conclusions

In this chapter, we present theoretical and experimental studies of a DNN-based approach for enhancing the tracking performance of black-box control systems for arbitrary feasible trajectories. We considered a MIMO, possibly nonlinear, system as our starting point. In order to achieve an identity map from the desired output to the actual output, we established that the DNN module in the proposed control architecture should approximate the output equation of the inverse dynamics of the baseline system. Due to the association with system inversion, the effectiveness of the proposed approach relies on two necessary conditions that the baseline system has *(i)* a well-defined vector relative degree and *(ii)* stable zero dynamics. Second, for the systems satisfying these two necessary conditions, we identified the necessary and sufficient inputs of the DNN module. Third, we verified the insights by repeating the quadrotor experiments in [48]. In particular, we showed that with the proposed DNN input selection, the DNN input dimension is reduced by two thirds while achieving similar or better performance on the 30 hand-drawn trajectories in [48]. Moreover, in contrast to the quadrotor baseline controller, for which the tracking error increased with the trajectory speed, the tracking errors of the DNN-enhanced systems remained small as the trajectory became more aggressive. By using a richer training dataset, we also showed that the proposed DNN module reduced RMS error by approximately 62% on the average of the 30 testing hand-drawn trajectories. Fourth, using an argument similar to the small gain theorem, we proved that, for systems with stable zero dynamics, the overall DNN-enhanced control system is input-to-state stable if the DNN modeling error is sufficiently small. Fifth, we explored via both theory and experiments the effectiveness of the difference learning scheme for improving the efficiency of the training of the DNNs in the proposed approach. In particular, we derived a necessary condition for the effectiveness of the difference learning approach, and verified this condition via experiments. For the quadrotor impromptu tracking experiments, we showed that the DNN trained with the difference learning scheme is able to achieve comparable tracking performance of a DNN module trained without the difference learning scheme with approximately 15 times less data.

Chapter 3

Learning an Approximate Inverse to Enhance Non-minimum Phase Systems

3.1 Introduction

In the previous chapter, we introduced an add-on inverse dynamics learning approach to enhance the tracking performance of robot systems. One necessary condition for the approach to be effective is that the baseline system needs to be minimum phase. For many practical problems ranging from flexible robot arm end-effector tracking [77] to car backward driving [61] and to aircraft control [78], the input-output dynamics are non-minimum phase (i.e., the inverse dynamics are inherently unstable). The non-minimum phase nature poses challenges in classical control design [61] and prohibits the direct application of inversion-based approaches.

In this chapter, we again consider the task of *impromptu tracking* [48] and extend the inverse dynamics learning approach to non-minimum phase systems. In particular, informed by control theory, we (1) propose an approach for learning a stable approximate inverse model for a non-minimum phase system, (2) prove stability of the learning-enhanced architecture, and (3) provide theoretical insights on the inverse approximation utilized by the learning module to achieve performance enhancement, and (4) experimentally demonstrate the efficacy of the proposed approach for non-linear systems on *(i)* an inverted pendulum on a cart system and *(ii)* a modified non-minimum phase quadrotor system. For the quadrotor experiments, the generalizability of the learned inverse is verified using arbitrary, hand-drawn trajectories.

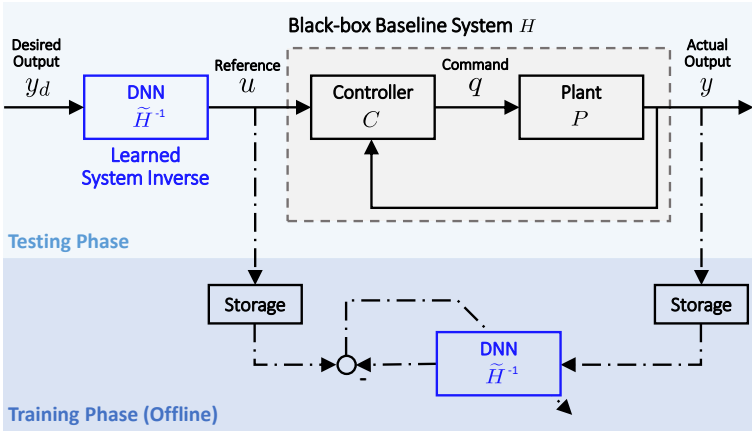


Figure 3.1: An illustration of the proposed DNN-enhanced control architecture for output trajectory tracking. A stable baseline control system is treated as a black box and a DNN module is pre-cascaded to the baseline system to adjust reference signals to improve the tracking performance.

3.2 Related Work

In the literature, various model-based inversion approaches have been proposed to resolve the instability issue associated with the system inverse of non-minimum phase systems. These approaches are based on *(i)* pre-actuation [27] or *(ii)* inverse approximation [79, 80]. In the pre-actuation approach, first proposed in [27], a bounded input is ensured by pre-loading the system state to a desired initial state designed for the particular desired trajectory. Though exact tracking can be achieved with bounded input signals, the solutions are trajectory-specific and require significant setup time in order to reach the desired initial condition [81]. On the other hand, in the inverse approximation approaches, stability of the inverse is ensured by replacing the unstable components of the inverse dynamics with a stable approximation that is capable of achieving precise tracking (see [80, 79] and the references therein). As compared with the pre-actuation approaches, the approximate inversion approaches are more robust against modeling errors and consequent instability issues. Moreover, since the inversion is system-specific, the approximate inversion approaches can be more easily generalized to impromptu tracking tasks. However, due to the model-based nature of both approaches, the effectiveness depends on sufficiently accurate system models. This limitation motivates the investigation of learning techniques, which leverage data to improve the performance of model-based approaches.

For *minimum phase systems*, different inverse dynamics learning approaches have been studied. In the previous chapter, a DNN-based control architecture (Fig. 2.1)

was proposed to enhance the tracking performance of minimum phase black-box systems (i.e., systems whose dynamical models are not available or not sufficiently accurate). With experiments on quadrotors, it was shown that the proposed approach led to an average of 62% tracking error reduction over 30 arbitrary, hand-drawn trajectories, as compared to the baseline controller. In addition to our previous work, the potential of utilizing inverse learning for high-accuracy tracking has been demonstrated using different robotic platforms and learning techniques (e.g., Gaussian processes (GPs) and locally weighted projection regression (LWPR)), see for instance [82, 83, 84]. Nevertheless, the applicability of these inversion-based learning approaches to non-minimum phase systems has not been studied, and systematically extending inverse dynamics learning schemes to non-minimum phase systems is still an open problem.

Previously, for *non-minimum phase systems*, a DNN-based adaptive feedback error learning approach has been proposed to learn an inverse of the open-loop plant for enhancing tracking [85, 86]. In this approach, the DNN training requires the plant or a good model of the plant in place, which may not always be desired in the initial training phase or available in practice. Moreover, similar to the adaptive inverse learning approaches discussed in Chapter 2, this approach is more susceptible to instability issues, especially when the DNN is not well-initialized [55].

In this chapter, we present a learning-based approach that constructs an approximate inverse of a non-minimum phase, feedback-stabilized system based only on input-output data. We show the connection between the proposed learning approach and a common model-based approximate inversion approach for linear systems [79, 80]. The proposed approach shares the same core concept as the model-based approach; yet, without requiring a detailed model, the proposed approach leads to better performance and is applicable to nonlinear systems.

3.3 Problem Formulation

We aim to provide an inversion-based learning approach for enhancing the tracking performance of non-minimum phase systems in impromptu tracking tasks. The proposed approach should satisfy the following objectives:

(O1) the overall system, including the learning module, is input-to-output stable [87];

(O2) the learning module relies only on the input-output data rather than a system model;

(O3) with the learning module, the root-mean-square (RMS) tracking error is reduced for impromptu tracking tasks, compared to the baseline system.

We consider the inversion-based learning architecture shown in Fig. 3.1, which consists of a baseline system and a pre-cascaded, learned system inverse module enhancing the tracking performance via modifying the reference signal u. In the training phase, the input-output data, u and y, generated from the baseline system is stored and used to construct a training dataset that typically has y and u at selected time steps as the labeled inputs and u at the current time step as the labeled output. When later using the trained module in the testing phase, the desired trajectory y_d is given to the learned inverse model as input (in place of y) to compute a reference u that is sent to the baseline system.

The considered architecture is different from typical inversion-based feedforward architectures where the inverse of the open-loop plant P is used and the output signal from the inverse is directly applied to the plant [82, 86]. By learning the inverse of a stabilized baseline system, the proposed architecture decouples the performance enhancement problem from the plant stabilization problem, which simplifies the design, analysis, and practical implementation.

We first motivate our proposed approach by analyzing linear time-invariant (LTI), single-input-single-output (SISO) systems and then extend our discussion to nonlinear SISO systems. For linear systems, we represent the baseline feedback control system by the transfer function

$$H(z) = \frac{Y(z)}{U(z)} = \frac{N(z)}{D(z)} = \frac{1 + \sum_{i=1}^{n-r} \alpha_i z^i}{\sum_{i=0}^{n} \beta_i z^i}, \tag{3.1}$$

where $U(z)$ and $Y(z)$ are the z-transforms of the input and output of the system, $N(z)$ and $D(z)$ are the numerator and denominator polynomials, n is the order of the system, r is the relative degree of the system, and $\alpha_i, \beta_i \in \mathbb{R}$ are scalar constants. For nonlinear systems, we consider the control affine nonlinear system:

$$\begin{aligned} x(k+1) &= f(x(k)) + g(x(k))\, u(k), \\ y(k) &= h(x(k)), \end{aligned} \tag{3.2}$$

where $k \in \mathbb{Z}_{\geq 0}$ is the discrete time index, $x \in \mathbb{R}^n$ is the state, $u \in \mathbb{R}$ is the input, $y \in \mathbb{R}$ is the output, and $f(\cdot)$, $g(\cdot)$, $h(\cdot)$ are nonlinear smooth functions (i.e., functions for which all orders of differentiation exist and are continuous).

In deriving a solution for our problem, we assume:

(A1) the underlying plant is stabilizable and the baseline system is stable;

(A2) at any time instant k, the current and future values of the desired trajectory are known up to time $k+n$, where n is the order of the baseline system;

(A3) the learned inverse dynamics module are feedforward neural networks (FNNs) with (A3a) finite weights and biases and (A3b) continuous activation functions $\sigma(\cdot)$.

Assumptions (A1) through (A3) are not restrictive in practice. For (A1), well-developed control methods, including model-free controllers (e.g., PID controllers), can be used to stabilize a system even in the absence of a dynamical model. It should be noted that, although it is possible to design a stabilizing controller using standard control techniques, achieving high-accuracy tracking on arbitrary feasible trajectories remains to be challenging. Our goal is to employ an inversion-based approach to enhance the impromptu tracking performance of the baseline control system. For (A2), a preview of n time steps of the desired trajectory is typically available, and this assumption does not prevent combinations with on-line trajectory generation and adaptation algorithms. Moreover, for (A3), even though we use FNNs in this chapter, the proposed approach can be potentially realized with other nonlinear regression techniques (e.g., GPs and LWPR). Assumption A3a can always be satisfied with standard DNN training algorithms, and assumption A3b holds for all common DNN activation functions (e.g., rectified linear units (ReLU), tanh, and sigmoid).

3.4 Non-minimum Phase System Inverse Learning

For non-minimum phase systems, one approach to resolve the instability issue in inversion-based approaches is to utilize stable inverse approximations. In this section, we adapt this concept to unknown, possibly nonlinear baseline systems using a DNN-based control architecture (Fig. 3.1).

Given the control architecture in Fig. 3.1, in Chapter 2, it is shown that for a minimum phase system with a well-defined relative degree r, exact tracking (i.e., $y(k+r) = y_d(k+r)$) can be achieved by training the DNN to model the *exact inverse dynamics* of the baseline system. As shown in Chapter 2, to learn the exact inverse of system (3.2), the proper selection of inputs $\mathcal{I}$ and outputs $\mathcal{O}$ of the DNN module are $\mathcal{I} = \{x(k), y_d(k+r)\}$ and $\mathcal{O} = \{u(k)\}$. For LTI systems, based on the

representation (3.1), the inputs of the DNN module can be selected as

$$\mathcal{I} = \{y_d(k{-}n{+}r : k{+}r), u(k{-}n{+}r : k{-}1)\}, \tag{3.3}$$

where consecutive time indices are abbreviated with ':'. In practice, when applying these results to design the DNN module, only basic system properties (i.e., n and r) are needed. A system's order n can be determined from basic physics laws, and the relative degree r can be determined from simple step-response experiments. Although the *exact inverse learning* approach can be conveniently implemented in practice [48], its effectiveness is restricted to *minimum phase systems*.

3.4.1 The Proposed Approach: DNN Input Modification

We propose a learning approach that achieves stability (O1) and performance enhancement (O3) through modifying the DNN input selection. We first consider the linear baseline system (3.1), for which the exact inverse is

$$H^{-1}(z) = \frac{U(z)}{Y_d(z)} = \frac{D(z)}{N(z)} = \frac{\sum_{i=0}^{n} \beta_i z^i}{1 + \sum_{i=1}^{n-r} \alpha_i z^i}, \tag{3.4}$$

where $Y_d(z)$ is the z-transform of the desired output $y_d(k)$. For non-minimum phase systems, at least one root of the denominator $N(z)$ is outside of the unit circle, which is the source of instability that prevents the direct application of the inverse learning scheme in (3.3). If the input of the DNN module is selected such that the unstable dynamics associated with $N(z)$ cannot be learned, then the instability issues would not arise. Eqn. (3.4) can be written as $H^{-1}(z) = \frac{U(z)}{Y_d(z)} = \frac{\sum_{i=0}^{n} \beta_i z^{i-n}}{z^{-n} + \sum_{i=1}^{n-r} \alpha_i z^{i-n}}$; by applying the inverse z-transform (denoted by $\mathcal{Z}^{-1}\{\cdot\}$) to $H^{-1}(z)$, we have

$$\sum_{i=0}^{n} \beta_i \mathcal{Z}^{-1}\{z^{i-n} Y_d(z)\} = \mathcal{Z}^{-1}\{z^{-n} U(z)\} + \sum_{i=1}^{n-r} \alpha_i \mathcal{Z}^{-1}\{z^{i-n} U(z)\} \tag{3.5}$$

$$\Leftrightarrow \quad \sum_{i=0}^{n} \beta_i y_d(k+i-n) = u(k-n) + \sum_{i=1}^{n-r} \alpha_i u(k+i-n). \tag{3.6}$$

By shifting the signals in (3.6) forward by n, we can then relate the input and output of the inverse system as

$$u(k) = \sum_{i=0}^{n} \beta_i y_d(k+i) - \sum_{i=1}^{n-r} \alpha_i u(k+i), \tag{3.7}$$

or

$$u(k) = F(\underbrace{y_d(k:k{+}n)}_{\text{from } D(z)}, \underbrace{u(k{+}1:k{+}n{-}r)}_{\text{from } N(z)}), \tag{3.8}$$

where $F(\cdot)$ denotes a generic multi-variable function. From (3.8), it can be seen that the unstable dynamics associated with $N(z)$ are reflected in the dependency of $u(k)$ on the sequence of reference signals $u(k{+}1:k{+}n{-}r)$.

Proposed Input-Output Selection. Based on (3.8), we propose the following DNN input-output selection:

$$\mathcal{I} = \{y_d(k:k{+}n)\} \text{ and } \mathcal{O} = \{u(k)\}, \tag{3.9}$$

where the sequence of u is removed from the input $\mathcal{I}$ to prevent the DNN module from learning the unstable dynamics associated with $N(z)$.

Note that, while the proposed input-output selection is derived based on linear systems, when applying the proposed approach to nonlinear systems, the DNN module learns an approximate inverse of the nonlinear baseline system rather than a linearized baseline system. This is due to the fact that the DNN module is directly trained with the input-output data generated by the nonlinear baseline system.

3.4.2 Stability of the Proposed Approach

The proposed approach was derived from (3.1) to guarantee stability for the LTI systems. In this subsection, we prove stability for nonlinear systems using assumptions (A1) and (A3).

Theorem 3.4.1 (Stability of the Approximate Inverse Learning Approach)**.** *Consider the inversion-based learning control architecture in Fig. 3.1 and the nonlinear system (3.2). Under assumptions (A1) and (A3), the learning module input-output selection in (3.9) ensures that the overall control system (from y_d to y) is input-to-output stable.*

Proof. From (3.9), the learning module approximates a mapping from $\mathcal{I} = \{y_d(k:k{+}n)\}$ to $\mathcal{O} = \{u(k)\}$. For a typical L-layer DNN with $n+1$ inputs and 1 output, by denoting $\zeta_0(k) = [y_d(k)\ y_d(k+1)\ \cdots\ y_d(k+n)]^\intercal$ as the network input at time k, the output of a neuron i in a hidden layer l, denoted by $\zeta_{l,i}(k)$, can be expressed as $\zeta_{l,i}(k) = \sigma\left(\sum_{j=1}^{N_{l-1}} W_{l,ij}\zeta_{l-1,j}(k) + b_{l,i}\right)$, where $\sigma(\cdot)$ is the activation function, $l \in \mathbb{N}$, $1 \le l \le L-1$, is the layer index, $N_l \in \mathbb{N}$ is the number of neurons

in layer l, $\zeta_l \in \mathbb{R}^{N_l}$ is the output of the layer l, $W_l \in \mathbb{R}^{N_l \times N_{l-1}}$ and $b_l \in \mathbb{R}^{N_l}$ are the weights and bias associated with layer l, $\zeta_{l,i}$ and $\zeta_{l-1,j}$ are the i-th element of the vector ζ_l and the j-th element of the vector ζ_{l-1}, $W_{l,ij}$ is the i-th row and j-th column element of the matrix W_l, and $b_{l,i}$ is the i-th element of the vector b_l. The output of the network is $\widetilde{F}(\zeta_0(k)) = \sum_{j=1}^{N_{L-1}} W_{L,1j}\zeta_{L-1,j}(k) + b_{L,1}$. By assumptions A3a and A3b, the network parameters w and b are bounded, and σ is continuous; hence, the output of each neuron i in layer l (i.e., $\zeta_{l,i}$) is continuous in ζ_0. Moreover, since $\widetilde{F}$ is a composition of $\zeta_{l,i}$, $\widetilde{F}$ is also continuous in ζ_0. Since every continuous function from a compact space into a metric space is bounded, the network output $u(k)$ is bounded for bounded input $\zeta_0(k)$. Furthermore, by assumption (A1), the baseline system is input-to-output stable; thus, for any bounded desired trajectory y_d, the output $u(k)$ of the DNN is bounded, and the overall system from y_d to y is input-to-output stable. □

Note that the input-to-output stability of the DNN module and the overall DNN-enhanced system rely on the fact that the proposed DNN module is a continuous, static mapping. This stability result holds for both linear and nonlinear systems and is independent of the DNN regression errors.

3.4.3 Insights on Performance Enhancement

Given that the stability (O1) is achieved through the input selection of the learning module in (3.9), in this subsection we address the performance enhancement objective (O3).

Remark 3.4.1 (Approximate Inverse Learning)**.** *For system* (3.1)*, given a sufficiently high sampling rate, the input selection in* (3.9) *enables the DNN to learn an approximate inverse, where the sequence of reference signals in the input of the exact inverse map is approximated by* $u(k)$*.*

In order to clarify the insight above, we first present a toy example. Consider a linear function with input $\xi = [\xi_1\ \xi_2\ ...\ \xi_m]^\intercal \in \mathbb{R}^m$ and output $\upsilon \in \mathbb{R}$: $\upsilon = F_1(\xi)$. If a particular input ξ_p is correlated to the output υ by the linear function $\upsilon = F_2(\xi_p)$ and $\frac{\partial F_1}{\partial \xi_p} \neq \frac{dF_2}{d\xi_p}$, then υ can be re-expressed as a linear function of the remaining components of the vector ξ: $\upsilon = F_3(\widetilde{\xi})$, where $\widetilde{\xi} := [\xi_1\ ...\ \xi_{p-1}\ \xi_{p+1}\ ...\ \xi_m]^\intercal$. This implies that a regression model for the output υ can be found with either ξ or $\widetilde{\xi}$ as the input. This simple discussion can be generalized to the case when the removal of the dimension ξ_p *does not* lead to a one-to-many map from $\widetilde{\xi}$ to υ; a regression model can be constructed in a lower-dimensional input space to uniquely determine the output υ for a given $\widetilde{\xi}$. An illustration is shown in Fig. 3.2. When a component of

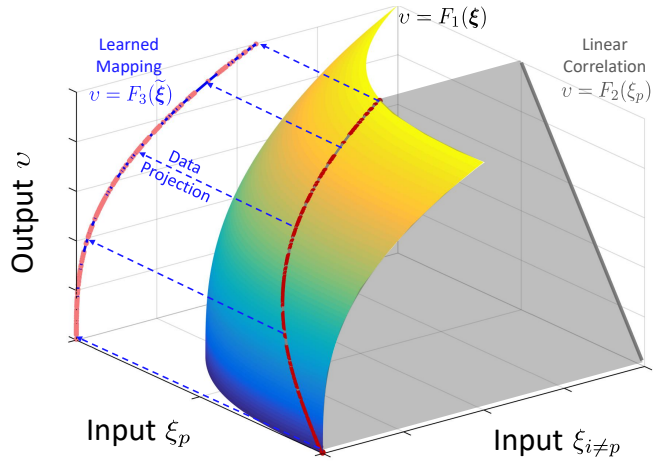


Figure 3.2: Illustration of data projection in the approximate inverse learning.

the input vector is related to the output by the function F_2, the data points generated by F_1 are restricted to the intersection of the manifolds defined by F_1 and F_2. When ξ_p is removed from the input of the dataset, the data points are projected onto a lower-dimensional space that is orthogonal to ξ_p.

For training, since an arbitrary smooth trajectory can be expressed as a superposition of sinusoidal functions, without loss of generality, we consider in our discussion below a single sinusoidal training trajectory of the form $u(t) = A\sin(\frac{2\pi}{T}t) + b$, where t denotes continuous time. It can be shown using Taylor series expansion of $u(t)$ that at time step k, future references $u(k+p)$ for $p = 1, ..., n-r$ can be related to the current reference $u(k)$ by

$$u(k+p) = u(k) + \sum_{i=1}^{\infty} \left(\frac{2\pi p \Delta t}{T}\right)^i c_i(k), \tag{3.10}$$

where Δt denotes the sampling time and $|c_i(k)| \leq \frac{A}{i!}$. Given that p is typically a small positive number bounded by $n-r$, if Δt is sufficiently small as compared to the period of the trajectory T, then from (3.10), at a particular time step k, the future reference $u(k+p)$ and $u(k)$ are approximately correlated by the identity function. Given this approximate correlation and by the result above, though dependent reference components are removed from the DNN input based on the selection in (3.9), the DNN can still learn a regression model to output a reference u that best matches that in the training dataset. Hence, the DNN acts as an approximate inverse from output y to input u that reduces the error between y_d and y. From (3.10), the error involved in consecutive reference signal approximations and the inherent regression

error in the learned inverse model is smaller for smaller Δt (i.e., higher sampling frequency).

For nonlinear systems, to achieve exact tracking, the learning module should model the output equation of the inverse dynamics, and $u(k)$ should be a nonlinear function of $x(k)$ and $y_d(k+r)$ (see Sec. 3.4); however, for non-minimum phase systems, the internal instability of $x(k)$ can cause numerical issues. One trivial solution is to remove the state $x(k)$ from the DNN input and use $\mathcal{I} = \{y_d(k+r)\}$. Instead, we suggest to use the same proposed input selection as in (3.9). A rough conjecture for this selection is as follows. Since smooth nonlinear systems can be approximated by piecewise affine/linear systems with arbitrary accuracy [71], one can always represent the considered, smooth nonlinear system as an aggregation of local, n-dimensional, affine/linear models defined on local regions of a cover/partition of the nonlinear system state space. Since all models have order n, by following the derivation in Sec. 3.4.1 for each local model, one obtains the same input selection as in (3.9) for each local model. Thus, it is reasonable to select the inputs for the DNN as in (3.9) even for nonlinear systems. The effectiveness of the proposed input selection for nonlinear systems is validated with simulations and experiments in Sec. 4.6 and 3.6, respectively.

3.4.4 Connection to the ZOS Approach

In this subsection, we show a connection between the proposed approach and a model-based approximate inverse approach for linear systems, the zero-order series (ZOS) approach [79]. In the ZOS approach, the transfer function polynomials associated with the unstable zeros are approximated by zero-order Taylor series [79]. In particular, by re-expressing (3.1) as $H(z) = \frac{N_s(z)N_u(z)}{D(z)}$, the ZOS approximate inverse is

$$\widetilde{H}^{-1}_{\text{ZOS}}(z) = \frac{D(z)}{N_u(z)|_{z=1} N_s(z)}, \tag{3.11}$$

where $N_s(z)$ and $N_u(z)$ denote the numerator polynomials with stable and unstable zeros, respectively.

Remark 3.4.2 (Connection to the ZOS Approximate Inverse)**.** *For linear systems, the approximation of the sequence of reference signals with the current reference $u(k)$ is equivalent to approximating the numerator of the transfer function $N(z)$ in (3.1) with $N(z)|_{z=1}$. With the input selection in (3.9), the proposed learning approach achieves stability (O1) and performance enhancement (O3) in a similar manner as*

the model-based ZOS approach in (3.11).

The time-domain representation of the exact inverse in (3.4) is shown in (3.7). When $u(k+i)$ for $i=1,...,n-r$ are approximated by $u(k)$ as in the proposed approach, we obtain $\sum_{i=0}^{n}\beta_i y(k+i) \approx \left(1+\sum_{i=1}^{n-r}\alpha_i\right)u(k)$, or $H^{-1}(z) \approx \frac{\sum_{i=0}^{n}\beta_i z^i}{1+\sum_{i=1}^{n-r}\alpha_i} = \frac{D(z)}{N(z)|_{z=1}}$ in the z-domain. By comparing the latter expression with the ZOS approximation in (3.11), it can be seen that they both achieve stability by approximating unstable zero dynamics at $z=1$, and compensating for the delays introduced by the dynamics associated with the poles ($D(z)$) to improve tracking performance.

Note that the generalizability of the DNN depends on the invariance of the phase and magnitude errors of the transfer function $\frac{Y(z)}{Y_d(z)} = \frac{N(z)}{N(z)|_{z=1}}$ with respect to the frequency of the desired trajectory; it can be shown that the generalizability is better if the zeros (the roots of $N(z)$) are further away from $z=1$. Moreover, similar to the ZOS approach [80], we expect that the proposed learning approach is more effective for enhancing the tracking performance of desired trajectories with frequencies less than the frequency of the zeros.

3.5 Simulation Results

We use an inverted pendulum on a cart system (pendulum-cart system) to illustrate the efficacy of the proposed approach for nonlinear non-minimum phase systems.

3.5.1 Simulation Setup

The pendulum-cart system has two degrees of freedom – the cart linear position η and the pendulum angular position θ. By applying Lagrange's equations, a dynamics model of the pendulum-cart system can be obtained [88]:

$$\begin{aligned}\ddot{\eta} &= \frac{q + mg\sin\theta\cos\theta - ml\dot{\theta}^2\sin\theta}{M + m\sin^2\theta}\\ \ddot{\theta} &= \frac{q\cos\theta + (M+m)g\sin\theta - ml\dot{\theta}^2\sin\theta\cos\theta}{l\left(M + m\sin^2\theta\right)},\end{aligned} \tag{3.12}$$

where M and m are the masses of the cart and the pendulum, respectively, l is the effective length of the pendulum relative to the pivot point, and q is the force applied to the cart. By defining the state of the system as $x = [\eta\ \dot{\eta}\ \theta\ \dot{\theta}]^\intercal$, its input as the force q, and its output as the full state $y = x$, the nonlinear state-space representation of

the pendulum-cart system can be written in the control affine form:

$$\dot{x} = f_1(x_2, x_3, x_4) + g_1(x_3)\, q, \;\; y = x, \tag{3.13}$$

where $x_2 = \dot{\eta}$, $x_3 = \theta$, and $x_4 = \dot{\theta}$. The control objective is to compute a control input q such that the cart tracks a desired trajectory $\eta_d(t)$ while the pendulum is balanced at the upright position. The desired output is $y_d(t) = [\eta_d(t) \quad \dot{\eta}_d(t) \quad 0 \quad 0]^\intercal$. Through linearizing the system (3.13) at $\eta = \eta_d$, $\dot{\eta} = 0$, $\theta = 0$, $\dot{\theta} = 0$, and $q = 0$, the pole placement technique can be used to find a stabilizing controller $q(t) = K_1(u(t) - y(t))$, where u is the reference of the baseline system and for our simulations $K_1 = [-0.8678 \;\; -1.808 \;\; 25.46 \;\; 4.140]$.

A learning module, pre-cascaded to the baseline system as in Fig. 3.1, is designed based on (3.9) to enhance the performance of the cart position tracking. Given the desired trajectory η_d (a component of y_d), at a time instance k, the learning module computes an adjusted reference signal η_r (a component of u) to be sent to the baseline system. The $\dot{\eta}_r$ component in u is generated from the η_r trajectory. A DNN with 2 hidden layers of 5 hyperbolic tangent neurons is used for learning the approximate inverse of the baseline system. Assuming that the baseline system succeeds to stabilize the pendulum at the upright position, then from (3.12), the dynamics associated with η may be approximated by a second-order system; by (3.9), the input and output of the learning module are selected to be $\mathcal{I} = \{\eta_d(k : k{+}2)\}$ and $\mathcal{O} = \{\eta_r(k)\}$. The learning module is executed at sampling intervals of 0.015 s. The module is trained on 30 sinusoidal trajectories with different combinations of amplitudes $\{0.5, 1.0, 1.5, 2.0, 2.5, 3.0\}$ m and periods $\{5, 10, 15, 20, 25\}$ s. The training dataset consists of pairs of $(\mathcal{I} = \{\eta(k : k{+}2)\}, \mathcal{O} = \{\eta_r(k)\})$ randomly sampled from the 30 training trajectories with equal proportions. Validation of the DNN model is performed on 30% of the training dataset; additional validation of the learning module is done by running the overall system on untrained trajectories.

3.5.2 Results

The tracking performance of the baseline system and the learning-enhanced system are compared in Fig. 3.3 for test sinusoidal trajectories with frequencies different from those used in training. From Fig. 3.3, although the baseline system is capable of stabilizing the pendulum-cart system, the tracking error increases with decreasing periods of desired trajectories. In contrast, when the proposed learning module is added to the baseline system, the tracking error is approximately maintained at a

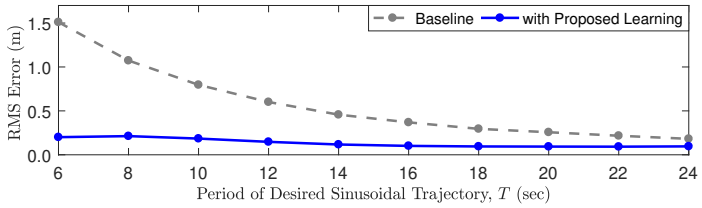


Figure 3.3: The RMS tracking error of the baseline and the learning-enhanced system for desired trajectories of the form $\eta_d(t) = \frac{5}{2}\sin\left(\frac{2\pi}{T}t\right)$, where the periods T are different from those used for training. The RMS error reduction achieved by the learning module ranges from 47% to 87%. A video for $T = 12$ s can be found at: http://tiny.cc/fq0mny.

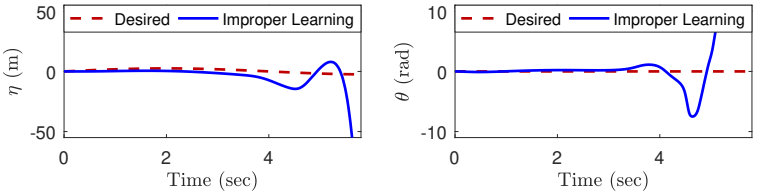


Figure 3.4: Illustration of the adverse effect caused by the inclusion of an additional reference component in the input $\mathcal{I}$ of the learning module.

smaller constant value over the range of trajectory periods covered by the training dataset, which shows the generalizing capabilities of the learning approach.

Fig. 3.4 shows the adverse impact when a single past reference is included in the proposed input selection of the learning module, i.e., when $\mathcal{I} = \{\eta_d(k:k{+}2), u(k{-}1)\}$. It can be seen that when the additional information is included, the pendulum-cart system quickly becomes unstable. Thus, for non-minimum phase systems, the input selection of the learning module is essential; the inclusion of unnecessary inputs can prevent not only the learning approach but also the baseline system from being functional. From this example, it is interesting to see that, for non-minimum phase systems, *the DNN trained with less inputs leads to a better performance*. In contrast to typical DNN applications (e.g., image classification), for control applications, the training objective (e.g., minimizing regression error) and performance objective (e.g., minimizing tracking error) may not coincide. Consequently, DNN training algorithms may not phase out unnecessary input dimensions to achieve a good performance.

3.6 Experimental Results

The effectiveness of the proposed approach is further verified using pendulum-cart and quadrotor experiments. Note that, in the experiments, the criterion we use

for evaluating tracking performance is the RMS tracking error, which characterizes tracking performance over entire trajectories.

3.6.1 Pendulum-Cart Experiments

3.6.1.1 Experiment Setup

The setup is similar to that of the simulation (Sec. 3.5.1), except that the input force q is replaced by the input voltage v to the cart motor. By using a simple voltage-to-force model $q(t) = -7.74\dot{\eta}(t) + 1.73v(t)$ [89], system (3.13) can be re-expressed as

$$\dot{x} = f_2(x_2, x_3, x_4) + g_2(x_3)\, v, \;\; y = x, \tag{3.14}$$

where $\eta(t)$ and $\theta(t)$ are measured, and $x(t)$ is estimated with a full-state observer. A controller $v(t) = K_2(u(t) - y(t))$ with $K_2 = [-105.6 \;\; -55.04 \;\; 130.7 \;\; 23.67]$ is run at 1 kHz.

We compare the proposed learning approach with the baseline system and the model-based ZOS approach. In the experiments, the learning module is run at 70 Hz; the design and training procedure for the inverse-learning module are similar to that of the simulations (see Sec. 4.6). The training dataset is constructed from 18 sinusoidal trajectories with combinations of amplitudes $\{0.04, 0.06, 0.08\}$ m and periods $\{5, 6, 7, 8, 9, 10\}$ s. The ZOS approach is implemented based on the linearized state-space model of system (3.14). From the linearized system, a discrete-time transfer function from the reference η_r to the output η can be determined. By applying (3.11), the ZOS approximate inverse is obtained: $\widetilde{H}^{-1}_{\text{ZOS}}(z) = \frac{z^4 - 3.5217z^3 + 4.6504z^2 - 2.7290z + 0.6005}{0.00137z^2 - 0.0001066z - 0.001066}$. For the experimental comparison, the ZOS approximate inverse $\widetilde{H}^{-1}_{\text{ZOS}}(z)$ replaces the learning module in Fig. 3.1.

3.6.1.2 Results

Figure 3.5 compares the tracking performance of the baseline, the ZOS, and the proposed learning-based systems on a test trajectory $\eta_d(t) = \frac{117}{2000}\sin(\frac{2\pi}{5}t) + \frac{13}{2000}\sin(\frac{4\pi}{11}t)$, which was not included in the training phase. The stability objective is achieved by all three systems, and the pendulum position is kept approximately at the upright position. From the cart position $\eta(t)$ plot, the proposed DNN (blue) effectively compensates for the phase and magnitude errors in the baseline system response (gray). For this test trajectory, the learning module reduces the RMS tracking error by 60%.

In contrast, by comparing the $\eta(t)$ of the ZOS approach (green) with the baseline

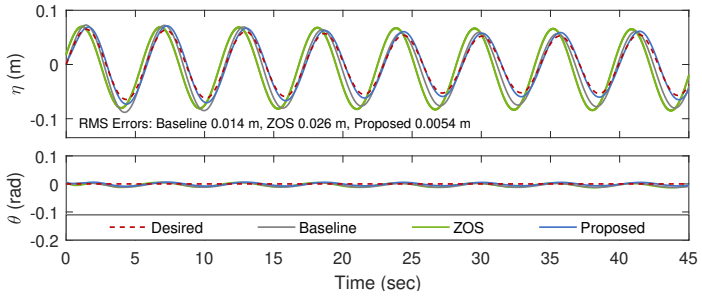


Figure 3.5: The cart position η and the pendulum angle θ of the baseline, the ZOS, and the proposed learning-based systems on a test trajectory $\eta_d(t) = \frac{117}{2000}\sin(\frac{2\pi}{5}t) + \frac{13}{2000}\sin(\frac{4\pi}{11}t)$.

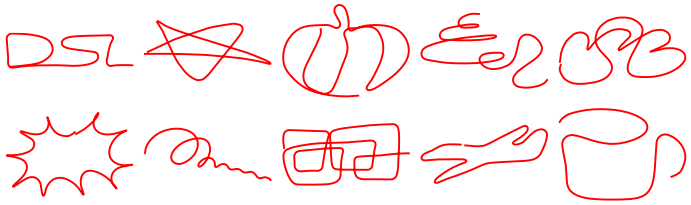


Figure 3.6: Illustrations of 10 hand-drawn test trajectories used for evaluating the tracking performance of the quadrotor controllers.

response (gray), the addition of the approximate inversion led to worse tracking performance. Though the linearized state-space model is sufficiently accurate for deriving a baseline controller that stabilizes the pendulum-cart system, the application of the model-based system inversion approach requires a much more detailed and accurate system model. Thus, in comparison with the ZOS approach, *the proposed DNN-based learning approach (blue) is capable of achieving a better performance without relying on a detailed dynamic model of the baseline system.*

3.6.2 Quadrotor Experiments

The efficacy of the proposed approach on higher degree-of-freedom systems is demonstrated using quadrotor vehicles. In this set of experiments, the objective is to enhance a baseline controller of a quadrotor for tracking arbitrary, hand-drawn trajectories (Fig. 3.6) in one shot.

3.6.2.1 Experiment Setup

The state vector of the quadrotor system consists of the positions $\mathbf{p} = (x, y, z)$, velocities $\mathbf{v} = (\dot{x}, \dot{y}, \dot{z})$, roll-pitch-yaw Euler angles $\boldsymbol{\theta} = (\phi, \theta, \psi)$, and rotational velocities $\boldsymbol{\omega} = (p, q, r)$. The control objective is to control the position of the quadrotor to track

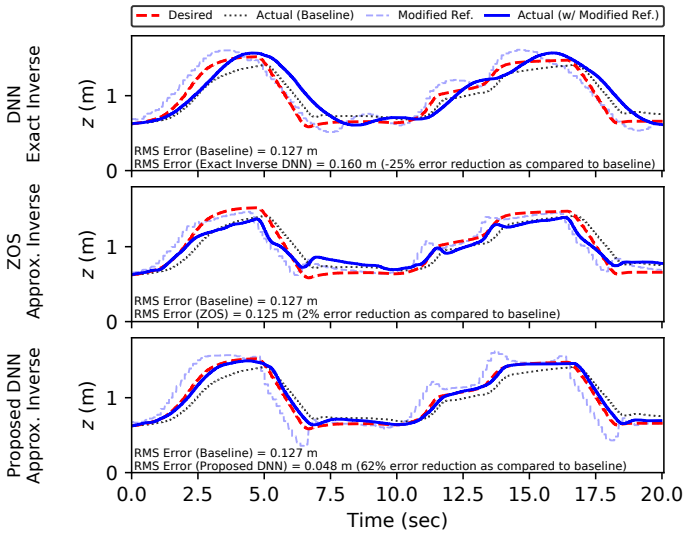


Figure 3.7: Comparison of the DNN exact inverse approach (Chapter 2), the ZOS approximate inverse approach, and the proposed DNN approximate inverse approach for enhancing the tracking performance of the modified non-minimum phase quadrotor system. The desired z-position trajectory is from the first hand-drawn test trajectory shown in Fig. 3.6 (left, top).

a desired trajectory $\mathbf{p}_d(t)$. The baseline tracking controller is a standard nonlinear controller composed of a nonlinear transformation and PD control running at 70 Hz. For the purpose of studying non-minimum phase systems, non-minimum phase zeros at 1.2 are introduced to the baseline system by modifying the baseline z position and velocity references (z_r and $\dot{z}_r$). Note that, in this chapter, we purposely introduce a non-minimum phase zero to the baseline system for evaluating our proposed approach; in practice, this non-minimum phase nature can occur in apparent minimum phase robotic systems when the sampling rate is high [90].

In the experiments, we examine three inversion-based approaches that adapt the reference signals of the baseline controller $\mathbf{p}_r$ and $\mathbf{v}_r$ to reduce the tracking error between the desired position $\mathbf{p}_d$ and the actual position $\mathbf{p}$:

(M1) DNN exact inverse learning: the learning-based approach effective for minimum phase systems;

(M2) ZOS approximate inverse: a model-based approach for non-minimum phase systems;

(M3) DNN approximate inverse learning: the proposed learning-based approach with input-output selection based on (3.9).

The inverse blocks receive the desired position $\mathbf{p}_d$ and desired velocity $\mathbf{v}_d$ as inputs,

and compute the adjusted position reference $\mathbf{p}_r$ and velocity reference $\mathbf{v}_r$ for the baseline system. For comparison purposes, the DNN training and architecture are similar to that outlined in Sec. 2.6.1. In particular, the DNNs are fully-connected feedforward networks with 4 hidden layers of 128 ReLUs. During the training phase, the baseline system is used to track a 400-second, 3-dimensional sinusoidal trajectory, and the input-output data of the baseline system is collected at 7 Hz. The training dataset of the DNN consists of $(\mathcal{I}, \mathcal{O})$ pairs randomly sampled from the input-output data of the baseline system. Overall, 90% of the dataset is used for training, and the remainder of the dataset is used for validation. For evaluating the effectiveness and generalizability of the inversion-based approaches, test trajectories generated from arbitrary hand drawings are utilized (Fig. 3.6).

3.6.2.2 Results

We first examine the three inversion-based approaches for enhancing the tracking performance of the modified non-minimum phase quadrotor baseline system, where non-minimum phase zeros are introduced in the dynamics associated with the z-direction. The implementation of (M1) follows from that in Sec. 2.6.1. The implementation of (M2) is based on the approximation of the dynamics of the baseline system with decoupled second-order linear systems; by applying (3.11), the ZOS approximate inverse is found to be $H_{\text{ZOS}}^{-1}(z) = \frac{z^3-1.713z^2+0.7493z}{0.2692z-0.2331}$, and is applied to adjust the position and velocity references z_r and $\dot{z}_r$. In the implementation of (M3), we need to estimate the system order n. We assume that the quadrotor has decoupled double-integrator dynamics in the x, y, and z directions. By further accounting for the experimentally determined time delays in each direction and applying (3.9), the inputs and outputs of the DNN module are selected to be $\mathcal{I} = \{x_d(k{+}1 : k{+}7) - x_d(k), y_d(k{+}1 : k{+}7) - y_d(k), z_d(k{+}1 : k{+}5) - z_d(k), \dot{x}_d(k{+}1 : k{+}6) - \dot{x}_d(k), \dot{y}_d(k{+}1 : k{+}6) - \dot{y}_d(k), \dot{z}_d(k{+}1 : k{+}4) - \dot{z}_d(k)\}$ and $\mathcal{O} = \{x_r(k) - x_d(k), y_r(k) - y_d(k), z_r(k) - z_d(k), \dot{x}_r(k) - \dot{x}_d(k), \dot{y}_r(k) - \dot{y}_d(k), \dot{z}_r(k) - \dot{z}_d(k)\}$. Following the discussion in Chapter 2, in the implementations of (M1) and (M3), we utilized a difference learning scheme (i.e., training with relative positions and velocities) to improve training efficiency.

Figure 3.7 shows a comparison of the three inversion-based approaches for a test trajectory in the z-direction, $z_d(t)$, from the first hand drawing shown in Fig. 3.6. From the top panel, as expected, due to the inherent instability of the inverse, the approach (M1) does not lead to an improved tracking performance. Instead, it introduces undesired oscillations in the system response and leads to worse performance as

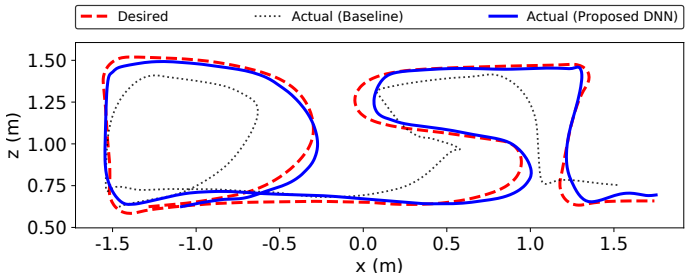


Figure 3.8: Example of the performance enhancement achieved by the proposed DNN approximate inverse approach for the modified non-minimum phase quadrotor system. Here, the proposed DNN leads to a 67% error reduction.

compared with the baseline controller. We next consider (M2) shown in the middle panel. From the computed reference z_r (light blue dotted line), it can be seen that the model-based system approximate inversion tends to compensate for the delay in the system response; however, with the linearized model, the approximate inverse H_{ZOS}^{-1} cannot effectively reduce the magnitude error of the system response. In contrast, for the proposed approach (M3), shown in the bottom panel, the reference computed by the DNN module efficaciously compensates for the tracking errors of the baseline response. With (M3), the RMS tracking error in the z-direction is reduced by approximately 62%, while the percentage reductions for (M1) and (M2) are approximately -25% and 2%, respectively.

Figure 3.8 shows the tracking performance of the proposed approach (M3) on the hand-drawn test trajectory corresponding to that shown in Fig. 3.7. On this hand-drawn test trajectory, the proposed approach reduces the 3-dimensional RMS tracking error by 67%. The generalizability of the proposed approach is tested on 10 hand-drawn trajectories (Fig. 3.6), which are not seen during the training phase. Fig. 3.9 shows a summary of the 3-dimensional RMS errors of the non-minimum phase baseline quadrotor tracking system (dark blue bars) and the system enhanced by the proposed DNN approximate inverse learning (light blue bars). On average, 60% error reduction is achieved by the proposed DNN module. In addition, the dark and light yellow bars in Fig. 3.9 show that the proposed DNN also effectively enhances the performance of the original minimum phase quadrotor system studied in Chapter 2.

Note that, with the proposed approach, it is expected that the performance enhancement of the DNN module is better for input trajectory frequencies closer to those seen in the training phase; in practice, the DNN inverse module should be trained on a dataset that sufficiently covers the operational space.

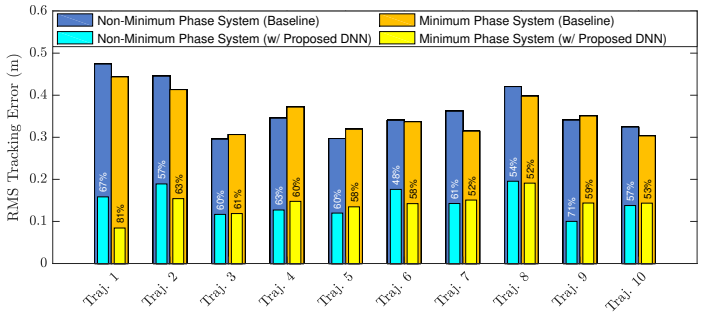


Figure 3.9: RMS tracking errors on 10 hand-drawn test trajectories (shown in Fig. 3.6) for the modified non-minimum phase quadrotor system and the original minimum phase quadrotor system. The percentage above each bar indicates the error reduction achieved by the addition of the proposed DNN module. On average, the DNN modules lead to approximately 60% error reductions for both the non-minimum phase and minimum phase systems.

3.7 Conclusions

Many robotic systems can exhibit non-minimum phase behaviours; in this chapter, we present a learning-based approach to enhance the impromptu tracking performance of non-minimum phase systems. In our approach, a learning module approximates the inverse of a stabilized baseline system, and the stability of the learning module is ensured through appropriate input selection. As demonstrated with experiments on a pendulum-cart and quadrotor system, the proposed approach, requiring only input-output data of the baseline system, leads to significantly better performance as compared to the ZOS approximate inverse, one of the typical model-based approaches in the literature.

Chapter 4

Active Training Trajectory Generation to Improve Sampling Efficiency

4.1 Introduction

In previous chapters, we studied a DNN-based approach that enhances the trajectory tracking performance of black-box control systems. In particular, a DNN module is trained to approximate the inverse dynamics of the underlying system, and at test time, it is pre-cascaded to the system to enhance the tracking performance (Fig. 2.1). While we verified the efficacy of our approach with extensive experiments, one open question that requires further exploration is a systematic trajectory generation approach for training the DNN inverse dynamics module.

In addition to our work, various neural network (NN)-based control architectures have been proposed in the control literature. In these works, a common assumption is that the NNs are trained on datasets that sufficiently cover the operational space [51]. In practical applications, this assumption often results in a trial-and-error process of collecting data, training the model, testing in experiment, and repeating this process until satisfactory control performance is achieved based on a representative dataset. This trial-and-error process can lead to unnecessary training and related costs on physical robots, or safety risks in industrial applications. This fact motivates us to investigate approaches that guide the data collection process towards experiments that are the most informative for (D)NN-based model learning.

From the machine learning literature, a concept that can be adopted for informa-

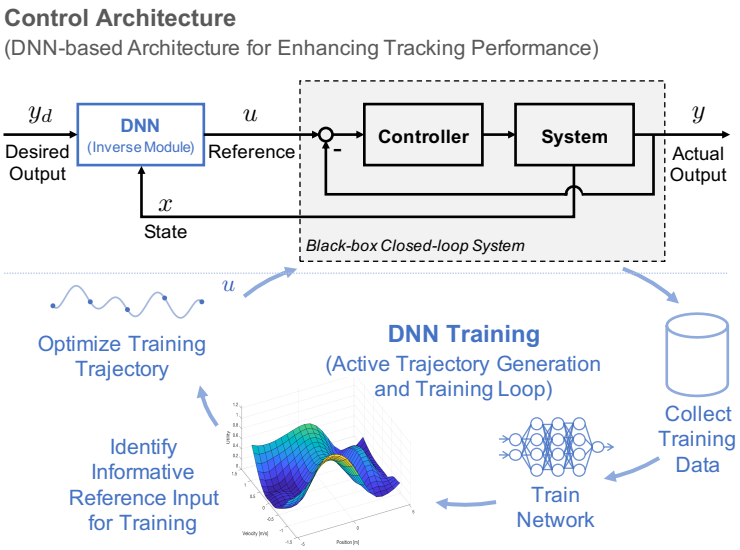


Figure 4.1: A DNN-based control architecture *(top)* was proposed in Chapter 2 to enhance the tracking performance of black-box, closed-loop control systems. In this chapter, we study an episode-based active trajectory generation approach *(bottom)* for systematically training the DNN inverse dynamics module. With the proposed approach, the DNN module is trained in a closed-loop manner, where in each episode, informative points for training the DNN module are identified and a smooth trajectory is generated for collecting the data in the next set of robotics experiments.

tive DNN training data collection is active learning [91]. With active learning, instead of passively training the learner using a pre-collected dataset, the learner is allowed to 'actively select' the most informative points to query. This concept has been applied to various machine learning problems such as segmentation and image/text classification with the goal of saving the time and cost associated with manually labelling datasets [91]. In this chapter, we adopt the idea of active learning and propose an optimization framework that allows us to systematically generate feasible and informative trajectories for training DNN inverse dynamics models. With the proposed active trajectory generation framework, we aim to (1) improve the efficiency of the data collection and (2) provide a means for monitoring when the training of DNN-based robot model learning is sufficient.

4.2 Related Work

The concept of active learning is closely tied to the theory of optimal experimental design (OED), which is a branch of statistics that outlines the mathematical foundations and statistical criterion for automating query selections [92]. The theory of OED has been discussed in a wide range of contexts including neuroscience [93], system

identification [94], structural optimization [95], and experimental economics [96]. In these different applications, the shared goal for OED is to maximize the information content gathered about an underlying process of interest with limited experiments.

While OED can encompass experimental conditions in a broader sense, active learning focuses on input design for black-box models. As shown in [97], common active learning heuristics such as uncertainty sampling can be connected mathematically to OED optimality criteria. Examples of active learning can be found in classification tasks, where unlabelled images or text are selected heuristically to minimize prediction errors [98, 99], or in regression tasks [100, 101, 102], where regions lacking data are identified for further exploration. As discussed in [103], in addition to improved data efficiency, with controlled input selection, active learning is also a way to enhance generalization of black-box models. Despite its advantages, it should be noted that typical frameworks of active learning provide a set of informative points but do not account for issues such as continuity or feasibility constraints that are critical to collecting data on physical robots for model learning.

In the literature, one approach to account for feasibility constraints is to parametrize an input trajectory and formulate an optimization problem for trajectory generation [104]. This approach has been utilized to generate excitation trajectories for identifying dynamic parameters of manipulators [105, 106, 107]. In particular, the identification of the manipulator dynamics is written as a linear regression problem. The joint trajectories are parametrized as splines [106] or harmonic functions [107, 105], and the trajectory parameters are optimized to minimize the model parameter uncertainty or the condition number of the linear regression model. In this chapter, we similarly formulate an optimization problem for generating input trajectories. However, we present a method that incorporates an active learning objective for training generic inverse dynamics models parametrized as DNNs.

4.3 Problem Statement

We consider the DNN-enhanced control architecture shown in Fig. 4.1. In this learning-based approach, a DNN module is trained offline to approximate the inverse dynamics of a baseline closed-loop system. The trained DNN is then pre-cascaded to the baseline system at test time to enhance its tracking performance. The goal of this work is to derive a framework that allows us to design informative trajectories for systematically training the DNN inverse module.

We consider a baseline closed-loop system represented by

$$\begin{aligned} x(k+1) &= f\left(x(k)\right) + g\left(x(k)\right)u(k), \\ y(k) &= h\left(x(k)\right), \end{aligned} \tag{4.1}$$

where k is the discrete time index, $x \in \mathbb{R}^n$ is the system state, $u \in \mathbb{R}$ is the reference of the closed-loop baseline system, $y \in \mathbb{R}$ is the output, and f, g, and h are smooth functions with consistent dimensions. System (4.1) is said to have a relative degree r around an operating point (x_0, u_0) if *(i)* $\frac{\partial}{\partial u} h \circ f^p(f(x(k)) + g(x(k))u(k)) = 0$, $\forall p = 0, ..., r-2$ for each point in the neighbourhood of (x_0, u_0), and *(ii)* $\frac{\partial}{\partial u} h \circ f^{r-1}\big(f(x(k)) + g(x(k))u(k)\big) \neq 0$ at (x_0, u_0), where $h \circ f$ is the composition of the functions h and f, and $f^p(\cdot)$ denotes the pth composition of f, with $f^0(x(t)) = x(t)$ [59]. As shown in Chapter 2, if a system has a well-defined relative degree r, and $y(k+r)$ is affine in $u(k)$, we can derive the reference for exact tracking (i.e., $y(k+r) = y_d(k+r)$):

$$u(k) = \mathcal{G}^{-1}\big(x(k)\big)\big(y_d(k+r) - \mathcal{F}\left(x(k)\right)\big), \tag{4.2}$$

where $\mathcal{F}\left(x(k)\right) = h \circ f^r(x(k))$ and $\mathcal{G}\left(x(k)\right) = \frac{\partial}{\partial u} h \circ f^{r-1}(f(x(k)) + g(x(k))u(k))$. When the exact dynamics of the baseline system (4.1) is unknown but it is minimum phase and has a well-defined relative degree, then we can train a DNN module to approximate (4.2) to effectively enhance the tracking performance of the baseline system:

$$u(k) = F_\theta\left(x(k), y_d(k+r)\right), \tag{4.3}$$

where $F_\theta(\cdot)$ denotes the nonlinear function represented by the DNN module, and θ denotes the DNN module parameters. This approach is generally applicable to systems such as quadrotors and robot manipulators.

We consider an episode-based training trajectory generation framework outlined in Alg. 4.3.1 for systematically training the DNN inverse module. In each episode, we first identify the informative points for training the DNN module and formulate an optimization problem to generate a training trajectory $\{u(k)\}$, $k = 0, ..., K$, where K is the predefined training trajectory length. The generated training trajectory is sent to the baseline system, and the input-output response data $\{x(k), y(k), u(k)\}$ is recorded. The training dataset for the DNN model is extended using the obtained system response data, where the paired input and output of the training dataset are $\xi = [x(k), y(k+r)]$ and $\gamma = u(k)$, respectively. Given this dataset, the DNN module is trained with standard stochastic gradient descent (SGD) algorithms. In the next

Algorithm 4.3.1 Episode-based Training Trajectory Optimization for DNN Inverse Dynamics Model Learning

1: Initialize with some trajectory $\{u(k)\}$ for $k = 1, ..., K$
2: **while** not converged **do**
3: Run the training trajectory on the baseline system and record the input-output response data $\{x(k), y(k), u(k)\}$ for $k = 0, ..., K$
4: Extend DNN training dataset with input $\xi = [x(k), y_d(k + r)]$ and output $\gamma = u(k)$
5: Train DNN inverse dynamics model on the dataset
6: Select training points that maximize an active learning utility function
7: Form a quadratic program (QP) to obtain a feasible training trajectory $\{u(k)\}$ for $k = 0, ..., K$
8: **end while**

sections, we derive the details of the proposed active training trajectory generation in Ln. 6-7 of Alg. 4.3.1.

4.4 Background on Active Learning for DNNs

In this section, we provide a brief summary of the active learning literature for DNNs to facilitate our discussion.

4.4.1 DNN Model Preliminaries

We consider a L-layer fully-connected DNN denoted by $\gamma = F_\theta(\xi)$, where ξ is the input to the network, γ is its output, and $\theta = (w_1, ..., w_L, b_1, ..., b_L)$ is an augmented vector of weights and biases parametrizing the network. We can express an L-layer network F_θ as

$$\begin{aligned} \zeta_0 &= \xi, \\ \zeta_l &= \sigma\left(w_l \zeta_{l-1} + b_l\right), \quad \forall l = 1, ..., L-1, \\ \gamma &= w_L \zeta_{L-1} + b_L, \end{aligned} \tag{4.4}$$

where $w_l \in \mathbb{R}^{N_l \times N_{l-1}}$ and $b_l \in \mathbb{R}^{N_l}$, N_l is the number of neurons in a hidden layer, $\sigma : \mathbb{R}^{N_l} \mapsto \mathbb{R}^{N_l}$ is the element-wise activation operation applied in a hidden layer, and ζ_l for $l = 1, ..., L-1$ is the output of a hidden layer. Given a training dataset with D paired input-output points $\mathcal{D} = \{\xi_d, \gamma_d\}_{d=1}^{D}$, SGD algorithms can be used to find a set of parameters θ that minimizes the network prediction error.

4.4.2 Predictive Uncertainty Estimation for DNNs

Common active learning heuristics are based on the uncertainty of the learner's output predictions (i.e., the predictive uncertainty). For each input, there is a corresponding prediction of mean and variance of the output. Intuitively, with active learning, we wish to collect data at inputs which the learner is uncertain about. We review three common predictive uncertainty estimation techniques for DNNs.

Fisher Information: We assume that the conditional probabilities $p(\gamma \mid \xi, \theta)$ at distinct inputs ξ are independent Gaussian distributions with expectations $\mathbb{E}_{p(\gamma \mid \xi,\theta)}[\gamma \mid \xi, \theta] = F_\theta(\xi)$, where $\mathbb{E}_{p(\gamma \mid \xi,\theta)}[\,\cdot\,]$ denotes the expectation over $p(\gamma \mid \xi, \theta)$. We can approximate the predictive variance at a given input ξ as

$$\mathbb{V}_{p(\gamma \mid \xi,\theta)}[\gamma \mid \xi, \theta] = \nabla_\theta F_\theta(\xi)^T M^{-1} \nabla_\theta F_\theta(\xi), \tag{4.5}$$

where ∇_θ denotes the gradient with respect to the parameters θ, and $M = \frac{1}{S^2}\sum_{d=1}^{D} \nabla_\theta F_\theta(\xi_d) \nabla_\theta F_\theta(\xi_d)^T$ with $S^2 = \frac{1}{2D}\sum_{d=1}^{D} ||F_\theta(\xi_d) - \gamma_d||^2$ approximates the Fisher information matrix, which the inverse characterizes the lower bound on the parameter uncertainties [108].

Bagging: Bagging is a method providing an empirical estimate of the predictive uncertainty for DNNs. In typical implementations, an ensemble of DNN models is trained, and randomization is introduced to the DNN ensemble via randomized batch samples and/or initial weights [109]. The predictive variance at a given input is estimated based on the empirical variance of the DNN outputs.

Dropout Approximate Inference: An alternative ensemble uncertainty estimation technique is based on dropout approximate inference, which can be interpreted as a variational approximation of the Bayesian inference performed with deep Gaussian processes [100]. In this approach, a single DNN is trained with stochastic dropout [76]. The predictive variance of the DNN at test time is estimated based on multiple forward passes with independently sampled dropout units:

$$\mathbb{V}_{p(\gamma \mid \xi,\theta)}[\gamma \mid \xi, \theta] = \hat{S}^2 + \tau^{-1} I, \tag{4.6}$$

where $\hat{S}^2$ is an empirical estimate of the predictive variance found via multiple forward passes, $\tau = \frac{p_{\text{keep}} l^2}{2K\lambda}$ is the model precision, p_{keep} is the dropout Bernoulli distribution parameter, K is the data size, and I is the identity matrix [100].

4.4.3 Measures of Informativeness

With active learning, our goal is to collect data $\mathcal{D}$ that maximizes the information we gain about the underlying process. We denote a generic active learning problem as

$$\xi^* = \arg\max_{\xi} \quad U(\xi, \theta), \tag{4.7}$$

where U denotes an utility function that measures the amount of information provided by querying ξ. We briefly review the common active learning heuristics for DNNs. More thorough discussions can be found in review papers such as [91].

Uncertainty Sampling: Uncertainty sampling is an active learning heuristic that encourages selecting an input ξ^* about which the model is currently the most uncertain. The utility function can be written as $U_{\text{US}} = \mathbb{H}(\gamma \,|\, \xi, \theta)$, where $\mathbb{H}(\cdot)$ denotes the entropy of a random variable and is a monotonic function of variance for a Gaussian distribution. Intuitively, uncertainty sampling encourages the selection of an input ξ^* that the model is currently the most uncertain about.

Ensemble-based Approach: In an ensemble-based approach, multiple DNN models are trained for making predictions, and the extent of disagreement is used as the measure of information. This approach can be thought as a variation of uncertainty sampling, where the uncertainty is estimated by the empirical variance of the DNN ensemble [91].

Information Gain: Information gain characterizes the expected regression error reduction for an unbiased learner [110]. It is defined as $U_{IG} = \mathbb{E}_{p(\xi')}\left[\mathbb{H}(\gamma \,|\, \xi', \theta) - \mathbb{E}_{p(\gamma \,|\, \xi, \theta)}[\mathbb{H}(\gamma \,|\, \xi', \theta^+)]\right]$, where θ^+ is the parameter if the DNN is trained on a candidate input ξ, where the second term in the expectation represents the expected entropy of the learner after the new data point is added for training.

4.5 Active Training Trajectory Generation

In this section, we formulate the active training trajectory generation framework for DNN inverse dynamics learning.

4.5.1 Spline Trajectory Generation

We adopt a trajectory generation framework similar to [104]. In particular, we consider reference trajectories parametrized by Nth-order polynomial splines $T_s(t) = \sum_{n=0}^{N} p_{s,n} t^n$ joined at prescribed times $\{t_1, ..., t_{S-1}\}$, where T_s is the sth polynomial

of the spline, t is the continuous time, and $p_{s,n}$ are the coefficients of the sth polynomial. The smoothness of the reference trajectories is enforced by penalizing the mth derivative of the spline:

$$\begin{aligned} \min_{p} \quad & \int_{t_0}^{t_S} \left|\left|T^{(m)}(t)\right|\right|^2 dt \\ \text{s.t.} \quad & T_s(\tau) = \bar{T}(\tau), \forall s = 1, ..., S, \tau = \{t_{s-1}, t_s\}, \\ & T_s^{(l)}(t_s) = T_{s+1}^{(l)}(t_s), \forall s = 1, ..., S-1, \\ & \qquad\qquad\qquad\qquad l = 1, ..., l_{\max}, \end{aligned} \tag{4.8}$$

where $T(t)$ denotes the spline trajectory, $\bar{T}(t)$ is the predefined waypoint at t, $||\cdot||$ denotes the Euclidean norm, $p = (p_1, ..., p_S)$ is an augmented vector with p_s containing the coefficients of the sth polynomial segment (in ascending order), $t_0, ..., t_S$ are the times corresponding to the interior points of the spline trajectory, and $l_{\max}$ is the order of continuity enforced at the interior points.

By substituting the definition of $T_s(t)$ into (4.8) and computing the derivatives, one can show that the spline trajectory generation in (4.8) can be formulated as

$$\begin{aligned} \min_{p_1,...,p_S} \quad & \sum_{s=1}^{S} p_s^T Q_s p_s \\ \text{s.t.} \quad & A_s p_s - b_s = 0, \quad \forall s = 1, ..., S, \\ & Ap = 0, \end{aligned} \tag{4.9}$$

where Q_s is a matrix enforcing smoothness of the sth polynomial segment, and the pairs $(A_s,\, b_s)$ and the matrix A enforce continuity constraints at the interior points of the spline. The matrix Q_s for the sth polynomial is

$$Q_s = \left[\begin{array}{c|c} 0_{m\times m} & 0_{m\times(N-m+1)} \\ \hline 0_{(N-m+1)\times m} & \widetilde{C}^T \widetilde{Q}_s \widetilde{C} \end{array}\right], \tag{4.10}$$

where $\widetilde{C} = \text{diag}(c(m), ..., c(N))$, $c(q) = \prod_{i=0}^{l-1} q - i$, and

$$\widetilde{Q}_s = \begin{bmatrix} t_s - t_{s-1} & \cdots & \frac{t_s^{N-m+1} - t_{s-1}^{N-m+1}}{N-m+1} \\ \vdots & \ddots & \vdots \\ \frac{t_s^{N-m+1} - t_{s-1}^{N-m+1}}{N-m+1} & \cdots & \frac{t_s^{2(N-m)+1} - t_{s-1}^{2(N-m)+1}}{2(N-m)+1} \end{bmatrix}. \tag{4.11}$$

For the continuity constraints, we note that the lth derivative of T_s evaluated at time t is $T_s^{(l)}(t) = A_{slt}\, p_s$, where $A_{slt} = \begin{bmatrix} 0_l & c(l) & c(l+1)t & \cdots & c(N)t^{N-l} \end{bmatrix}$, where 0_l is a zero row vector with dimension indicated by the subscript. The zero-order continuity constraint can be enforced by setting

$$A_s = \begin{bmatrix} A_{s0t_{s-1}} \\ A_{s0t_s} \end{bmatrix} \text{ and } b_s = \begin{bmatrix} \bar{T}(t_{s-1}) \\ \bar{T}(t_s) \end{bmatrix}. \tag{4.12}$$

The higher-order continuity constraints at the interior points are introduced with A. For instance, the lth-order continuity enforced at t between the sth and $(s+1)$th polynomials can be introduced by augmenting the following row to the matrix

$$[A]_i = \begin{bmatrix} 0_{(s-1)(N+1)} & A_{slt} & -A_{(s+1)lt} & 0_{(S-s-1)(N+1)} \end{bmatrix}, \tag{4.13}$$

where $[A]_i$ denotes the ith row of A.

4.5.2 Integrating Active Learning and Trajectory Optimization

In this subsection, we integrate the active learning concept discussed in Sec. 4.4 and the spline trajectory generation approach presented in Sec. 4.5.1.

Based on the approaches discussed in Sec. 4.4, in each episode, we estimate the DNN module predictive uncertainty and calculate the utility over the DNN input space. This allows us to identify a set of points in the DNN input space that are the most informative for training the DNN module:

$$\xi^* = \arg\max_{\xi} \quad U(\xi, \theta), \tag{4.14}$$

where $\xi = [x(k), y(k+r)]$ and $\gamma = u(k)$ for our problem.

In contrast to typical active learning applications, where the input ξ^* is directly used as the next point to query, for our inverse learning problem, we need to identify the informative DNN outputs γ^* (i.e., the informative references) to be sent to the baseline system. To this end, given the informative input(s) ξ^*, we evaluate the corresponding outputs of the DNN module $\gamma^* = F_\theta(\xi^*)$ and the associated standard deviations $\Delta\gamma$ at these inputs. A set of candidate DNN outputs γ_s^* are sampled from $\mathcal{N}(\gamma^*, \alpha\Delta\gamma)$, where $\mathcal{N}(\gamma^*, \alpha\Delta\gamma)$ denotes a Gaussian distribution with a mean of γ^* and a standard deviation of $\alpha\Delta\gamma$, and $\alpha \geq 1$ is a parameter that controls the

range of exploration. Intuitively, if the uncertainty estimates of the DNN module are sufficiently accurate, the samples γ_s^* encourage explorations over regions where the DNN module is currently uncertain, where the range of exploration is proportional to the extent of uncertainty. We treat γ_s^* as the informative reference points that are used in the generation of the training trajectory for the DNN inverse module.

The samples γ_s^* are incorporated into the spline trajectory generation algorithm as constraints at the interior points. In particular, we formulate the following problem:

$$\begin{aligned} \min_{p_1,\ldots,p_S} \quad & \sum_{s=1}^{S} p_s^T Q_s p_s \\ \text{s.t.} \quad & A_s p_s = \gamma_s^*, \quad \forall s = 1, ..., S, \\ & Ap = 0, \\ & A_{slt_i} p_s \leq b_{sl,\max}, \quad \forall t_i \in \mathcal{T}, \end{aligned} \tag{4.15}$$

where $\mathcal{T}$ is a set of sample times along the trajectory, and the inequality constraints introduced with the pairs $(A_{slt_i}, b_{sl,\max})$ can be used to account for additional feasibility constraints (e.g., bounds on velocities and accelerations).

Note that we optimize a continuous-time trajectory $T(t)$. The trajectory is discretized at a defined sampling interval Δt to obtain the sequence of references $\{u(k)\}$ for training.

4.6 Simulation Results

In this section, we use a numerical example to illustrate the proposed active training trajectory generation. We consider a minimum phase baseline system represented by

$$\begin{aligned} x(k+1) &= \begin{bmatrix} 0 & 1 \\ -0.15 & 0.8 \end{bmatrix} x(k) + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u(k) \\ y(k) &= \begin{bmatrix} -0.2 & 0.5 \end{bmatrix} x(k). \end{aligned} \tag{4.16}$$

Following (4.3), we can train a DNN module with input $\xi = [x(k), y_d(k+1)]$ and output $\gamma = u(k)$ to approximate the exact inverse of the baseline system (4.2). In Chapter 2, with a similar system, we showed that the DNN inverse learning approach can reduce the tracking error of the baseline system to approximately zero.

Although being effective at test time, the DNN module in our previous work was trained on hand-designed sinusoidal trajectories; the quality of the training could only

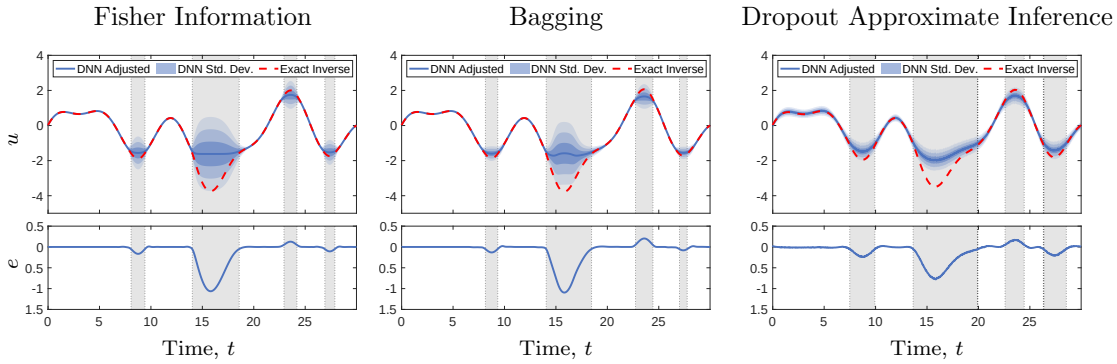


Figure 4.2: A comparison of the references u and the tracking errors, $e = y_d - y$, of three DNN-enhanced systems implemented based on different uncertainty estimation techniques: *(i)* Fisher information, *(ii)* bagging, and *(iii)* dropout approximate inverse. The results correspond to a test trajectory $y_d(t) = \frac{3}{5}\cos\left(\frac{\pi}{3}t\right) + \sin\left(\frac{\pi}{5}t\right) + \sin\left(\frac{\pi}{10}t\right) - \frac{3}{5}$. From the plots, it can be seen that at time steps where the DNNs are uncertain (blue shadings in the top row), we correspondingly observe relatively large tracking errors (regions shaded in grey); conversely, at time steps where the DNNs are certain, the tracking errors of the DNN-enhanced systems are close to zero (unshaded regions). Given the correlation between the DNN uncertainty and the tracking error, we can then exploit the uncertainty information to efficiently design trajectories for training the DNN inverse modules.

be validated when the DNN module is tested on the baseline system. In the following simulation study, we illustrate the active-learning-based framework for systematically designing DNN training trajectories. This framework allows us to *(i)* infer training quality prior to testing the DNN module on the system and *(ii)* efficiently collect data that is needed for good generalization. In the following subsections, we first present a set of simulations to examine the correlation between the predictive uncertainty of the DNN module and the system's tracking performance (Sec. 4.6.1) and then utilize the uncertainty estimates in the proposed active learning framework for DNN training trajectory design (Sec. 4.6.2).

4.6.1 DNN Predictive Uncertainty Estimation

In this subsection, we examine three techniques for estimating the DNN inverse dynamics module's predictive uncertainty: *(i)* Fisher-information-based estimation, *(ii)* bagging, and *(iii)* dropout approximate inference (see Sec. 4.4.2).

4.6.1.1 Architecture of the DNN Inverse Module

For the three techniques we implement in this simulation study, the DNNs are three-layer feedforward networks, and each hidden layer of the networks has ten hyperbolic tangent neurons. There are overall 161 weight and bias parameters in each network. The DNN modules are initially trained on a sinusoidal trajectory $y_d(t) = \sin\left(\frac{\pi}{10}t\right)$ sampled at $\Delta t = 0.015$. The training dataset is constructed from the system response data with paired input and output $\mathcal{D} = \{x(k), y(k+1); u(k)\}_{k=1}^{K}$, where K is the size of the training dataset. Standard SGD algorithms are used for optimizing the network parameters.

4.6.1.2 Implementation of the Different Uncertainty Estimation Techniques

The Fisher-information-based approach is implemented for a DNN module with a single network. The predictive uncertainty of the DNN module at test time is computed based on (4.5), where the module input and output are $\xi = [x(k), y_d(k+1)]$ and $\gamma = u(k)$, respectively. The inverse of the Fisher information matrix M is independent of the input ξ, and we pre-compute M^{-1} from the training data to minimize the computational load at test time. For the bagging approach, we use a committee of 20 DNNs trained with different randomly sampled initial parameters [109]. The uncertainty of the DNN module at test time is estimated based on the empirical variance of the committee outputs. For the dropout approximate inference approach, we train a DNN with stochastic dropout [76, 100]. The uncertainty of the DNN at test time is estimated using (4.6). Here, we use 300 forward passes with independent weight dropout samples, and the dropout probability is 0.05.

4.6.1.3 Simulation Results

We show the results of the DNN predictive uncertainty estimation techniques on a test trajectory $y_d(t) = \frac{3}{5}\cos\left(\frac{\pi}{3}t\right) + \sin\left(\frac{\pi}{5}t\right) + \sin\left(\frac{\pi}{10}t\right) - \frac{3}{5}$. This test trajectory differs from the training trajectory, and the DNN inputs encountered at test time are only partially covered by the training dataset. Fig. 4.2 shows the predictions of the DNN modules (top row) and the resulting tracking errors of the corresponding DNN-enhanced systems (bottom row). The time intervals where the systems have relatively larger tracking errors are shaded in grey. From the unshaded regions of the plots, we see that when the uncertainty of the DNN module is small, the reference computed by the DNN module coincides with the reference computed based on the exact inverse

of the system (4.2), and the tracking error of the DNN-enhanced system is close to zero. From the shaded regions, for each technique, we see a correlation between the uncertainties of the DNN modules and the tracking errors of the DNN-enhanced systems. Given this correlation, we can then exploit the uncertainty information to efficiently design trajectories for training the DNN inverse module.

4.6.2 Active Training Trajectory Generation

In Chapter 2, we verified the DNN inverse learning approach by training the DNN module using hand-designed trajectories. In this subsection, we illustrate the proposed active training trajectory generation with the system considered in Sec. 4.6.1.

4.6.2.1 Simulation Setup

In this simulation study, we examine four approaches for training trajectory generation:

(M1) *Baseline approach:* We consider a baseline training trajectory generation approach that resembles that shown in Chapter 2. In particular, in consecutive episodes, the DNN module is trained on sinusoidal trajectories with increasing amplitudes and frequencies. For the results in this subsection, the amplitudes of the sinuosoidal trajectories range from 1 to 5.5, and the frequencies range from 0.02 Hz to 0.2 Hz.

(M2) *Fisher-information-based approach:* In each episode, the uncertainty of the DNN module is estimated based on (4.5) and is used in the active trajectory generation framework proposed in Sec. 4.5.2. The DNN module is a three-layer feedforward network with ten hyperbolic neurons in each hidden layer.

(M3) *Bagging-based approach:* A committee of 20 DNNs is used to estimate the predictive uncertainty of the DNN module in each training episode, and the proposed active trajectory generation framework is similarly applied. The architectures of the DNNs are identical to that used in (M2). To reduce the computation load in the testing phase, we use a subset of five DNNs for reference computation.

(M4) *Dropout-inference-based approach:* A single network is trained with stochastic dropout, and the uncertainty of the DNN module is estimated based on the dropout approximate inference technique [100]. The DNN architecture is identi-

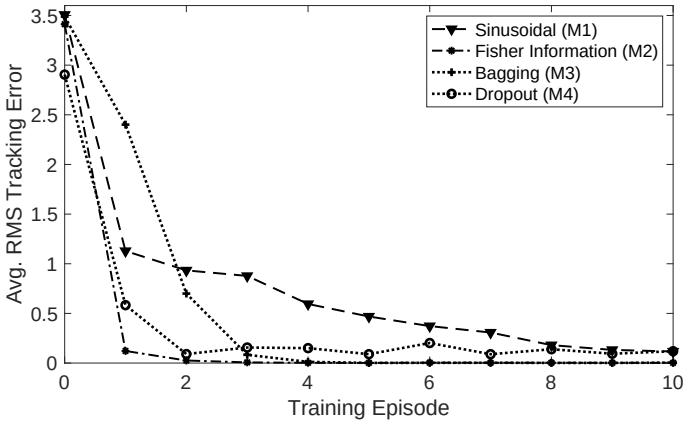


Figure 4.3: The average RMS tracking errors of the DNN-enhanced systems on ten test trajectories for an increasing number of the DNN training episodes.

cal to that used in (M2), and the proposed active training trajectory generation approach is similarly applied.

For approaches (M2)-(M4), we use a grid search to identify informative points for training the DNN modules. For system (4.16), the input space of the DNN module is $\xi = [x(k), y_d(k+1)]$. In order for the approach to be efficient, we restrict the grid search to a subset of the DNN input space that excludes regions with implausible combinations of $x(k)$ and $y_d(k+1)$.

4.6.2.2 Simulation Results

We test the generalizability of the DNN modules trained with the different training trajectory generation approaches on ten test trajectories. The test trajectories have the form of $y_d(t) = \alpha_1 \sin\left(\frac{2\pi}{\beta_1}t\right) - \alpha_2 \cos\left(\frac{2\pi}{\beta_2}t\right) + \alpha_2$, and the parameters (α_i, β_i) are randomly generated from uniform distributions $\alpha_i \sim \mathcal{U}(0,5)$ and $\beta_i \sim \mathcal{U}(5,50)$ for $i = \{1,2\}$, where $\mathcal{U}$ denotes a uniform distribution. We initialize the training with a sinusoidal trajectory $y_d(t) = \sin\left(\frac{\pi}{10}t\right)$. Fig. 4.3 shows the average root-mean-square (RMS) tracking error of the DNN-enhanced systems on the ten test trajectories after the algorithm in Alg. 4.3.1 is ran for an increasing number of training episodes. From the plot, it can be seen that all the training trajectory generation approaches are effective in the sense that the corresponding DNN-enhanced systems converge to small RMS tracking errors. In comparison, the DNN modules trained with the active training trajectory generation approaches (M2)-(M4) generally have faster convergence as compared to the baseline sinusoidal trajectory generation approach (M1). This means that a lower number of training trajectories is needed to learn an effective

inverse model.

It should be noted that even though, with (M1), the designed sinusoidal trajectories in the training dataset approximately cover the frequencies and amplitudes of the test trajectories, the DNN module has relatively high generalization errors in the last episodes. The imperfect result of (M1) is partially due to the fact that the references corresponding to the exact inverse of the system do not necessarily lie within the desired output space which the training trajectories are initially designed to cover. This mismatch between the exact references and desired outputs makes designing training trajectories based on strategies such as (M1) non-intuitive. The active trajectory generation approaches circumvent this issue by incorporating the learning module in the loop and actively identifying the uncertain references (the DNN outputs) that are required to cover the operational space of interest.

In addition to the improved learning efficiency, it should be noted that, with the active trajectory generation approaches, the evaluation of the utility function over the DNN input space in each episode provides us with a means to monitor or infer the quality of training prior to testing the DNN module on the baseline system. On the contrary, with typical hand-designed training trajectory generation approaches, the DNN training process is open-loop, and the quality of training may incorrectly be assumed to be good, which is unsafe for practical applications.

4.7 Conclusions

In this chapter, we introduced an active trajectory generation framework for systematically training DNNs that represent inverse dynamics modules deployed to enhance the tracking performance of black-box baseline systems. In simulation, we showed that, by using an active trajectory generation and training approach (Fig. 4.1), we can significantly improve the data efficiency for training the DNN inverse module. Moreover, the proposed active training trajectory generation framework allows us to infer the training quality of the DNN module prior to testing on the physical system, which can be important for safe operation in practical applications.

Chapter 5

Cross-Robot Experience Transfer to Improve the Performance of Similar Robots

5.1 Introduction

Machine learning techniques have been applied to many robot control problems with the goal of achieving high performance in the presence of uncertainties in the dynamics and the environment [111]. Due to the cost associated with data collection and training, approaches such as manifold alignment [112, 113, 114] and learning invariant features [115, 116] have been proposed to transfer knowledge between robots and thereby increase the efficiency of robot learning. In these approaches, datasets on a set of sample tasks are initially collected from both robots. They are then used for finding a mapping offline to transfer knowledge from a source robot to a target robot. This transferred knowledge is expected to speed up the training of the target robot and enhance its performance in untrained tasks [117].

In this chapter, we consider the problem of impromptu trajectory tracking, in which robots are required to track arbitrary trajectories accurately from the first attempt [48]. Model-based techniques such as model predictive control (MPC) or the linear-quadratic regulator (LQR) can be used to solve tracking problems; however, applying these techniques to achieve high tracking performance can be difficult as they rely on sufficiently accurate dynamics models or can be time-consuming to tune. In Chapter 2, we proposed a deep neural network (DNN)-based approach to enhance the tracking performance of black-box robot control systems. In particular, we showed

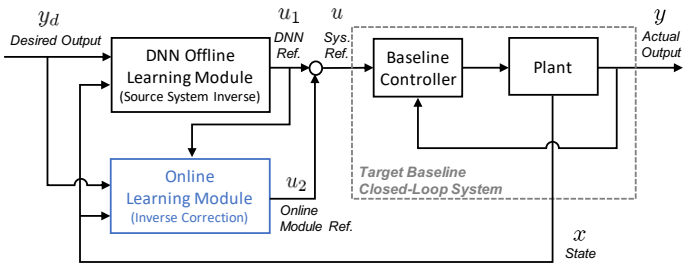


Figure 5.1: Block diagram of the DNN-enhanced control architecture with online learning. The DNN module represents the inverse dynamics of a source system and is previously trained offline with a sufficiently rich dataset. During the testing phase, the DNN module is leveraged to enhance the tracking performance of a target system that shares some dynamic similarities with the source system. An online learning module (trained based on small sets of real-time data) further adjusts the reference generated by the DNN module to allow the target system to achieve high-accuracy tracking on arbitrary trajectories from the first attempt (i.e., impromptu tracking). A video of this work can be found here: http://tiny.cc/dnnTransfer

that we can effectively enhance the tracking performance of a robot by training a DNN inverse dynamics module offline and then pre-cascading the module to the baseline system at test time.

Motivated by recent work in transfer learning, in this chapter, we study the feasibility of leveraging the DNN model trained on one robot to enhance the performance of another robot in impromptu tracking tasks. In contrast to the existing approaches, where transfer mappings are usually found offline (e.g., [113, 115]), we propose an online learning approach (Fig. 5.1) that allows a target robot using the DNN module from a source robot to achieve high-accuracy tracking *impromptu*—i.e., without additional data collection and training on sample tasks. With the online learning approach, we aim to significantly reduce the data recollection and training time usually required for enhancing the target robot performance. In this chapter, we (1) analytically derive the form of the mapping for the online module that allows the target system to achieve exact tracking, (2) present first results on characterizing system similarity between source and target systems and how it relates to the stability of the proposed overall learning system given modeling uncertainties, and (3) verify the effectiveness of the proposed approach in simulation and impromptu trajectory tracking experiments on quadrotors.

5.2 Related Work

The problem of knowledge transfer or transfer learning has been studied in different application domains (e.g., natural language processing [118], computer vision [119],

and robot control [113]). The common goal is to leverage existing data to accelerate and improve subsequent learning processes such that the costs (and potential risks) associated with data recollection can be reduced [120, 117]. In robotics, two directions of knowledge transfer have been considered: *(i)* transfer across tasks and *(ii)* transfer across robots. The former typically considers the transfer of knowledge from a source task to a target task to be performed by a single robot (e.g., [121, 122, 123]), while the latter considers the transfer of knowledge from a source robot to a target robot (e.g., [112, 113, 124, 114, 115, 116]). In this chapter, we will focus on the latter. We aim to transfer the inverse dynamics model trained on one robot to enhance the tracking performance of another robot. The transferred inverse dynamics model is expected to generalize to arbitrary trajectories.

In the robot learning literature, and especially in reinforcement learning (RL), different approaches have been proposed to address the problem of knowledge transfer across different robots or domains. One of the approaches for cross-domain transfer is manifold alignment, where data from the source and target systems are collected for a set of sample tasks and are mapped to corresponding feature spaces (e.g., through dimensionality reduction) from which a transformation mapping between the source and target systems is found. This offline mapping can then be used to translate the policies trained on the source robot to the policies for the target robot [112], or map the data collected on the source robot to the target robot for model learning [113]. Extension hereto [124, 114] derive an optimal mapping for data transfer across robots from a control theory perspective. Other related work aims to learn and exploit a common feature space between the source and target robots while performing similar tasks [115, 116]. In [116], it is shown that the approach can effectively transfer control policies across different quadrotor platforms for autonomous navigation.

In addition to the above, there are a few other lines of relevant work involving knowledge transfer. One of them is sim-to-real [125, 126], where the low-cost data from a simulation is exploited for accelerating the training on physical robots. Moreover, in meta learning, the learning parameters are optimized for initializing subsequent learning [127]. In [18], modularity in learning has also been proposed to maximize the utility of learned models.

Although recent literature demonstrates the possibility of transferring knowledge across robots, we address two additional aspects in our work. The first aspect is *impromptu* knowledge transfer without a-priori data collection on target systems. The second aspect is the impact of dynamic system similarity on the feasibility of knowledge transfer. An open question in the transfer learning literature is the issue

of negative transfer (i.e., when the transfer adversely affects the target system) [117]. While researchers have investigated task similarity in the context of task transfer problems [128], discussions on system similarity for transferring knowledge across robots are rare. In this chapter, we present theoretical results that associate system similarity to the feasibility of knowledge transfer across robots.

5.3 Problem Formulation

We consider the control architecture in Fig. 5.1 and study the knowledge transfer problem that allows the DNN module trained on a source robot system to enhance the impromptu tracking performance of a target robot system that has different dynamics. In this chapter, we consider closed-loop robot systems represented as

$$\begin{aligned} x(k+1) &= f\left(x(k)\right) + g\left(x(k)\right) u(k), \\ y(k) &= h\left(x(k)\right), \end{aligned} \tag{5.1}$$

where $k \in \mathbb{Z}_{\geq 0}$ is the discrete time index, $x \in \mathbb{R}^n$ is the state of the system, $u \in \mathbb{R}$ and $y \in \mathbb{R}$ are the input and output of the system, respectively, and $f(\cdot)$, $g(\cdot)$, and $h(\cdot)$ are smooth functions. We assume that:

(A1) The source and target systems are input-to-state stable [129].

(A2) The source and target systems *(i)* have well-defined and the same relative degree, and *(ii)* are minimum phase.

(A3) The desired trajectory y_d is bounded, and a preview of $y_d(k+r)$ is available at time step k.

Note that (A1) and (A2) are necessary for safe operations and for applying the DNN inverse learning. In (A2), we also assume that the source and target systems have the same relative degree to simplify the analysis. This condition holds, for instance, if the two robots have similar structures but different parameters (e.g., masses and dimensions). For (A3), the relative degree of a system is typically a small integer bounded by the system order, and a preview of r time steps of the desired trajectory can typically be achieved by online and offline trajectory generation algorithms.

5.4 Theoretical Results

In this section, we consider the control architecture in Fig. 5.1 and provide theoretical results related to the knowledge transfer problem. We denote u_1 as the reference from the DNN module trained on the source system and u_2 as the reference from the online learning module. The overall reference to the target baseline system $u(k)$ is given by

$$u(k) = u_1(k) + u_2(k). \tag{5.2}$$

Below we derive an expression of $u_2(k)$ for achieving exact tracking in Sec. 5.4.1, propose a characterization of system similarity in Sec. 5.4.2, and analyze the stability of the overall system in the presence of uncertainties in Sec. 5.4.3.

5.4.1 Reference Adaptation for Exact Tracking

In this subsection, we derive an expression for $u_2(k)$ such that $u(k)$ achieves exact tracking $y(k+r) = y_d(k+r)$, where y and y_d are the desired and actual outputs of the target system, and r is the system relative degree.

A common approach for high-accuracy trajectory tracking is to adapt the reference input of a nominal controller based on the observed tracking errors. For instance, in PD-type iterative learning control (ILC), proportional and derivative tracking error terms are added to the reference in each iteration to improve the tracking performance over a sequence of trials [130]. In distal teacher inverse dynamics learning, the tracking error is proposed as the cost function for updating the weights of a neural-network-based controller online to achieve improved tracking [53]. In this chapter, we similarly consider an online learning approach that adapts the reference of the DNN module $u_1(k)$ based on the tracking error. In particular, we justify below that the reference $u_2(k)$ can be approximated by

$$u_2(k) = \alpha\, e_p(k+r), \tag{5.3}$$

where α is an adaptation gain, and $e_p(k+r)$ is a prediction of the tracking error r time steps ahead.

As shown in Chapter 2, the input and output of a nonlinear target system (5.1) can be related as follows:

$$y(k+r) = \mathcal{F}_t\left(x(k)\right) + \mathcal{G}_t\left(x(k)\right) u(k), \tag{5.4}$$

where $\mathcal{F}_t\left(x(k)\right) = h_t \circ f_t^r(x(k))$ and $\mathcal{G}_t\left(x(k)\right) = \frac{\partial}{\partial u} h_t \circ f_t^{r-1}(f_t(x(k)) + g_t(x(k))u(k))$, and $f_t(\cdot)$, $g_t(\cdot)$, and $h_t(\cdot)$ are the corresponding nonlinear functions in (5.1).

In addition to the target system, we consider a source system, which the DNN module is trained on. The input-output equation of this system is similarly represented in the form of (5.4). The underlying function approximated by the DNN of the source system is

$$u_1(k) = \mathcal{G}_s^{-1}\big(x(k)\big)\big(y_d(k+r) - \mathcal{F}_s\left(x(k)\right)\big), \tag{5.5}$$

where $\mathcal{F}_s\left(x(k)\right)$ and $\mathcal{G}_s\left(x(k)\right)$ are defined analogously to those of the target system. By substituting (5.2) and (5.5) into (5.4), one can see that the ideal reference $u_2(k)$ for achieving exact tracking is

$$u_2(k) = \alpha^* e_p^*(k+r), \tag{5.6}$$

where $\alpha^* = \mathcal{G}_t^{-1}\left(x(k)\right)$ and

$$e_p^*(k+r) = y_d(k+r) - \mathcal{F}_t\left(x(k)\right) - \mathcal{G}_t\left(x(k)\right) u_1(k). \tag{5.7}$$

Remark 5.4.1 (Ideal Mapping for Exact Tracking)**.** *In order to achieve exact tracking, the online learning module should predict the tracking error of the target system that would result from applying $u_1(k)$. The predicted error is scaled by a gain $\alpha^* = \mathcal{G}_t^{-1}(x(k))$, where $\mathcal{G}_t(x(k)) = \frac{\partial y(k+r)}{\partial u(k)}$.*

The error prediction in (5.7) depends on the current state $x(k)$, the reference $u_1(k)$ from the DNN module, and the future desired output $y_d(k+r)$. When the dynamics of the source and the target systems are not known, one may use supervised learning to train a model online to approximate (5.7). We present a general approach for training this online model in Remark 5.4.2.

Remark 5.4.2 (Online Learning for Error Prediction)**.** *For training an online model to approximate (5.7), at each time step k, one may construct a dataset with paired inputs $\{x(p-r), u(p-r), y_d(p)\}$ and outputs $\{y_d(p) - y(p)\}$ over the past N time steps $p = k-N, ..., k$. The error $e_p(k+r)$ can then be predicted using the online model with input $\mathcal{I} = \{x(k), u_1(k), y_d(k+r)\}$.*

Given the predicted error $e_p(k+r)$, another component to be determined for computing $u_2(k)$ is the gain α. With an online model $F\left(x(k), u_1(k), y_d(k+r)\right)$ approximating (5.7), it can be shown that α^* can be obtained from $\hat{\alpha}^* = -\left(\partial F/\partial u_1\right)^{-1}$.

In practice, due to noise in the systems, the online estimation of α^* can be non-trivial. In Sec. 5.4.3, we provide an analysis to examine the stability of the overall system when α^* is approximated by a constant and also when the estimation of $e_p^*(k+r)$ by the online model is inexact.

5.4.2 System Similarity

The concept of task similarity has been introduced in the RL literature to address the issue of negative knowledge transfer in task transfer learning problems [128]. In this subsection, we propose a characterization of system similarity for impromptu knowledge transfer problems, where an inverse module is transferred across two robot systems.

We consider two systems are similar if at any given state $x(k)$, the application of an input $u(k)$ to the systems results in similar outputs $y(k+r)$ [131]. For the similarity discussion, we assume linear or linearized source and target systems to simplify our analysis:

$$\begin{aligned} x(k+1) &= Ax(k) + Bu(k), \\ y(k) &= Cx(k), \end{aligned} \tag{5.8}$$

where $x \in \mathbb{R}^n$ is the state, $u \in \mathbb{R}$ is the input, $y \in \mathbb{R}$ is the output, and $(A,\ B,\ C)$ are constant matrices. It can be shown that the input and output of system (5.8) are related by

$$y(k+r) = \mathcal{A}x(k) + \mathcal{B}u(k). \tag{5.9}$$

where $\mathcal{A} = CA^r$ and $\mathcal{B} = CA^{r-1}B$, and r is the relative degree of system (5.8). From (5.9), the input-output relationship is fully characterized by $\mathcal{A}$ and $\mathcal{B}$, which can be thought as the state-to-output gain vector and the input-to-output gain, respectively.

Based on the relationship in (5.9), we define a vector S to characterize the similarity of the source and target systems:

$$S = \begin{bmatrix} S_1 & S_2 \end{bmatrix}, \tag{5.10}$$

where $S_1 = 1 - \mathcal{B}_t\mathcal{B}_s^{-1}$, $S_2 = \mathcal{A}_t - \mathcal{B}_t\mathcal{B}_s^{-1}\mathcal{A}_s$, and the subscripts s and t denote the source and the target system. The terms S_1 and S_2, respectively, characterize the differences in the input-to-output gain and state-to-output gain vector of the source and target systems. Note that $S = 0$ if and only if $\mathcal{A}_t = \mathcal{A}_s$ and $\mathcal{B}_t = \mathcal{B}_s$ (i.e., the state-to-output and input-to-output gains of the systems are identical).

5.4.3 Stability in the Presence of Uncertainties

In this subsection, we use the concept of system similarity and analyze the stability of the target system when the gain α^* is approximated by a constant α and the prediction of the future error $e_p^*(k+r)$ is not exact. We focus on system (5.8) and make the following assumptions:

(A4) The output of the offline DNN $u_1(k)$ corresponds to the inverse of the source system $u_1(k) = \mathcal{B}_s^{-1}\left(y_d(k+r) - \mathcal{A}_s x(k)\right)$, where $\mathcal{A}_s$ and $\mathcal{B}_s$ are the gains of the source system, and $x(k)$ and $y_d(k+r)$ are the state and desired output of the target system.

(A5) The error in the prediction $\Lambda = e_p^*(k+r) - e_p(k+r)$ can be bounded as follows: $\Lambda \leq \beta_1||y_d(k+r)|| + \beta_2||x(k)|| + \beta_3$, where β_1, β_2, and β_3 are positive constants, and $||\cdot||$ is the Euclidean norm.

In addition, by (A1), the target system is input-to-state stable. It can be shown that the state of system (5.8) can be bounded as follows: $||x||_\infty \leq L_1||u||_\infty + L_2||x_0||$, where $||x||_\infty = \sup_k\{||x(k)||\}$, $||u||_\infty = \sup_k\{||u(k)||\}$, and L_1 and L_2 are positive constants.

Theorem 5.4.1 (Stability of the Target System with Transferred Inverse)**.** *Consider a target system represented by* (5.8) *and the control architecture in Fig. 5.1, where the reference of the online learning module $u_2(k)$ has the form of* (5.3)*. Under* (A1)*,* (A4)*, and* (A5)*, the overall system is bounded-input-bounded-state (BIBS) stable if*

$$|\alpha|\left(||S_2|| + \beta_1\right) < \frac{\beta_4}{L_1}, \tag{5.11}$$

where $\beta_4 = 1 - L_1\,||\mathcal{A}_s\mathcal{B}_s^{-1}||$.

Proof. At a time step k, the output of the online learning module is $u_2(k) = \alpha e_p(k+r)$, where α is a constant gain and $e_p(k+r)$ is the predicted tracking error. The adjusted reference $u(k)$ sent to the target baseline system is $u(k) = u_1(k) + \alpha\, e_p(k+r)$, where $u_1(k)$ is the output of the offline DNN module. By (A4) and (A5), we can write $u(k)$ as

$$u(k) = \mathcal{B}_s^{-1}\big(y_d(k+r) - \mathcal{A}_s x(k)\big) + \alpha\left(e_p^*(k+r) - \Lambda\right). \tag{5.12}$$

For a target system represented by (5.8), $e_p^*(k+r)$ in (5.7) can be written as $e_p^*(k+r) = y_d(k+r) - \mathcal{A}_t x(k) - \mathcal{B}_t u_1(k) = y_d(k+r) - \mathcal{A}_t x(k) - \mathcal{B}_t\mathcal{B}_s^{-1}\left(y_d(k+r) - \mathcal{A}_s x(k)\right)$. By substituting the expression of $e_p^*(k+r)$ into (5.12), we obtain $u(k) = \left(\mathcal{B}_s^{-1} + \alpha S_1\right) y_d(k+$

$r) - (\mathcal{A}_s\mathcal{B}_s^{-1} + \alpha S_2)\, x(k) - \alpha\Lambda$. Moreover, by (A1) and (A5), we can relate $||x||_\infty$ to $||y_d||_\infty = \sup_k\{||y_d(k)||\}$ by the following inequality:

$$\begin{aligned} ||x||_\infty \leq L_1 \Big(\big(\big|\mathcal{B}_s^{-1}\big| + |\alpha|\,|S_1| + \beta_1|\alpha|\big)\,||y_d||_\infty \\ + \big(\big|\big|\mathcal{A}_s\mathcal{B}_s^{-1}\big|\big| + |\alpha|\,||S_2|| + \beta_2|\alpha|\big)\,||x||_\infty\Big) + L_1\beta_3|\alpha| + L_2||x_0||. \end{aligned} \tag{5.13}$$

From (5.13), if $1 - L_1\left(||\mathcal{A}_s\mathcal{B}_s^{-1}|| + |\alpha|\,||S_2|| + \beta_2|\alpha|\right) > 0$, or equivalently $|\alpha|\left(||S_2|| + \beta_2\right) < \frac{\beta_4}{L_1}$, then the state of the system can be bounded as follows:

$$||x||_\infty \leq \frac{L_1\left(|\mathcal{B}_s^{-1}| + |\alpha|\,|S_1| + \beta_1|\alpha|\right)||y_d||_\infty + L_1\beta_3|\alpha| + L_2||x_0||}{1 - L_1\left(||\mathcal{A}_s\mathcal{B}_s^{-1}|| + |\alpha|\,||S_2|| + \beta_2|\alpha|\right)}. \tag{5.14}$$

Now, if y_d and hence $||y_d||_\infty$ are bounded, then the system state is bounded, and the overall system is BIBS stable. □

Recall that, in (5.11), α is the gain of the online learning module, S_2 characterizes the similarity between the two systems, β_1 is associated with the uncertainty in the error prediction, and L_1 can be thought of as a characterization of the aggressiveness of the target system. The condition in (5.11) can be interpreted for two scenarios: (i) when $|\alpha| = 0$ (i.e., the online module is inactive) and *(ii)* when $|\alpha| \neq 0$ (i.e., the online module is active). In scenario *(i)*, the condition in (5.11) reduces to $L_1 < \frac{1}{||\mathcal{A}_s\mathcal{B}_s^{-1}||}$, which can be interpreted as an upper bound on the relative aggressiveness of the source and target systems. When this condition is satisfied, the target system with the source system DNN module is stable. In scenario *(ii)*, when the online learning module is active, the condition in (5.11) implies that if the source and target systems are more similar, that is $||S_2||$ is closer to 0, then there will be a greater margin for selecting α and higher tolerance for having uncertainties in the online prediction model. Moreover, based on the condition in (5.11), one may use probabilistic learning techniques to estimate the uncertainties in the predicted error $e_p(k+r)$ and calculate an upper bound on the magnitude of the fixed gain α for stability.

Remark 5.4.3 (Nonlinear Systems)**.** *For nonlinear systems (5.1) with inputs and outputs related by (5.4), one can relate the outputs of the source and target systems by* $y_t(k+r) = \vartheta_1\left(x(k)\right) y_s(k+r) + \vartheta_2\left(x(k)\right)$*, where* $\vartheta_1\left(x(k)\right) = \mathcal{G}_t\left(x(k)\right)\mathcal{G}_s^{-1}\left(x(k)\right)$ *and* $\vartheta_2\left(x(k)\right) = \mathcal{F}_t\left(x(k)\right) - \mathcal{G}_t\left(x(k)\right)\mathcal{G}_s^{-1}\left(x(k)\right)\mathcal{F}_s\left(x(k)\right)$*. The relation between* $y_t(k+r)$ *and* $y_s(k+r)$ *can be used for characterizing the similarity for the nonlinear systems.*

5.5 Simulation Results

In this section, we illustrate the proposed online learning approach with a simulation example. In Chapter 2, we considered a minimum phase closed-loop baseline system represented by

$$\begin{aligned} x(k+1) &= \begin{bmatrix} 0 & 1 \\ -0.15 & 0.8 \end{bmatrix} x(k) + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u(k), \\ y(k) &= \begin{bmatrix} -0.2 & 1 \end{bmatrix} x(k), \end{aligned} \tag{5.15}$$

and showed that a DNN module can be designed to enable the system to achieve exact tracking on untrained trajectories. In the following simulation study, we consider system (5.15) as the source system and leverage its offline DNN module to enhance the tracking performance of a target system that is represented by

$$\begin{aligned} x(k+1) &= \begin{bmatrix} 0 & 1 \\ -0.24 & 1 \end{bmatrix} x(k) + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u(k), \\ y(k) &= \begin{bmatrix} -0.1 & 1 \end{bmatrix} x(k). \end{aligned} \tag{5.16}$$

Note that the source system (5.15) and the target system (5.16) are minimum phase and have relative degrees of 1. The source system has two poles at $\{0.3, 0.5\}$ and a zero at 0.2, while the target system has two poles at $\{0.4, 0.6\}$ and a zero at 0.1. When implementing the learning modules, we assume that the systems are black boxes, and we rely on only their input-output data and basic properties (e.g., relative degree).

5.5.1 Simulation Setup

In the proposed control architecture (Fig. 5.1), there are two learning modules placed in parallel to enhance the tracking performance of the target system: *(i)* an offline learning module that represents the inverse dynamics of the source system, and *(ii)* an online learning module that further adjusts the reference signals by predicting the future tracking error of the target system. The details the offline and online learning modules are discussed respectively in the following subsections.

5.5.1.1 Offline Learning of Inverse Module

The offline inverse module is trained on a source system (e.g., a system that is similar to the target system or a simulator), from which abundant data has been collected. The collected data can often be compactly represented by parametric regression tech-

niques. For the source system (5.15), we train the DNN module as outlined in Chapter 2 and transfer this inverse module to enhance the target system (5.16) with the proposed online learning approach. The DNN module of the source system is a 3-layer feedforward network with 20 hyperbolic tangent neurons in each hidden layer. The input and output of the DNN module are $\mathcal{I} = \{x(k), y_d(k+1)\}$ and $\mathcal{O} = \{u_1(k)\}$. The training dataset is constructed from the source system's response on 25 sinusoidal trajectories with different combinations of frequencies and amplitudes; Matlab's Neural Network Toolbox is used to train the DNN model parameters offline.

5.5.1.2 Error Prediction with Online Learning

The online error prediction module is a local model trained on a small dataset constructed from the latest observations of the target system. The objective of incorporating the online module is to achieve fast adaptation to the dynamic differences between the source and target systems. In the simulation, a Gaussian process (GP) regression model is utilized for learning the error prediction module online. Based on Remark 5.4.2, the input and output of the online module are selected to be $\mathcal{I} = \{x(k), u_1(k), y_d(k+1)\}$ and $\mathcal{O} = \{e_p(k+1)\}$, respectively. At each time step k, a fixed-sized training dataset is constructed based on the latest 15 observations; in particular, the input and output are $\{(x(p-1), u(p-1), y_d(p))\}$ and $\{y_d(p) - y(p)\}$ for $p = k-15, ..., k$. For the simulation, the GP model uses the squared-exponential kernel $K(\xi, \xi') = \sigma_1^2 \exp\left(-\frac{1}{2}\sum_i \frac{(\xi_i - \xi_i')^2}{l_i^2}\right)$ and polynomial explicit basis functions $\{1, \xi_i, \xi_i^2\}$, where ξ denotes the input to the module and ξ_i denotes the i-th component of ξ, l_i is the length scale associated with the input dimension ξ_i, and σ_1^2 is the prior variance [132]. The length scales l_i are identical for all input dimensions in the simulation; the hyperparameters of the kernel function and the coefficients of the basis functions are optimized online with Matlab's Gaussian Process Regression toolbox. The gain α^* is estimated based on the online error prediction module as $\hat{\alpha}^* = -\left(\partial F_{\text{gpr}}/\partial u_1\right)^{-1}$, where F_{gpr} denotes the function represented by the GP regression model.

5.5.2 Results

Figures 5.2 and 5.3 show the performance of the learning modules and the target system on a test trajectory $y_d(t) = \sin\left(\frac{2\pi}{8}t\right) + \cos\left(\frac{2\pi}{16}t\right) - 1$, where $t = 1.5 \times 10^{-3}k$ is the continuous-time variable. This test trajectory is not previously used in the training of the offline learning module.

Figure 5.2 compares the predicted error from the online module and the analytical

error prediction of the target system computed based (5.7). It can be seen that the online module designed based on Remark 5.4.2 is able to accurately predict the error of the target system that would result from applying the reference u_1 alone. On the test trajectory, the root-mean-square (RMS) error of the online module prediction is approximately 2.9×10^{-7}.

Figure 5.3 shows the outputs of the target system when *(i)* the baseline controller is applied (grey), *(ii)* the baseline system is enhanced by the offline module alone (green), and *(iii)* the baseline system is enhanced by both the online and the offline modules (blue). As compared to the baseline system, the offline module alone reduces the RMS tracking error of the target system from 3.97 to 0.44. The online module further reduces the RMS tracking error to 9×10^{-5}. Applying the offline and the online learning modules jointly allows the target system to achieve approximately exact tracking on a test trajectory that is not seen by the source or the target system a-priori.

5.6 Quadrotor Experiments

With impromptu tracking of hand-drawn trajectories as the benchmark problem [48], we illustrate the proposed online learning approach for transferring the DNN module trained on a source quadrotor system, the Parrot ARDrone 2.0, to a target quadrotor system, the Parrot Bebop 2. A demo video can be found here: http://tiny.cc/dnnTransfer

5.6.1 Experiment Setup

In Chapter 2, with the ARDrone as the testing platform, it is shown that a DNN module trained offline can effectively enhance the impromptu tracking performance of the quadrotor on arbitrary hand-drawn trajectories. In this chapter, we leverage the DNN module trained on the ARDrone to enhance the impromptu tracking performance of the Bebop and further apply the proposed online learning approach (Remark 5.4.2) to achieve high-accuracy tracking.

5.6.1.1 Control Objective and Baseline Control System

The dynamics of a quadrotor vehicle can be characterized by 12 states: translational positions $\mathbf{p} = (x, y, z)$, translational velocities $\mathbf{v} = (\dot{x}, \dot{y}, \dot{z})$, roll-pitch-yaw angles $\boldsymbol{\theta} = (\phi, \theta, \psi)$, and rotational velocities $\boldsymbol{\omega} = (p, q, r)$. The objective is to design a

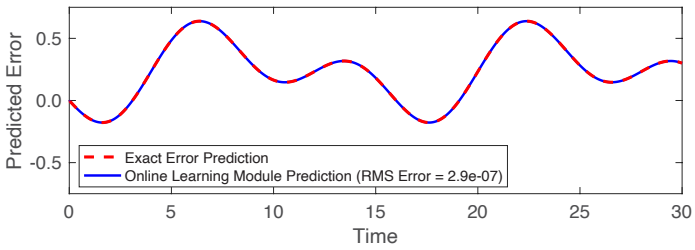


Figure 5.2: A plot of the error prediction from the online learning module. The error predicted by an online module trained based on Remark 5.4.2 (blue) coincides with the exact error prediction computed based on 5.7 (red).

control system such that the position of the quadrotor $\mathbf{p}_a$ tracks desired trajectories $\mathbf{p}_d$ generated from arbitrary hand drawings. In our work, we use the RMS error as the measure for evaluating tracking performance.

The baseline control systems of the quadrotor platforms, the ARDrone and the Bebop, have an offboard position controller running at 70 Hz and an onboard attitude controller running at 200 Hz. The offboard position controller receives the reference position $\mathbf{p}_r$ and reference velocity $\mathbf{v}_r$, and computes attitude commands ϕ_{cmd} and θ_{cmd}, yaw rate command r_{cmd}, and z-velocity command $\dot{z}_{\text{cmd}}$. The onboard attitude controller receives the commands from the offboard controller, and computes the desired thrusts of the four motors of the vehicle. In the experiments, we apply the offline and online learning modules to enhance the tracking performance of the baseline controller of the Bebop (the target system). In the design of the learning modules, we assume that the high-level dynamics of the ARDrone and the Bebop are decoupled in the x, y, and z directions.

5.6.1.2 DNN Module Trained on ARDrone (Source System)

In Chapter 2, a DNN module is trained offline to approximate the inverse of the ARDrone baseline system dynamics. Based on the theoretical insights in Chapter 2, the input and output of the DNN module are determined to be $\mathcal{I}_1 = \{x_d(k+4) - x_a(k), y_d(k+4) - y_a(k), z_d(k+3) - z_d(k), \dot{x}_d(k+3) - \dot{x}_a(k), \dot{y}_d(k+3) - \dot{y}_a(k), \dot{z}_d(k+2) - \dot{z}_a(k), \boldsymbol{\theta}_a(k), \boldsymbol{\omega}_a(k)\}$ and $\mathcal{O}_1 = \{\mathbf{p}_r(k) - \mathbf{p}_a(k), \mathbf{v}_r(k) - \mathbf{v}_a(k)\}$. The DNN module consists of fully-connected feedforward networks with 4 hidden layers of 128 rectified linear units (ReLU). The training dataset of the DNN module is constructed from the ARDrone baseline system response on a 400-second, 3-dimensional sinusoidal trajectory. At a sampling rate of 7 Hz, approximately 2,800 pairs of data points are collected for training. The DNN module is implemented using Tensorflow in Python.

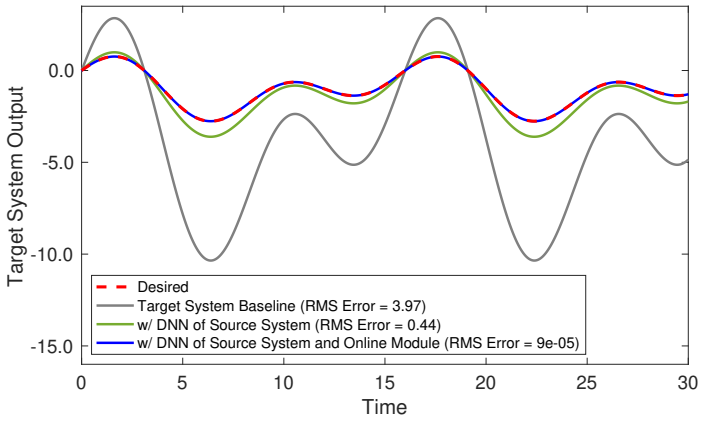
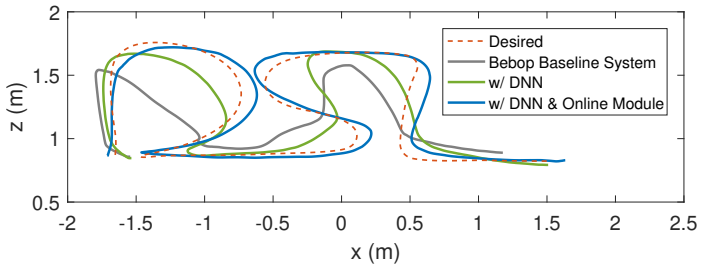


Figure 5.3: Plots of the target system output when *(i)* the baseline controller (grey), *(ii)* the baseline controller with the offline learning module (green), and *(iii)* the baseline controller with both the offline and online learning modules (blue) are applied. Due to system similarity, the offline learning module (trained on the source system) significantly reduces the tracking error of the target system. With the further incorporation of the online learning module, exact tracking is approximately achieved.

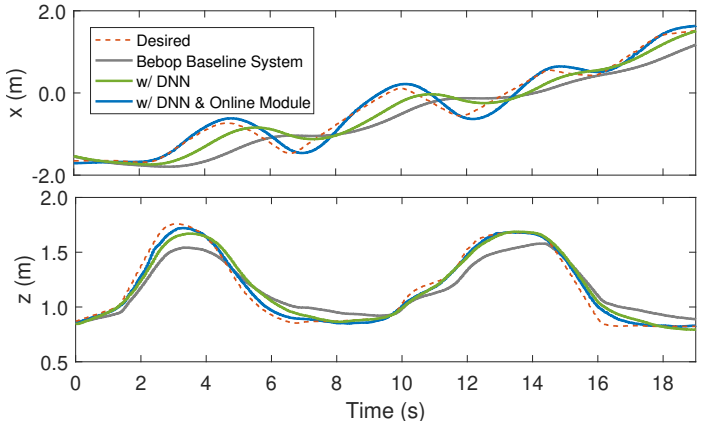
As shown in Sec. 2.6, for 30 hand-drawn test trajectories, this offline DNN module is able to reduce the impromptu tracking error of the ARDrone baseline system by 62% on average.

5.6.1.3 Online Learning for Bebop (Target System)

Based on Remark 5.4.2, the input and output of the online learning module are $\mathcal{I}_2 = \{\mathbf{p}_a(k), \mathbf{v}_a(k), \boldsymbol{\theta}_a(k), \boldsymbol{\omega}_a(k), \mathbf{p}_r(k), \mathbf{v}_r(k), x_d(k+4), y_d(k+4), z_d(k+3), \dot{x}_d(k+3), \dot{y}_d(k+3), \dot{z}_d(k+2)\}$ and $\mathcal{O}_2 = \{x_e(k+4), y_e(k+4), z_e(k+3), \dot{x}_e(k+3), \dot{y}_e(k+3), \dot{z}_e(k+2)\}$, where $(\cdot)_e$ denotes the predicted position and velocity tracking errors of the Bebop system when the offline DNN trained on the ARDrone system is used. In the experiment, in order to make online learning more efficient, instead of predicting the position and velocity errors directly, we train a GP model to predict the position of the Bebop $\mathbf{p}_a(k+r) = [x_a(k+4), y_a(k+4), z_a(k+3)]$ and computes the predicted error by subtracting the predicted position from future desired position $\mathbf{p}_d(k+r) - \mathbf{p}_a(k+r)$, where $\mathbf{p}_d(k+r) = [x_d(k+4), y_d(k+4), z_d(k+3)]$. The predicted position errors are used to compute the corrections for the position components; the velocity reference corrections are numerically approximated with a first-order finite difference scheme. For the experiments, the online learning module is implemented by using the GPy library in Python. We use a standard squared-exponential kernel with a fixed length scale l for all input dimensions, prior variance σ_1^2, and zero mean Gaussian measurement noise with variance σ_2^2 [132]. At each time step k, the most recent 40



(a) Path of the target system in the x-z plane.



(b) Position trajectories of the target system.

Figure 5.4: Comparison of three control strategies for the Bebop target system: The RMS error is 0.42 m for the Bebop baseline system (grey), 0.26 m for the baseline system enhanced by the ARDrone DNN (green), and 0.14 m for the baseline system further enhanced by the online learning module (blue).

observations are used for constructing the training dataset. The hyperparameters of the GP model are $l = 20$, $\sigma_1^2 = 1$, and $\sigma_2^2 = 2 \times 10^{-5}$; these values are manually tuned a-priori for our experimental setup. If computational resources permit, we expect finer tuning of the hyperparameters online would lead to lower generalization errors and better tracking performance. Due to the measurement noise in the experiment, instead of estimating the parameter α online, we used constant gains $\alpha = (5, 5, 0.5)$ for the x, y, and z directions.

5.6.2 Results

Figure 5.4 compares the tracking performance of three control strategies on the Bebop on one of the test hand-drawn trajectories. When comparing the performance of the Bebop system enhanced by the ARDrone DNN (green) and the performance of the Bebop baseline system (grey), the ARDrone DNN reduces the delay and the

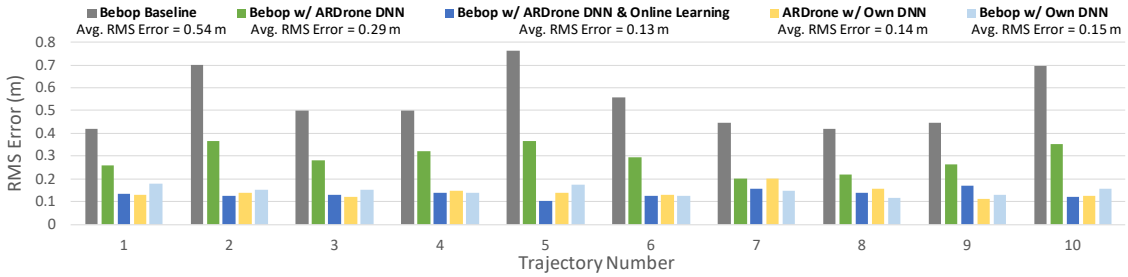


Figure 5.5: Tracking performance of the target system (Bebop) on 10 hand-drawn trajectories. The ARDrone DNN module alone (green) and the ARDrone DNN module with the online learning module (blue) reduce the tracking error of the Bebop baseline system (grey) by 46% and 74% on average, respectively. With the proposed online learning approach, the average RMS error of the Bebop (blue) is comparable to cases where the ARDrone and the Bebop are enhanced by their own offline DNN modules (yellow and light blue).

amplitude errors in the Bebop tracking response. Along this particular trajectory, the DNN module alone reduces the RMS tracking error of the Bebop from approximately 0.42 m to 0.26 m. When further comparing with the performance of the DNN-enhanced system with the addition of the online learning module (blue), the tracking of the Bebop, especially in the x-direction, is brought close to the desired trajectory. With the online learning module, the RMS tracking error is reduced to approximately 0.14 m. Note that, from the plots in Fig. 5.4, when the online learning module is applied, there are small overshoots at the locations with larger curvatures. The overshoots may be reduced with online tuning of the GP hyperparameters and online estimations of the α parameters.

Figure 5.5 summarizes the performance errors of the three control strategies on 10 hand-drawn trajectories. When compared with the Bebop baseline system performance (grey), the direct application of the transferred DNN module (green) reduces the RMS tracking error of the Bebop baseline system by an average of 46%. With the addition of the online learning module (blue), an average of 74% RMS tracking error reduction is achieved. Two additional sets of results are included for comparison: *(i)* the performance of the ARDrone enhanced by the DNN module trained on the ARDrone system (yellow) and *(ii)* the performance of the Bebop enhanced by a DNN module trained on the Bebop system (light blue). Without requiring further data collection and offline training, the inclusion of the online learning module effectively reduces the RMS tracking error of the Bebop to values that are comparable to those of the cases where the quadrotors are enhanced by their own offline DNN modules. These results demonstrate the efficiency of the proposed online learning

module to leverage past experience and reduce data re-collection and training.

5.7 Conclusions

In this chapter, we consider the impromptu tracking problem and propose an online learning approach to efficiently transfer a DNN module trained on a source robot system to a target robot system. In the theoretical analysis, we derive an expression of the online module for achieving exact tracking. Then, based on a linear system formulation, we propose an approach for characterizing system similarity and provide insights on the impact of the system similarity on the stability of the overall system in the knowledge transfer problem.

We verify our approach experimentally by applying the proposed online learning approach to transfer a DNN inverse dynamics module across two quadrotor platforms (Parrot ARDrone and Bebop). On 10 arbitrary hand-drawn trajectories, the DNN module of the source system reduces the tracking error of the target system by an average of 46%. The incorporation of the online module further reduces the tracking error and leads to an average of 74% error reduction. These experimental results show that the proposed online learning and knowledge transfer approach can efficaciously circumvent data recollection on the target robot, and thus, the costs and risks associated with training new robots to achieve higher performance in impromptu tasks.

Chapter 6

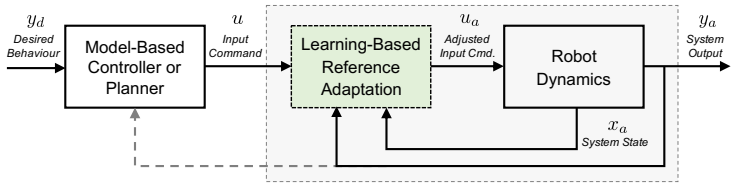
Lipschitz Network Adaptation to Bridge the Model-Reality Gap

6.1 Introduction

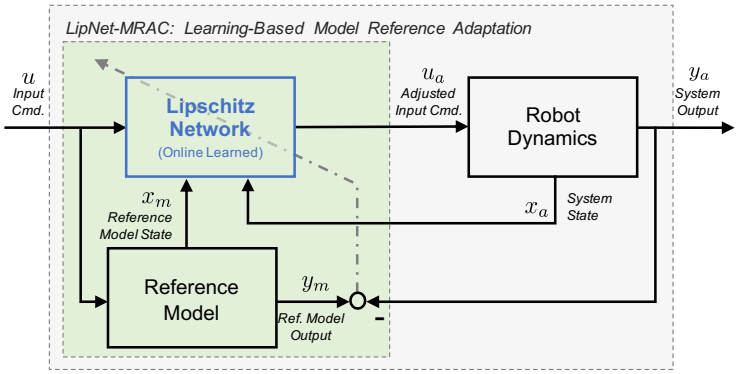
Advances in hardware and algorithms have enabled robots to enter more complex environments and perform increasingly versatile tasks such as home and healthcare services, search and rescue, aerial package delivery, and industrial inspections. In these applications, robots need to cope with unmodeled dynamics, external time-varying disturbances, and other adverse factors such as communication latency. These practical issues pose challenges to the design of controllers using standard model-based techniques.

In the literature, common model-based control techniques include, but are not limited to, model predictive control (MPC) and linear quadratic regulators (LQR). These approaches are effective when the dynamics model of the robot system is sufficiently accurate and the operating environment does not change significantly over time. When these conditions are not met, model-based designs can lead to suboptimal or unsafe behaviour [3]. While there exist robust approaches that account for uncertainties by considering worst-case scenarios, these robust techniques can be often overly conservative [133, Ch. 17].

An alternative approach to cope with dynamics uncertainty is to enable the system to adapt. One particular set of adaptive approaches is model reference adaptive control (MRAC), which aims to make the controlled system behave similarly to a desired reference model despite unknown disturbances [134]. Although classical adaptive control approaches techniques provide stability guarantees, they usually assume a



(a) High-level block diagram



(b) Detailed block diagram of the model reference adaptation module

Figure 6.1: Block diagrams of the proposed approach. The grey box represents the robot system equipped with the proposed learning-based model reference adaptation module (green box). The reference sent to the robot system is adapted online by a Lipschitz network (blue box) such that the response from the input u to the output y_a resembles the response of a reference model, which can be used in the outer model-based controller or planner. A video of the flying inverted pendulum experimental results can be found here: http://tiny.cc/lipnet-pendulum

particular system structure that limit the range of robotic applications to which they can be applied [134].

Inspired by recent advances in machine learning techniques, we propose a novel learning-based MRAC approach that bridges the model-reality gap and enables us to leverage the power and simplicity of model-based control techniques, even in dynamic and uncertain conditions. In particular, we consider the hierarchical architecture illustrated in Fig. 6.1, where a low-level adaptive module (green box) modifies the input to the system such that the system's input-output response resembles that of the reference model and a high-level controller is designed based on the reference model to achieve a desired robot behaviour. In contrast to existing MRAC approaches such as [134, 135, 136], we leverage the expressive power of deep neural networks (DNNs) to capture a broader range of unmodelled dynamics and guarantee stability by exploiting the Lipschitz property of a special type of DNN called Lipschitz network

(LipNet) [19]. In this chapter, we (1) present a Lipschitz network adaptive control approach that makes a nonlinear robot system, with possibly unknown dynamics, behave as a reference model, (2) derive a condition that guarantees stability of the proposed approach by exploiting the Lipschitz properties of the LipNet module, and (3) experimentally demonstrate the efficacy of the proposed approach for bridging the model-reality gap by balancing an inverted pendulum on a quadrotor platform despite dynamics uncertainties.

6.2 Related Work

The rich literature on model-based control approaches shows the effectiveness, safety, and simplicity of these techniques for cases when the dynamics model of system is accurate and the operating environment is static. The model-reality gap is a crucial factor that prevents traditional model-based approaches to be directly applicable to robot systems that are subject to uncertain dynamics and disturbances. One can think of three approaches to address the model-reality gap [3]: *(i)* robustness, *(ii)* adaptation, and *(iii)* anticipation.

The *robustness* approach aims to design control laws that are stable for a range of unknown dynamics and disturbances that may affect the robotic system. Robust control approaches include, but are not limited to, sliding-mode control [137], robust MPC [138], H_∞ control [139], as well as more recent domain randomization techniques for sim-to-real transfer [126]. While robust approaches typically guarantee stability and safety in the presence of unmodelled dynamics and disturbances, their performance can be conservative.

In contrast, *adaptation* approaches address the model-reality gap by adapting or learning online using data collected by the robot. Adaptive controllers such as MRAC [134] and $\mathcal{L}_1$ adaptive control [140] are fast and able to handle unmodelled dynamics. In order to further improve performance, learning-based controllers that leverage past experience are being proposed. Non-parametric approaches include learning-based controllers using Gaussian Processes (GPs) [141], which leverage past experience to learn a better system model, but can be computationally expensive resulting in a slow response to changes in the environment. While there are newer learning MPC approaches using Bayesian linear regression (BLR) [142], formal guarantees are not given.

Anticipation approaches address the model-reality gap by learning offline. In DNN-based inverse control, a mapping from desired output to actual output is learned

offline and used to improve tracking performance of a quadrotor. In reinforcement learning (RL), latent variables that represent the environment are used to anticipate changes in the environment for off-policy RL [143]. While anticipation approaches are effective for addressing the uncertainties for a broad range of systems, they typically lack the adaptivity to cope with changes during real-time execution.

Adaptive controllers handle unmodelled dynamics and disturbances without the need for conservative control laws or significant amounts of past experience to learn offline. Due to their expensiveness and fixed cost for online inference, neural networks (NNs) are emerging as attractive options for implementing adaptive frameworks on resource-constrained robot platforms. Neural networks have previously been used in online inverse control, but they suffered from a lack of robustness against disturbances [53] and the need for appropriate initialization in order to converge [55]. They have also been used to relax the assumptions of conventional MRAC (e.g., [144]). However, earlier studies often use radial basis function (RBF) NNs, which require a sufficient preallocation of basis functions over the operating domain; the desired theoretical guarantees do not hold outside of the targeted operating domain. Recently, an asynchronous DNN MRAC framework was proposed to mitigate the limitation of RBF NNs by learning "features" at a slower timescale [145]; but, the approach only considers systems with additive input uncertainties.

In this chapter, we consider a more general class of control-affine nonlinear systems and leverage the expressiveness of DNNs to learn complex dynamic uncertainties. The stability of the adapted system is guaranteed by exploiting the Lipschitz properties of the Lipschitz network. We demonstrate our approach in flying inverted pendulum experiments.

6.3 Problem Formulation

We consider robot systems whose dynamics can be represented in the following form:

$$\begin{aligned} x_a(k+1) &= f_a(x_a(k)) + g_a(x_a(k))\, u_a(k), \\ y_a(k) &= h_a(x_a(k)), \end{aligned} \tag{6.1}$$

where the subscript a denotes the actual robot system, $k \in \mathbb{Z}_{\geq 0}$ is the discrete-time index, $x_a \in \mathbb{R}^n$ is the system state, $u_a \in \mathbb{R}$ is the system input, $y_a \in \mathbb{R}$ is the system output, and f_a, g_a, and h_a are smooth nonlinear functions that are possibly unknown a priori. Our goal is to design a learning-based control law such that the robot behaves as a reference model, which can subsequently be leveraged when designing

the outer-loop controller or planner.

The reference model can have the following form:

$$\begin{aligned} x_m(k+1) &= f_m(x_m(k)) + g_m(x_m(k))\, u_m(k), \\ y_m(k) &= h_m(x_m(k)), \end{aligned} \tag{6.2}$$

where the subscript m denotes the reference model, the reference model state x_m, input u_m, and output y_m are defined analogously as in (6.1), and f_m, g_m, and h_m are smooth nonlinear functions. Note that the reference model has a generic control-affine form. Practically, one could use a nonlinear reference model that best captures our prior knowledge about the robot system. Alternatively, to simplify the outer-loop controller design, one may choose a linear reference model

$$\begin{aligned} x_m(k+1) &= A_m x_m(k) + B_m u_m(k), \\ y_m(k) &= C_m x_m(k), \end{aligned} \tag{6.3}$$

where (A_m, B_m, C_m) are constant matrices with consistent dimensions, and use well-established linear control tools.

We consider a control architecture as shown in Fig. 6.1b. Without loss of generality, we assume that the inputs to the robot system and the reference model are

$$u_a(k) = u(k) + \delta u(k) \text{ and } u_m(k) = u(k), \tag{6.4}$$

where $u(k)$ is the input command computed by the outer-loop model-based controller. The objective of model reference adaptation is to learn the input adjustment $\delta u(k)$ such that the output of the robot system (6.1), $y_a(k)$, tracks the output of the reference system (6.2), $y_m(k)$.

We make the following assumptions: *(i)* the dynamics of the robot system is minimum phase (i.e., has stable forward and inverse dynamics) and has a well-defined relative degree, and *(ii)* the reference model is stable and has the same relative degree as the robot system. As discussed in Chapter 2, the first assumption is necessary to safely apply an inverse dynamics learning approach and is satisfied by closed-loop stabilized robot systems such as quadrotors and manipulators. The second assumption is also not restrictive, as the relative degree of a robot system can be estimated from experiments or inferred from our prior knowledge, and the reference model can be designed to satisfy this assumption.

Note that the choice of reference model is generally problem-dependent. For

instance, it can be chosen to achieve the desired transient or steady-state response or maximize the stability margin of the system. There is usually a tradeoff in selecting a reference model. Choosing a fast reference model could potentially favour the design of the outer-loop controller, while choosing a slower reference model would make the adaptation task easier. In practice, one should choose a sufficiently fast reference model to fulfill the desired higher-level objective and ensure that the reference model is feasible for the robot to follow.

6.4 Methodology

In this section, we present our proposed LipNet-based MRAC (LipNet-MRAC) approach to enforce a robot to behave as a predefined reference model. To facilitate our discussion, in Sec. 6.4.1, we present a brief background on the LipNet [19]. In Sec. 6.4.2, we derive an ideal model reference adaptation law based on the dynamics model of the robot system. In Sec. 6.4.3, we introduce an online algorithm to learn the model reference adaptation law with a LipNet when the robot dynamics are unknown, and in Sec. 6.4.4, we derive a Lispchitz condition that guarantees stability of the proposed LipNet-MRAC approach.

6.4.1 Background on Lipschitz Networks

In contrast to conventional feedforward networks whose Lipschitz constants are often difficult to estimate [65], LipNets have exact, predefined Lipschitz constants that the designer can choose freely [19]. Setting and knowing the Lipschitz constant is critical for guaranteeing stability of NN-based control frameworks [16, 146].

We consider an M-layer neural network $F_\theta(\xi)$ that can be expressed as follows:

$$F_\theta(\xi) = W^M \sigma(W^{M-1}\sigma(\dots\sigma(W^1\xi + b^1)) + b^{M-1}) + b^M\,, \tag{6.5}$$

where ξ is the input of the network, $\{W^1, W^2, ..., W^M\}$ are the weights matrices, $\{b^1, b^2, ..., b^M\}$ are the bias vectors, θ denotes an augmented vector of the network weight and bias parameters, and $\sigma(\cdot)$ is the activation function.

In contrast to conventional networks, LipNets enforce exact Lispchitz constraints by ensuring that the input-output gradient norm is preserved by each linear and activation layer: $||J_l^T z_l|| = ||z_l||$, where z_l and J_l are the input and the input-output Jacobian of layer l, and $||\cdot||$ is the Euclidean norm of a vector. To realize gradient norm preservation, [19] proposes to *(i)* orthonormalize the weight matrices in each linear

layer such that the weight matrices have singular values of 1 exactly, and *(ii)* use a gradient-preserving activation function *GroupSort* that sorts the input to the hidden layer. More specifically, the *GroupSort* activation function divides the input to the hidden layer into groups and sorts the values of each group in ascending or descending order. As an example, with full sort, we have $[1, 2, 3, 4]^T = \text{GroupSort}([3, 2, 4, 1]^T)$.

Since the *GroupSort* activation function only permutates the inputs to the layer, the input-output gradient norm of the GroupSort layer is 1. By design, the overall network has a Lipschitz constant of 1. The 1-Lipschitz network can be extended to approximate a function with an arbitrary Lipschitz constant by scaling the output of the network by the desired Lipschitz constant [19]. In contrast to spectral normalization approaches, where the weight matrices of the network are scaled by their spectral norms, LipNets have exact Lipschitz constants, which reduces the conservatism for imposing Lipschitz constraints.

6.4.2 Model Reference Adaptation Law

In this subsection, using the representations of the robot system (6.1) and the reference model (6.2), we derive the model reference adaptive law to be approximated by the LipNet.

To facilitate our discussion, we introduce the notion of system relative degree. We define $f_a \circ g_a$ as the composition $f_a(g_a(\cdot))$ of the functions f_a and g_a, and f_a^i as the ith composition of the function f_a with $f_a^0(x) = f_a(x)$ and $f_a^i(x) = f_a^{i-1}(x) \circ f_a(x)$. As discussed in Chapter 2, a nonlinear system (6.1) is said to have a relative degree of r, if r is the smallest integer such that $\frac{\partial}{\partial u_a} h_a \circ f_a^{r-1}(f_a(x) + g_a(x)u_a(x)) \neq 0$ in a neighbourhood of an operating point $(\bar{x}_a, \bar{u}_a)$. Intuitively, for a discrete-time system, the relative degree r defines the number of sample delays between applying an input u_a to the system and seeing a corresponding change in the output y_a.

By leveraging the definition of relative degree, we can relate the input u_a and output y_a of the robot system (6.1):

$$y_a(k + r) = \mathcal{F}_a(x_a(k)) + \mathcal{G}_a(x_a(k))\, u_a(k), \tag{6.6}$$

where $\mathcal{F}_a(x_a(k)) = h_a \circ f_a^r(x_a(k))$ and $\mathcal{G}_a(x_a(k)) = \frac{\partial}{\partial u_a} h_a \circ f_a^{r-1}(f_a(x_a(k)) + g_a(x_a(k))u_a(k))$. This input-output relationship allows us to predict the future output of the robot system $y_a(k + r)$ based on the current input $u_a(k)$ and state $x_a(k)$.

By assuming that the reference model is designed to have the same relative de-

gree r, we can similarly derive the input-output equation of the reference system:

$$y_m(k+r) = \mathcal{F}_m(x_m(k)) + \mathcal{G}_m(x_m(k))\, u_m(k), \tag{6.7}$$

where $\mathcal{F}_m(x_m(k))$ and $\mathcal{G}_m(x_m(k))$ are defined analogously to $\mathcal{F}_a(x_a(k))$ and $\mathcal{G}_a(x_a(k))$ for (6.1). For a linear reference system (6.3), the input-output equation reduces to

$$y_m(k+r) = \mathcal{A}_m x_m(k) + \mathcal{B}_m u_m(k), \tag{6.8}$$

where $\mathcal{A}_m = C_m A_m^r$ and $\mathcal{B}_m = C_m A_m^{r-1} B_m$.

Recall the architecture in Fig. 6.1, where the inputs to the robot system and the reference model are defined by (6.4). In order to make the robot system behave like the reference model, we enforce the outputs of the robot system and the reference model to be identical. In particular, by setting $y_a(k+r) = y_m(k+r)$ and solving for $\delta u(k)$, one can show that the ideal input adjustment $\delta u(k)$ for model reference adaptation is

$$\delta u(k) = \mathcal{G}_a^{-1}(x_a(k))\big(\mathcal{F}_m(x_m(k)) - \mathcal{F}_a(x_a(k)) + \mathcal{G}_m(x_m(k))\, u(k)\big) - u(k). \tag{6.9}$$

When the robot dynamics is unknown, we can treat the ideal adjustment (6.9) as a nonlinear function that maps from the robot system state $x_a(k)$, the reference model state $x_m(k)$, and the input signal $u(k)$ to the input adjustment $\delta u(k)$:

$$\delta u(k) = F_\theta(x_a(k), x_m(k), u(k)). \tag{6.10}$$

6.4.3 Online Learning of the Model Reference Adaptation Law

In this subsection, we outline an online learning algorithm to discover the ideal adaptation law (6.9) *via* a Lipschitz network when the robot dynamics (6.1) is not known.

We define the performance error of the neural network as the difference between the output of the robot system (6.1) and the output of the reference model (6.2): $E(k) = y_m(k+r) - y_a(k+r)$. At each time step k, the parameters of the Lipschitz network are updated to minimize the squared error cost function:

$$\mathcal{J}(k) = \frac{1}{2} E^2(k) = \frac{1}{2} \left(y_m(k+r) - y_a(k+r)\right)^2. \tag{6.11}$$

We use the following gradient-based approach to update the network parameters,

$\theta(k+1) = \theta(k) + \Delta\theta(k)$. The change in the network parameters is:

$$\Delta\theta(k) = -\lambda \nabla_\theta \mathcal{J}(k) = \lambda H(k) G(k) E(k), \tag{6.12}$$

where $\lambda > 0$ is the learning rate, $G(k) = \nabla_\theta F_\theta(x_a(k), x_m(k), u_m(k))$ is the gradient of the network output with respect to its parameters, and $H(k) = \nabla_{u_a} y_a(k+r)$ is the input-output gradient of the robot system.

To realize the online adaptation law (6.12), we need to predict the system output $y_a(k+r)$ and estimate the input-to-output gradient $\nabla_{u_a} y_a(k+r)$. Similar to [53, 147], we can simultaneously learn a forward model for the robot system to estimate $y_a(k+r)$ and $\nabla_{u_a} y_a(k+r)$ (see (6.6)):

Remark 6.4.1 (Forward Model Learning)**.** *At time k, one can construct a paired dataset with inputs $\{y_a(p-r), u_a(p-r)\}$ and outputs $\{y_a(p)\}$ based on the latest N time steps $p = \{k-N, ..., k\}$ and use standard supervised learning to train a forward model (e.g., a BLR model) as a local approximator of (6.6). The model can be then used to estimate $y_a(k+r)$ and $\nabla_{u_a} y_a(k+r)$ by setting the input to $(x_a(k), u_a(k))$.*

We note that inaccuracies in the forward dynamics model could, in general, lower the adaptation performance but will not jeopardize the stability of the adapted system. As will be shown in Sec. 6.4.4, the stability of the proposed LipNet-MRAC approach is guaranteed if the Lipschitz constant of the LipNet satisfies a small-gain-type condition.

In the case where a prediction model is not available, one could still apply the proposed algorithm for model reference adaptation but with a sample delay of r steps, which is typically a small integer for robot systems such as quadrotors. For a linear system, $\nabla_{u_a} y_a(k+r)$ is a constant that can be factored into the learning rate λ as a tuning parameter, and its estimation is not required.

6.4.4 Stability Analysis

In this subsection, we provide stability guarantees of the system including the model reference adaptation law by exploiting the Lipschitz property of the learning module. In the stability analysis, we make the following assumptions:

(A1) The state of the robot system can be bounded by $||x_a||_l \leq \gamma ||u_a||_l + \beta$, where γ and β are positive constant scalars, $||\cdot||_l$ denotes the l_2 signal norm, and the variables without the time indices (k) denote the corresponding signals.

(A2) The input adjustment computed by the adaptation module satisfies $\delta u(k) = 0$ for $(x_a(k), x_m(k), u(k)) = (0, 0, 0)$.

(A3) The state of the reference system x_m is bounded (i.e., $||x_m||_l < \infty$).

Assumption (A1) holds for finite-gain l_2 stable systems and is common assumption in small-gain-type theorems, which are the basis of the proof presented below. The scalar γ is an upper bound on the input-to-state gain of the robot system, and the scalar β is a constant value associated with the initial state of the robot system. As shown in the proof below, β affects the upper bound on the state of the system but does not impact the stability of the adapted system. Assumption (A2) is true for any robot and reference systems satisfying $\mathcal{F}_a(0) = 0$ and $\mathcal{F}_m(0) = 0$. This condition is not restrictive and can be practically enforced by removing the bias vectors from the LipNet architecture. Assumption (A3) can be satisfied by a proper choice of stable reference system.

Theorem 6.4.1 (Stability of the LipNet-MRAC Approach)**.** *Consider the proposed LipNet-MRAC approach shown in Fig. 6.1 (grey box). Under assumptions* (A1)-(A3)*, the dynamics of the adapted system from u to x_a is finite-gain l_2 stable if $L < 1/\gamma$, where L is the Lipschitz constant of the LipNet, which we are free to choose, and γ is an upper bound on the input-to-state gain of the robot system.*

Proof. By assumption (A1), the state of the robot system can be bounded as follows: $||x_a||_l \leq \gamma||u_a||_l + \beta = \gamma||u + \delta u||_l + \beta \leq \gamma||u||_l + \gamma||\delta u||_l + \beta$. Moreover, by assumption (A2) and the Lipschitz property of the LipNet, at any instance, the input adjustment computed by the LipNet can be bounded as $||\delta u(k)|| \leq L||\xi(k)||$, where L is the Lipschitz constant of the network, and $\xi = [x_a^T, x_m^T, u]^T$ denotes the network input. It follows that $||\delta u||_l = \left(\sum_{k=0}^{\infty} ||\delta u(k)||^2\right)^{1/2} \leq \left(\sum_{k=0}^{\infty} L^2||\xi(k)||^2\right)^{1/2} = L||\xi||_l \leq L||x_a||_l + L||x_m||_l + L||u||_l$. Using the upper bound on $||\delta u||_l$, we obtain $||x_a||_l \leq \gamma(1 + L)||u||_l + \gamma L||x_a||_l + \gamma L||x_m||_l + \beta$. It can be shown that, if $L < 1/\gamma$ is satisfied, the state of the robot system can be bounded by $||x_a||_l \leq \big(\gamma(1 + L)||u||_l + \gamma L||x_m||_l + \beta\big)/(1 - \gamma L)$. Since, by assumption (A3), $||x_m||_l$ is bounded, the dynamics of the adapted system from u to x_a is finite-gain l_2 stable [67] (cf. Fig. 6.1b). □

Theorem 6.4.1 provides an upper bound on the Lipschitz constant of the adaptive network module to guarantee stability. This Lipschitz condition can be enforced via the architecture design of the LipNet (Sec. 6.4.1). To enforce the Lipschitz condition, we require an estimate of the upper bound of the system gain γ, which can be

estimated from system input-output data [68, 69] or chosen conservatively based on our prior knowledge of the system. Overestimating γ will lead to a smaller, more conservative Lipschitz constant for the LipNet, but the overall adapted system will remain stable. However, if the system gain γ is underestimated, there is no guarantee that the overall system is stable.

6.5 Simulation Results

In this section, we present a numerical example to illustrate the proposed LipNet-MRAC approach.

6.5.1 Simulation Setup

We consider the following system:

$$\begin{aligned} x_a(k+1) &= \begin{bmatrix} 1 & T \\ -T & 1-T \end{bmatrix} x_a(k) + d(x_a(k)) + \begin{bmatrix} 0 \\ 0.6T \end{bmatrix} u(k), \\ y_a(k) &= \begin{bmatrix} 1 & 1 \end{bmatrix} x_a(k), \end{aligned} \tag{6.13}$$

where $d(x_a(k)) = 0.1T \begin{bmatrix} x_{a,1}(k)\sin(x_{a,1}(k)), & 0 \end{bmatrix}^T$ with $T = 0.01$ and $x_{a,1}(k)$ being the first element of $x_a(k)$. The gain of system (6.13) has an upper bound of $\gamma = 1.12$. The system (6.13) has a relative degree of 1. The reference model is

$$\begin{aligned} x_m(k+1) &= \begin{bmatrix} 1 & T \\ -0.25T & 1-T \end{bmatrix} x_m(k) + \begin{bmatrix} 0 \\ T \end{bmatrix} u_m(k), \\ y_m(k) &= \begin{bmatrix} 0.25 & 0.25 \end{bmatrix} x_m(k)\,. \end{aligned} \tag{6.14}$$

The reference system (6.14) also has a relative degree of 1.

Our goal is to design an adaptive module such that the system output (6.13) tracks the output of the reference model (6.14). In the discussion below, we first illustrate the efficacy of using the proposed adaptive LipNet-MRAC approach to make a nonlinear system (6.13) behave as a linear reference system (6.14) without knowing the dynamics model of the nonlinear system a priori. We then show the benefit of using the proposed LipNet-MRAC approach by comparing it to a learning-based MRAC approach with a conventional feedforward network architecture (NNet). Both the LipNet and the NNet have a depth and width of 3 and 20. The LipNet has

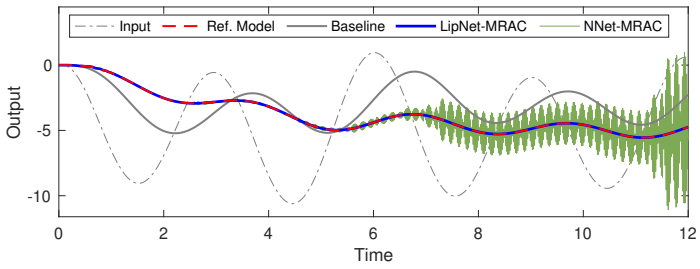


Figure 6.2: The proposed LipNet-MRAC approach effectively enforces the input-output response of system (6.13) to behave as the the reference model (6.14) (blue and red). The baseline response without adaptation is shown in grey. In contrast, with the conventional NNet-MRAC (green), closed-loop stability is not guaranteed. This simulation corresponds to one test input trajectory $u(k) = \sin\left(\frac{2\pi}{5}kT\right) + 5\cos\left(\frac{2\pi}{3}kT\right) - 5$ with $T = 0.01$. The learning rate is set to 33 for both approaches.

FullSort hidden layers and orthogonalized linear layers [19], while the NNet has tanh hidden layers and standard linear layers. The same adaptation scheme (Sec. 6.4.3) is applied to update the network parameters. The initial parameters of the networks are randomly sampled from the standard normal distribution. We compare the two approaches over ten randomly-initialized trials. To satisfy Theorem 6.4.1 with the proposed LipNet-MRAC approach, the Lipschitz constant of the LipNet is set to $1/\gamma = 0.89$ to guarantee stability.

6.5.2 Results

Figure 6.2 shows the response of the system (6.13) when using *(i)* the proposed LipNet-MRAC approach, and *(ii)* a learning-based MRAC approach with a conventional feedforward network architecture (abbreviated as NNet). By comparing the baseline response of system (6.13) (grey line) and the response of system (6.13) with the adaptive LipNet (blue line), we can see that the proposed approach effectively enforces the dynamics of system (6.13) to behave as the reference model (red dashed line) as desired. With the conventional NNet (green line), stability is not guaranteed.

Figure 6.3 compares the adaptation error when different learning rates are used with NNet and the proposed LipNet. For each learning rate, the plot shows the mean and standard deviation of the root-mean-square (RMS) error over ten randomly-initialized trials. NNet has one ideal learning rate for which the mismatch between the system and the reference model is the lowest. Searching for this ideal learning rate requires trial-and-error and the system can be destabilized for higher values. For the LipNet-MRAC approach, the stability of the system is not jeopardized, regardless

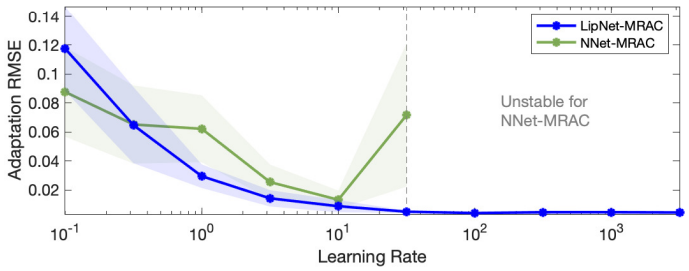


Figure 6.3: The performance of the proposed LipNet-MRAC approach and the NNet-MRAC on one test trajectory when different learning rates are used. Using the LipNet-MRAC approach, we can always guarantee stability, and the adaptation performance asymptotically approaches a lower bound as the learning rate increases. However, with the NNet-MRAC approach, there is an ideal learning rate that needs to be carefully chosen, which can be challenging to find when we do not know the robot dynamics a priori. The solid lines and the shades show the means and one standard deviations for ten trials with different initial network parameters.

of the chosen learning rate. Higher learning rates generally allow for faster adaptation to any mismatches between the reference model and the robot system. As a result, the adaptation RMSE for the LipNet-MRAC case asymptotically approaches an ideal value as the learning rate increases. By encoding the Lipschitz condition (Theorem 6.4.1) in the LipNet design, we can safely increase the learning rate for faster adaptation while guaranteeing stability a priori despite network parameter initialization.

6.6 Experimental Results

We demonstrate the proposed LipNet-MRAC approach through flying inverted pendulum experiments. A video of the quadrotor experiments presented in this section can be found here: http://tiny.cc/lipnet-pendulum

6.6.1 Experimental Setup

The goal of the experiment is to stabilize an inverted pendulum on a quadrotor vehicle (the Bebop) while hovering and tracking a trajectory in the xy-plane. The state of the quadrotor consists of the translational positions of its centre of mass (COM) (p_x, p_y, p_z), the translational velocities (v_x, v_y, v_z), the roll-pitch-yaw Euler angles (ϕ, θ, ψ), and the angular velocities $(\omega_x, \omega_y, \omega_z)$. We model the pendulum as a point mass [148]. To capture the dynamics of the pendulum, we define four additional states $(r, s, \dot{r}, \dot{s})$, which correspond to the positions and velocities of the COM of the

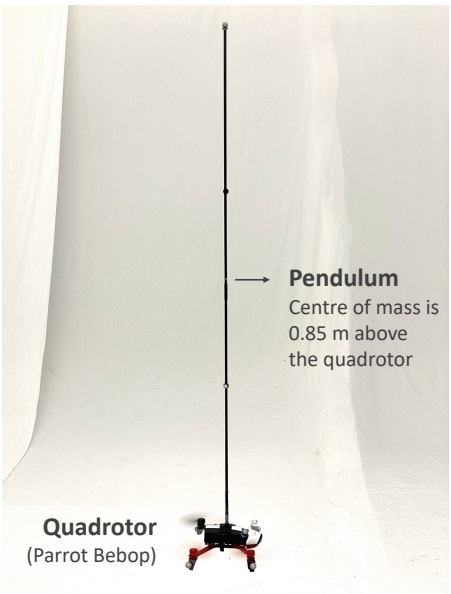


Figure 6.4: We demonstrate our proposed approach using a flying inverted pendulum, where a quadrotor (Parrot Bebop) balances a pendulum while hovering at a fixed point or tracking a trajectory.

pendulum relative to the positions and velocities of the COM of the quadrotor along the x and the y axes—the pendulum is balanced in the upright position when both r and s are zero. An illustration of the experimental setup is shown in Fig. 6.4.

By assuming that the quadrotor is stabilized at a constant height (i.e., $v_z = 0$), we can represent the translational dynamics of the flying inverted pendulum system in the form below:

$$x_a(k+1) = f_a(x_a(k)) + g_a(x_a(k))\, a_a(k), \tag{6.15}$$

where the state $x_a = [p_{x,a}, v_{x,a}, r_a, \dot{r}_a, p_{y,a}, v_{y,a}, s_a, \dot{s}_a]^T$ is an augmentation of the pertinent states of the quadrotor and the pendulum, and the input $a_a = [a_{a,x}, a_{a,y}]^T$ is the actual acceleration of the quadrotor [148].

To design a stabilizing controller for the quadrotor-pendulum system in (6.15), one could first design a controller to compute the required acceleration of the quadrotor to stabilize the quadrotor-pendulum dynamics (6.15) and then use an inner-loop attitude controller to ensure that the desired acceleration is achieved [148]. However, in our experiments, we do not have access to the attitude control of the off-the-shelf quadrotor. We instead apply the proposed LipNet-MRAC approach outside of the attitude control loop to make the acceleration dynamics of the quadrotor behave as

a predefined reference model:

$$a_m(k+1) = A_m a_m(k) + B_m u_m(k), \tag{6.16}$$

where $u_m = [u_{m,x}, u_{m,y}]^T$ is the acceleration command. The reference model is then incorporated into the overall quadrotor-pendulum dynamics model as an extended system:

$$\xi_a(k+1) = \begin{bmatrix} f_a(x_a(k)) + g_a(x_a(k))a_a(k) \\ A_m a_a(k) \end{bmatrix} + \begin{bmatrix} 0 \\ B_m \end{bmatrix} u(k), \tag{6.17}$$

where $\xi_a = [x_a^T, a_a^T]^T$ is the state of the extended system, and the input u is the acceleration command of the quadrotor. In our experiments, the parameters of the reference model were chosen to maximize the time-delay margin of the augmented system (i.e., the maximum number of sample delays that the system can tolerate before becoming unstable). Note that, as discussed in Chapter 2, we can estimate the relative degree of an uncertain robot system from simple experiments. In our case, the robot system and the reference model have a relative degree of 1.

Given the model in (6.17), we can use a standard model-based controller to design a feedback control law for stabilizing the quadrotor-pendulum system. In this work, we use a standard linear quadratic regulator (LQR) of the form $u(k) = K\widetilde{\xi}_a(k)$, where K is the controller gain designed based on (6.17), $\widetilde{\xi}_a(k)$ is the error in the extended state relative to a desired state, which is constant for stabilization tasks and time-varying for tracking tasks. Note that, to compensate for the input-output delay present in the quadrotor system, we introduced a lead compensator with a forward prediction in the closed-loop system. Similar to the LQR controller, the parameters of the lead compensator are determined based on (6.17).

To ensure that the acceleration dynamics of the quadrotor follow the reference model (6.16), we assume decoupled quadrotor acceleration dynamics in the x- and y-directions and use the proposed LipNet-MRAC approach outlined in Sec. 6.4. In the experiments, the adaptive LipNets have depths of 3 and widths of 20. By observing the input-output responses of the baseline quadrotor attitude controller on a set of sinusoidal trajectories, the quadrotor system gain γ is estimated to be 0.68. Based on Theorem 6.4.1, we conservatively set the Lipschitz constant of the LipNets to 0.8. To train the LipNet online, we simultaneously fit a local BLR model to approximate the forward acceleration dynamics. The parameters of the LipNet are updated to minimize the cost (6.11) with $\lambda = 0.8$.

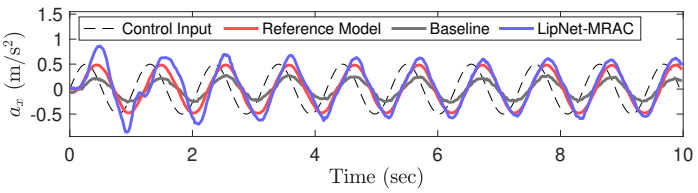


Figure 6.5: Given the control input (dashed line), the proposed LipNet-MRAC approach allows the actual acceleration of the quadrotor (blue) to closely follow the output of the reference model (red). The response of the baseline system without adaptation is shown in grey. A similar result is observed for the acceleration tracking in the y-direction.

With the proposed LipNet-MRAC, the acceleration command from the LQR controller is adjusted by the adaptive LipNet (Fig. 6.1b) and the overall acceleration command sent to the quadrotor is $u_a(k) = u(k)+\delta u(k)$, where $\delta u(k)$ is the adjustment computed by the LipNet. Using the Euler parameterization of the attitude angles, the acceleration command $u = [u_x, u_y]^T$ is converted to the attitude commands based on the following transformations: $\theta_c = \arctan(u_x/g)$ and $\phi_c = \arctan\left(-u_y/\sqrt{u_x^2+g^2}\right)$, where θ_c and ϕ_c are the commanded pitch and roll angles, and g is the acceleration due to gravity. The attitude commands are sent to the Bebop quadrotor onboard controller at a rate of 50 Hz.

Our experiments consist of *(i)* verifying efficacy of the proposed LipNet-MRAC for making the quadrotor system behave as a reference model and *(ii)* demonstrating the LipNet-MRAC in closed-loop control for a flying inverted pendulum.

6.6.2 LipNet-MRAC for Predictable Acceleration Dynamics

We first show that the proposed LipNet-MRAC can make the acceleration dynamics of the Bebop quadrotor behave as different predefined reference models. For simplicity of the outer-loop controller design, we choose linear reference acceleration models of the following form:

$$a_m(k+1) = \begin{bmatrix} \beta_{m,x} & 0 \\ 0 & \beta_{m,y} \end{bmatrix} a_m(k) + \begin{bmatrix} \alpha_{m,x} & 0 \\ 0 & \alpha_{m,y} \end{bmatrix} u_m(k), \tag{6.18}$$

where $\tau_m = (\alpha_{m,x}, \beta_{m,x}, \alpha_{m,y}, \beta_{m,y})$ are model parameters.

To illustrate the idea of our proposed approach, we first set the reference model parameters to $\tau_m = (0.35, 0.65, 0.35, 0.65)$. Figure 6.5 shows the quadrotor system response with the baseline controller, and with the LipNet-MRAC on one test tra-

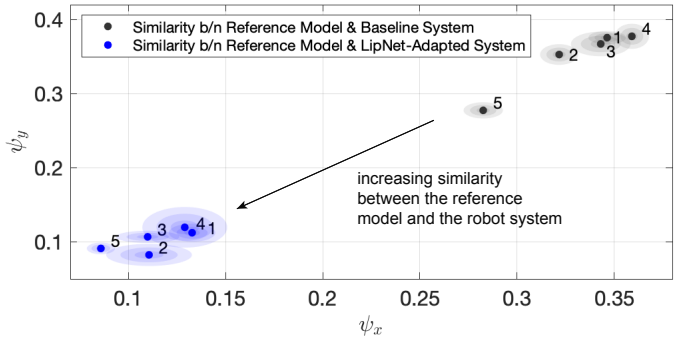


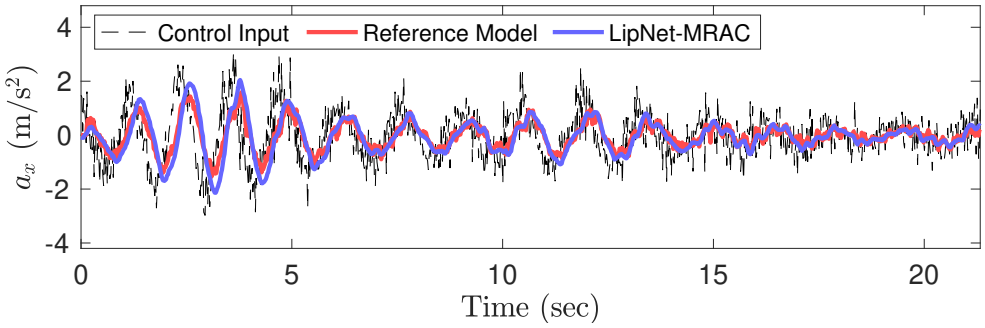
Figure 6.6: The proposed LipNet-MRAC approach can effectively enforce the robot system to behave as the randomly selected reference models. The plot shows a comparison of the system similarity between the five reference models and *(i)* the system with the baseline controller (grey), and *(ii)* the system with the proposed LipNet-MRAC module (blue). Smaller values of ψ_x and ψ_y indicate a higher system similarity between the reference model and the system in terms of the ν-gap metric [133]; the dots and shaded areas in the plot correspond to the means and 3σ error bounds of the ν-gap estimates obtained based on the algorithm outlined in [149].

jectory. As can be seen from the plot, the adaptive LipNet brings the acceleration response of the quadrotor system close to the given reference model.

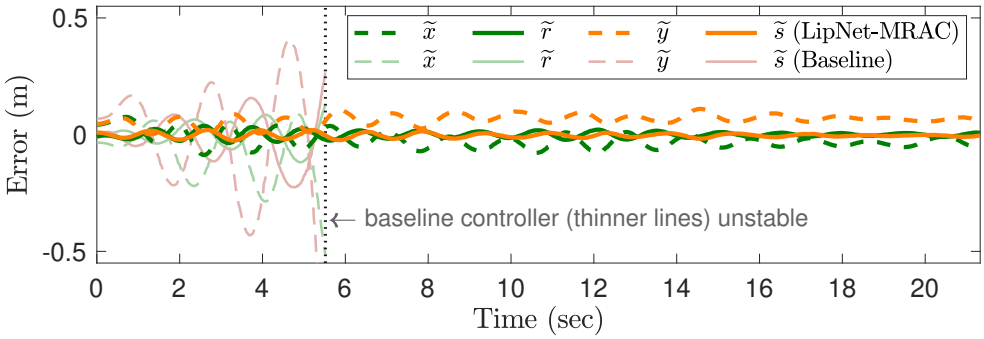
To further demonstrate the efficacy of the LipNet-MRAC approach, we randomly sample five sets of model parameters τ_m and apply the LipNet without any fine tuning of the learning algorithm parameters. To formally evaluate the performance of the reference model adaptation approach, we use the ν-gap metric from robust control [133] to measure the 'distance' between the reference model and the quadrotor system response with and without LipNet adaptation. Intuitively, two dynamical systems that are close in term of the ν-gap can be stabilized by the same controller. Figure 6.6 shows the estimated ν-gap metric using experimental data from the quadrotor and the iterative algorithm outlined in [149]. A smaller ν-gap value indicates a higher similarity between the reference model and the quadrotor system. The plot shows that the LipNet-MRAC approach can reliably make the quadrotor system behave close to the five reference models. In the next subsection, we apply LipNet-MRAC to the flying inverted pendulum problem.

6.6.3 Inverted Pendulum on a Quadrotor Experiments

An LQR stabilization controller is designed based on the dynamics in (6.17), where the reference acceleration model has the form of (6.18). In the controller design process, we expect that the quadrotor system behaves as the reference model; we

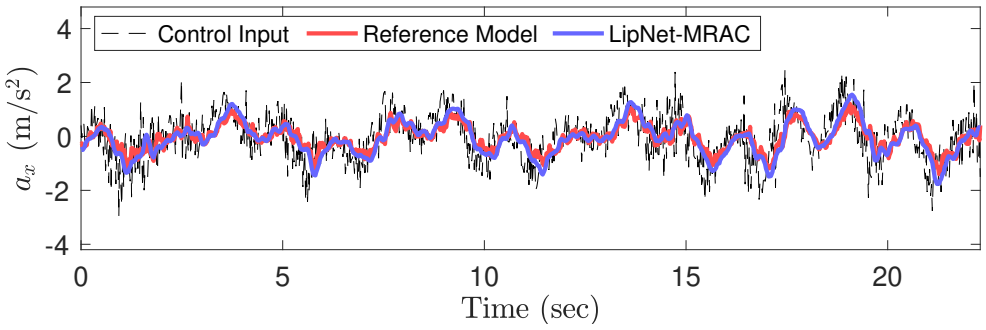


(a) Performance validation of the underlying LipNet-MRAC module. The actual acceleration of the quadrotor (blue) closely follows the output acceleration of the reference model (red). Similar result is observed for a_y. The control input signal (black) is generated by the high-level LQR controller.

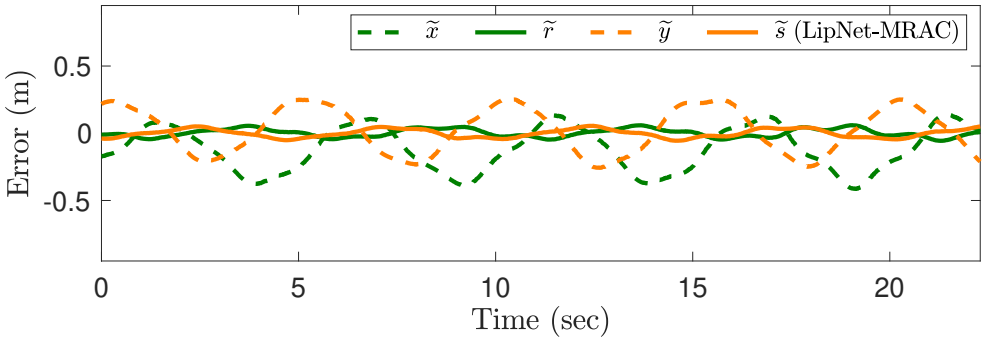


(b) The error of the quadrotor positions $\widetilde{x}$ and $\widetilde{y}$ (dashed lines) and the pendulum relative positions $\widetilde{r}$ and $\widetilde{s}$ (solid lines) when the quadrotor balances the pendulum while hovering at a fixed position. The proposed LipNet-MRAC approach (thicker lines) enables the pendulum to be balanced in the upright position despite dynamics uncertainties. Without the LipNet-MRAC, due to the model-reality gap, the baseline controller alone (thinner lines) cannot stabilize the flying inverted pendulum system.

Figure 6.7: Quadrotor balancing a pendulum while hovering.



(a) Performance validation of the underlying LipNet-MRAC module. The actual acceleration of the quadrotor (blue) closely follows the output acceleration of the reference model (red). Similar result is observed for a_y. The control input signal (black) is generated by the high-level LQR controller.



(b) The LipNet-MRAC approach allows the quadrotor to balance the pendulum while tracking a circular trajectory with a radius of 0.25 m and angular velocity of 1.25 rad/sec. The RMS error in the quadrotor positions (dashed lines) and the pendulum positions (solid lines) are 0.27 m and 0.04 m, respectively.

Figure 6.8: Quadrotor balancing a pendulum while tracking a trajectory.

do not need to explicitly model the acceleration dynamics of the quadrotor system or modify the default attitude controller onboard of the quadrotor platform. Our experiments encompass the following tests: *(i)* pendulum stabilization, *(ii)* pendulum stabilization with wind and tap disturbances, and *(iii)* pendulum stabilization while tracking circular trajectories.

We first show results for the case when the quadrotor is commanded to hover at a fixed point while balancing the pendulum. Figure 6.7a shows the acceleration response of the quadrotor system. It can be seen that, as desired with the LipNet-MRAC, the actual acceleration of the quadrotor follows the output of the reference model. As compared to the baseline system of the quadrotor, the acceleration reference model has an input-to-output gain closer to unity, which facilitates the outer-loop LQR controller design. Figure 6.7b shows the resulting errors in the pendulum and quadrotor system. Given the predictable behaviour of the acceleration dynamics, we see that the outer-loop pendulum controller can successfully balance the pendulum while keeping the quadrotor position error close to zero. The RMS error in the quadrotor and pendulum positions are 0.08 m and 0.02 m, respectively. On the contrary, if we use the baseline controller alone, there is a model-reality gap and the overall system is not stable (lighter lines in Fig. 6.7b). As we demonstrate in the supplementary video, the proposed LipNet-MRAC-based controller design is even able to maintain the pendulum in the upright position when wind disturbances are applied to the quadrotor or a gentle force is applied to the pendulum.

Next, we show the case when the quadrotor is commanded to track a circular trajectory of radius 0.25 m and angular frequency 1.25 rad/sec while balancing a pendulum. Figure 6.8a shows the acceleration response of the quadrotor system, which closely tracks the output of the reference model despite the sharp changes in the input signal. Figure 6.8b shows the position errors of the pendulum and the quadrotor. The quadrotor is able to track the circular trajectory while keeping the pendulum balanced. The RMS error in the quadrotor and pendulum positions are 0.27 m and 0.04 m, respectively. In the supplementary video, we show that the quadrotor can successfully track circular trajectories with angular frequencies up to 2.09 rad/sec, while keeping the pendulum balanced.

6.7 Conclusions

In this chapter, we presented a neural model reference adaptive approach (LipNet-MRAC) to make nonlinear systems with possibly unknown dynamics behave as a

predefined reference model. By leveraging the representative power of DNNs, the proposed approach can be applied to a larger class of nonlinear systems than other approaches in the literature. Moreover, we derive a certifying Lipschitz condition that guarantees the stability of the overall adaptive LipNet framework. We applied the proposed approach to a flying inverted pendulum. Our experiments show that the proposed approach is able to make the dynamics of an unknown black-box quadrotor system behave in a predictable manner, which facilitates the outer-loop pendulum stabilization controller synthesis. By complementing a standard controller with the proposed LipNet-MRAC, we successfully stabilized an inverted pendulum with an off-the-shelf quadrotor platform whose dynamics are not known a priori.

Chapter 7

Summary and Future Work

In this dissertation, we explored learning-based control approaches that combine control theory and machine learning for high-performance robot control system designs. While there are various machine learning techniques that we could have used in our control architecture, we focus on methods that exploit the modeling capability of neural networks and derive theoretical insights to guide safe and efficient implementations of neural networks for enhancing the performance of robot control systems. In contrast to typical learning-based control techniques, in our work, we leverage the baseline control system available to the robot and thereby make the overall learning approach more data-efficient and less prone to instability issues—both are critical for real-world applications.

7.1 Novel Contributions of This Thesis

In Chapter 2, we introduced a novel neural network inverse dynamics learning approach to enhance the trajectory tracking performance of robot control systems. We used control theory to derive the input and output that are required for the neural network to learn the full inverse map as well as conditions that are necessary for the approach to be safely applied to physical robot systems. In experiments, we demonstrated that, by using our theoretical insights, we effectively trained a neural network using offline data to approximate the inverse dynamics of a quadrotor system; over 30 arbitrary hand-drawn trajectories, the theory-guided neural network design leads to an average performance improvement of 62% .

In Chapter 3, we extended the inverse learning framework to non-minimum phase systems (i.e., systems with unstable inverse dynamics). To the best our knowledge, this is the first work showing the feasibility of learning stable approximate inverse

dynamics for non-minimum phase systems. Through both theory and experiments, we showed that, by carefully selecting the input and output of the neural network module, we can train an approximate inverse dynamics model of an unknown non-minimum phase baseline system directly from its input-output data to efficiently enhance its tracking performance. In Chapter 4, we further introduced an active training trajectory generation approach to systematically collect data for training neural network inverse dynamics modules.

Chapter 2 through Chapter 4 present a simple, yet effective neural network inverse learning approach for enhancing the tracking performance of single robots when their dynamics are not fully known. Inspired by the transfer learning literature, in Chapter 5, we studied a novel online learning approach that allows the neural network inverse module trained on a source robot to be used to improve the performance of a target robot that is structurally similar but dynamically different (e.g., from one quadrotor to another quadrotor with different mass or aerodynamic properties). In quadrotor trajectory tracking experiments, we showed that, with a minimal amount of data collected online, the proposed transfer learning approach effectively reduced the tracking error of a target robot by an average 74% over 10 trajectories. The resulting performance is comparable to the case when the neural network inverse dynamics module of the target system was fully trained from scratch.

Finally, in Chapter 6, we explored the feasibility of using an online adaptive neural network to enforce an uncertain robot system to behave as a predefined reference model. The approach can be combined with a model-based planner or controller design to realize a higher-level planning or control objective. In this work, we derived an online adaptation law for training a neural network to achieve model reference adaptation and exploited the Lipschitz property of a special type of neural network, the Lipschitz network [19], to guarantee the stability of the adapted system. This approach was successfully verified in challenging flying inverted pendulum experiments, where the proposed adaptive Lipschitz adaptation approach was combined with a standard model-based controller (i.e., an LQR) to enable an uncertain quadrotor system balancing an inverted pendulum while hovering or tracking circular trajectories. Our work is the first work demonstrating the efficacy of using Lipschitz networks for the closed-loop control of uncertain robot systems with theoretical guarantees.

7.2 Future Work

In this dissertation, we demonstrated the potential of incorporating control and machine learning to improve the performance of robot control systems. There are several interesting extensions:

- **Broadening the Class of Systems.** In our work, we have considered robot systems that can be modeled as continuous, control-affine nonlinear systems. One possible extension is to explore the applicability of the inverse dynamics learning for systems with hybrid dynamics (e.g., bipedal robots or quadrotors with load suspension), where multiple dynamic equations defined on different regions of the state space must be considered [150]. One trivial approach is to apply the current results to each dynamical system and train a set of independent neural network modules to enhance the tracking performance. Accounting for transitions across the boundaries of the dynamic regions is an open question. The hierarchical structure in some typical hybrid control approaches (e.g., supervisory control) naturally encourages a hierarchical learning structure for dealing with this class of systems capturing more complex dynamics. This idea requires further investigation in the context of inverse dynamics learning.

- **Perception-Based Learning Control.** The neural network add-on learning approaches presented in this dissertation rely on state feedback from the robot system. Practical robots often require reliable decision making using perception inputs (e.g., camera images or LiDAR scans). Estimating the state of the robot from perception data can be challenging, especially in visually-degraded environments. Another potential extension to the work in this dissertation is to investigate methodologies that systematically account for perception errors and develop provably safe frameworks that seamlessly integrate perception feedback for performing versatile tasks in uncertain environments. This may, for instance, require novel Bayesian learning techniques (e.g., Bayesian deep learning [151]) to efficiently characterize the unknowns and thereby facilitate reliable reasoning and decision making in unstructured environments.

- **Similarity Characterization for Across-Domain Experience Transfer.** In Chapter 5 and our follow-up works [149, 152], we hint at the importance of characterizing dynamics similarity when we intend to transfer the learning experience across two different domains, either from simulation to real experiments

or between two dynamically different robots. While there exist alternative data-driven approaches in the literature for characterizing domain similarity (e.g., by using likelihood measures based on trajectories collected from the source and target domains [153]), these approaches often suffer from scalability issues and lack closed-loop theoretical guarantees. Another interesting direction could be generalizing the notion of domain similarity presented in our work beyond trajectory tracking tasks and leverage the similarity notion to provide theoretical bounds on the transfer performance for practical applications.

Bibliography

[1] D. P. Bertsekas, *Dynamic Programming and Optimal Control*. Athena Scientific, 2000.

[2] F. Allgöwer and A. Zheng, *Nonlinear Model Predictive Control*. Springer, 2012.

[3] L. Brunke, M. Greeff, A. W. Hall, Z. Yuan, S. Zhou, J. Panerati, and A. P. Schoellig, "Safe learning in robotics: From learning-based control to safe reinforcement learning," *Annual Review of Control, Robotics, and Autonomous Systems*, 2021.

[4] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 82–97, 2012.

[5] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 25, pp. 1097–1105, 2012.

[6] M. Tan, R. Pang, and Q. V. Le, "Efficientdet: Scalable and efficient object detection," in *Proc. of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 10 781–10 790.

[7] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *Journal of Machine Learning Research*, vol. 17(39), pp. 1–40, 2016.

[8] P. Abbeel, A. Coates, M. Quigley, and A. Y. Ng, "An application of reinforcement learning to aerobatic helicopter flight," *Advances in Neural Information Processing Systems*, vol. 19, 2007.

[9] M. Zucker, N. Ratliff, M. Stolle, J. Chestnutt, J. A. Bagnell, C. G. Atkeson, and J. Kuffner, "Optimization and learning for rough terrain legged locomotion," *International Journal of Robotics Research (IJRR)*, vol. 30, no. 2, pp. 175–191, 2011.

[10] I. Akkaya, M. Andrychowicz, M. Chociej, M. Litwin, B. McGrew, A. Petron, A. Paino, M. Plappert, G. Powell, R. Ribas *et al.*, "Solving rubik's cube with a robot hand," *arXiv preprint arXiv:1910.07113*, 2019.

[11] N. Roy, I. Posner, T. D. Barfoot, P. Beaudoin, Y. Bengio, J. Bohg, O. Brock, I. Depatie, D. Fox, D. E. Koditschek, T. Lozano-Perez, V. K. Mansinghka, C. J. Pal, B. A. Richards, D. Sadigh, S. Schaal, G. S. Sukhatme, D. Thérien, M. Toussaint, and M. van de Panne, "From machine learning to robotics: Challenges and opportunities for embodied intelligence," 2021.

[12] C. J. Ostafew, A. P. Schoellig, and T. D. Barfoot, "Robust constrained learning-based nmpc enabling reliable mobile robot path tracking," *International Journal of Robotics Research (IJRR)*, vol. 35, no. 13, pp. 1547–1563, 2016.

[13] L. Hewing, J. Kabzan, and M. N. Zeilinger, "Cautious model predictive control using Gaussian process regression," *IEEE Transactions on Control Systems Technology*, vol. 28, no. 6, pp. 2736–2743, 2019.

[14] J. F. Fisac, N. F. Lugovoy, V. Rubies-Royo, S. Ghosh, and C. J. Tomlin, "Bridging Hamilton-Jacobi safety analysis and reinforcement learning," in *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, 2019, pp. 8550–8556.

[15] M. Jin and J. Lavaei, "Control-theoretic analysis of smoothness for stability-certified reinforcement learning," in *Proc. of the IEEE Conference on Decision and Control (CDC)*, 2018, pp. 6840–6847.

[16] S. Zhou, M. K. Helwa, and A. P. Schoellig, "Deep neural networks as add-on modules for enhancing robot performance in impromptu trajectory tracking," *The International Journal of Robotics Research*, vol. 39(12), pp. 1397–1418, 2020.

[17] Y. Li, "Deep reinforcement learning: An overview," *arXiv preprint arXiv:1701.07274*, 2017.

[18] C. Devin, A. Gupta, T. Darrell, P. Abbeel, and S. Levine, "Learning modular neural network policies for multi-task and multi-robot transfer," in *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 2169–2176.

[19] C. Anil, J. Lucas, and R. Grosse, "Sorting out Lipschitz function approximation," in *Proc. of the International Conference on Machine Learning (ICML)*, 2019, pp. 291–301.

[20] Y. Liu and G. Nejat, "Robotic urban search and rescue: A survey from the control perspective," *Journal of Intelligent & Robotic Systems*, vol. 72(2), pp. 147–165, 2013.

[21] T. Brogårdh, "Present and future robot control development — An industrial perspective," *Annual Reviews in Control*, vol. 31(1), pp. 69–79, 2007.

[22] J. Nikolic, M. Burri, J. Rehder, S. Leutenegger, C. Huerzeler, and R. Siegwart, "A UAV system for inspection of industrial facilities," in *Proc. of the IEEE Aerospace Conference*, 2013, pp. 1–8.

[23] K. Åström and T. Hägglund, "Revisiting the Ziegler–Nichols step response method for PID control," *Journal of Process Control*, vol. 14(6), pp. 635–650, 2004.

[24] B. A. Francis and W. M. Wonham, "The internal model principle of control theory," *Automatica*, vol. 12(5), pp. 457–465, 1976.

[25] A. Liniger, A. Domahidi, and M. Morari, "Optimization-based autonomous racing of 1:43 scale RC cars," *Optimal Control Applications and Methods*, vol. 36(5), pp. 628–647, 2015.

[26] R. Hirschorn, "Invertibility of multivariable nonlinear control systems," *IEEE Transactions on Automatic Control*, vol. 24(6), pp. 855–865, 1979.

[27] S. Devasia, D. Chen, and B. Paden, "Nonlinear inversion-based output tracking," *IEEE Transactions on Automatic Control*, vol. 41(7), pp. 930–942, 1996.

[28] J.-J. E. Slotine and W. Li, "On the adaptive control of robot manipulators," *International Journal of Robotics Research*, vol. 6(3), pp. 49–59, 1987.

[29] M. W. Spong, "On the robust control of robot manipulators," *IEEE Transactions on Automatic Control*, vol. 37(11), pp. 1782–1786, 1992.

[30] S. Levine, N. Wagener, and P. Abbeel, "Learning contact-rich manipulation skills with guided policy search," in *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 156–163.

[31] X. Da, R. Hartley, and J. W. Grizzle, "Supervised learning for stabilizing underactuated bipedal robot locomotion, with outdoor experiments on the wave field," in *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 3476–3483.

[32] P. Drews, G. Williams, B. Goldfain, E. A. Theodorou, and J. M. Rehg, "Aggressive deep driving: Combining convolutional neural networks and model predictive control," in *Proc. of the Conference on Robot Learning (CoRL)*, 2017, pp. 133–142.

[33] S. Tang and V. Kumar, "Autonomous flight," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 1(1), pp. 29–52, 2018.

[34] S. Bansal, A. K. Akametalu, F. J. Jiang, F. Laine, and C. J. Tomlin, "Learning quadrotor dynamics using neural network for flight control," in *Proc. of the IEEE Conference on Decision and Control (CDC)*, 2016, pp. 4653–4660.

[35] D. A. Bristow, M. Tharayil, and A. G. Alleyne, "A survey of iterative learning control," *IEEE Control Systems*, vol. 26(3), pp. 96–114, 2006.

[36] A. P. Schoellig, F. L. Mueller, and R. D'Andrea, "Optimization-based iterative learning for precise quadrocopter trajectory tracking," *Autonomous Robots*, vol. 33(1-2), pp. 103–127, 2012.

[37] A. Tayebi, "Adaptive iterative learning control for robot manipulators," *Automatica*, vol. 40(7), pp. 1195–1203, 2004.

[38] B. Kiumarsi, F. L. Lewis, H. Modares, A. Karimpour, and M.-B. Naghibi-Sistani, "Reinforcement Q-learning for optimal tracking control of linear discrete-time systems with unknown dynamics," *Automatica*, vol. 50(4), pp. 1167–1175, 2014.

[39] T. Zhang, G. Kahn, S. Levine, and P. Abbeel, "Learning deep control policies for autonomous aerial vehicles with MPC-guided policy search," in *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 528–535.

[40] Y. P. Pane, S. P. Nageshrao, and R. Babuška, "Actor-critic reinforcement learning for tracking control in robotics," in *Proc. of the IEEE Conference on Decision and Control (CDC)*, 2016, pp. 5819–5826.

[41] D. Nguyen-Tuong and J. Peters, "Local Gaussian Process regression for real-time model-based robot control," in *Proc. of the IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2008, pp. 380–385.

[42] M. K. Helwa, A. Heins, and A. P. Schoellig, "Provably robust learning-based approach for high-accuracy tracking control of Lagrangian systems," *arXiv preprint arXiv:1804.01031*, 2018.

[43] Z. Yan and J. Wang, "Robust model predictive control of nonlinear systems with unmodeled dynamics and bounded uncertainties based on neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 25(3), pp. 457–469, 2014.

[44] W. He, Y. Chen, and Z. Yin, "Adaptive neural network control of an uncertain robot with full-state constraints," *IEEE Transactions on Cybernetics*, vol. 46(3), pp. 620–629, 2016.

[45] S. Iplikci, "Support Vector Machines-based generalized predictive control," *International Journal of Robust and Nonlinear Control*, vol. 16(17), pp. 843–862, 2006.

[46] D. Nguyen-Tuong and J. Peters, "Using model knowledge for learning inverse dynamics," in *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, 2010, pp. 2677–2682.

[47] S. Schaal, C. G. Atkeson, and S. Vijayakumar, "Scalable techniques from nonparametric statistics for real time robot learning," *Applied Intelligence*, vol. 17(1), pp. 49–60, 2002.

[48] Q. Li, J. Qian, Z. Zhu, X. Bao, M. K. Helwa, and A. P. Schoellig, "Deep neural networks for improved, impromptu trajectory tracking of quadrotors," in *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 5183–5189.

[49] F. L. Mueller, A. P. Schoellig, and R. D'Andrea, "Iterative learning of feedforward corrections for high-performance tracking," in *Proc. of the IEEE In-*

ternational Conference on Intelligent Robots and Systems (IROS), 2012, pp. 3276–3281.

[50] S. Zhou, M. K. Helwa, and A. P. Schoellig, "Design of deep neural networks as add-on blocks for improving impromptu trajectory tracking," in *Proc. of the IEEE Conference on Decision and Control (CDC)*, 2017, pp. 5201–5207.

[51] K. J. Hunt, D. Sbarbaro, R. Żbikowski, and P. J. Gawthrop, "Neural networks for control systems — A survey," *Automatica*, vol. 28(6), pp. 1083–1112, 1992.

[52] H. Suprijono, W. Wahab, and B. Kusumoputro, "Optimized direct inverse control to control altitude of a small helicopter," in *MATEC Web of Conferences*, vol. 34. EDP Sciences, 2015.

[53] M. I. Jordan and D. E. Rumelhart, "Forward models: Supervised learning with a distal teacher," *Cognitive Science*, vol. 16(3), pp. 307–354, 1992.

[54] M. Kawato, "Feedback-error-learning neural network for supervised motor learning," in *Advanced Neural Computers*. Elsevier, 1990, pp. 365–372.

[55] F.-C. Chen and H. K. Khalil, "Adaptive control of a class of nonlinear discrete-time systems using neural networks," *IEEE Transactions on Automatic Control*, vol. 40(5), pp. 791–801, 1995.

[56] S. S. Ge and J. Zhang, "Neural-network control of nonaffine nonlinear system with zero dynamics by state and output feedback," *IEEE Transactions on Neural Networks*, vol. 14(4), pp. 900–918, 2003.

[57] Y. Zhang, G. Tao, and M. Chen, "Adaptive neural network based control of noncanonical nonlinear systems," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 27(9), pp. 1864–1877, 2016.

[58] Z.-P. Jiang and Y. Wang, "Input-to-state stability for discrete-time nonlinear systems," *Automatica*, vol. 37(6), pp. 857–869, 2001.

[59] T.-J. Jang, H.-S. Ahn, and C.-H. Choi, "Iterative learning control for discrete-time nonlinear systems," *International Journal of Systems Science*, vol. 25(7), pp. 1179–1189, 1994.

[60] M. Sun and D. Wang, "Analysis of nonlinear discrete-time systems with higher-order iterative learning control," *Dynamics and Control*, vol. 11(1), pp. 81–96, 2001.

[61] J. B. Hoagg and D. S. Bernstein, "Nonminimum-phase zeros — much to do about nothing — classical control revisited Part II," *Control Systems*, vol. 27(3), pp. 45–57, 2007.

[62] H. Sussmann, "Limitations on the stabilizability of globally-minimum-phase systems," *IEEE Transactions on Automatic Control*, vol. 35(1), pp. 117–119, 1990.

[63] J. L. Giesbrecht, H. K. Goi, T. D. Barfoot, and B. A. Francis, "A vision-based robotic follower vehicle," in *SPIE 7332, Unmanned Systems Technology XI*, 2009, pp. 73 321O1–73 321O12.

[64] M. K. Helwa and A. P. Schoellig, "On the construction of safe controllable regions for affine systems with applications to robotics," in *Proc. of the IEEE Conference on Decision and Control (CDC)*, 2016, pp. 3000–3005.

[65] M. Fazlyab, A. Robey, H. Hassani, M. Morari, and G. J. Pappas, "Efficient and accurate estimation of Lipschitz constants for deep neural networks," in *Proc. of the Conference on Neural Information Processing Systems (NeurIPS)*, 2019, pp. 11 427–11 438.

[66] B. Francis and P. Khargonekar, *Robust Control Theory*, ser. The IMA volumes in mathematics and its applications. Springer, 1995.

[67] H. K. Khalil, "Nonlinear systems third edition," *Patience Hall*, 2002.

[68] K. Van Heusden, A. Karimi, and D. Bonvin, "Data-driven estimation of the infinity norm of a dynamical system," in *Proc. of the IEEE Conference on Decision and Control (CDC)*, 2007, pp. 4889–4894.

[69] M. R. James and S. Yuliar, "Numerical approximation of the H_∞ norm for nonlinear systems," *Automatica*, vol. 31(8), pp. 1075–1086, 1995.

[70] G. F. Franklin, J. D. Powell, A. Emami-Naeini, and J. D. Powell, *Feedback control of dynamic systems*. Addison-Wesley Reading, MA, 1994, vol. 3.

[71] M. K. Helwa and P. E. Caines, "Epsilon controllability of nonlinear systems on polytopes," in *Proc. of the IEEE Conference on Decision and Control (CDC)*, 2015, pp. 252–257.

[72] M. Dahleh, M. A. Dahleh, and G. Verghese, *Lectures on Dynamic Systems and Control*. Department of Electrical Engineering and Computer Science, Massachuasetts Institute of Technology, 2004.

[73] C. M. Bishop, *Pattern recognition and machine learning*. Springer, 2006.

[74] P.-J. Bristeau, F. Callou, D. Vissiere, and N. Petit, "The navigation and control technology inside the ARDrone micro UAV," *IFAC Proceedings Volumes*, vol. 44(1), pp. 1477–1484, 2011.

[75] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint:1412.6980*, 2014.

[76] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014.

[77] A. De Luca, P. Lucibello *et al.*, "Inversion techniques for trajectory control of flexible robot arms," *Journal of Field Robotics*, vol. 6(4), pp. 325–344, 1989.

[78] S. A. Al-Hiddabi and N. H. McClamroch, "Tracking and maneuver regulation control for nonlinear nonminimum phase systems: Application to flight control," *IEEE Transactions on Control Systems Technology*, vol. 10(6), pp. 780–792, 2002.

[79] B. P. Rigney, L. Y. Pao, and D. A. Lawrence, "Nonminimum phase dynamic inversion for settle time applications," *IEEE Transactions on Control Systems Technology*, vol. 17(5), pp. 989–1005, 2009.

[80] J. J. E. Slotine and W. Li, *Applied nonlinear control*. Prentice Hall, New Jersey, 1991.

[81] Y. Zhang, Q. Zhu, and R. Xiong, "Pre-action and stable inversion based precise tracking for non-minimum phase system," in *Proc. of the IEEE Conference on Decision and Control (CDC)*, 2016, pp. 5682–5687.

[82] A. S. Polydoros, L. Nalpantidis, and V. Krüger, "Real-time deep learning of robotic manipulator inverse dynamics," in *Proc. of the IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2015, pp. 3442–3448.

[83] D. Nguyen-Tuong, J. Peters, M. Seeger, and B. Schölkopf, "Learning inverse dynamics: a comparison," in *European Symposium on Artificial Neural Networks*, no. EPFL-CONF-175477, 2008.

[84] C. Williams, S. Klanke, S. Vijayakumar, and K. M. Chai, "Multi-task Gaussian Process learning of robot inverse dynamics," in *Proc. of the Advances in Neural Information Processing Systems (NIPS)*, 2009, pp. 265–272.

[85] S. Jung and S. S. Kim, "Control experiment of a wheel-driven mobile inverted pendulum using neural network," *IEEE Transactions on Control Systems Technology*, vol. 16(2), pp. 297–303, 2008.

[86] A. de Almeida Neto, W. R. Neto, L. C. S. Góes, and C. Nascimento, "Feedback-error-learning for controlling a flexible link," in *Proc. of the IEEE Brazilian Symposium on Neural Networks*, 2000, pp. 273–278.

[87] E. D. Sontag and Y. Wang, "Notions of input to output stability," *Systems & Control Letters*, vol. 38(4), pp. 235–248, 1999.

[88] A. M. Bloch, N. E. Leonard, and J. E. Marsden, "Controlled Lagrangians and the stabilization of mechanical systems I: The first matching theorem," *IEEE Transactions on Automatic Control*, vol. 45(12), pp. 2253–2270, 2000.

[89] Quanser Consulting Inc., "IP02 self-erecting inverted pendulum user's guide," 1996, Available at: http://www.mecatronica.eesc.usp.br/wiki/upload/1/11/Manual_SelfErecting.pdf.

[90] J. A. Butterworth, L. Y. Pao, and D. Y. Abramovitch, "The effect of nonminimum-phase zero locations on the performance of feedforward model-inverse control techniques in discrete-time systems," in *Proc. of the American Control Conference*, 2008, pp. 2696–2702.

[91] B. Settles, "Active learning literature survey," *Technical Report, University of Wisconsin, Madison*, vol. 52(55-66), 2010.

[92] V. V. Fedorov, *Theory of Optimal Experiments*. Elsevier, 1972.

[93] R. Lorenz, R. P. Monti, I. R. Violante, C. Anagnostopoulos, A. A. Faisal, G. Montana, and R. Leech, "The automatic neuroscientist: a framework for optimizing experimental design with closed-loop real-time fMRI," *NeuroImage*, vol. 129, pp. 320–334, 2016.

[94] L. Ljung, "System identification," in *Signal Analysis and Prediction*. Springer, 1998.

[95] A. J. G. Schoofs, "Experimental design and structural optimization," in *Structural Optimization*. Springer, 1988, pp. 307–314.

[96] J. A. List, S. Sadoff, and M. Wagner, "So you want to run an experiment, now what? some simple rules of thumb for optimal experimental design," *Experimental Economics*, vol. 14(4), pp. 439–457, 2011.

[97] B. Settles, "Active learning," *Synthesis Lectures on Artificial Intelligence and Machine Learning*, vol. 6(1), pp. 1–114, 2012.

[98] Y. Gal, R. Islam, and Z. Ghahramani, "Deep Bayesian Active Learning with Image Data," in *Proc. of the International Conference on Machine Learning (ICML)*, 2017.

[99] S. Tong and D. Koller, "Support vector machine active learning with applications to text classification," *Journal of Machine Learning Research (JMLR)*, vol. 2, pp. 45–66, 2001.

[100] Y. Gal and Z. Ghahramani, "Dropout as a Bayesian approximation: Representing model uncertainty in deep learning," in *Proc. of the International Conference on Machine Learning (ICML)*, 2016, pp. 1050–1059.

[101] D. A. Cohn, Z. Ghahramani, and M. I. Jordan, "Active learning with statistical models," *Journal of Artificial Intelligence Research (JAIR)*, vol. 4, pp. 129–145, 1996.

[102] R. Burbidge, J. J. Rowland, and R. D. King, "Active learning for regression based on query by committee," in *Proc. of the International Conference on Intelligent Data Engineering and Automated Learning (IDEAL)*. Springer, 2007, pp. 209–218.

[103] D. Cohn, L. Atlas, and R. Ladner, "Improving generalization with active learning," *Machine Learning*, vol. 15(2), pp. 201–221, 1994.

[104] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, 2011, pp. 2520–2525.

[105] G. Calafiore, M. Indri, and B. Bona, "Robot dynamic calibration: Optimal excitation trajectories and experimental parameter estimation," *Journal of Robotic Systems*, vol. 18(2), pp. 55–68, 2001.

[106] W. Rackl, R. Lampariello, and G. Hirzinger, "Robot excitation trajectories for dynamic parameter estimation using optimized b-splines," in *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, 2012, pp. 2042–2047.

[107] J. Swevers, C. Ganseman, D. Bilgin, J. De Schutter, and H. Van Brussel, "Optimal robot excitation and identification," *IEEE Transactions on Robotics and Automation*, vol. 13(5), pp. 730–740, 1997.

[108] D. A. Cohn, "Neural network exploration using optimal experiment design," *Neural Networks*, vol. 9(6), pp. 1071–1083, 1996.

[109] B. Lakshminarayanan, A. Pritzel, and C. Blundell, "Simple and scalable predictive uncertainty estimation using deep ensembles," in *Proc. of Advances in Neural Information Processing Systems (NIPS)*, 2017, pp. 6402–6413.

[110] S. Geman, E. Bienenstock, and R. Doursat, "Neural networks and the bias/variance dilemma," *Neural Computation*, vol. 4(1), pp. 1–58, 1992.

[111] D. Nguyen-Tuong and J. Peters, "Model learning for robot control: a survey," *Cognitive Processing*, vol. 12(4), pp. 319–340, 2011.

[112] H. B. Ammar, E. Eaton, P. Ruvolo, and M. Taylor, "Unsupervised cross-domain transfer in policy gradient reinforcement learning via manifold alignment," in *Proc. of the AAAI Conference on Artificial Intelligence*, 2015.

[113] B. Bócsi, L. Csató, and J. Peters, "Alignment-based transfer learning for robot models," in *Proc. of the International Joint Conference on Neural Networks (IJCNN)*, 2013, pp. 1–7.

[114] M. K. Helwa and A. P. Schoellig, "Multi-robot transfer learning: A dynamical system perspective," in *Proc. of the IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2017, pp. 4702–4708.

[115] A. Gupta, C. Devin, Y. Liu, P. Abbeel, and S. Levine, "Learning invariant feature spaces to transfer skills with reinforcement learning," in *Proc. of the International Conference on Learning Representations*, 2017.

[116] S. Daftry, J. A. Bagnell, and M. Hebert, "Learning transferable policies for monocular reactive MAV control," in *Proc. of the International Symposium on Experimental Robotics*. Springer, 2016, pp. 3–11.

[117] M. E. Taylor and P. Stone, "Transfer learning for reinforcement learning domains: A survey," *Journal of Machine Learning Research (JMLR)*, vol. 10, pp. 1633–1685, dec 2009.

[118] J. Blitzer, R. McDonald, and F. Pereira, "Domain adaptation with structural correspondence learning," in *Proc. of the Association for Computational Linguistics Conference on Empirical Methods in Natural Language Processing*, 2006, pp. 120–128.

[119] Z. Wang, Y. Song, and C. Zhang, "Transferred dimensionality reduction," in *Machine Learning and Knowledge Discovery in Databases*, W. Daelemans, B. Goethals, and K. Morik, Eds., 2008, pp. 550–565.

[120] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 10, pp. 1345–1359, Oct 2010.

[121] J. Fu, S. Levine, and P. Abbeel, "One-shot learning of manipulation skills with online dynamics adaptation and neural network priors," *arXiv preprint arXiv:1509.06841*, 2016.

[122] K. Pereida, D. Kooijman, R. R. P. R. Duivenvoorden, and A. P. Schoellig, "Transfer learning for high-precision trajectory tracking through L1 adaptive feedback and iterative learning," *International Journal of Adaptive Control and Signal Processing*, 2018.

[123] M. Hamer, M. Waibel, and R. D'Andrea, "Knowledge transfer for high-performance quadrocopter maneuvers," in *Proc. of the IEEE Conference on Intelligent Robots and Systems (IROS)*, 2013, pp. 1714–1719.

[124] K. Pereida, M. K. Helwa, and A. P. Schoellig, "Data-efficient multi-robot, multi-task transfer learning for trajectory tracking," *IEEE Robotics and Automation Letters*, vol. 3(2), pp. 1260–1267, 2018.

[125] A. Marco, F. Berkenkamp, P. Hennig, A. P. Schoellig, A. Krause, S. Schaal, and S. Trimpe, "Virtual vs. real: Trading off simulations and physical experiments in reinforcement learning with Bayesian optimization," in *Proc. of the IEEE*

International Conference on Robotics and Automation (ICRA), 2017, pp. 1557–1563.

[126] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-real transfer of robotic control with dynamics randomization," in *Prof. of the IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 3803–3810.

[127] C. Finn, T. Yu, T. Zhang, P. Abbeel, and S. Levine, "One-shot visual imitation learning via meta-learning," in *Proc. of Machine Learning Research (PMLR)*, vol. 78, 2017, pp. 357–368.

[128] A. Lazaric, "Transfer in reinforcement learning: a framework and a survey," in *Reinforcement Learning*. Springer, 2012, pp. 143–173.

[129] E. D. Sontag, "Input to state stability: Basic concepts and results," in *Nonlinear and optimal control theory*. Springer, 2008, pp. 163–220.

[130] A. Hock and A. P. Schoellig, "Distributed iterative learning control for a team of quadrotors," 2016.

[131] M. Whorton, L. Yang, and R. Hall, *Similarity Metrics for Closed Loop Dynamic Systems*, ser. AIAA Guidance, Navigation and Control Conference and Exhibit. American Inst. of Aeronautics and Astronautics, 2008.

[132] C. E. Rasmussen, *Gaussian processes for machine learning*. MIT Press, 2006.

[133] K. Zhou and J. C. Doyle, *Essentials of Robust Control*. Prentice Hall Upper Saddle River, 1998.

[134] S. Sastry and M. Bodson, *Adaptive Control: Stability, Convergence and Robustness*. Courier Corporation, 2011.

[135] J. Cooper, J. Che, and C. Cao, "The use of learning in fast adaptation algorithms," *International Journal of Adaptive Control and Signal Processing*, vol. 28(3-5), pp. 325–340, 2014.

[136] G. Chowdhary, H. A. Kingravi, J. P. How, and P. A. Vela, "Bayesian nonparametric adaptive control using Gaussian processes," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 26(3), pp. 537–550, 2014.

[137] F. Chen, R. Jiang, K. Zhang, B. Jiang, and G. Tao, "Robust backstepping sliding-mode control and observer-based fault estimation for a quadrotor UAV," *IEEE Transactions on Industrial Electronics*, vol. 63(8), pp. 5044–5056, 2016.

[138] D. Limón, I. Alvarado, T. Alamo, and E. F. Camacho, "Robust tube-based MPC for tracking of constrained linear systems with additive disturbances," *Journal of Process Control*, vol. 20(3), pp. 248–260, 2010.

[139] X. Lyu, J. Zhou, H. Gu, Z. Li, S. Shen, and F. Zhang, "Disturbance observer based hovering control of quadrotor tail-sitter vtol UAVs using H_∞ synthesis," *IEEE Robotics and Automation Letters*, vol. 3(4), pp. 2910–2917, 2018.

[140] N. Hovakimyan and C. Cao, $\mathcal{L}_1$ *Adaptive Control Theory: Guaranteed Robustness with Fast Adaptation*. Society for Industrial and Applied Mathematics, 2010.

[141] C. J. Ostafew, A. P. Schoellig, T. D. Barfoot, and J. Collier, "Learning-based nonlinear model predictive control to improve vision-based mobile robot path tracking," *Journal of Field Robotics*, vol. 33(1), pp. 133–152, 2016.

[142] C. D. McKinnon and A. P. Schoellig, "Learn fast, forget slow: Safe predictive learning control for systems with unknown and changing dynamics performing repetitive tasks," *IEEE Robotics and Automation Letters*, vol. 4(2), pp. 2180–2187, 2019.

[143] A. Xie, J. Harrison, and C. Finn, "Deep reinforcement learning amidst lifelong non-stationarity," *arXiv preprint arXiv:2006.10701*, 2020.

[144] G. Lightbody and G. Irwin, "Direct neural model reference adaptive control," *IEEE Proceedings-Control Theory and Applications*, vol. 142(1), pp. 31–43, 1995.

[145] G. Joshi and G. Chowdhary, "Deep model reference adaptive control," in *Proc. of the IEEE Conference on Decision and Control (CDC)*, 2019, pp. 4601–4608.

[146] G. Shi, X. Shi, M. O'Connell, R. Yu, K. Azizzadenesheli, A. Anandkumar, Y. Yue, and S.-J. Chung, "Neural lander: Stable drone landing control using learned dynamics," in *Proc. of the International Conference on Robotics and Automation (ICRA)*, 2019, pp. 9784–9790.

[147] S. Zhou, M. K. Helwa, A. P. Schoellig, A. Sarabakha, and E. Kayacan, "Knowledge transfer between robots with similar dynamics for high-accuracy impromptu trajectory tracking," in *Proc. of the European Control Conference (ECC)*, 2019, pp. 1–8.

[148] M. Hehn and R. D'Andrea, "A flying inverted pendulum," in *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, 2011, pp. 763–770.

[149] M. J. Sorocky, S. Zhou, and A. P. Schoellig, "Experience selection using dynamics similarity for efficient multi-source transfer learning between robots," in *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, 2020.

[150] P. J. Antsaklis, "A brief introduction to the theory and applications of hybrid systems," in *Proc. of the IEEE Special Issue on Hybrid Systems: Theory and Applications*, 2000, pp. 879–887.

[151] E. Daxberger, A. Kristiadi, A. Immer, R. Eschenhagen, M. Bauer, and P. Hennig, "Laplace redux-effortless bayesian deep learning," *Advances in Neural Information Processing Systems*, vol. 34, 2021.

[152] M. J. Sorocky, S. Zhou, and A. P. Schoellig, "To share or not to share? performance guarantees and the asymmetric nature of cross-robot experience transfer," *IEEE Control Systems Letters*, vol. 5(3), pp. 923–928, 2021.

[153] A. Rajeswaran, S. Ghotra, B. Ravindran, and S. Levine, "Epopt: Learning robust neural network policies using model ensembles," 2017.