ROBOT MANIPULATORS INTELLIGENT MOTION AND FORCE CONTROL

Shuyang Chen

Submitted in Partial Fullfillment of the Requirements
for the Degree of

DOCTOR OF PHILOSOPHY

Approved by:
John T. Wen, Chair
Mamadou Diagne
Agung Julius
Sandipan Mishra

Department of Mechanical, Aerospace, and Nuclear Engineering
Rensselaer Polytechnic Institute
Troy, New York

[May 2021]
Submitted April 2021

© Copyright 2021
by
Shuyang Chen
All Rights Reserved

CONTENTS

LIST OF TABLES vi

LIST OF FIGURES viii

ACKNOWLEDGMENT xiv

ABSTRACT xv

1. INTRODUCTION 1

1.1 Background and Motivation 1

1.2 Problem Statement 2

1.3 Related Work 3

1.3.1 Robot Manipulators Motion Control 3

1.3.2 Robot Manipulators Force Control 5

1.4 Thesis Contributions 6

1.5 Publications Resulting from This Thesis 8

2. ROBOT MANIPULATORS KINEMATIC CONTROL 10

2.1 Kinematic Control Strategy 10

2.1.1 Equality Constraints 11

2.1.2 Inequality Constraints 12

2.2 Experimental Results 13

2.2.1 System Description 13

2.2.2 Implementation and Results 14

2.3 Summary 17

3. GRADIENT-BASED ILC FOR NONLINEAR SYSTEMS 18

3.1 Generic ILC Formulation 18

3.2 Gradient-Based ILC 19

3.3 Robustness Condition and Convergence Analysis 21

3.4 Gradient-Based ILC for Trajectory Tracking Control with EGM 23

3.5 Summary 24

4. ROBOT MANIPULATORS MOTION CONTROL THROUGH DYNAMICS COMPENSATION . . . 25
4.1 Introduction . . . 25
4.2 Multi-Layer Neural Networks as a Feedforward Dynamics Compensator . . . 26
4.2.1 Approach Overview . . . 26
4.2.2 Problem Statement . . . 28
4.2.3 Methodology . . . 29
4.2.3.1 Iterative Learning Control . . . 30
4.2.3.2 Feedforward Neural Networks . . . 30
4.2.3.3 Transfer Learning . . . 32
4.2.4 Simulation Results with IRB6640-180 . . . 33
4.2.5 Experimental Results with Physical IRB6640-180 Robot . . . 39
4.2.6 Simulation Results with Other Robots . . . 40
4.2.7 Discussion . . . 41
4.3 Recurrent Neural Networks as a Feedforward Dynamics Compensator . . . 42
4.3.1 Approach Overview . . . 43
4.3.2 Methodology . . . 44
4.3.2.1 Proposed Control Law . . . 44
4.3.2.2 Feedforward Controller using a URNN and Gradient-Based ILC 44
4.3.2.3 Feedforward Controller using a BRNN . . . 46
4.3.2.4 Resolved Velocity Controller for Human Teleoperation . . . 47
4.3.3 Results . . . 47
4.3.4 Discussion . . . 52
4.4 Neural Adaptive Control . . . 53
4.4.1 Approach Overview . . . 54
4.4.2 Problem Statement . . . 55
4.4.3 Methodology . . . 56
4.4.3.1 Controller Design . . . 56
4.4.3.2 Neural Network Design . . . 57
4.4.3.3 Online Backpropagation for Regressor Update . . . 59
4.4.4 Simulation Results . . . 60
4.4.5 Experimental Results with a Physical Baxter Robot . . . 62
4.4.5.1 Experiment I: Feasibility Study . . . 62
4.4.5.2 Experiment II: Comparative Study . . . 63
4.4.6 Neural Adaptive Controller for the ABB Robot . . . 65
4.5 Summary . . . 66

5. ROBOT MANIPULATORS HYBRID FORCE AND MOTION CONTROL 68
5.1 Introduction 68
5.2 MIMO ILC for Robotic Deep Rolling on a Flat Sample 70
5.2.1 System Input/Output Map 71
5.2.2 System Characterization 71
5.2.3 Experimental Results 73
5.2.3.1 Experimental Setup 73
5.2.3.2 Implementations and Comparison 74
5.2.4 Discussion 78
5.3 SISO ILC for Robotic Deep Rolling on Surface with Complex Geometries 80
5.3.1 System Input/Output Map 80
5.3.2 Surface Parametrization 81
5.3.3 Generalized Damper Force Controller 82
5.3.4 Experimental Results 83
5.3.4.1 Rolling Under a Singular-Free Configuration 83
5.3.4.2 Rolling Under a Near-Singular Configuration 85
5.4 Summary 87
6. CONCLUSIONS AND FUTURE WORK 89
6.1 Conclusions 89
6.2 Future Work 90
REFERENCES 92
APPENDICES
A. EGM DYNAMICS CHARACTERIZATION 98
B. ADJOINT OF A LINEAR TIME-INVARIANT SYSTEM 100
C. INVERSE OF A NON-MINIMUM-PHASE SISO SYSTEM 102
D. NEURAL ADAPTIVE CONTROL FOR ABB EGM 105
E. SUPPLEMENTARY FILES 108
E.1 Permissions for Section 2.2 108

LIST OF TABLES

3.1 Comparison of the ℓ_2 norm of tracking errors of a single-joint trajectory, by using ILC implemented with RobotStudio and with the fitted linear model $G(s)$ in Appendix A. The initial tracking error is 31.6230°. The linear model offers a reliable approximation of the linear regime of EGM. 24

4.1 Mean squared testing error of different NN architectures. The architecture with two hidden layers and 100 nodes in each layer produces the best testing accuracy. 32

4.2 ℓ_2 and ℓ_∞ norm of tracking errors of the multi-joint sinusoidal trajectory without and with NNs compensation in RobotStudio. The trajectory tracking accuracy of all joints is improved with the NN feedforward compensator. 35

4.3 ℓ_2 and ℓ_∞ norm of tracking errors of the MoveIt! planned joint trajectories without and with NNs compensation in RobotStudio. The NN feedforward compensator improves the trajectory tracking accuracy for all joints. 37

4.4 ℓ_2 norm of tracking errors in three Cartesian axes. The NN feedforward controller improves the trajectory tracking accuracy in all Cartesian directions. 37

4.5 RMSE tracking errors of the straight line using 4 different approaches in RobotStudio. A similar tracking accuracy to MoveL is achieved by running extra 3 ILC iterations using the commanded trajectory produced by the NNs as an initial guess for ILC in (3.7). 39

4.6 ℓ_2 norm of tracking errors in radian of the sinusoidal trajectory without and with transfer learning. The table demonstrates the benefits of transfer learning for tracking a fast trajectory. 40

4.7 Both neural-learning control approaches achieve better tracking of the 7-dimensional sinusoidal joint trajectory than the baseline controller, in terms of ℓ_2 and ℓ_∞ norm of tracking errors. 49

4.8 Both neural-learning control approaches achieve better tracking of the 7-dimensional random joint trajectory than the baseline controller, in terms of ℓ_2 and ℓ_∞ norm of tracking errors. 49

4.9 Both neural-learning control approaches achieve better tracking of the Cartesian trajectory in all three Cartesian axes than the baseline controller, in terms of ℓ_2 and ℓ_∞ norm of tracking errors. 50

4.10 The BRNN feedforward controller achieves real-time dynamics compensation and produces better tracking of the 7-dimensional user teleoperated joint trajectory than the baseline controller, in terms of ℓ_2 and ℓ_∞ norm of tracking errors. 54

4.11 Connection between neighboring network layers. 58

4.12 In the feasibility study, the neural adaptive controller presents a better tracking of the sinusoidal multi-joint trajectory in Figure 4.28 than the baseline controller, in terms of ℓ_2 and ℓ_∞ norm of tracking errors. 64

4.13 Frobenius norm of tracking errors (in radian) for all 7 joints of the multi-joint sinusoidal trajectory in Figure 4.29 using the neural adaptive controller with (system I) and without (system II) online learning of the regressor. The regressor online learning scheme speeds up the tracking error convergence starting from $t = 75\ s$. 65

4.14 The neural adaptive controller presents a better tracking of a sinusoidal multi-joint trajectory for the ABB robot EGM interface than the baseline PD controller, in terms of RMS and ℓ_∞ norm of tracking errors. 66

5.1 RMS and L_∞ norm of trapezoidal force tracking errors (1000 N peak force) in z with the ILC algorithm. The RMS force tracking error monotonically reduces by over 78 % after 9 iterations. 77

5.2 RMS and L_∞ norm of trapezoidal velocity tracking errors in y. 77

5.3 RMS and L_∞ norm of trapezoidal force tracking errors (2000 N peak force) in z with the ILC algorithm. The RMS force tracking error monotonically reduces by over 73 % after 7 iterations. 79

5.4 RMS and L_∞ norm of force tracking errors along the curved surface normal of Figure 5.14. The RMS force tracking error monotonically reduces by over 71 % after 30 iterations. 85

5.5 RMS and L_∞ norm of force tracking errors of Figure 5.16. The RMS force tracking error monotonically reduces by over 90 % after 49 iterations. 86

LIST OF FIGURES

1.1 Examples of industrial robot-assisted manufacturing: ABB robots for float glass unloading (left) and KUKA robots for automobile assembly lines (right). Image credit: Wikimedia Commons. 1

1.2 Examples of robot-assisted manufacturing projects conducted at RPI. An ABB robot is used for wind turbine blades assembly (left) and material deep rolling (right). . . . 2

2.1 Control barrier function σ_i as a function of the inequality constraint h_{I_i}. When h_{I_i} is violated ($h_{I_i} \leqslant \eta_i$), σ_i is designed to drive h_{I_i} toward a feasible region. 12

2.2 Kinematic parameters of ABB IRB6640 180/255 robot. Reprinted by permission from ASME: "Software Framework for Robot-Assisted Large Structure Assembly," S. Chen, Y. Peng, J. Wason, J. Cui, G. Saunders, S. Nath, and J. T. Wen [71], Paper No: MSEC2018-6689. 14

2.3 Use of bounding boxes to speed up the collision checking. (a) Simplified collision models of the robot, vacuum gripper and testbed. (b) Robot collision model rendered in SolidWorks. Reprinted by permission from ASME: "Software Framework for Robot-Assisted Large Structure Assembly," S. Chen, Y. Peng, J. Wason, J. Cui, G. Saunders, S. Nath, and J. T. Wen [71], Paper No: MSEC2018-6689. 15

2.4 Top-down view of the simulated collision avoidance behavior during the object transportation through robot teleoperation using the kinematic controller. The green and red arrows denote the actual commanded velocity computed by the kinematic controller and the desired velocity, respectively. (a) Robot initial pose. (b) The actual commanded velocity is the same as the desired velocity when there is no risk of collision with the wall. (c) The actual commanded velocity is different from the desired velocity and drags the object away from the wall when there exists a risk of collision. (d) The magnitude of the commanded velocity is larger when the object is closer to the wall. (e) The magnitude of the commanded velocity reduces when the object is further from the wall. (f) The actual commanded velocity is again the same as the desired velocity when there exists no risk of collision with the wall. 16

2.5 EGM motion control performance is compromised by the nonlinear robot inner loop dynamics. The output (blue curve) shows a significant tracking error in both amplitude and phase compared with the desired trajectory (red curve). 17

3.1 Block diagram of one iteration of the gradient-based ILC algorithm, which uses time reversal filtering to generate the gradient descent directions with the physical system. 21

3.2 Trajectory tracking improvement with the gradient-based ILC after 8 iterations for the fast trajectory in Figure 2.5. 23

4.1 Overview of the first neural-learning trajectory tracking control approach. 27

4.2 Robot dynamics with closed loop joint servo control. The goal of the neural-learning control approach is to find a possibly non-causal feedforward compensator $\mathcal{G}^{\dagger}$ using MNNs. 29

4.3 Magnitude frequency response plot of a local LTI approximation of EGM transfer matrix. The frequency response is identified using a series of single frequency sinusoidal signals. For all subplots, the horizontal axis is frequency (rad/s), and the vertical axis is magnitude (dB). Locally the EGM dynamics has a nearly diagonal structure. 31

4.4 The architecture of the employed neural network that has two fully-connected hidden layers and 100 nodes in each hidden layer. The input and output layer contain 50 and 25 nodes, respectively. 32

4.5 Comparison of responses between RobotStudio (red curve) and the physical robot (blue curve). Inputs are sinusoids with amplitude of 2° and frequency of 2 rad/sec and 10 rad/sec. (a) RobotStudio captures the dynamics of the physical robot well when commanded a slow trajectory. (b) Discrepancy exists between RobotStudio dynamics and the physical robot when commanded a fast trajectory. 33

4.6 Comparison of tracking performance without and with the NN compensation for a chirp signal in RobotStudio. (a) Uncompensated case: desired output (also the input into the inner loop), RobotStudio output, and linear model output. (b) Compensation with NN: desired output, RobotStudio output, and the input generated by the NN. The NN feedforward controller addresses the issues of lag effect and amplitude discrepancies induced by robot inner loop dynamics. 34

4.7 The generalization capability of the NN feedforward controller is demonstrated by applying to tracking a pure random trajectory for joint 1 in RobotStudio. 34

4.8 The NN feedforward controller improves the trajectory tracking accuracy for a multi-joint sinusoidal trajectory in RobotStudio through robot inner loop dynamics compensation. 35

4.9 The NN feedforward controller improves the trajectory tracking accuracy for a multi-joint trajectory planned by MoveIt! in RobotStudio through robot inner loop dynamics compensation. The inset figures clearly demonstrate the improvement of tracking. 36

4.10 The NN feedforward controller improves the trajectory tracking accuracy for a Cartesian square trajectory in x-y plane with z constant in RobotStudio. 37

4.11 Improved tracking performance of Figure 4.10 in individual Cartesian axis. Figures (a)-(c) denote the tracking performance in x, y, z directions, respectively. With the NN compensation, the tracking errors are significantly reduced for all axes, as reported in Table 4.4. 38

4.12 (a) Uncompensated case: desired output (also the input into the inner loop) and the physical robot output. (b) Compensation with NN: desired output, physical robot output, and the input generated by the NN. With transfer learning, the tracking errors for the same chirp signal as the one in simulation (Figure 4.6) are reduced for joint 1 of the physical robot. 39

4.13 (a) Uncompensated case. (b) Compensation using simulation-trained NN. (c) Compensation using the NN updated by transfer learning. Transfer learning plays a key role for tracking a fast sinusoidal trajectory for joint 1 of the physical robot. The ℓ_2 norm of tracking error of each case is shown in Table 4.6. 40

4.14 IRB120 robot tracking a chirp (top left) and a sinusoidal (top right) signal in RobotStudio. IRB6640-130 robot tracking a chirp (bottom left) and a sinusoidal (bottom right) signal in RobotStudio. The trained NNs using IRB6640-180 also improve the tracking accuracy of a chirp and a sinusoidal joint trajectories for IRB120 and IRB6640-130 in RobotStudio, possibly because these robots have similar inner loop dynamics. 41

4.15 Slight coupling of Baxter joints. The dashed and solid curves indicate desired and actual joint angles, respectively. Different colors represent response of different joints. We only command joint motion of joint 4 (the blue sinusoid in the middle of the figure). The figure shows that some joints other than joint 4 have also been activated, although they are not commanded any motion. 42

4.16 RNN-based feedforward learning control. Robot Raconteur [87] is a communication framework that is used to coordinate different components. 43

4.17 Overall control architecture. The feedforward control input q_f is produced by either using ILC implemented with the trained URNN, or directly generated by the trained BRNN. The commanded joint position q_c consists of q_f and a proportional feedback control. 44

4.18 Unrolled RNNs structures. h_i represents the internal state at each time step. (a) Unrolled causal unidirectional RNN. The URNN takes past inputs and produces a future output. (b) Unrolled non-causal bidirectional RNN. The BRNN takes past and future inputs and produces an output at current time step. 45

4.19 Comparison of tracking performance with using the baseline controller, and the RNN feedforward controllers of a multi-joint sinusoidal trajectory. The plots of joint 5 to 7 are zoomed in to better visualize different tracking performance. Both URNN and BRNN feedforward controllers achieve better tracking than the baseline controller. . . 48

4.20 Comparison of tracking performance with using the baseline controller and the RNN feedforward controllers of a multi-joint random trajectory. The plots of joint 5 to 7 are zoomed in to show the difference. The horizontal axis is time in second, and the vertical axis is the angle in radian for all plots. Both URNN and BRNN feedforward controllers achieve better tracking than the baseline controller. 50

4.21 Comparison of tracking performance using the baseline controller and the RNN feedforward controllers for a Cartesian square trajectory in $x-y$ plane with constant z. Both URNN and BRNN feedforward controllers achieve better tracking than the baseline controller. 51

4.22 Comparison of average tracking errors over all 7 joints of the sinusoidal joint trajectory in Figure 4.19. The blue curved denotes the average tracking error using the ILC+URNN approach. The red curved denotes the average tracking error with implementing 2 more ILC runs on Baxter, using the feedforward input trajectory generated by ILC+URNN approach as the initial guess for (3.7). The tracking performance is further improved by running extra ILC iterations. 52

4.23 Comparison of tracking performance with using the baseline controller and the BRNN feedforward controller of a teleoperated multi-joint trajectory. The plots of joint 1, 2, 5 and 7 are zoomed in to better visualize the difference. The BRNN feedforward controller achieves better tracking than the baseline controller. 53

4.24 Overall neural adaptive control architecture. 54

4.25 Network architecture that expresses robot dynamics as a linear-in-parameter form for the neural adaptive controller development. 58

4.26 Schematics of the regressor online learning. The update of the regressor weights is at a slower rate compared with the output layer weights adaptation. 60

4.27 Comparison of absolute tracking errors of a sinusoidal trajectory for the single pendulum with and without online backpropagation of the regressor. The large errors from $t = 0 \sim 2$ s (the initial tracking error is 0.25 radian at $t = 0$ and the mean error is 0.0461 radian from $t = 0 \sim 2$ s) are not plotted to better visualize the later small errors. The regressor online learning speeds up the tracking error convergence. 61

4.28 Comparison of tracking performance with the baseline PD controller and the neural adaptive controller for a multi-joint sinusoidal trajectory. The neural adaptive controller presents a better tracking than the baseline controller, which demonstrates the feasibility of the assumptions in Section 4.4.2. 63

4.29 Comparison of absolute tracking errors using the developed neural adaptive controller with (blue curve) and without (red curve) the regressor online learning for tracking a multi-joint sinusoidal trajectory, with an unknown load attached at the robot end effector. The regressor online learning finishes at $t = 75$ s and $t = 100$ s, which speeds up the tracking error convergence. 65

5.1 Robotic deep rolling with a flat sample. 69

5.2 Structure of the input/output map for ILC implementation. 71

5.3 Experimental setup for robotic deep rolling with a flat sample. 73

5.4 The open-loop position-based control achieves good force tracking for a constant 1000 N desired force in z. 74

5.5 (a) Force tracking performance with a full generalized mass-spring-damper impedance controller. (b) Force tracking performance with a generalized damper controller. The tracking performance of using the generalized damper (only keeps K_p in (5.9)) is similar but slightly worse than using a full generalized mass-spring-damper controller (keeps all gains in (5.9)). 75

5.6 Force tracking performance of the same trapezoidal force profile in z as in Figure 5.5 over 9 ILC iterations. Results from the second roll in each iteration for gradient descent direction computation are omitted. The RMS force tracking error reduces by over 78 % after 9 ILC iterations, as summarized in Table 5.1. 76

5.7 Comparison of tracking of the desired velocity profile (red) with the impedance controller (black), 0 ILC iteration (blue) and 9 ILC iterations (green). 78

5.8 Gradient update for force over the nine ILC iterations. The gradient converges with increase number of ILC iterations. 78

5.9 Sketch of a curved sample for robotic deep rolling (unit: inch). Image credit: Ken Myer. 79

5.10 Structure of the SISO system for ILC implementation. 80

5.11 Experimental setup in RobotStudio for curved sample deep rolling. 81

5.12 Experimental setup for robotic deep rolling with a curved sample. 83

5.13 Experimental setup for robotic deep rolling with a curved sample without singularity issues. 84

5.14 Force tracking performance in surface normal (1000 N desired force) over 30 ILC iterations. The RMS force tracking error reduces by over 71 % after 30 iterations. . . 85

5.15 Gradient update for commanded force magnitude along surface normal over 30 iterations. The gradient converges with increase number of ILC iterations. 86

5.16 Force tracking performance in surface normal (1000 N desired force) over 49 ILC iterations. The RMS force tracking error reduces by over 90 % after 49 iterations. . . 87

5.17 The time-varying learning rate α for gradient-based ILC. α is small at high curvature part of the curved sample to suppress oscillations. 87

5.18 Gradient update for commanded force magnitude along surface normal over 49 iterations. The gradient converges with increase number of ILC iterations. 88

A.1 EGM step responses with different input amplitudes (2 and 4 degrees) at two different robot configurations. 98

A.2 The fitted linear model for approximation of the locally linear EGM dynamics. (a) Comparison of step responses between EGM and the identified linear model. (b) Bode plot of the fitted linear model has a limited bandwidth and a large phase lag. . . 99

C.1 Three regions of convergence of $X(s)$ in s plane. 102

ACKNOWLEDGMENT

I would like to express great thanks to my advisor, Professor John T. Wen, for his support, guidance and encouragement during my PhD journey. I would like to thank other committee members, Professor Mamadou Diagne, Professor Agung Julius, and Professor Sandipan Mishra, for their highly appreciated advice. I also would like to thank Glenn Saunders, William Lawler, Steve Rock, and Ken Myer, for their important technical support. Thanks to Susan Miller and Beth Macey for them always being there to help out with all kinds of student affairs.

Great thanks to my brilliant labmates for their generous help and inspiring discussions: David Carabis, Yuan-Chih Peng, Kimberly Oakes, Jinda Cui, Honglu He, Alex Elias, Burak Aksoy, Dev Jivani, Jiawei Yin, Qi Jin, Jun Dong, Jinyu Su, Ruijie Geng, Brian Park, and many other friendly people at RPI who have helped me.

Love and deepest appreciation to my parents Longzhi Hou and Qixiang Chen, for their unconditional support. Thanks for all their concerns and love during the dark days. Finally, I would like to give my special thanks to my girlfriend Luyi Wang, for her accompany during the last five years. I feel very fortunate to meet her in my life.

ABSTRACT

Robot manipulators have been widely utilized in industrial automation to improve product quality and productivity. Common robotic tasks in an industrial setting include assembly, welding, surface treatment, packaging and labeling. The various tasks also stimulate the development of sophisticated control algorithms and dexterous tools for robot manipulators.

Typically for industrial robots, users have no access to motor torque control. Instead, a robot may be controlled by an external position or velocity command interface. This outer-loop external interface facilitates sensor-guided and robotic shared motion control. However, the robot trajectory tracking performance is affected by the unknown robot dynamics and joint servo controllers. The performance is further compromised by time delays in transmitting the external command as a setpoint to the inner control loop. Thus, it is necessary to compensate for the internal dynamics to improve the motion control performance. However, it is in general a difficult problem to analytically identify a highly nonlinear and coupled robot dynamical model. Instead, a universal function approximator such as a neural network may be adopted for modeling the system dynamics.

This thesis investigates the application of optimization and deep learning to robotics motion and force control. First, by leveraging a position control interface, a kinematic controller is developed by solving a quadratic-programming problem. However, the motion control performance is obviously compromised by the unknown internal dynamics when the robot is commanded a fast trajectory. To improve the motion control accuracy, we then develop a model-free gradient-based iterative learning control (ILC) algorithm with the gradient descent directions generated by the physical system. However, the algorithm may be time-consuming considering iterative updates, and the learned representation is not transferable to a new trajectory. This motivates the use of neural networks to approximate the system dynamics to improve the time efficiency and generalization of ILC. And a neural-learning control framework is proposed to address robot motion control in dynamics level through dynamics compensation using neural networks. Finally, the control scenario is extended from robot motion control to hybrid force and motion control, motivated by a robotic deep rolling task. We apply the model-free ILC algorithm to deep rolling on flat and curved surfaces.

CHAPTER 1
INTRODUCTION

1.1 Background and Motivation

Robot manipulators have been widely used for industrial tasks including assembly [1], welding [2], deburring [3], painting [4], packaging [5] and labeling [6]. They offer highly desirable features such as high repeatability, high speed, high flexibility, and large load capacities. For typical industrial robots, users do not have access to their motor torque control but only have access to an external position or velocity control interface. For example, MotoPlus of Yasakawa Motoman (2 ms sampling rate), Robot Sensor Interface (RSI) of Kuka (12 ms sampling rate) [7], Low Level Interface (LLI) of Stäubli (4 ms sampling rate), and Externally Guided Motion (EGM) of ABB (4 ms sampling rate) [8]. Examples of industrial robot-assisted manufacturing process are demonstrated in Figure 1.1.

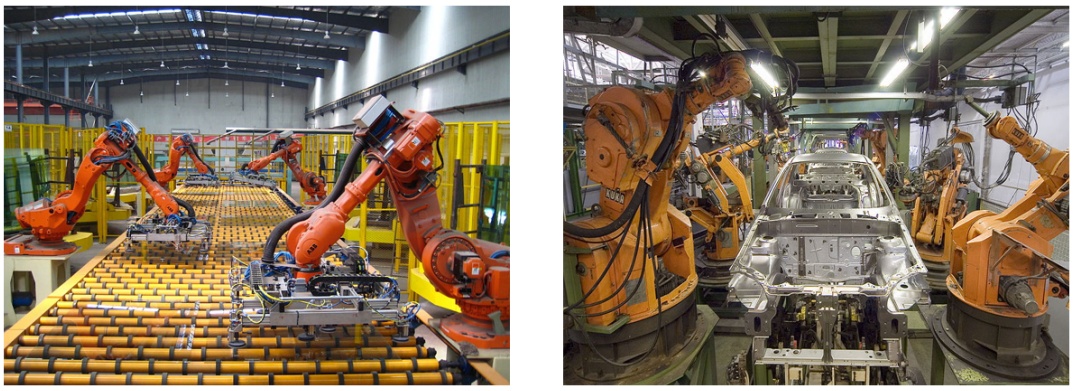


Figure 1.1: Examples of industrial robot-assisted manufacturing: ABB robots for float glass unloading (left) and KUKA robots for automobile assembly lines (right). Image credit: Wikimedia Commons.

Typical industrial robotic tasks can be classified into two categories. The primary task is essentially robot motion control in which the robot manipulators (possibly carrying a load) are

Portions of this chapter previously appeared in: S. Chen and J. T. Wen, "Neural-learning trajectory tracking control of flexible-joint robot manipulators with unknown dynamics," in *IEEE/RSJ IROS*, Nov. 2019, pp. 128-135.

Portions of this chapter previously appeared in: S. Chen and J. T. Wen, "Adaptive neural trajectory tracking control for flexible-joint robots with online learning," in *IEEE ICRA*, May 2020, pp. 2358-2364.

Portions of this chapter previously appeared in: S. Chen, Z. Wang, A. Chakraborty, M. Klecka, G. Saunders, and J. T. Wen, "Robotic deep rolling with iterative learning motion and force control," *IEEE Robot. Autom. Lett.*, vol. 5, no. 4, pp. 5581-5588, Oct. 2020.

commanded to track a given trajectory, such as robot-assisted wind turbine blades assembly (as shown in Figure 1.2 (left)). Robot manipulators are also adopted for tasks when contact happens between the environment and the robot end effector, such as deburring and rolling (Figure 1.2 (right)). When contact happens, both force and motion control requirements may be satisfied at the same time to guarantee accurate surface treatment without any damage. For both categories, the control performance is affected by the unknown robot dynamics and joint servo controllers, and by time delays in transmitting the external command as a setpoint to the inner control loop.

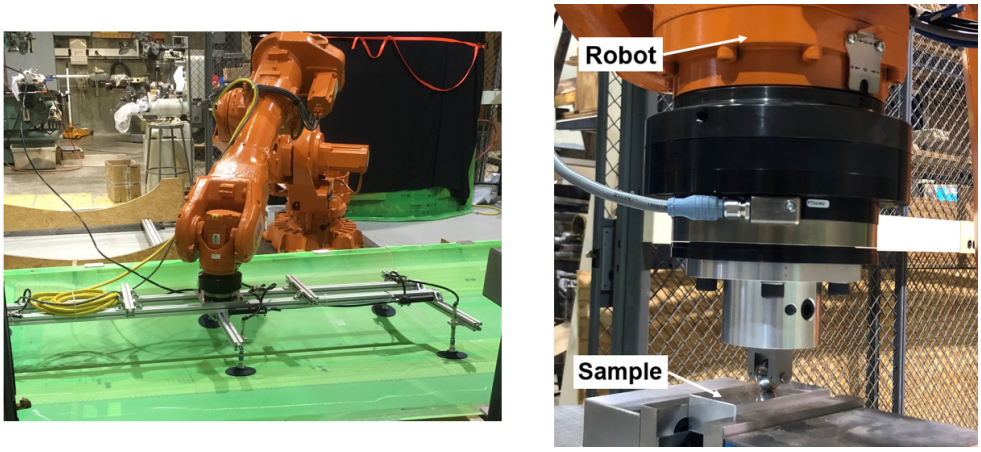


Figure 1.2: Examples of robot-assisted manufacturing projects conducted at RPI. An ABB robot is used for wind turbine blades assembly (left) and material deep rolling (right).

1.2 Problem Statement

This thesis investigates the application of optimization and deep learning techniques to robot motion and force control under outer-loop position control with unknown dynamics. The purpose is to improve the control performance for a robot-assisted industrial manufacturing task such as large structures assembly and deep rolling. The control challenges mainly come from the unknown dynamics of the robot, robot joint deflection, unknown robot joint servo controller, uncertainty of the contact, and motion-induced inertial force. We explore the use of deep learning for robot internal dynamics compensation, considering the ability of a neural network as a universal approximator. The thesis objective is to develop generic and effective control algorithms for a broad range of tasks including:

1. A human teleoperates a robot to transport a massive object in a constrained environment

with tight position and safety requirements. This scenario is common in industry such as the assembly of wind turbine blades.

2. A robot manipulator is commanded to track a given desired trajectory under the outer loop joint position control. The tracking performance is highly compromised by the unknown robot dynamics, especially for an aggressive commanded trajectory.

3. A robot is commanded to conduct a deep rolling task, which is a surface treatment process that applies specified pressure along a given path to the metal part surface to enhance its fatigue life. Deep rolling is essentially a hybrid force and motion control problem. Typically users only have access to a position or velocity control interface, and the contact may be unknown. In addition, the target sample may have complex geometry.

1.3 Related Work

1.3.1 Robot Manipulators Motion Control

Motivated by the need for fast and precise motion in industrial manufacturing, robot tracking control has long been of interest [9]. The typical control architecture is based on joint torque control which is challenging to implement on industrial robots. The controller structure usually consists of a linear stabilizing feedback controller and a feedforward compensation for trajectory tracking. If the robot dynamics is expressed in the linear-in-parameter form, an adaptive controller in joint-torque level has been developed to achieve asymptotic tracking [10]. However, accurate dynamical models of robot manipulators are rarely available, particularly for flexible-joint manipulators. For such robots, a simplified linearizable reduced-order model may be used [11] for control design, with model parameters obtained from offline identification. To further remove the requirement for the knowledge of the robot dynamics, a P-type iterative learning control (ILC) [12] may be used, but the learned representation is not transferable to new trajectories. To improve the generalization capability, numerous schemes have been proposed to control a robot through dynamics identification by wavelet networks [13], Gaussian Processes [14], Locally Weighted Projection Regression (LWPR), fuzzy logic systems [15] and neural networks (NNs) [16]. Neural network based controller receives particular interests in recent years due to its capability as a universal approximator.

In a general framework of neural-learning control, NNs are used to either approximate the robot forward dynamics or inverse dynamics for controller design [17]–[19], and the control prob-

lem falls within the context of either reinforcement learning [20], [21], adaptive control [22], or optimal control [23]. Radial basis function (RBF) NNs, multi-layer feedforward NNs, and recurrent neural networks have been explored for trajectory tracking control [24]–[27]. The concept of incorporating fuzzy logic into NN control has also grown into a popular research topic [28]–[30].

Radial basis function NN is a popular choice in neural adaptive controller design since it naturally expresses the dynamics approximation into a linear-in-parameter form [31]. In [32], an adaptive control strategy in joint torque level using RBF NNs was proposed to compensate for the unknown dynamics of a Baxter robot with an unknown payload. However, typically it requires a large amount of parameters to approximate robot dynamics accurately by using the RBF NNs, and it is a non-trivial task to select the centers and shapes of RBF kernels. Also, users usually only have access to a position or velocity setpoint control for industrial robots. In [33], a neural adaptive control technique was introduced for a trilateral teleoperation system with Dual-master/Single-slave manipulators. Sun et al. [34] proposed a NN-based sliding mode adaptive controller that achieved robust trajectory tracking control for robot manipulators subject to uncertainties. However, a torque control interface was required and only simulation results were presented. In [35], [36], an adaptive controller based on a two-layer feedforward fully-connected NN was proposed for robot manipulators motion control. The adaptation of the unknown parameters was derived by the stability analysis and Taylor series expansion. However, the adaptation law may get much more complex with more NN layers. Adaptive neural control methods have also been investigated for humanoid robots [37], [38]. Other control frameworks that use general function approximators for linearizing controllers design can be referred to in [39]–[41].

There have been several approaches using NNs to approximate a system inverse for dynamical compensation [42]–[44]. None of these approaches consider the possible non-causality in stable dynamical inversion. In [45], a double-NN architecture was introduced, with one NN for the nonlinear system identification and another one learning the system inverse. But the structure is complex and leads to a large amount of parameters. It has been shown that RNN outperforms Gaussian Processes in performance of learning robot inverse dynamics with linear time complexity, given sufficient training data [46]. A thorough survey of neural-learning robot manipulators control can be referred to in [47].

In this thesis, we develop a neural-learning framework for industrial robot trajectory tracking control using a position control interface. Four data-driven algorithms are reported within the framework in a progressive manner. For some industrial robots, a high-fidelity dynamic simulator

is available to facilitate the training data collection. If such a simulator is not available, an extra NN may be trained to approximate the robot dynamics. For these approaches, a large amount of training data are required to obtain an accurate dynamics approximation using neural networks. Finally, a more data-efficient neural adaptive control algorithm is reported based on stability analysis, and a linear-in-parameter form expression of robot dynamics using a neural network.

1.3.2 Robot Manipulators Force Control

Classical techniques that tackled the force control problem mostly focused on model-based control such as feedback linearization [48], [49] and impedance control [50], [51], which aimed at achieving a desired mechanical impedance. For instance, Raibert *et al.* [52] proposed one of the earliest hybrid position/force control algorithm based on feedback linearization, and two complementary orthogonal subspaces were developed to decouple the control of position and force. When industrial robots are involved in force control, extra challenges are introduced such as unmodeled dynamics, uncertainties, and robot joint deflection. However, the aforementioned approaches rely on exact model and environment knowledge, and a robot joint torque control interface. Thus, they are not applicable to industrial robot force control.

Force control algorithms based on universal function approximators such as neural networks for robot dynamics model approximation have been proposed [53], [54]. For example, Rani *et al.* [55] designed a neural adaptive controller based on dynamics approximation using radial basis function NNs for hybrid force and position control of a robot manipulator subject to uncertainties. The parameter adaptation law and tracking error convergence were based on stability analysis. However, the control was in joint-torque level and only simulation results were presented with a planar two-link manipulator.

Iterative learning control performs well in motion control [56] and has been evaluated with industrial robots [57]. It has also been extended to impedance control, admittance control, and force control. For the case of impedance control [58]–[60], a learning controller was designed such that a target impedance behaviour was followed. For admittance control [61], an ILC algorithm was proposed to gradually eliminate the interaction force for a peg-in-hole assembly task. As for robot force control [62]–[64], evaluation of the proposed techniques with physical systems was limited. On the other hand, Jeon *et al.* [65] proposed a torque-level iterative learning controller for robot position and force control and evaluated it with a physical 2 DOF SCARA robot, with the tracking error convergence guaranteed by stability analysis. Tahara *et al.*[66] developed an

ILC method for force/position trajectory tracking with a robot subject to non-holonomic rolling constraints. However, the control input was at the torque level, and only simulation results were presented. A hybrid control strategy for a dual-arm manipulation task using a P-type ILC was proposed in [67], where the measured position and force errors were fused by a Kalman filter with an experimentally identified contact model, and the fused position error was utilized by ILC for manipulator position compensation. However, the tracking error convergence analysis was not presented. In addition, all of the aforementioned works lack an experimental evaluation for control with a large target force (over 1000 N), which is typical in a robot deep rolling [68] task.

Industrial robot force control remains an open research area due to challenges with unknown dynamics, environments and lack of robot torque control. In this thesis, we develop a model-free ILC algorithm that aims at addressing the robot deep rolling task using a position control interface. The goal is to relax the industrial robots' barrier to entry into surface treatment processes by improving the robot force control accuracy while retaining all the benefits of industrial robots, such as flexibility, and low-cost.

1.4 Thesis Contributions

The main contribution of this thesis is the design and implementation of novel learning control algorithms for robotics trajectory tracking control and hybrid force/motion control. We develop a gradient-based iterative learning control algorithm for both control scenarios, and also explore the use of neural networks for robot dynamics compensation. All control algorithms are based on a joint position control interface, which is available for most industrial robots. The specific contributions are itemized as follows:

1. The development of a resolved velocity controller that finds commanded joint position by solving a quadratic programming problem (Chapter 2). The controller can be used in collision-free teleoperation and path planning. Compared with many sampling-based path planners such as the rapidly exploring random tree, this kinematic controller plans the path directly in task space rather than in the configuration space. In addition, sampling-based path planners may produce non-intuitive solutions.

2. The development of a robust and model-free gradient-based iterative learning control algorithm (Chapter 3).

- We provide tracking error convergence conditions and robustness analysis for the algorithm.
- The standard ILC is mostly based on a linear proportional update. Also, the tracking error convergence requires some system properties such as passivity and relative degree. In contrast, this algorithm uses a trajectory-based gradient descent approach, and it finds the gradient descent directions of the entire trajectory using the physical system thus no explicit model information is required.
- This algorithm only requires a joint position control interface and is applicable to most industrial robots.
- Most ILC algorithms are only for motion control. This gradient-based ILC algorithm is also applicable to hybrid force/motion control, as demonstrated in this thesis for a robotic deep rolling task.

3. The development and implementation of a learning control framework for robotics joint position trajectory tracking control (Chapter 4).

   - Compared with many neural learning control approaches, this neural learning control framework only requires a joint position control interface rather than a torque-level control.
   - We explore different neural network architectures to approximate robot dynamics information, including multi-layer feedforward neural networks, recurrent neural networks, and a customized directed acyclic graph. These neural networks are used as a feedforward compensator for robot motion control.
   - We use the system domain knowledge such as causality as a guidance to select neural network architectures. We report the use of unidirectional recurrent neural networks to approximate the causal robot forward dynamics, and the use of multi-layer feedforward neural networks and bidirectional recurrent neural networks to model the non-causal behavior of the stable inverse of a non-minimum-phase or strictly proper system.
   - We develop a novel neural adaptive controller for the tracking control of a flexible-joint robot under the joint position control. This controller is based on a neural network which expresses the robot dynamics into a linear-in-parameter form (a regressor multiplied by an output layer). The parameters update of the regressor and output layer

happens at different rates for decoupling purposes, and this regressor online learning speeds up the tracking error convergence.

4. The development and implementation of a new algorithm for robotic deep rolling task, which is essentially a hybrid force and motion control problem (Chapter 5).

    - We apply the ILC algorithm described in Chapter 3 to rolling on a flat sample, by designing a system with a symmetric impulse response matrix. The key to the design is that, for rolling on the flat sample, the 1-dimensional motion and force are decoupled in different Cartesian directions, and the robot inner loop dynamics is nearly diagonal as seen from outer loop.
    - We extend the ILC algorithm to rolling on a curved sample where the motion and force are 3-dimensional in the robot world frame, by designing a single-input and single-output system, which naturally has a symmetric impulse response matrix.
    - This algorithm requires no knowledge of system dynamics and contact information, but uses the physical process to generate the information to iteratively update the input.

1.5 Publications Resulting from This Thesis

1. S. Chen, Y. Peng, J. Wason, J. Cui, G. Saunders, S. Nath, and J. T. Wen, "Software framework for robot-assisted large structure assembly," presented at the ASME 13th MSEC, College Station, TX, USA, June 18-22, 2018.

2. S. Chen and J. T. Wen, "Neural-learning trajectory tracking control of flexible-joint robot manipulators with unknown dynamics," in *IEEE/RSJ IROS*, Macau, China, 2019.

3. J. T. Wen, J. Wason, D. Kruse, Y. Peng, and S. Chen, "Collaborative industrial robot control: from safe motion to multi-robot manipulation," in *Recent Advances in Industrial Robotics, edited by Craig Schlenoff, Venkat Krovi, Satyandra Gupta, Advanced Manufacturing Book series*, 2019.

4. S. Chen and J. T. Wen, "Adaptive neural trajectory tracking control for flexible-joint robots with online learning," in *IEEE ICRA*, Paris, France, 2020.

5. S. Chen, Z. Wang, A. Chakraborty, M. Klecka, G. Saunders, and J. T. Wen, "Robotic deep rolling with iterative learning motion and force control," *IEEE Robot. Autom. Lett.*, vol. 5, no. 4, pp. 5581-5588, Oct. 2020.

6. S. Chen and J. T. Wen, "Industrial robot trajectory tracking control using multi-layer neural networks trained by iterative learning control," *Robotics*, 2021, 10(1), 50.

7. Y. Peng, S. Chen, D. Jivani, J. Wason, W. Lawler, G. Saunders, R. Radke, J. Trinkle, S. Nath, and J. T. Wen, "Sensor-guided assembly of segmented structures with industrial robots," *Appl. Sci.*, 2021, 11, 2669.

8. S. Chen, A. Elias, and J. T. Wen, "Robot deep rolling for surface with complex geometries," (under preparation).

CHAPTER 2
ROBOT MANIPULATORS KINEMATIC CONTROL

This chapter presents the development of a resolved velocity controller which allows robot manipulation without concern of collision or joint limits violation. By incorporating various sensors, this controller achieves sensor-based robot control such as compliance force control and position/image-based visual servoing. The controller assumes the availability of a joint position control interface and is readily adapted to other industrial robots. This chapter presents preliminary results of the collision avoidance feature of the controller for robot teleoperation using an ABB robot, and it motivates the necessity for robot dynamics compensation to improve the trajectory tracking performance.

2.1 Kinematic Control Strategy

In the context of outer-loop sensor-guided control, the adjustment of the robot joint motion is achieved through an external position control interface. The resolved velocity controller aims at determining the robot joint motion at each time instant by considering a series of constraints such as joint limits, singularity and obstacle avoidance, maintenance of a desired end-effector orientation, etc. Without the concern of singularity, the robot joint angular velocity $\dot{q}$ (rad/sec) can be computed from the desired end-effector spatial velocity V_d by:

$$\dot{q} = J(q)^{-1} V_d, \tag{2.1}$$

where q is the robot joint position (in radian), and $J(q)$ is the manipulator Jacobian matrix. The spatial velocity $V_d \in \mathbb{R}^6$ is in form of:

$$\begin{bmatrix} \omega_d \\ v_d \end{bmatrix}, \tag{2.2}$$

where $\omega_d \in \mathbb{R}^3$ and $v_d \in \mathbb{R}^3$ are desired angular velocity and linear velocity of robot end effector represented in the robot world frame. Considering the possibility of a singular manipulator Jacobian, instead of solving the inverse Jacobian directly, we design the resolved velocity controller by

Portions of this chapter previously appeared in: S. Chen *et al.*, "Software framework for robot-assisted large structure assembly," presented at the ASME 13th MSEC, College Station, TX, USA, Jun. 18-22, 2018.

solving the following constrained quadratic programming (QP) problem at each control instant:

$$\dot{q} = arg\min_{\dot{q}} ||J(q)\dot{q} - V_d||^2, \tag{2.3}$$

subject to

$$\begin{aligned} &\frac{dh_{E_i}}{dt} = \frac{\partial h_{E_i}}{\partial q}\dot{q} = -k_{E_i}h_{E_i},\ i = 1,\ldots,n_E \\ &\frac{dh_{I_i}}{dt} = \frac{\partial h_{I_i}}{\partial q}\dot{q} \geqslant \sigma_i(h_{I_i}),\ i = 1,\ldots,n_I \\ &\dot{q}_{\min} \leqslant \dot{q} \leqslant \dot{q}_{\max}, \end{aligned} \tag{2.4}$$

where h_{E_i} and h_{I_i} are equality and inequality constraints, which will be described in Sections 2.1.1 and 2.1.2, respectively. n_E and n_I are the number of equality and inequality constraints. $k_{E_i} > 0$ is the proportional control gain. $\dot{q}_{\min}$ and $\dot{q}_{\max}$ are the minimum and maximum joint velocity vectors. As a result, the solved robot joint motion $\dot{q}$ stays close to the desired end-effector velocity V_d but may be scaled down to respect the constraints. The control barrier function $\sigma_i(h_{I_i})$ drives the robot back to the feasible region when the inequality constraints are violated. QP is a convex optimization problem, and numerical computing packages are available to efficiently solve the unique solution, such as the `quadprog` function in Matlab, and Python packages `cvx, quadprog`. After solving for $\dot{q}$, the commanded joint position q_c at time $t+1$ is computed by:

$$q_c(t+1) = q_c(t) + \dot{q} \cdot dt, \tag{2.5}$$

where dt is the sampling rate of the external joint position control interface.

2.1.1 Equality Constraints

To avoid solving an optimal control problem, we convert the equality constraint $h_E(q)$ into a differential constraint to implement the feedback velocity control: $\frac{dh_E(q)}{dt} = \frac{\partial h_E(q)}{\partial q}\dot{q} = -k_E h_E$. In order to maintain a constant desired end-effector orientation R_d, we use the angular velocity ω_d as a feedback control (with optimization variable $\dot{q}$ derived from the robot manipulator kinematics) to drive the actual end effector orientation R to R_d:

$$\omega_d = J_r\dot{q} = -k_\omega \sin(\frac{\theta}{2})k, \tag{2.6}$$

where J_r is the rotational part of the manipulator Jacobian, and $k_\omega > 0$ is a proportional control gain. (θ, k) is the equivalent angle/axis representation for the orientation error RR_d^T.

2.1.2 Inequality Constraints

The inequality constraint $h_I(q)$ includes the joint position limits and the minimum distance between the robot and the obstacles in the workspace. We again use the differential form of the constraint for determination of the instantaneous feasible direction as in (2.4). The control barrier function σ_i, as shown in Figure 2.1, is a function of h_{I_i} such that h_{I_i} increases with the rate $\leq e_i$ when $h_{I_i} \leqslant \eta_i$, and gradually slows down until it becomes feasible, i.e., $h_{I_i} \geqslant \eta_i$:

$$\sigma_i(h_{I_i}) = \begin{cases} -tan(c_i\pi) & \text{if} \quad h_{I_i} \geqslant \eta_i + \varepsilon_i \\ -tan(\frac{c_i\pi(h_{I_i}-\eta_i)}{\varepsilon_i}) & \text{if} \quad \eta_i < h_{I_i} < \eta_i + \varepsilon_i \\ \frac{e_i(\eta_i - h_{I_i})}{\eta_i} & \text{if} \quad 0 \leqslant h_{I_i} \leqslant \eta_i \\ e_i & \text{otherwise} \end{cases}, \tag{2.7}$$

where $c_i, \eta_i, \varepsilon_i, e_i$ are positive constants. For the scenario of obstacle avoidance, the physical meaning of each constant is as follows: ε_i denotes the distance from the obstacle where the speed reduction occurs, and η_i infers the size of the buffer zone. e_i and c_i control the time rate of change of the optimization variable [69].

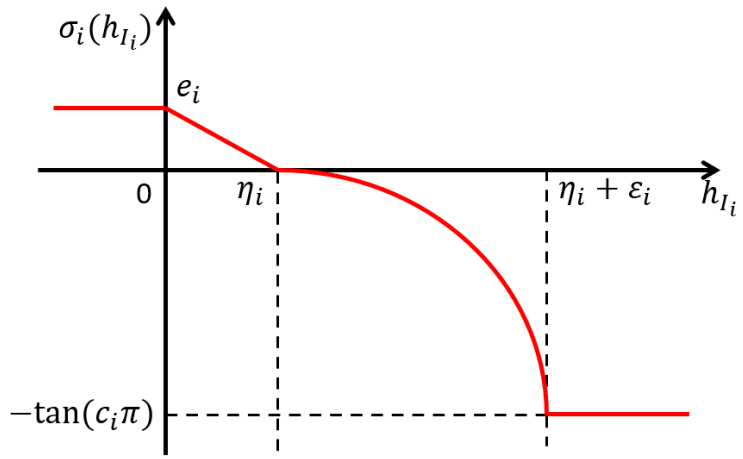


Figure 2.1: Control barrier function σ_i as a function of the inequality constraint h_{I_i}. When h_{I_i} is violated ($h_{I_i} \leqslant \eta_i$), σ_i is designed to drive h_{I_i} toward a feasible region.

The following inequality constraints are considered during implementation:

1. Joint position limits, which may be provided by the manufacturer:

$$q_{min} \leqslant q \leqslant q_{max}. \tag{2.8}$$

2. Minimum distance d between the robot and obstacles, which can be measured by a collision checker such as Tesseract or OpenRAVE, given the collision model of robot and the environment. Let p and p_{env} be the points on the robot and the environment that are the closest to collision at one time instant, then $d = \|p - p_{env}\| > d_{min}$, where $d_{min} > 0$ is a safety threshold.

2.2 Experimental Results

We adopt the kinematic controller to safely transport an object attached at the robot end-effector in a testbed. An operator provides the desired end-effector velocity V_d through an input device such as a joystick, to teleoperate the robot for the object transportation. The software and hardware components involved in the experiment are summarized below.

2.2.1 System Description

ABB IRB 6640/180 Robot

We employ the ABB IRB 6640/180 articulated robot [70], which has 6 degrees of freedom (DOF) with a farthest reach of 2.55 m. The robot features a high payload of 180 kg and high performance of 0.07 mm position repeatability and 0.7 mm path repeatability. The robot is also easy to maintain due to its simplified mechanism design, and it is widely used in industry for its relatively low cost. Figure 2.2 shows the robot kinematic parameters, which are provided by the manufacturer.

Externally Guided Motion Interface

Typical industrial robots offer an external interface to allow for the adjustment of the joint position or velocity. The ABB controller has an optional *Externally Guided Motion* module [8] which provides an external interface of commanded joint angles, $q_c \in \mathbb{R}^6$, and joint angle measurements $q \in \mathbb{R}^6$ at a 4 ms rate. EGM is a feature of RobotWare [8] (ABB's Robot Controller Software), and is also available in RobotStudio (ABB's high-fidelity dynamic simulator) [72]. A preliminary characterization of EGM dynamics is presented in Appendix A.

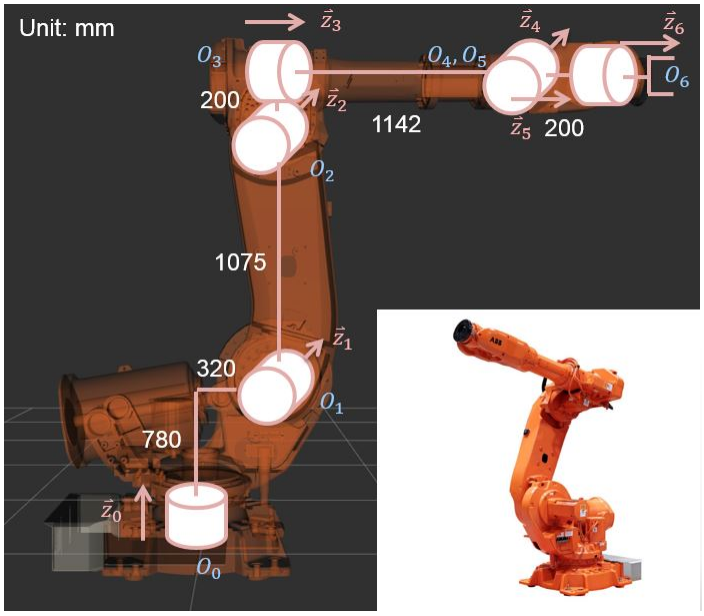


Figure 2.2: Kinematic parameters of ABB IRB6640 180/255 robot. Reprinted by permission from ASME: "Software Framework for Robot-Assisted Large Structure Assembly," S. Chen, Y. Peng, J. Wason, J. Cui, G. Saunders, S. Nath, and J. T. Wen [71], Paper No: MSEC2018-6689.

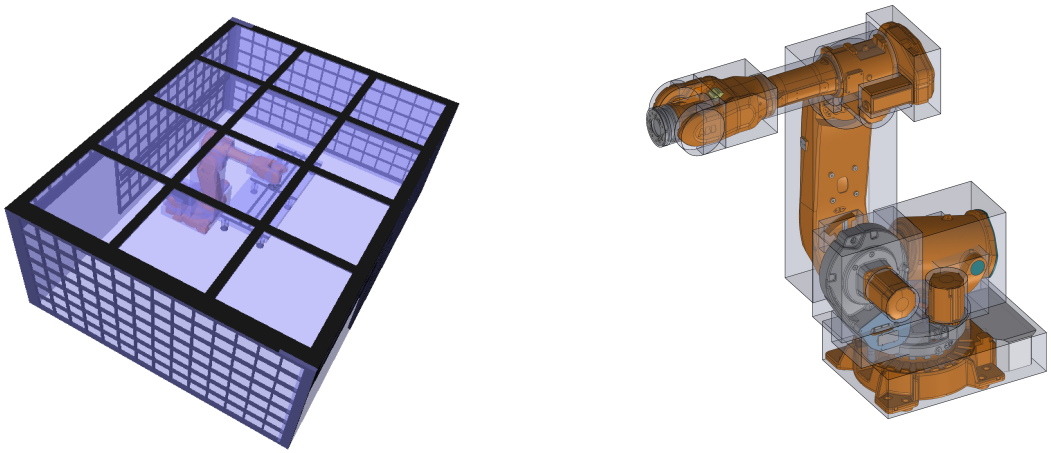
Tesseract

We use Tesseract as the collision checker to obtain the minimum distance d between the robot and the environment, as well as the corresponding position p on the robot, and p_{env} in the environment. To increase the collision checking speed, we adopt bounding boxes to approximate the geometry of the robot, end-effector tools such as a vacuum gripper, and the environment, as shown in Figure 2.3. With the simplified collision geometry, the collision checking rate is around 10 Hz.

2.2.2 Implementation and Results

We tune the control parameters in (2.7) and we set $d_{min}, c_i, \eta_i, \varepsilon_i, e_i$ to be 0.3, 0.5, 0.1, 0.15 and 0.001, respectively. Thus, the speed reduction begins at 0.45 m with a buffer size of 0.1 m from the obstacle, which indicates that the robot will start decelerating at 0.55 m from the obstacle. We adopt this group of parameters to guarantee that there is no collision during the object transportation as described in Figure 2.4, and the safety buffer zone is not too conservative such that the actual commanded velocity deviates a lot from the user desired spatial velocity.

Here, we teleoperate the robot using a joystick for an object transportation in a simulated



(a) Testbed collision model **(b) Robot collision model**

Figure 2.3: Use of bounding boxes to speed up the collision checking. (a) Simplified collision models of the robot, vacuum gripper and testbed. (b) Robot collision model rendered in SolidWorks. Reprinted by permission from ASME: "Software Framework for Robot-Assisted Large Structure Assembly," S. Chen, Y. Peng, J. Wason, J. Cui, G. Saunders, S. Nath, and J. T. Wen [71], Paper No: MSEC2018-6689.

testbed with a start position as shown in Figure 2.4 (a). The joystick generates a constant desired end-effector velocity V_d, including a constant linear velocity (denoted by the directed red arrow) and a constant angular velocity (denoted by the directed red circle). The actual commanded linear velocity and angular velocity solved using the kinematic controller are denoted by the green arrow and circle, respectively. We can observe that as the object is far from the wall and there is no risk of collision, the actual commanded velocity is the same as the desired constant velocity (as seen in Figure 2.4 (b)). As the object gets closer to the wall and falls within the safety buffer, the commanded velocity is in the direction that drags the object away from the wall (as shown in Figure 2.4 (c)), though the desired velocity remains the same. The closer the object is to the wall, the larger the commanded velocity is (as denoted by the magnitude of the arrow in Figure 2.4 (d)). When the object is further from the wall, the commanded velocity still drags the object away from the wall but has a smaller magnitude (as shown in Figure 2.4 (e)). When there is no risk of collision, the commanded velocity is again the same as the constant desired velocity (as shown in Figure 2.4 (f)).

Using the described kinematic controller, the robot stays close to the human commanded motion while remains inside a feasible region. The controller produces a commanded velocity to

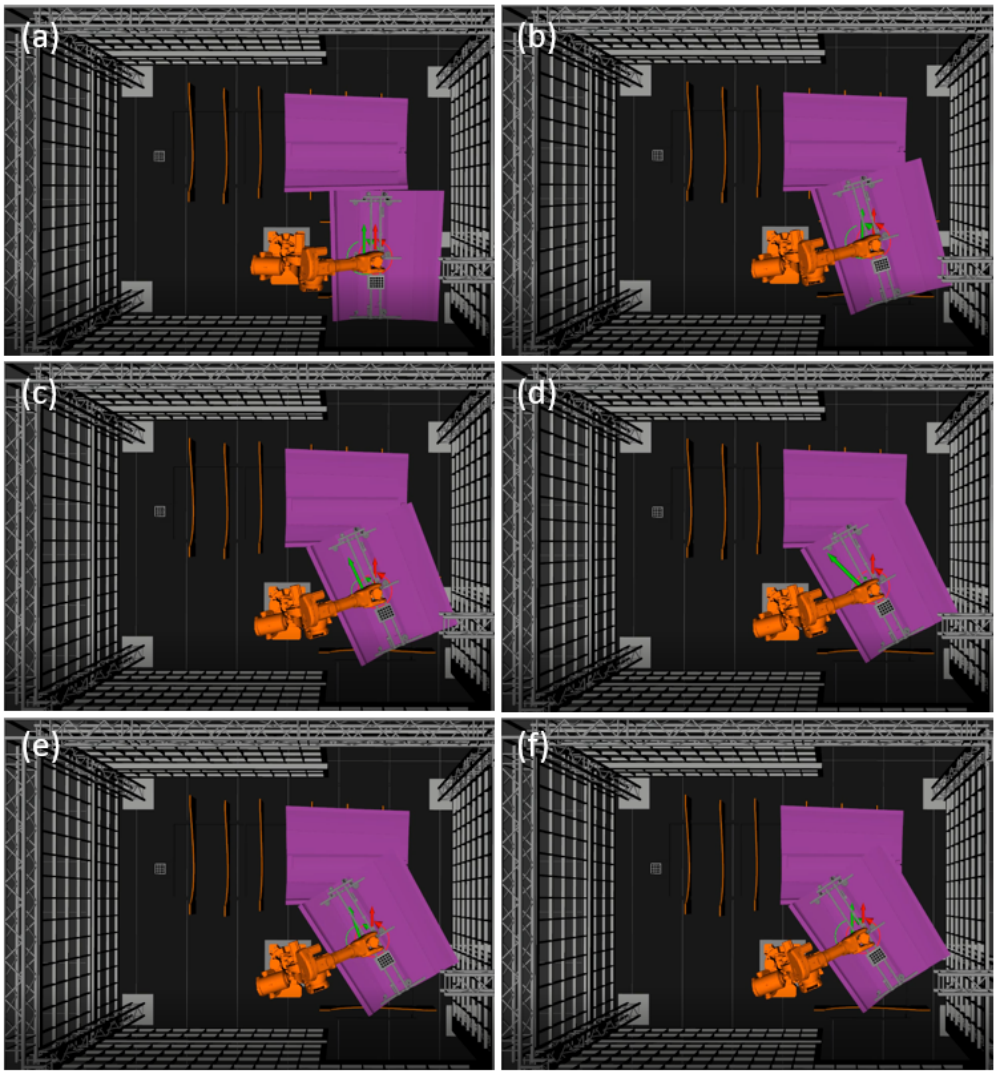


Figure 2.4: Top-down view of the simulated collision avoidance behavior during the object transportation through robot teleoperation using the kinematic controller. The green and red arrows denote the actual commanded velocity computed by the kinematic controller and the desired velocity, respectively. (a) Robot initial pose. (b) The actual commanded velocity is the same as the desired velocity when there is no risk of collision with the wall. (c) The actual commanded velocity is different from the desired velocity and drags the object away from the wall when there exists a risk of collision. (d) The magnitude of the commanded velocity is larger when the object is closer to the wall. (e) The magnitude of the commanded velocity reduces when the object is further from the wall. (f) The actual commanded velocity is again the same as the desired velocity when there exists no risk of collision with the wall.

avoid collision, but the robot may get stuck as there may be multiple nearest points to obstacles. In that case, the human operator is responsible for steering the robot away from the obstacle towards

a feasible direction.

2.3 Summary

This chapter reports a kinematic controller for adjustment of robot joint positions by solving a quadratic programming problem. The proposed resolved velocity controller finds the optimal solution abiding the designed penalty functions under a set of equality and inequality constraints. We demonstrate the collision avoidance feature of the controller through an object transportation task. The controller can be used in teleoperation and path planning. Compared with the potential field and many sampling-based path planners such as the rapidly exploring random tree (RRT), the resolved velocity controller allows motion planning in the task space rather than in the configuration space. Also, these sampling-based path planners may generate non-intuitive solutions.

However, we observe a remarkable trajectory tracking error (as shown in Figure 2.5) when commanded an aggressive trajectory using the kinematic controller to the EGM interface. The magnitude attenuation and phase lag are also identified from the Bode plot of the LTI approximation of the locally linear robot dynamics in Figure A.2b in Appendix A. This motivates the development of a gradient-based iterative learning control algorithm for robot inner loop dynamics compensation to improve the trajectory tracking accuracy.

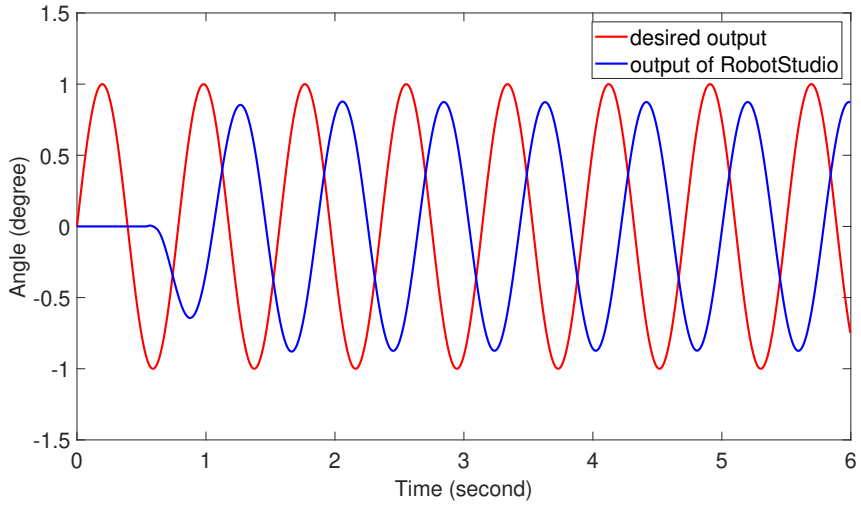


Figure 2.5: EGM motion control performance is compromised by the nonlinear robot inner loop dynamics. The output (blue curve) shows a significant tracking error in both amplitude and phase compared with the desired trajectory (red curve).

CHAPTER 3
GRADIENT-BASED ILC FOR NONLINEAR SYSTEMS

The compromised trajectory tracking accuracy due to the nonlinear robot inner loop dynamics motivates the development of a gradient-based iterative learning control algorithm in this chapter. Iterative learning control [73] is an effective control tool for improving the tracking performance and transient response of dynamic systems that operate in a repetitive manner. Compared with numerous traditional controllers, ILC can achieve perfect tracking even with the presence of unmodeled dynamics or parametric uncertainties. Essentially, ILC is a data-driven approach that aims to find the inverse of a system subject to predetermined uncertainty and external disturbances. The purpose of this chapter is to present a robust and model-free gradient-based ILC algorithm for a general nonlinear system, and also provides robustness condition and monotonic convergence analysis of the $\mathcal{L}_2$ norm tracking error in the presence of uncertainty and modeling errors.

3.1 Generic ILC Formulation

Let $L_2^n[0,T]$ denote $\mathbb{R}^n$ square integrable functions on $[0,T]$ with the inner product $< v,w >= \int_0^T v(t)^T w(t)\,dt$ and norm $\|v\|^2 = \int_0^T v(t)^T v(t)\,dt$. Consider the mapping of an input trajectory $\underline{u} \in L_2^n[0,T]$ to an output trajectory $\underline{y} \in L_2^n[0,T]$ with a fixed initial condition, denoted by $\underline{y} = \mathcal{G}(\underline{u})$ where $\mathcal{G} : L_2^n[0,T] \to L_2^n[0,T]$ is an operator, possibly nonlinear. Given a desired output $\underline{y}_d \in L_2^n[0,T]$, the goal is to find $\underline{u}$ to minimize $\left\|\underline{y}-\underline{y}_d\right\|$ where the norm is in the $L_2^n[0,T]$ sense. Iterative learning control is basically an iterative algorithm to update $\underline{u}$ to reduce the tracking error $\underline{e} := \underline{y} - \underline{y}_d$:

$$\underline{u}_{k+1} = \underline{u}_k - \alpha\mathcal{L}(\underline{e}_k), \tag{3.1}$$

where α is sometimes called the learning rate, and k is the iteration number. In the ILC literature, the update law $\mathcal{L}$ does not explicitly depend on the model parameters, but needs certain properties of $\mathcal{G}$ to ensure convergence (e.g., gain, passivity, positive realness). For instance, if the system has relative degree one or less, a classical proportional type (P-type) ILC has been developed [74]. Variations of the classical P-type ILC have also been investigated including PI-type [75] and PID-

Portions of this chapter previously appeared in: S. Chen, Z. Wang, A. Chakraborty, M. Klecka, G. Saunders, and J. T. Wen, "Robotic deep rolling with iterative learning motion and force control," *IEEE Robot. Autom. Lett.*, vol. 5, no. 4, pp. 5581-5588, Oct. 2020.

type ILC [76]. Convergence analysis based on the strictly proper and positive realness of the plant has also been reported [77]. Many ILC algorithms have been developed for linear systems [78], [79] where $\mathcal{L}$ is chosen to ensure the convergence of $\underline{e}$ to zero. Such technique has been applied to industrial robot motion control based on linearized analysis [80]. For nonlinear systems such as robots, passivity-based convergence condition has been developed in [73]. A thorough review of ILC is available in [81].

3.2 Gradient-Based ILC

The standard gradient descent algorithm to minimize $\|\underline{e}\|$ results in the following ILC update law:

$$\underline{u}_{k+1} = \underline{u}_k - \alpha G_k^* \underline{e}_k, \tag{3.2}$$

where G_k^* is the adjoint of $G_k := \nabla_{\underline{u}} \mathcal{G}(\underline{u}_k)$, the gradient (Fréchet derivative) of the nonlinear operator $\mathcal{G}$. Fréchet derivative is used here since $\underline{u}$ is a function of time. The learning rate α is a sufficiently small constant, which can be tuned through simulation or experiment. If α is too small, it may take a large amount of iterations until convergence. On the other hand, a large α may result in error divergence.

Consider the following state space representation of the nonlinear dynamical system $\mathcal{G}$:

$$\begin{aligned} \dot{x} &= f(x,u), \quad x(0) = x_0 \\ y &= h(x,u). \end{aligned} \tag{3.3}$$

Then G_k is a linear time varying (LTV) system – the linearization of $\mathcal{G}$ about the trajectory $\{(u(t),x(t)), t \in [0,T]\}$, and is described in the following state space form:

$$\begin{aligned} \delta\dot{x} &= \underbrace{\frac{\partial f(x(t),u(t))}{\partial x}}_{A(t)} \delta x + \underbrace{\frac{\partial f(x(t),u(t))}{\partial u}}_{B(t)} \delta u, \ \ \delta x(0) = 0 \\ \delta y &= \underbrace{\frac{\partial h(x(t),u(t))}{\partial x}}_{C(t)} \delta x + \underbrace{\frac{\partial h(x(t),u(t))}{\partial u}}_{D(t)} \delta u. \end{aligned} \tag{3.4}$$

Then G_k is an operator that maps from δu to δy. The adjoint G_k^* is also an LTV system. If G_k is approximately time-invariant, then the gradient and its adjoint, G_k and G_k^*, are related through their impulse response kernels as proved in Appendix B:

$$G_k^* = \mathcal{T} G_k^{\mathsf{T}} \mathcal{T}, \tag{3.5}$$

where $\mathcal{T}$ is the time reversal operator [82], and G_k^{T} is the LTV system with the transpose of the impulse response of G_k. Substituting into (3.2), we have

$$\underline{u}_{k+1} = \underline{u}_k - \alpha \mathcal{T} G_k^{\mathsf{T}} \mathcal{T} \underline{e}_k. \tag{3.6}$$

If the impulse response of G_k is symmetric, then $G_k^{\mathsf{T}} = G_k$. This includes the special case when $\mathcal{G}$ is diagonal. In this case, we have

$$\underline{u}_{k+1} = \underline{u}_k - \alpha \mathcal{T} G_k \mathcal{T} \underline{e}_k. \tag{3.7}$$

It may appear that model information, i.e., $\mathcal{G}$, is needed in implementing this update law. However, as proposed in [56] for a single-input/single-output high speed galvo positioning system, $\mathcal{T} G_k \mathcal{T} \underline{e}_k$ may be obtained by using the time reversal filter with the physical system itself instead of relying on a model. The desired output trajectory $\underline{y}_d$ is usually selected as the initial guess $\underline{u}_0$ to start the iteration.

Under the symmetric kernel assumption on $\nabla\mathcal{G}$, the $\mathcal{T} G_k \mathcal{T} \underline{e}_k$(here identical to $G_k^* \underline{e}_k$) term in (3.7) may be computed *without any knowledge of* $\mathcal{G}$ by using the algorithm below (illustrated in Figure 3.1):

Algorithm 1 *Given $\underline{u}_k$, apply the following steps to compute $G_k^* \underline{e}_k$:*

(a) Apply $\underline{u}_k$ to the physical system and obtain the output $\underline{y}_k = \mathcal{G}(\underline{u}_k)$ and tracking error $\underline{e}_k = \underline{y}_k - \underline{y}_d$.

(b) Time-reverse $\underline{e}_k(t)$ to $\tilde{\underline{e}}_k(t) := \underline{e}_k(T-t)$.

(c) Apply the augmented input $\underline{u}'_k(t) = \underline{u}_k(t) + \tilde{\underline{e}}_k(t)$ to the physical system and obtain the output $\underline{y}'_k = \mathcal{G}(\underline{u}'_k)$ under the same initial configuration as in step (a).

(d) Compute $\underline{e}'_k = \underline{y}'_k - \underline{y}_k$, and reverse it in time again to obtain the gradient direction $G_k^* \underline{e}_k(t) = \underline{e}'_k(T-t)$.

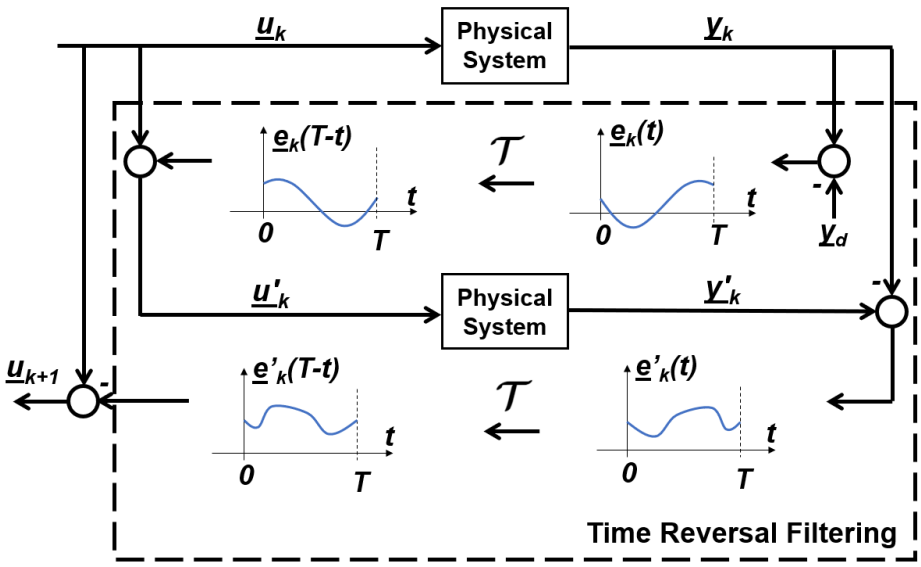


Figure 3.1: Block diagram of one iteration of the gradient-based ILC algorithm, which uses time reversal filtering to generate the gradient descent directions with the physical system.

3.3 Robustness Condition and Convergence Analysis

Consider the first order variation of $\delta \underline{e}$ resulting from a small change of the input $\delta \underline{u}$:

$$\delta \underline{e} = \underbrace{\nabla_{\underline{u}} \mathcal{G}(\underline{u})}_{:=G} \delta \underline{u}. \tag{3.8}$$

Substituting in the gradient update law

$$\delta \underline{u} = -\alpha G^* \underline{e}, \tag{3.9}$$

we have

$$\delta \underline{e} = -\alpha G G^* \underline{e}, \tag{3.10}$$

which guarantees $G^* \underline{e}$ converges to zero. In the iterative implementation, (3.9) becomes (3.2), provided α is sufficiently small. In the model-free implementation of the gradient descent algorithm, under the assumption that G is approximately time-invariant and has a symmetric impulse response matrix, (3.9) becomes

$$\delta \underline{u} = -\alpha \mathcal{T} G \mathcal{T} \underline{e}, \tag{3.11}$$

which becomes (3.7) in the iterative form. The convergence requires G to have a symmetric kernel, $G^{\mathsf{T}} = G$. However, this property may only be approximately true for the physical system. For example, for a nearly diagonal G, the diagonal portion satisfies the symmetric kernel condition, but we need to consider the effect of the off-diagonal terms. In this section, we derive a robustness bound for the anti-symmetric portion of the kernel to ensure the convergence of the tracking error in (3.11).

Decompose G into a symmetric kernel and an anti-symmetric kernel part:

$$G = G_s + G_a, \tag{3.12}$$

where $G_s^{\mathsf{T}} = G_s$ and $G_a^{\mathsf{T}} = -G_a$. It follows that

$$\begin{aligned} G_s^* &= \mathcal{T} G_s \mathcal{T} \\ G_a^* &= \mathcal{T} G_a^{\mathsf{T}} \mathcal{T} = -\mathcal{T} G_a \mathcal{T}. \end{aligned} \tag{3.13}$$

The update law (3.11) is now

$$\begin{aligned} \delta \underline{u} &= -\alpha \mathcal{T}(G_s + G_a)\mathcal{T}\underline{e} \\ &= -\alpha \mathcal{T}(G_s - G_a + 2G_a)\mathcal{T}\underline{e} \\ &= -\alpha \mathcal{T}((G_s + G_a)^{\mathsf{T}} + 2G_a)\mathcal{T}\underline{e} \\ &= -\alpha G^* \underline{e} + 2\alpha \mathcal{T} G_a^{\mathsf{T}} \mathcal{T}\underline{e} \\ &= -\alpha G^* \underline{e} + 2\alpha G_a^* \underline{e}. \end{aligned} \tag{3.14}$$

Substitute the update law (3.14) into the error variation equation (3.8), we have

$$\delta \underline{e} = -\alpha G G^* \underline{e} + 2\alpha G G_a^* \underline{e}. \tag{3.15}$$

Assume GG^* is positive definite (required for the convergence of $\underline{e}$ in the gradient algorithm to zero), i.e.,

$$\langle \underline{e}, GG^* \underline{e} \rangle \geq \gamma_G \|e\|^2 \tag{3.16}$$

for some $\gamma_G > 0$. If the anti-symmetric portion of G is sufficiently small:

$$\|G_a\| \leq \frac{\gamma_G}{2\|G\|}, \tag{3.17}$$

then the iteration (3.7) will converge.

3.4 Gradient-Based ILC for Trajectory Tracking Control with EGM

In this section, we evaluate the gradient-based ILC algorithm for trajectory tracking accuracy improvement with EGM. The ABB industrial robot inner-loop control tightly regulates each joint, and the inner loop dynamics is almost diagonal as seen from the outer loop. In addition, the robot inner loop dynamics is locally time invariant. Thus, we may directly apply the gradient-based ILC to compensating for the dynamics of each joint, which is a single-input/single-output (SISO) system and has a symmetric impulse response kernel. Figure 3.2 shows the trajectory tracking improvement through EGM dynamics compensation of the sinusoidal signal in Figure 2.5 using the gradient-based ILC in RobotStudio. The uncompensated output in Figure 2.5 shows a significant tracking error in both amplitude and phase due to the internal dynamics. Figure 3.2 illustrates the updated input $\underline{u}$ and output $\underline{y}$ with 8 ILC iterations. After the initial transient, $\underline{y}$ and $\underline{y}_d$ are almost indistinguishable.

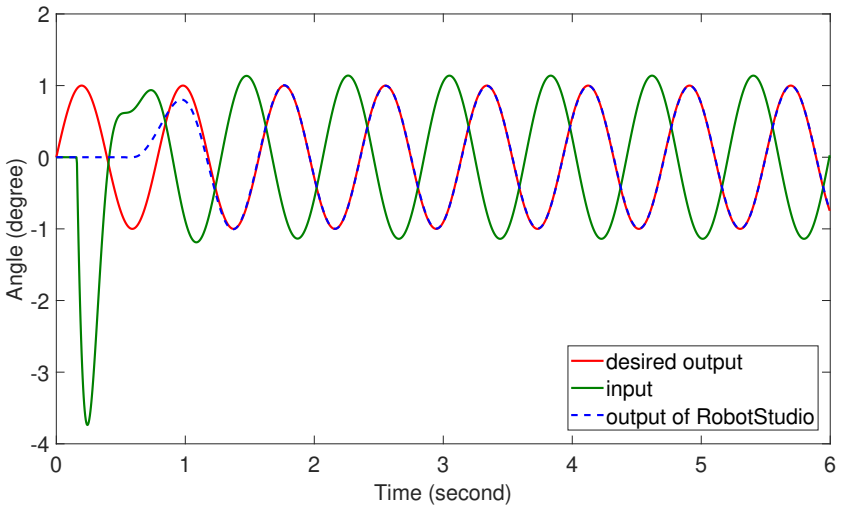


Figure 3.2: Trajectory tracking improvement with the gradient-based ILC after 8 iterations for the fast trajectory in Figure 2.5.

An LTI approximation of the locally linear region of robot inner loop dynamics $G(s)$ (as described in Appendix A) may be used to speed up the iteration. Table 3.1 compares the ℓ_2 norm of tracking error of a single-joint sinusoidal trajectory in RobotStudio, with either using RobotStudio or the identified linear model $G(s)$ to generate the gradient descent directions. As the amplitude and frequency of the desired output remains within the EGM linear regime, both approaches work

similarly well.

Table 3.1: Comparison of the ℓ_2 norm of tracking errors of a single-joint trajectory, by using ILC implemented with RobotStudio and with the fitted linear model $G(s)$ in Appendix A. The initial tracking error is 31.6230°. The linear model offers a reliable approximation of the linear regime of EGM.

Iteration #	Tracking Error Using RobotStudio (°)	Tracking Error Using Linear Model (°)
1	26.0699	25.0585
2	21.0735	20.0049
3	17.1372	15.9724
4	13.6738	12.7540
5	11.2821	10.1849

3.5 Summary

This chapter presents a data-driven gradient-based ILC algorithm that does not require explicit system information given that the system is time-invariant and has a symmetric impulse response matrix. The standard ILC is mostly based on a linear proportional update such as the P-type and PD-type ILC. In contrast, we use a trajectory-based gradient descent algorithm. Explicit model is not needed for the gradient update computation. Instead, we feed the time reversed tracking error profiles directly into the physical system to generate the gradient descent direction. Note that instead of estimating the full gradient operator G_k^*, we directly compute the gradient descent directions $G_k^* \underline{e}_k$, which is more computationally efficient.

This ILC algorithm can be regarded as a process of identifying the local linear system inverse. Thus, the learned optimal input is exclusive for a specific desired trajectory and is not applicable to a different one. In addition, the implementation of ILC on a physical system may be time-consuming considering that it may take several iterations to converge, and in each iteration we need to feed two different input trajectories to the system. Neural networks have attracted lots of attention of control community with their expressive power of generalization capability. Thus, it inspires the idea of learning forward system dynamics using neural networks as G_k in (3.7). On the other hand, a global dynamical inversion feedforward compensator may be identified using neural networks, by combining a large amount of local system inverse dynamics generated from ILC. These ideas of neural learning control are implemented and described in the following chapter.

CHAPTER 4
ROBOT MANIPULATORS MOTION CONTROL THROUGH DYNAMICS COMPENSATION

4.1 Introduction

Fast and precise robot motion is needed in many industrial applications. Most industrial robot motion controllers allow externally joint position/velocity commanded motion profile instead of torque-level control. This outer loop position/velocity control interface achieves better stability with an inner control loop. To demonstrate this, consider a torque-level control for an SISO system:

$$I\ddot{\theta} = \tau, \tag{4.1}$$

where I, θ, τ are the moment of inertia, joint position and input torque, respectively. With zero initial conditions, the transfer function $G_1(s)$ from input torque τ to joint position θ is:

$$G_1(s) = \frac{1}{Is^2}, \tag{4.2}$$

which is unstable. On the other hand, consider an outer loop position control with a proportional–derivative inner control loop for the SISO system:

$$I\ddot{\theta} = -k_p(\theta - \theta_c) - k_d(\dot{\theta} - \dot{\theta}_c), \tag{4.3}$$

where $\theta_c, \dot{\theta}_c$ are commanded joint position and velocity. $k_p > 0, k_d > 0$ are control gains of the inner control loop. Similarly, with zero initial conditions, we have the following transfer function $G_2(s)$ from θ_c to θ:

$$G_2(s) = \frac{k_d s + k_p}{Is^2 + k_d s + k_p}, \tag{4.4}$$

Portions of this chapter previously appeared in: S. Chen and J. T. Wen, "Neural-learning trajectory tracking control of flexible-joint robot manipulators with unknown dynamics," in *IEEE/RSJ IROS*, Nov. 2019, pp. 128-135.

Portions of this chapter previously appeared in: S. Chen and J. T. Wen, "Adaptive neural trajectory tracking control for flexible-joint robots with online learning," in *IEEE ICRA*, May 2020, pp. 2358-2364.

Portions of this chapter previously appeared in: S. Chen and J. T. Wen, "Industrial robot trajectory tracking control using multi-layer neural networks trained by iterative learning control," *Robotics*, vol.10, no.1, Mar. 2021, Art. no. 50.

which is always stable given positive gains k_p, k_d. For multi-axis articulated industrial robots, the inner loop control typically tightly regulates each joint, and the system is almost diagonal as seen from the outer loop with unity gains in low frequency, as demonstrated later in Figure 4.3. However, the trajectory tracking performance is affected by the unknown robot dynamics and joint servo controllers, as well as the time delays in transmitting the external command as a setpoint to the inner control loop. As previously discussed in Chapter 1, traditional robot motion control algorithms are based on the exact knowledge of robot dynamics [10], and a torque level control interface [31], [32]. However, in general it is a difficult task to analytically identify the dynamics of inner control loop, and access to a torque level control of an industrial robot.

This chapter focuses on articulated robot trajectory tracking control through inner loop dynamics compensation with neural networks. Various neural network architectures are explored to approximate the forward or inverse of the robot inner control loop dynamics, including feedforward multi-layer neural networks (MNNs), recurrent neural networks (RNNs), and a customized directed acyclic graph (DAG). Four neural-learning control algorithms are reported by applying these neural networks to classical control techniques including iterative learning control (Sections 4.2 and 4.3) and adaptive control (Section 4.4). Rather than torque level control, these control algorithms are based on a joint position control interface, which makes them more readily applicable to current industrial robots.

4.2 Multi-Layer Neural Networks as a Feedforward Dynamics Compensator

The first neural-learning control algorithm adopts MNNs as an outer loop feedforward controller to compensate for the inner loop dynamics for trajectory tracking control of an articulated industrial robot. The MNNs are used to approximate the non-causal stable inverse for the nonlinear non-minimum-phase robot inner loop dynamics. The gradient-based ILC algorithm described in Chapter 3 is used to generate the training set for the MNNs.

4.2.1 Approach Overview

Our approach, as illustrated in Figure 4.1, combines iterative learning control with multi-layer neural networks. ILC in (3.7) finds the commanded robot motion corresponding to a collection of desired robot joint trajectories. These input/output trajectory pairs are then used to train the inverse dynamics of the inner control loop (with desired robot joint trajectory as the input and com-

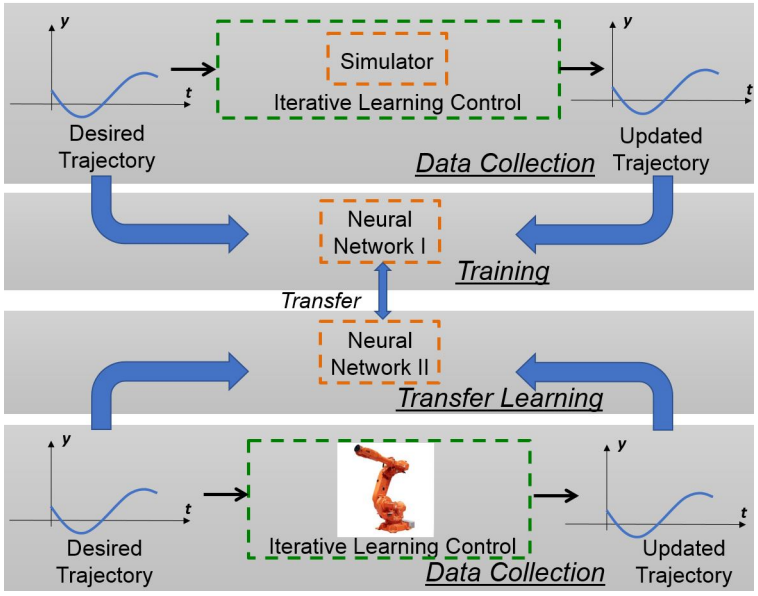


Figure 4.1: Overview of the first neural-learning trajectory tracking control approach.

manded robot motion as output) represented as an MNN (Neural Network I, *NN-I*, in Figure 4.1). As the inner loop may be non-minimum-phase (from time delay or non-collocated input/output map due to joint or structural flexibility), MNN needs to be able to approximate a non-causal system to avoid instability. We use a robot dynamics simulator from the robot vendor (RobotStudio from ABB) to rapidly generate a large set of training trajectories. The simulator is also used to guide the training process and evaluate the performance of the trained MNN. When the NN feedforward control is applied to the physical robot, the tracking performance invariably degrades due to the discrepancy between simulation and reality. To narrow this reality gap, the output layer of *NN-I* is adjusted using a moderate amount of additional data obtained from the physical robot. This process is similar to transfer learning [83]. Using the proposed approach, we have demonstrated improved tracking performance on a physical robot ABB IRB 6640-180 using the NN compensation. We have also applied the trained NNs (*NN-I*) to two other robot models in simulation and both show improved tracking. Compared to state-of-the-art of neural-learning controller design, the contributions of this approach include:

- *Non-Causal MNN for stable nonlinear inversion*: We showed the feasibility of using an MNN to approximate the non-causal stable inverse for the nonlinear non-minimum-phase robot inner loop dynamics.

- *Model-free Iterative Learning*: We applied the model-free gradient-based ILC law described in Chapter 3 to a nearly diagonal robot inner loop dynamics. This ILC is used to generate the training set for the MNN in a high-fidelity dynamic simulator.

- *Transfer Learning*: We narrowed the reality gap by updating the output layer of the simulation-trained MNN using the data from the physical robot.

- *Demonstration*: The improved performance using our approach is shown through simulation and experimental trials with a 6-degree-of-freedom (dof) nonlinear industrial robot.

- *Generalizability*: Different robots from the same vendor exhibit similar inner loop dynamics. We showed that the MNN trained for one robot improves the performance on other robot models from the same vendor as well.

4.2.2 Problem Statement

Consider an n-dof robot arm under closed loop joint servo control with input $u \in \mathbb{R}^n$, typically either the joint velocity command $\dot{q}_c$ or the joint position command q_c, and the output $q \in \mathbb{R}^n$, the measured joint position. The inner loop dynamics, $\mathcal{G}$, is a sampled-data nonlinear dynamical system possibly with communication transport delay and quantization effect. Our goal is to find a feedforward compensator $\mathcal{G}^\dagger$ (as shown in Figure 4.2) such that for a given desired joint trajectory q_d, the feedforward control $u = \mathcal{G}^\dagger q_d$ would result in asymptotic tracking: $q = \mathcal{G}u \to q_d$, as $t \to \infty$.

As $\mathcal{G}$ is assumed unknown, we will use a multi-layer neural network to approximate $\mathcal{G}^\dagger$. The plant $\mathcal{G}$ may be non-minimum-phase (e.g., due to time delay or structural flexibility with non-collocated sensors and actuators), so the causal inverse would be unstable. In this case, a stable inverse exists but is non-causal. This is demonstrated using an SISO non-minimum-phase system in Appendix C. Therefore, the neural network approximation of $\mathcal{G}\dagger$ must be able to represent the possible non-causal property.

Locally near a given configuration, $\mathcal{G}$ is a linear system, that may be represented by an auto-regressive/moving-average (ARMA) model in the sampled-data implementation. As the coefficients in the ARMA model may depend on the robot pose, a global representation of $\mathcal{G}$ is a nonlinear ARMA (NARMA) model. Similarly, $\mathcal{G}\dagger$ may also be represented by a *stable* but possibly non-causal NARMA model. In this case, a multi-layer feedforward network with a window of past and future inputs is a natural candidate to approximate $\mathcal{G}\dagger$. Note that since the desired trajectory q_d is assumed known, future q_d may be used in the non-causal implementation. It is

also possible to directly using a recurrent neural network (RNN) to directly capture the dynamics in $\mathcal{G}\dagger$. A bidirectional recurrent neural network (BRNN) may be needed to capture the non-causal behavior.

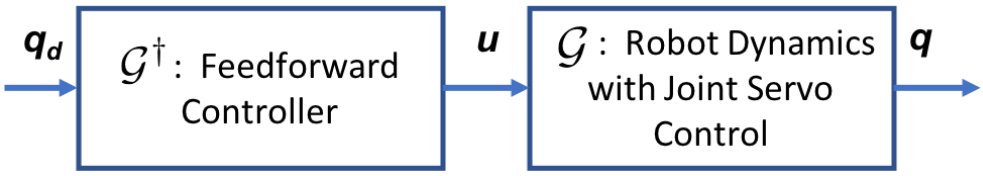


Figure 4.2: Robot dynamics with closed loop joint servo control. The goal of the neural-learning control approach is to find a possibly non-causal feedforward compensator $\mathcal{G}^\dagger$ using MNNs.

We use the ABB IRB6640-180 robot with EGM interface as described in Chapter 2 as a test platform. We assume the robot internal dynamics and delays are deterministic and repeatable, which is reasonable considering the high accuracy and repeatability of the robot. ABB also offers a high fidelity dynamical simulator, RobotStudio [72], with the same EGM feature included. We use this simulator to implement ILC and in turn generate a large amount of training data for $\mathcal{G}^\dagger$ implemented as a multi-layer NN.

4.2.3 Methodology

Though $\mathcal{G}u$ may be implemented in simulation and physical experiments, finding $\mathcal{G}^\dagger$ is challenging because $\mathcal{G}$ is difficult to characterize analytically. Our approach to the approximation of $\mathcal{G}^\dagger$ does not require the explicit knowledge of $\mathcal{G}$ and only uses $\mathcal{G}u$ for a specific input u. The key steps include:

1. *Model-free Gradient Descent-based ILC*: For a specific desired trajectory q_d, we apply the gradient-based ILC algorithm in (3.7) to find the corresponding optimal input trajectory u for each individual joint, which is a single-input/single-output system. Considering $\mathcal{G}$ is locally time invariant, the gradient descent direction can be obtained using the time reversal filtering as shown in Figure 3.1.

2. *Approximation of Dynamical System using a Feedforward Neural Network*: The first step generates multiple (q_d, u) trajectory pairs corresponding to $u = \mathcal{G}^\dagger q_d$. We would like to approximate $\mathcal{G}^\dagger$ by a feedfoward neural network and train the weights using (q_d, u) from the gradient-based ILC. To enable a feedforward net to approximate a possibly non-causal

dynamical system, we use a batch of past and future q_d to generate a batch of u. Furthermore, the q_d batch is shifted with respect to u to remove the effect of transient response.

3. *Improvement of Neural Network based on Physical Experiments*: We perform the first two steps using a dynamical simulator of $\mathcal{G}$ (in our case, RobotStudio). When the trained NN approximation of $\mathcal{G}^\dagger$ is applied to the physical system, tracking error may increase due to the discrepancy between the simulator and the physical robot. Instead of repeating the process all over again using the actual robot, we just retrain the output layer weights of the NN.

The rest of this section describes these steps in greater details.

4.2.3.1 Iterative Learning Control

We implement the gradient-based ILC algorithm described in Chapter 3 to search the gradient descent directions in RobotStudio for a single joint. As a result, no explicit model information is needed in the iterative input update. In addition, robot inner loop dynamics is nearly symmetric. To demonstrate this, we use a series of single frequency sinusoidal signals for frequency response identification of the EGM interface. The magnitude frequency response plot of an LTI approximation of the EGM dynamics is shown in Figure 4.3, which indicates that the EGM transfer matrix is almost diagonal with unity gain within the frequency range of $0 \sim 3$ rad/s. Therefore, the single-axis algorithm may be directly extended to the multi-axis case by using the same ILC procedure applied to all axes at the same time.

4.2.3.2 Feedforward Neural Networks

Despite the significantly improved tracking performance, ILC finds $u = \mathcal{G}^\dagger q_d$ only for a local specific q_d. It is therefore not suitable for run-time implementation. Our aim is to obtain a global approximation of $\mathcal{G}^\dagger$ directly by learning from a set of (u, q_d) obtained from ILC. To this end, we use multi-layer NNs to represent $\mathcal{G}^\dagger$. Because the decoupled nature of $\mathcal{G}$ (command of joint i only affects the motion of joint i), we have n separate NNs, one for each joint.

We use a feedforward net to approximate the inverse dynamics of EGM. At a given time t, the input to the NN is a segment of the desired joint trajectory $\underline{q}_d(t) := \{q_d(\tau) : \tau \in [t-T, t+T]\}$. The output is selected from a segment of u: $\underline{u} := \{u(\tau) : \tau \in [t, t+T]\}$. Updating a segment of the outer loop control instead of a single time sample reduces computation by avoiding frequent invocation of the NN. The choice of the future segment of $\underline{q}_d$ is to allow approximation of non-causal behavior needed in inverting non-minimum phase or strictly proper $\mathcal{G}$.

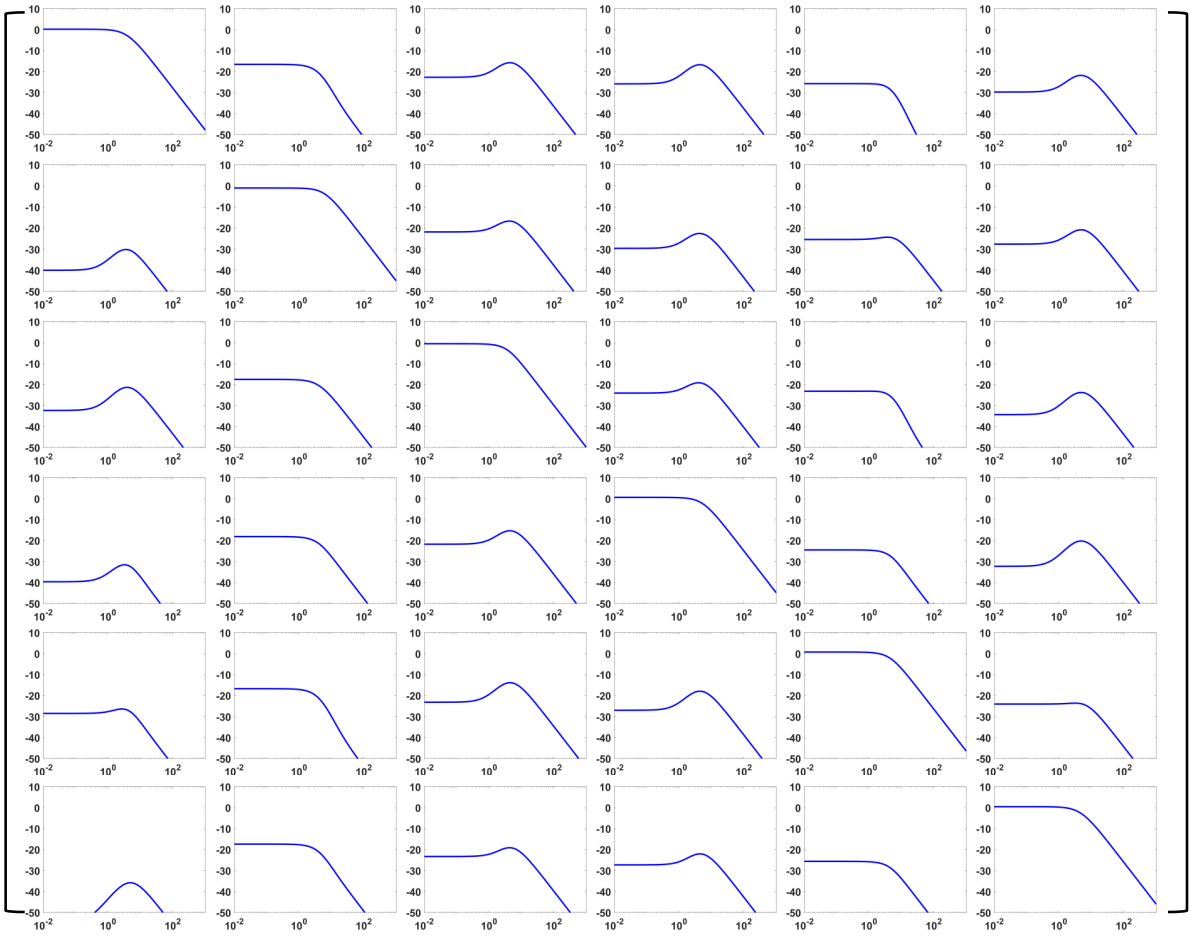


Figure 4.3: Magnitude frequency response plot of a local LTI approximation of EGM transfer matrix. The frequency response is identified using a series of single frequency sinusoidal signals. For all subplots, the horizontal axis is frequency (rad/s), and the vertical axis is magnitude (dB). Locally the EGM dynamics has a nearly diagonal structure.

We have explored various NN architectures, particularly the number of hidden layers and the number of nodes in each layer by experiments. The selection of T (which is proportional to the input dimension of NNs) should guarantee that the input of NNs contains enough information for the NNs to model the dynamics. On the other hand, it should not be too large or there exists a risk of network overfitting. Table 4.1 lists results of final mean squared testing errors with different neural network architectures.

After testing, we choose $T = 0.1$ s, which corresponds to 25 EGM samples with a 4 ms sampling rate. The input and output layers therefore contain 50 and 25 nodes, respectively. We select a fully-connected network including two hidden layers, and each hidden layer contains 100 nodes, as illustrated in Figure 4.4. The optimal inputs from ILC for 400 different desired trajecto-

Table 4.1: Mean squared testing error of different NN architectures. The architecture with two hidden layers and 100 nodes in each layer produces the best testing accuracy.

Number of Hidden Layers	Number of Nodes in Each Layer	Final Testing Error (°) (Average of Three Tests)
2	50/100 100/100	2.623 2.145
2	100/150 100/200	2.474 2.545
3	100/100/100 100/150/100	2.390 2.486
4	100/100/100/50	2.450

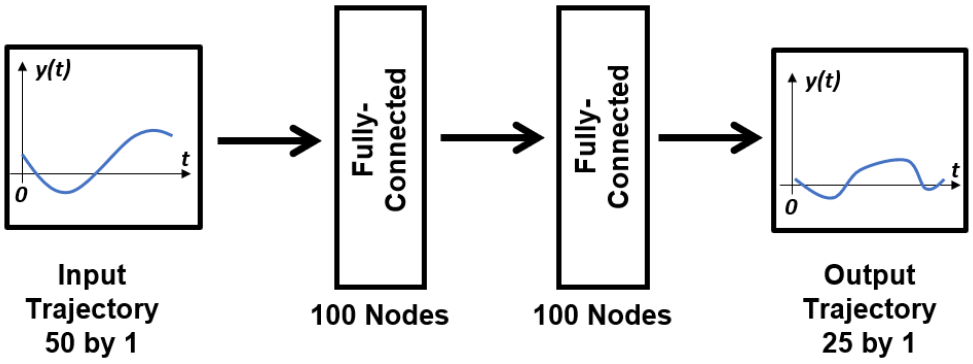
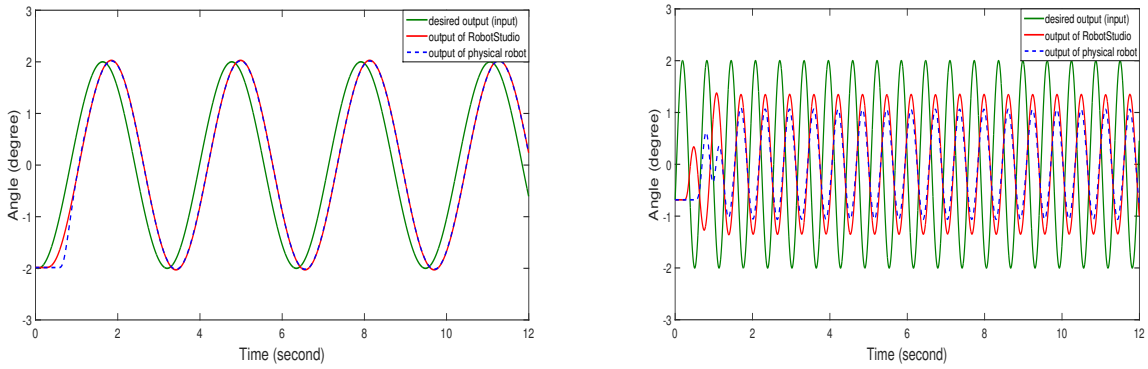


Figure 4.4: The architecture of the employed neural network that has two fully-connected hidden layers and 100 nodes in each hidden layer. The input and output layer contain 50 and 25 nodes, respectively.

ries, including sinusoids, sigmoids and trapezoids are collected in RobotStudio to train the NNs in `TensorFlow` on a NVIDIA GeForce GTX 1050 Ti GPU card, completing in around 45 minutes. The main python file runs on a desktop with an Intel Core i7-7700 CPU with 4 cores and 8 logical processors. We use a per-GPU batch size of 256, and use L_2 norm regularization to avoid overfitting, and the rectified linear unit (ReLU) function to introduce nonlinearity for the NNs. A fixed learning rate of 10^{-4} is employed for the training using `AdamOptimizer`.

4.2.3.3 Transfer Learning

Reality gap exists between RobotStudio and the physical robot due to model inaccuracies. RobotStudio captures the dynamics of the physical robot very well for a trajectory with low velocity, as in Figure 4.5a. However, for trajectories with large velocity and acceleration, there exist discrepancies between RobotStudio and physical robot responses (mainly in the output magnitude), as shown in Figure 4.5b.



(a) **(b)**

Figure 4.5: Comparison of responses between RobotStudio (red curve) and the physical robot (blue curve). Inputs are sinusoids with amplitude of 2° and frequency of 2 rad/sec and 10 rad/sec. (a) RobotStudio captures the dynamics of the physical robot well when commanded a slow trajectory. (b) Discrepancy exists between RobotStudio dynamics and the physical robot when commanded a fast trajectory.

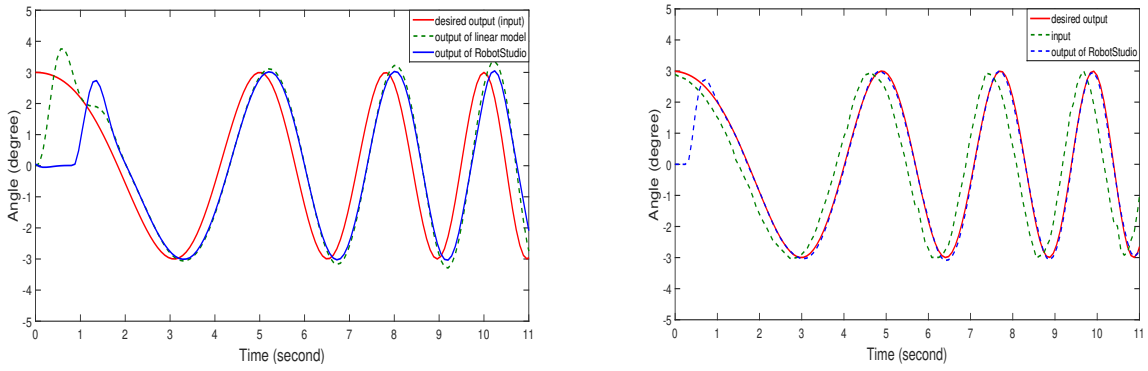
We use additional data from the physical robot to fine-tune the NNs trained from simulation data to narrow the reality gap between the simulator and the physical plant. We apply the ILC procedure on the physical robot for 20 different desired trajectories, and collect the optimal commanded robot motion. These input/output trajectory pairs are then used to fine-tune the output layer weights of the NNs. With the limited discrepancy of the response amplitude (as shown in Figure 4.5b), it may be feasible to only update the output layer using data from the physical robot.

4.2.4 Simulation Results with IRB6640-180

Single Joint Motion Tracking

As multiple sinusoids have been used for the NN training, we first test the generalization of the trained NNs for a chirp signal (multi-sinusoids) which is not part of the training set. Figure 4.6 presents the tracking result for joint 1 of the simulated robot with the trained NN. Figure 4.6a shows the uncompensated output that is lagging behind the desired output. The nonlinear effects are denoted by the deviation of the linear model output from the RobotStudio output. Figure 4.6b shows that, by compensating for the lag effect and amplitude discrepancies, the command input generated by the NN drives the output to the desired output closely after a brief initial transient.

We then apply the trained NNs to tracking a random trajectory generated from a uniform distribution for joint 1 of the simulated robot, as shown in Figure 4.7. The trained NN still works



(a) **(b)**

Figure 4.6: Comparison of tracking performance without and with the NN compensation for a chirp signal in RobotStudio. (a) Uncompensated case: desired output (also the input into the inner loop), RobotStudio output, and linear model output. (b) Compensation with NN: desired output, RobotStudio output, and the input generated by the NN. The NN feedforward controller addresses the issues of lag effect and amplitude discrepancies induced by robot inner loop dynamics.

for compensating for the lag effect and amplitude degradation, though may not as well as for the chirp signal.

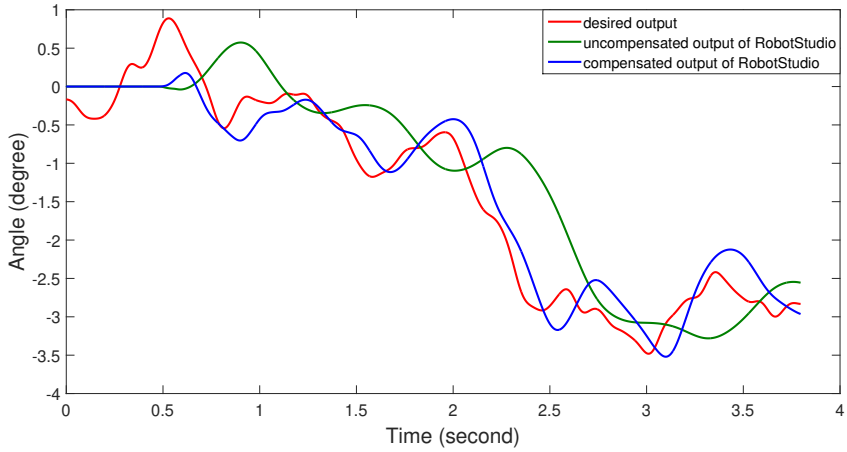
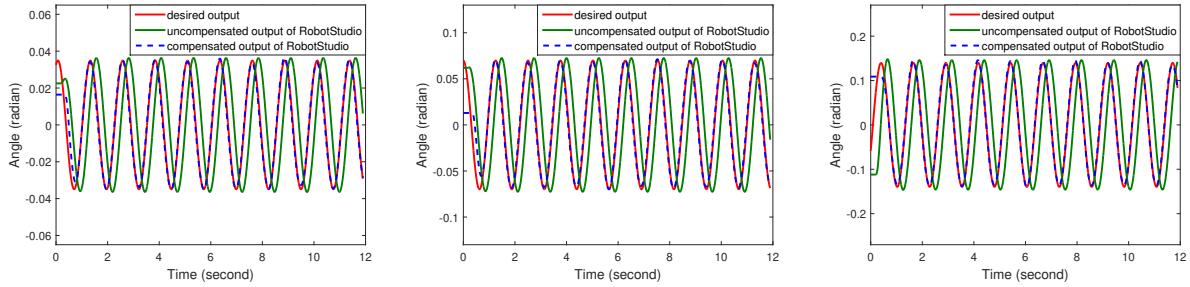


Figure 4.7: The generalization capability of the NN feedforward controller is demonstrated by applying to tracking a pure random trajectory for joint 1 in RobotStudio.

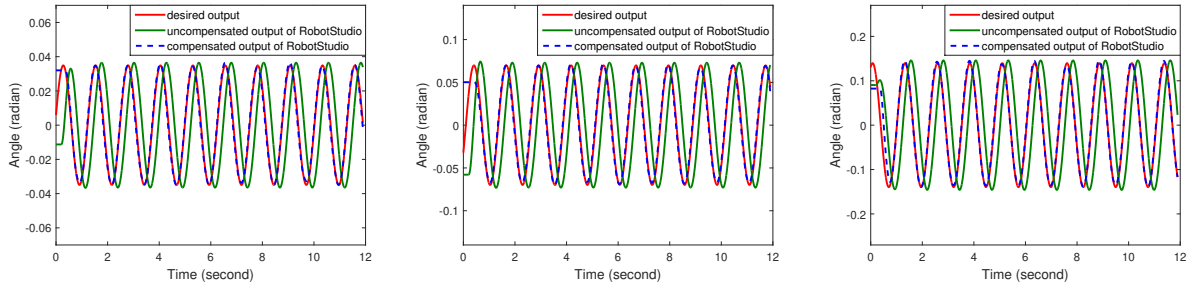
Multi-Joint Motion Tracking

Figure 4.8 shows the comparison of tracking an unseen 6-dimensional sinusoidal joint trajectory without and with NNs compensation. Table 4.2 lists the corresponding tracking errors in terms of ℓ_2 and ℓ_∞ norms (using error vectors at each sampling instant). The ℓ_∞ error is calculated

by ignoring the initial transient.



(a) Joint 1 **(b) Joint 2** **(c) Joint3**

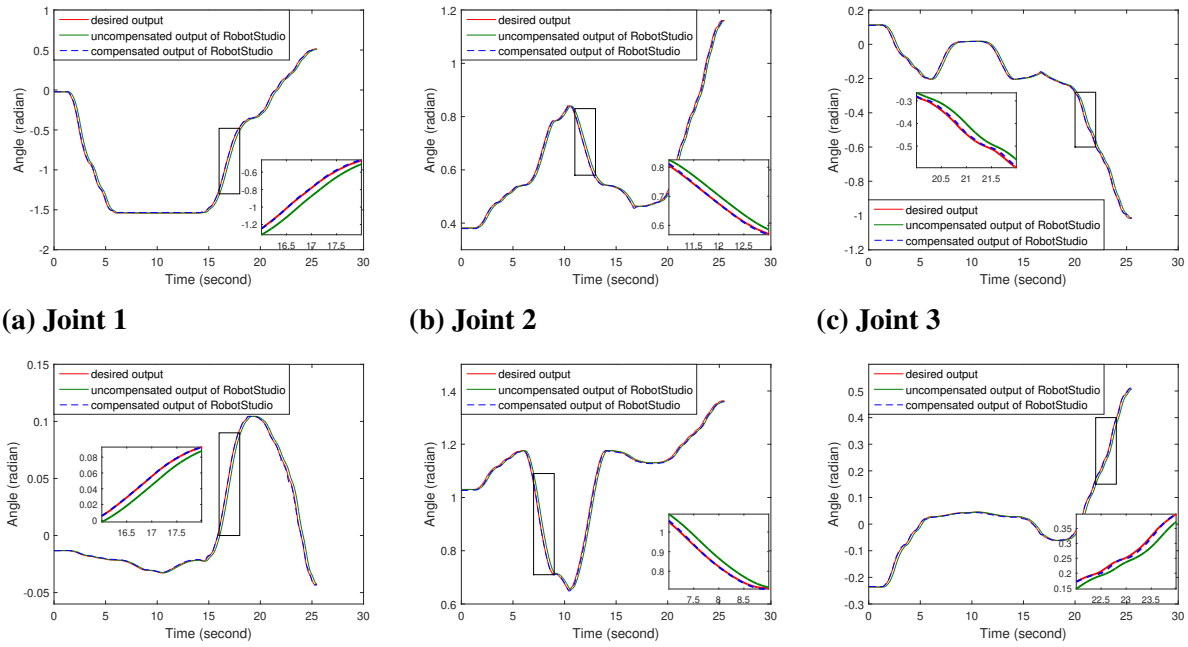


(d) Joint 4 **(e) Joint 5** **(f) Joint 6**

Figure 4.8: The NN feedforward controller improves the trajectory tracking accuracy for a multi-joint sinusoidal trajectory in RobotStudio through robot inner loop dynamics compensation.

Table 4.2: ℓ_2 and ℓ_∞ norm of tracking errors of the multi-joint sinusoidal trajectory without and with NNs compensation in RobotStudio. The trajectory tracking accuracy of all joints is improved with the NN feedforward compensator.

Joint	Original Tracking Error (rad)		Final Tracking Error (rad)	
	ℓ_2	ℓ_∞	ℓ_2	ℓ_∞
1	1.4944	0.0391	0.2056	0.0035
2	2.9935	0.0780	0.4993	0.0065
3	6.1128	0.1563	0.8384	0.0101
4	1.5143	0.0390	0.1421	0.0038
5	3.0542	0.0781	0.4555	0.0065
6	5.9629	0.1561	0.8245	0.0102



(a) Joint 1 **(b) Joint 2** **(c) Joint 3**

(d) Joint 4 **(e) Joint 5** **(f) Joint 6**

Figure 4.9: The NN feedforward controller improves the trajectory tracking accuracy for a multi-joint trajectory planned by MoveIt! in RobotStudio through robot inner loop dynamics compensation. The inset figures clearly demonstrate the improvement of tracking.

To further test the generalization capability of the trained NNs, we conduct another test of tracking a 6-dimensional joint trajectory planned by MoveIt! for a robot-assisted wind turbine blade assembly task [71] in RobotStudio. The comparison of tracking performance without and with NNs compensation is shown in Figure 4.9, with the corresponding tracking errors reported in Table 4.3.

Cartesian Motion Tracking

We also evaluate the performance of the trained NNs for tracking a Cartesian square trajectory (on x-y plane, with z constant of 1.9 m) with bounded velocity and acceleration (maximum velocity of 1 m/s and a maximum acceleration of 0.75 m/s^2). During the motion, the orientation of the robot end effector remains fixed. Figure 4.10 presents that, the trained NNs effectively correct the tracking errors and Figure 4.11 highlights the tracking performance in three Cartesian directions. The corresponding ℓ_2 norm of tracking errors in three Cartesian directions are summarized in Table 4.4.

Table 4.3: ℓ_2 and ℓ_∞ norm of tracking errors of the MoveIt! planned joint trajectories without and with NNs compensation in RobotStudio. The NN feedforward compensator improves the trajectory tracking accuracy for all joints.

Joint	Original Tracking Error (rad)		Final Tracking Error (rad)	
	ℓ_2	ℓ_∞	ℓ_2	ℓ_∞
1	3.5937	0.1497	0.5959	0.0278
2	1.3251	0.0443	0.3992	0.0180
3	1.4463	0.0493	0.2846	0.0112
4	0.3303	0.0127	0.0542	0.0033
5	1.4650	0.0511	0.4773	0.0154
6	0.9629	0.0392	0.1894	0.0078

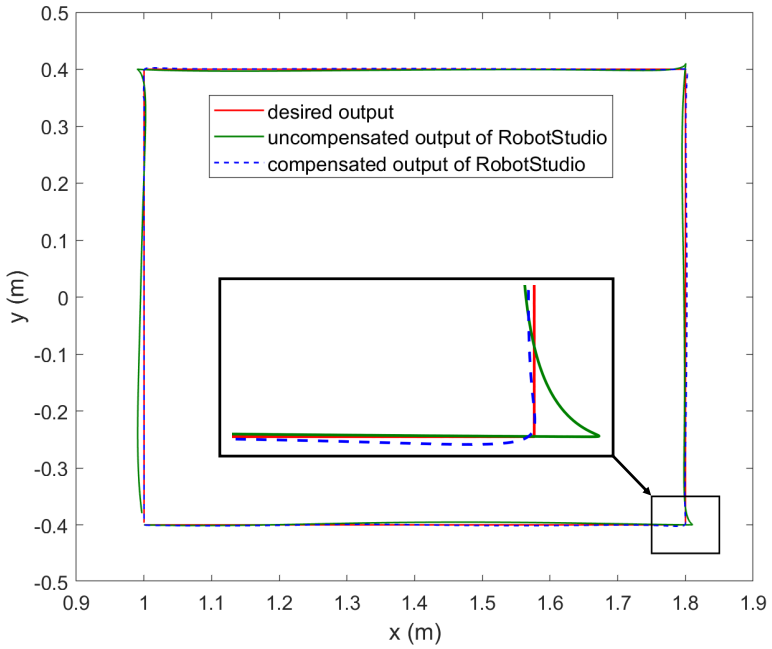


Figure 4.10: The NN feedforward controller improves the trajectory tracking accuracy for a Cartesian square trajectory in x-y plane with z constant in RobotStudio.

Table 4.4: ℓ_2 norm of tracking errors in three Cartesian axes. The NN feedforward controller improves the trajectory tracking accuracy in all Cartesian directions.

Cartesian Axes	Original Tracking Error (m)	Final Tracking Error (m)
x	3.2794	0.7938
y	3.1475	0.2490
z	0.1338	0.0737

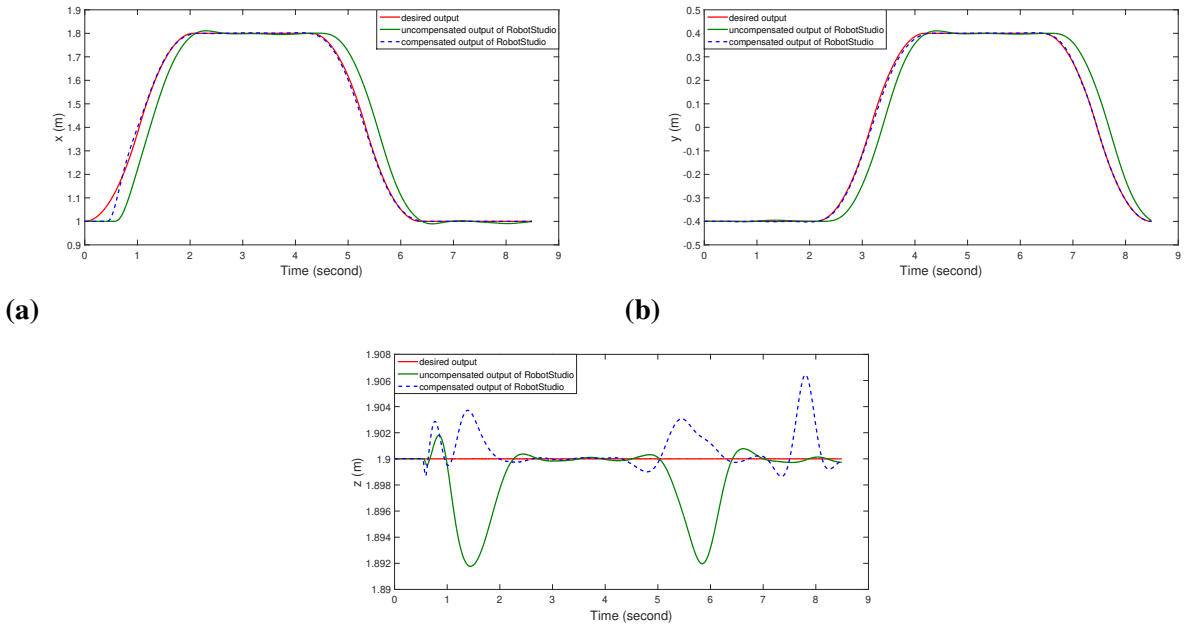


Figure 4.11: Improved tracking performance of Figure 4.10 in individual Cartesian axis. Figures (a)-(c) denote the tracking performance in x, y, z directions, respectively. With the NN compensation, the tracking errors are significantly reduced for all axes, as reported in Table 4.4.

Comparison with MoveL Command

We compare the NN compensation with a built-in ABB controller motion command MoveL, which is used to move the tool center point linearly to a given destination through linear interpolation. Considering the $0.6 \sim 0.7$ second initial dead time of EGM, we conduct the tracking experiments of a straight line with total movement time of 5 seconds, using MoveL, EGM without NN compensation, EGM with NN compensation, as well as running 3 ILC iterations with using the input generated by NN as the initial guess $\underline{u}^{(0)}$ for (3.7). The corresponding RMSE tracking errors are summarized in Table 4.5.

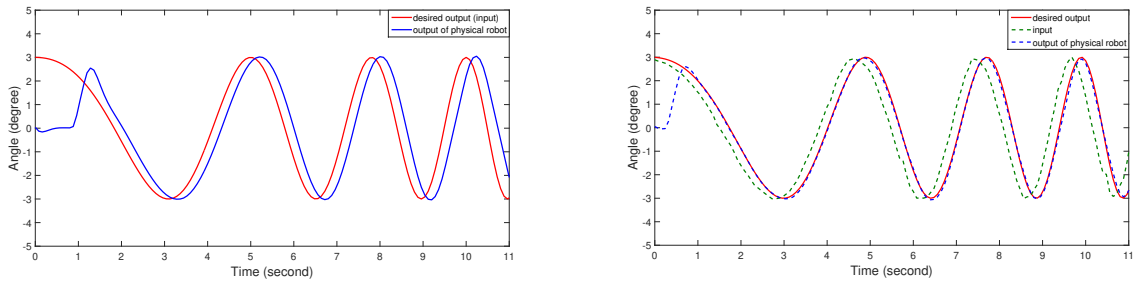
The motion command MoveL is part of the ABB RAPID programming [84] and it achieves accurate tracking, but it is for a fixed path and cannot be modified on the fly. So it is not applicable to real-time collision avoidance or sensor-based motion control. EGM, on the other hand, allows for sensor-guided motion with high sampling rate, but the tracking performance is compromised by the delay and internal dynamics. Table 4.5 indicates that the NN compensation produces a similar level of tracking performance compared with MoveL, and at the same time it retains the advantages of sensor-guided motion of EGM.

Table 4.5: RMSE tracking errors of the straight line using 4 different approaches in RobotStudio. A similar tracking accuracy to MoveL is achieved by running extra 3 ILC iterations using the commanded trajectory produced by the NNs as an initial guess for ILC in (3.7)**.**

RMSE (m)	MoveL	EGM without NN	EGM with NN	Extra 3 ILCs
x	0.0027	0.0200	0.0110	0.0083
y	0.0027	0.0200	0.0104	0.0081
z	0.0027	0.0200	0.0105	0.0081

4.2.5 Experimental Results with Physical IRB6640-180 Robot

We first evaluate the tracking performance of the same chirp signal as we did in simulation for joint 1 of the physical robot, as shown in Figure 4.12. The command input is produced by the NNs updated through transfer learning. Figure 4.12a shows the uncompensated output with the robot output lagging behind the desired output. Figure 4.12b shows that, by compensating for the lag effect, the command input generated by the NNs can drive the output to the desired output closely. We have noticed that for desired trajectories with low velocity profiles, there exists negligible difference between the generated input by NNs trained by simulation data and NNs updated by transfer learning.



(a) **(b)**

Figure 4.12: (a) Uncompensated case: desired output (also the input into the inner loop) and the physical robot output. (b) Compensation with NN: desired output, physical robot output, and the input generated by the NN. With transfer learning, the tracking errors for the same chirp signal as the one in simulation (Figure 4.6) are reduced for joint 1 of the physical robot.

However, for desired trajectories with large velocity and acceleration profiles, transfer learning plays a key role in narrowing the reality gap. Figure 4.13 illustrates the tracking of a sinusoidal signal with angular frequency of 8 rad/s (with different amplitude and phase from training data).

The comparison between Figures 4.13b and 4.13c demonstrates that, transfer learning is necessary to correctly compensate for the phase lagging and magnitude degradation for the physical robot. Table 4.6 summarizes the corresponding ℓ_2 norm of tracking errors.

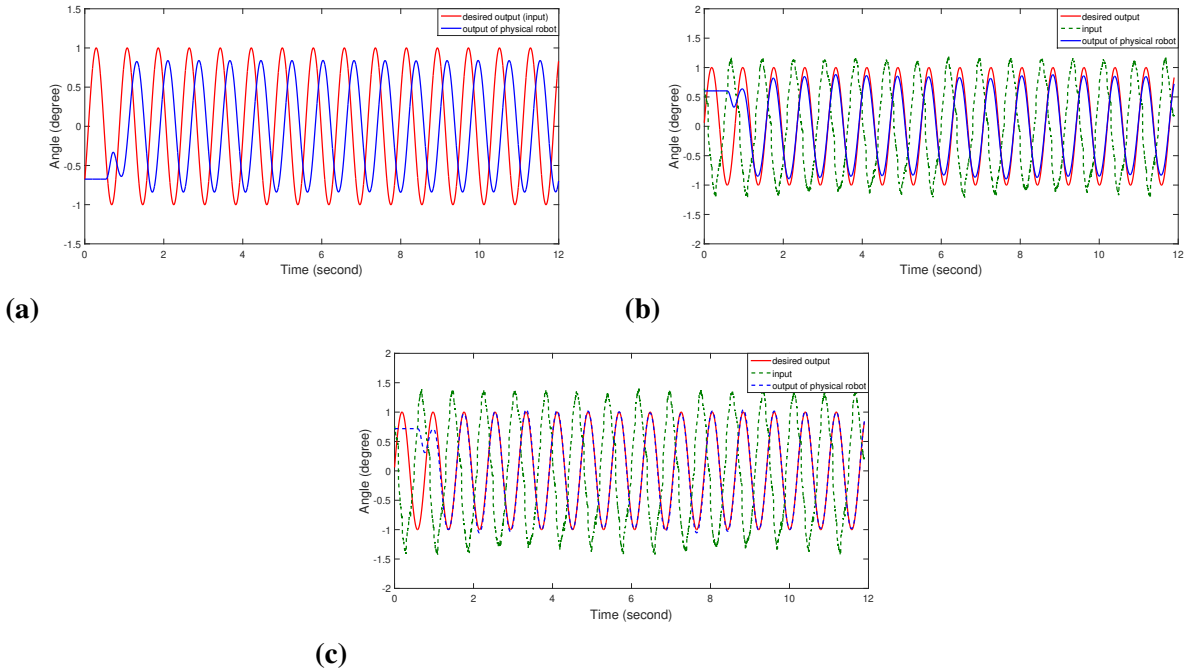


Figure 4.13: (a) Uncompensated case. (b) Compensation using simulation-trained NN. (c) Compensation using the NN updated by transfer learning. Transfer learning plays a key role for tracking a fast sinusoidal trajectory for joint 1 of the physical robot. The ℓ_2 norm of tracking error of each case is shown in Table 4.6.

Table 4.6: ℓ_2 norm of tracking errors in radian of the sinusoidal trajectory without and with transfer learning. The table demonstrates the benefits of transfer learning for tracking a fast trajectory.

Uncompensated Case	without Transfer Learning	with Transfer Learning
1.0242	0.2445	0.2388

4.2.6 Simulation Results with Other Robots

One natural question that arises is whether the trained NNs are able to compensate for the inner loop dynamics for other robot models. We apply the trained NNs to two robot models IRB120 and IRB6640-130 in RobotStudio, and evaluate the tracking for a chirp trajectory and a sinusoidal

trajectory with angular frequency of 8 rad/s and magnitude of 5 degrees, as shown in Figure 4.14. The figure demonstrates that the trained NNs effectively compensate for the lag effect and amplitude degradation for these two robot models, possibly because these robots exhibit similar inner loop dynamics to the IRB6640-180 robot.

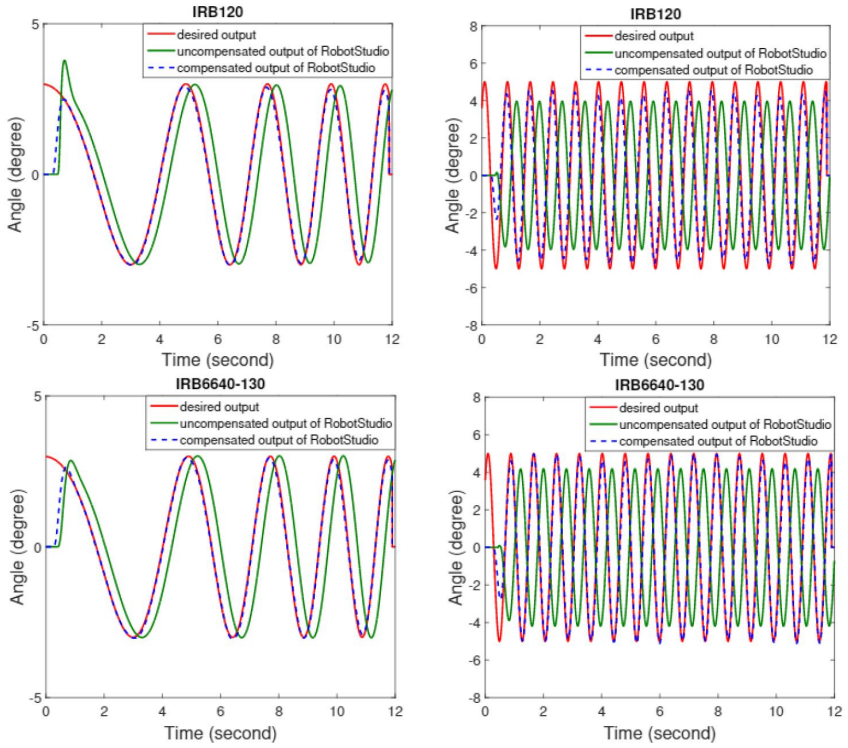


Figure 4.14: IRB120 robot tracking a chirp (top left) and a sinusoidal (top right) signal in RobotStudio. IRB6640-130 robot tracking a chirp (bottom left) and a sinusoidal (bottom right) signal in RobotStudio. The trained NNs using IRB6640-180 also improve the tracking accuracy of a chirp and a sinusoidal joint trajectories for IRB120 and IRB6640-130 in RobotStudio, possibly because these robots have similar inner loop dynamics.

4.2.7 Discussion

This section presents a neural-learning control approach that combines neural networks and iterative learning control to improve the trajectory tracking performance for a multi-axis articulated industrial robot. MNNs are trained through ILC to approximate the non-causal stable inverse of the robot inner loop dynamics. When using the trained MNNs for inference, the non-causality of MNNs requires the knowledge or prediction of desired trajectory in future time steps. A robot dynamics simulator is available from the robot vendor to rapidly implement the ILC and generate

the training data. However, not all physical robots offer a dynamics simulator such as the Baxter robot, which makes it tedious to implement ILC on the physical robot for a large amount of trajectories. The next section focuses on the use of recurrent neural networks to directly capture the causal forward dynamics of the inner loop dynamics for a flexible joint robot, which are adopted to generate the gradient descent directions for ILC.

4.3 Recurrent Neural Networks as a Feedforward Dynamics Compensator

The first neural-learning control technique does not apply to a Baxter robot due to the following issues. First, there is no dynamics simulator available for Baxter to rapidly generate the training data. Then, the joints of the ABB robot are decoupled (as seen from Figure 4.3), thus we train individual NN for each joint. The joints of Baxter, on the other hand, are slightly coupled due to joint flexibility, as shown in Figure 4.15. Therefore, we must use one NN to capture the dynamics of all 7 joints. A unidirectional recurrent neural network (URNN) is used to approximate the causal forward robot inner loop dynamics by training data collected directly from the physical robot. Training the URNN is essentially a simulator-building process and the trained URNN will be used as G_k in (3.7) to speed up the ILC implementation. We also explore the use of a bidirectional RNN (BRNN) [85] to capture the non-causal behavior of the stable inverse dynamics with the same set of training data.

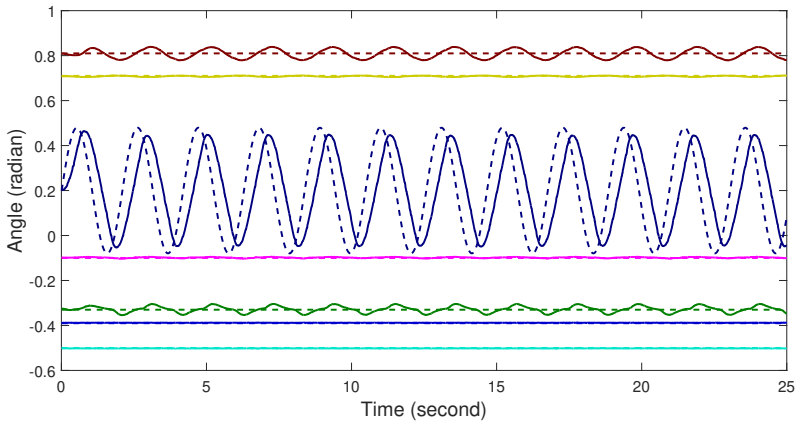


Figure 4.15: Slight coupling of Baxter joints. The dashed and solid curves indicate desired and actual joint angles, respectively. Different colors represent response of different joints. We only command joint motion of joint 4 (the blue sinusoid in the middle of the figure). The figure shows that some joints other than joint 4 have also been activated, although they are not commanded any motion.

4.3.1 Approach Overview

This section considers two neural-learning controllers for the outer loop tracking control for a flexible-joint Baxter robot using recurrent neural networks. RNNs have been widely used to model temporal dependencies in sequential data with their internal recurrent structure [86]. Here, we propose and compare two learning-control approaches based on a URNN and a BRNN, which are adopted as a feedforward controller. Figure 4.16 illustrates the overall control architecture.

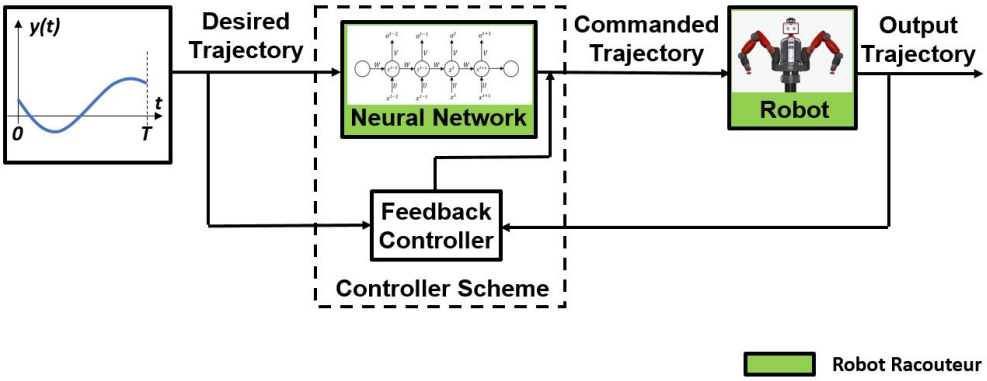


Figure 4.16: RNN-based feedforward learning control. Robot Raconteur [87] is a communication framework that is used to coordinate different components.

The first approach combines a URNN with the gradient-based ILC. We collect the actual robot joint motion by commanding a set of desired robot joint trajectories. These input/output trajectory pairs are then used to train a URNN which represents the causal forward dynamics of the inner control loop (with desired robot joint trajectory as the input and actual robot motion as output). This URNN is treated as a dynamics simulator and is used for the gradient-based ILC implementation to rapidly find the optimal commanded motion for a desired robot joint trajectory. The second approach explores the use of a BRNN to model the non-causal stable inverse of the inner loop dynamics, which may be non-minimum-phase due to joint flexibility. The BRNN is trained using the exact same set of training data as the URNN but flip the input/output pairs. Given an entire desired trajectory, both approaches are able to find the optimal input trajectory. However, for a desired trajectory that is not entirely determined such as an online planned trajectory via user teleoperation, the second dynamical inversion approach based on BRNN is preferred with its capability of real-time inference. The comparison and validation of both approaches are shown through experimental trials with the Baxter robot. To our best knowledge, there are few works exploring and comparing using URNN/BRNN as a feedforward compensator for robot manipulators trajectory tracking control with unknown dynamics.

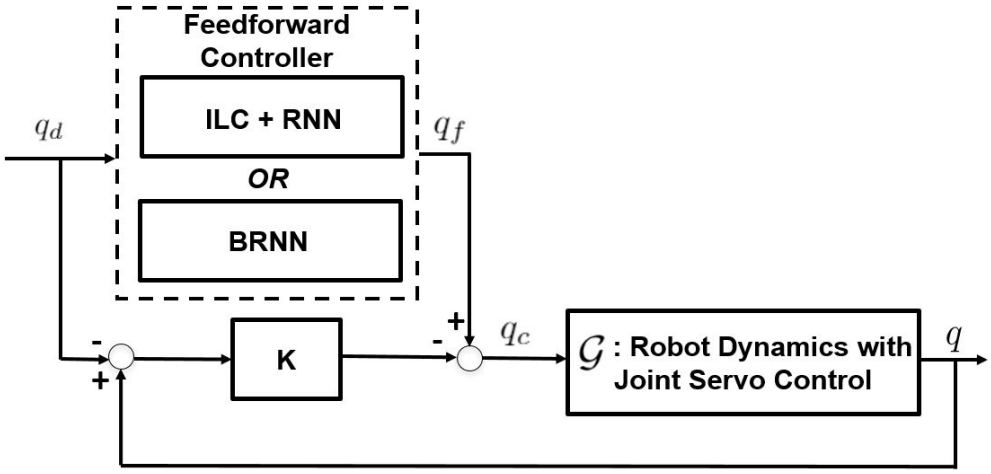


Figure 4.17: Overall control architecture. The feedforward control input q_f is produced by either using ILC implemented with the trained URNN, or directly generated by the trained BRNN. The commanded joint position q_c consists of q_f and a proportional feedback control.

4.3.2 Methodology

4.3.2.1 Proposed Control Law

We propose the following controller which consists of a feedforward and a feedback components as indicated in Figure 4.17.

$$q_c(t+1) = q_f(t+1) - k \cdot (q(t) - q_d(t)), \tag{4.5}$$

where $q_d(t)$ is the desired joint position at time t, and $q(t)$ is the actual joint position. $q_f(t+1)$ is the feedforward control command at time $t+1$, which is generated either by implementing ILC with the trained URNN that models the manipulator forward inner loop dynamics, or produced by the BRNN inference. $q_c(t+1)$ is the resulting commanded joint input for robot manipulator at time $t+1$. On the other hand, the baseline feedback controller is:

$$q_c(t+1) = q_d(t+1) - k \cdot (q(t) - q_d(t)). \tag{4.6}$$

4.3.2.2 Feedforward Controller using a URNN and Gradient-Based ILC

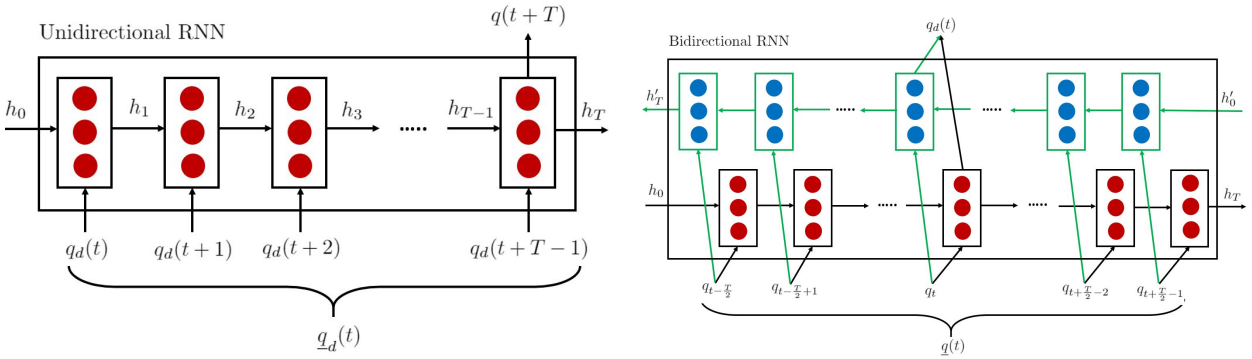
The feedforward control input q_f can be generated by implementing ILC with a URNN, which represents the causal forward dynamics.

Data Collection

We collect the actual robot joint motion of the left arm of Baxter for 500 commanded joint trajectories q_d, among which 100 trajectories are purely random (generated from a uniform distribution) and the remaining 400 are sinusoids with varying magnitudes and frequencies. These desired trajectories respect the joint limits of the Baxter robot, and they are able to cover a feasible portion of Baxter joint workspace. Each trajectory q_d contains 2500 joint setpoints for all 7 joints (thus a 7 by 2500 matrix) and is commanded to Baxter at 100 Hz. The output joint trajectory q of the manipulator is collected at the same sampling rate. This set of training data (q_d, q) are then used to train a URNN.

Unidirectional RNN Training

We train a unidirectional RNN as shown in Figure 4.18a to represent the forward inner loop dynamics of the manipulator, which addresses both the system causality and the coupling of Baxter joints.



(a) **(b)**

Figure 4.18: Unrolled RNNs structures. h_i represents the internal state at each time step. (a) Unrolled causal unidirectional RNN. The URNN takes past inputs and produces a future output. (b) Unrolled non-causal bidirectional RNN. The BRNN takes past and future inputs and produces an output at current time step.

Specifically, during training, the input to the URNN is a segment of the desired joint trajectory q_d containing T time steps setpoints $\underline{q}_d(t) := \{q_d(\tau) : \tau \in [t, t+T-1]\}$. Each $q_d(\tau)$ is a 7 by 1 vector considering all 7 joints of Baxter. The output of the URNN is the actual joint position $q(t+T)$ (a 7 by 1 vector) at time $t+T$ on the output trajectory q. For each commanded input q_d and output q trajectory containing 2500 samples, we can collect $2500-T$ input/output pairs $(\underline{q}_d(t), q(t+T))$ for network training.

We choose $T = 50$ to guarantee that the input to URNN contains rich dynamics information

for the URNN to model. This results in total 1225000 samples, and we use 80 % of the samples for training and 20 % for testing. We train a 4-layer unidirectional RNN with Gated Recurrent Units (GRUs) in `TensorFlow`. The training happens on a NVIDIA GeForce GTX 1050 Ti GPU card. The main python file runs on a desktop with an Intel Core i7-7700 cpu with 4 cores and 8 logical processors. We use a per-GPU batch size of 256, and use dropout [88] of 0.5 to avoid overfitting. An initial learning rate of 10^{-3} is employed for the training using `AdamOptimizer`. The training process converges after 10000 training iterations.

After training, the URNN generates a mapping from a segment of 50 time steps ($T = 50$) desired joint position setpoints (a 7 by 50 matrix) $\underline{q}_d(t)$ to a predicted output $q_o(t+50)$ (a 7 by 1 vector) at one future time step:

$$\underline{q}_d(t) := \{q_d(\tau) : \tau \in [t, t+49]\} \rightarrow q_o(t+50). \tag{4.7}$$

Iterative Learning Control

We implement the gradient-based ILC in (3.7) with the trained URNN as G_k to generate the gradient descent directions. The desired trajectory q_d is used as the initial guess to start the iteration. Although it is much faster to implement ILC with the URNN compared with ILC implementation on the physical system, it still takes several seconds until convergence. Thus, it only applies to an entirely given trajectory and does not work for a real-time compensation scenario such as teleoperation.

4.3.2.3 Feedforward Controller using a BRNN

The feedforward control input q_f can also be generated by filtering q_d through a trained BRNN, which approximates the non-causal manipulator inverse inner loop dynamics. In this case, there are no iterations required and it is suitable for real-time compensation.

Data Collection

Instead of collecting training data through ILC as we did in Section 4.2, we use the same set of training data as for URNN, but swap the input and output pairs. A segment of the output trajectory q is used as input and a segment of the desired trajectory q_d is used as the corresponding output of the BRNN, inspired by [89]. The idea behind this is that, if q_d and q are corresponding input and output trajectories for a dynamical system $\mathcal{G}$, then with input of q, the stable system dynamical inversion would generate q_d.

Bidirectional RNN Training

Here, we train a BRNN as shown in Figure 4.18b to represent the stable inverse of the manipulator inner loop dynamics, which is possibly non-minimum-phase due to joint flexibility. The input to the BRNN during training is a segment of an output joint trajectory q containing T time steps $\underline{q}(t) := \{q(\tau) : \tau \in [t - T/2, t + T/2 - 1]\}$. The output of the BRNN is the commanded joint position $q_d(t)$ at time t. This selection of input/output pair captures the non-causality of the stable inverse dynamics. For each input q_d and output trajectory q containing 2500 joint position samples, we can collect 2501-T training pairs $(\underline{q}(t), q_d(t))$.

Similar to the training of the unidirectional RNN, we choose $T = 50$, which results in 1225500 samples, and we use 80 % for training and 20 % for testing. With a per-GPU batch size of 256, we train a 2-layer BRNN with GRU cells in `TensorFlow` by `AdamOptimizer` with an initial learning rate of 1×10^{-3}.

After training, the BRNN provides a mapping from a sequence of 50 time steps of desired joint position setpoints (a 7 by 50 matrix) $\underline{q}_d(t)$ to a feedforward control input $q_f(t)$ (7 by 1):

$$\underline{q}_d(t) := \{q_d(\tau) : \tau \in [t - 25, t + 24]\} \rightarrow q_f(t). \tag{4.8}$$

4.3.2.4 Resolved Velocity Controller for Human Teleoperation

We demonstrate the real-time dynamics compensation capability of the trained BRNN for a teleoperated trajectory. The developed resolved velocity controller in (2.3) as described in Chapter 2 is adopted for human teleoperation using a joystick. The teleoperated trajectory will be used as input to the BRNN to produce the feedforward control input q_f in a real-time manner.

Robot Raconteur (RR) [87] is a communication framework for robotics and the Internet of Things based on a server/client communication scheme. We use RR to coordinate robot manipulator control and communication, joystick teleoperation, and neural network inference by developing different RR services, as denoted by the green blocks in Figure 4.16. The overall coordination is conducted by a MATLAB script that connects to these services as a client through TCP/IP.

4.3.3 Results

Multi-Joint Sinusoidal Trajectory Tracking

Figure 4.19 compares the tracking of an unseen sinusoidal joint trajectory using the baseline feedback controller in (4.6), as well as both neural-learning feedforward controllers. Table 4.7

reports the corresponding tracking errors in terms of ℓ_2 and ℓ_∞ norms (using the error vectors at each sampling instant) for each joint. The feedforward control input obtained by either of the two neural-learning approaches improves the tracking performance significantly compared with the baseline feedback controller (in average by over 50 % of improvement). Also, the second approach works slightly better than the first one, possibly due to the initial transient response of ILC.

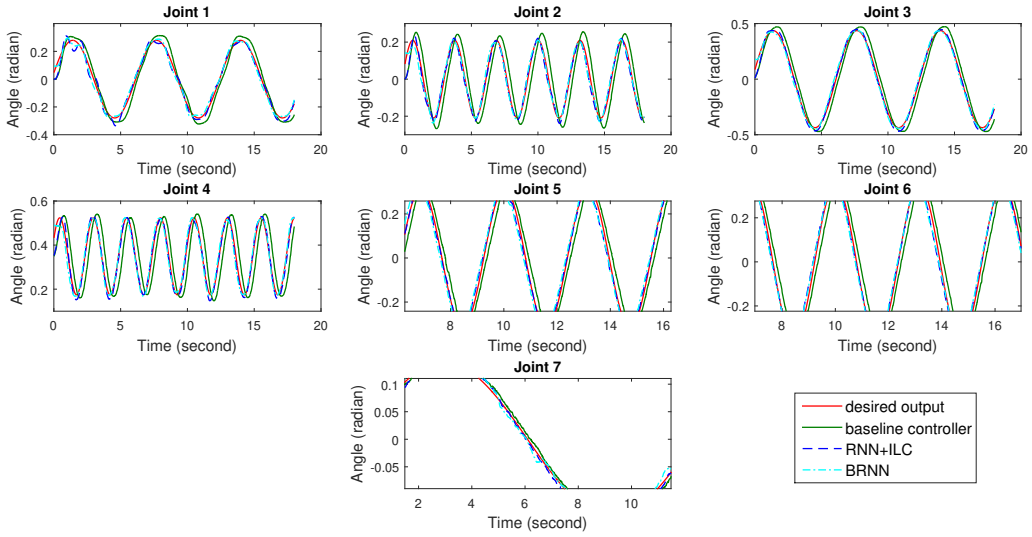


Figure 4.19: Comparison of tracking performance with using the baseline controller, and the RNN feedforward controllers of a multi-joint sinusoidal trajectory. The plots of joint 5 to 7 are zoomed in to better visualize different tracking performance. Both URNN and BRNN feedforward controllers achieve better tracking than the baseline controller.

Table 4.7: Both neural-learning control approaches achieve better tracking of the 7-dimensional sinusoidal joint trajectory than the baseline controller, in terms of ℓ_2 and ℓ_∞ norm of tracking errors.

Joint	Baseline Controller Error (rad)		URNN + ILC + Feedback Error (rad)		BRNN + Feedback Error (rad)	
	ℓ_2	ℓ_∞	ℓ_2	ℓ_∞	ℓ_2	ℓ_∞
1	2.3816	0.0973	1.3622	0.0884	0.8088	0.0590
2	4.0453	0.1296	1.2072	0.1124	0.7163	0.0545
3	3.4910	0.1029	1.2245	0.0942	0.7328	0.0727
4	4.1703	0.1437	0.9874	0.0962	0.5944	0.0391
5	2.5834	0.0917	1.0075	0.0672	0.7247	0.0530
6	2.9346	0.0955	1.1818	0.0808	0.7121	0.0593
7	0.3359	0.0133	0.2845	0.0178	0.2523	0.0155

Random Multi-Joint Trajectory Tracking

We also evaluate the neural-learning controller for tracking an unseen random joint trajectory as shown in Figure 4.20. Table 4.8 summarizes the corresponding tracking errors. The feedforward control input generated by both neural-learning approaches improves the tracking performance obviously compared with the baseline feedback controller (in average by over 40 % of improvement).

Table 4.8: Both neural-learning control approaches achieve better tracking of the 7-dimensional random joint trajectory than the baseline controller, in terms of ℓ_2 and ℓ_∞ norm of tracking errors.

Joint	Baseline Controller Error (rad)		URNN + ILC + Feedback Error (rad)		BRNN + Feedback Error (rad)	
	ℓ_2	ℓ_∞	ℓ_2	ℓ_∞	ℓ_2	ℓ_∞
1	4.1746	0.2515	2.1068	0.1495	2.1974	0.1478
2	3.8348	0.2129	1.9292	0.1131	1.7717	0.1357
3	3.7421	0.2193	2.1502	0.1642	1.9836	0.1458
4	2.9880	0.1614	1.9791	0.1366	1.5006	0.1015
5	1.9861	0.1156	1.7605	0.0974	1.3623	0.1179
6	2.2318	0.167	1.7216	0.1153	1.3887	0.1223
7	1.8733	0.1448	1.3336	0.1104	1.2803	0.0948

Cartesian Trajectory Tracking

Figure 4.21 demonstrates the results of using two neural-learning controllers to track a Carte-

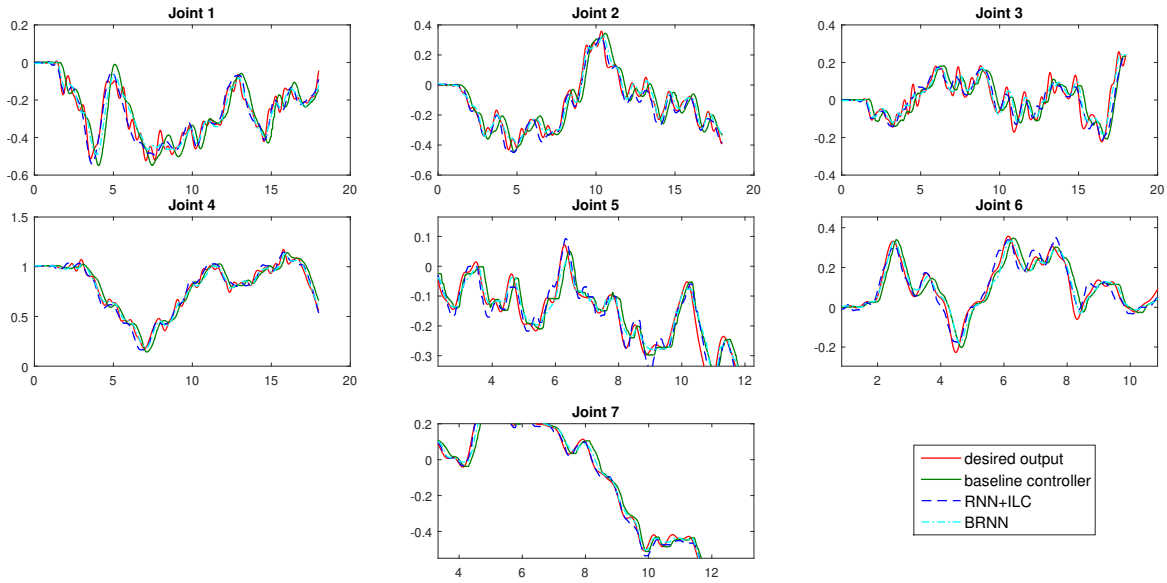


Figure 4.20: Comparison of tracking performance with using the baseline controller and the RNN feedforward controllers of a multi-joint random trajectory. The plots of joint 5 to 7 are zoomed in to show the difference. The horizontal axis is time in second, and the vertical axis is the angle in radian for all plots. Both URNN and BRNN feedforward controllers achieve better tracking than the baseline controller.

sian trajectory, which is a square in the $x-y$ plane with $z = 0.2\ m$. During the motion, the orientation of the end-effector is fixed. Table 4.9 compares the corresponding tracking errors in each Cartesian direction. The figure and the table indicate that the tracking performance is significantly improved with the feedforward control inputs (in average over 50 %), as compared with the baseline feedback controller.

Table 4.9: Both neural-learning control approaches achieve better tracking of the Cartesian trajectory in all three Cartesian axes than the baseline controller, in terms of ℓ_2 and ℓ_∞ norm of tracking errors.

Axis	Baseline Controller Error (m)		URNN + ILC + Feedback Error (m)		BRNN + Feedback Error (m)	
	ℓ_2	ℓ_∞	ℓ_2	ℓ_∞	ℓ_2	ℓ_∞
x	0.4478	0.0363	0.1306	0.0124	0.1223	0.0129
y	0.5694	0.0459	0.1808	0.0199	0.1685	0.0200
z	0.0582	0.0084	0.0455	0.0068	0.0273	0.0023

It is also demonstrated that the tracking performance for all test cases above can be further

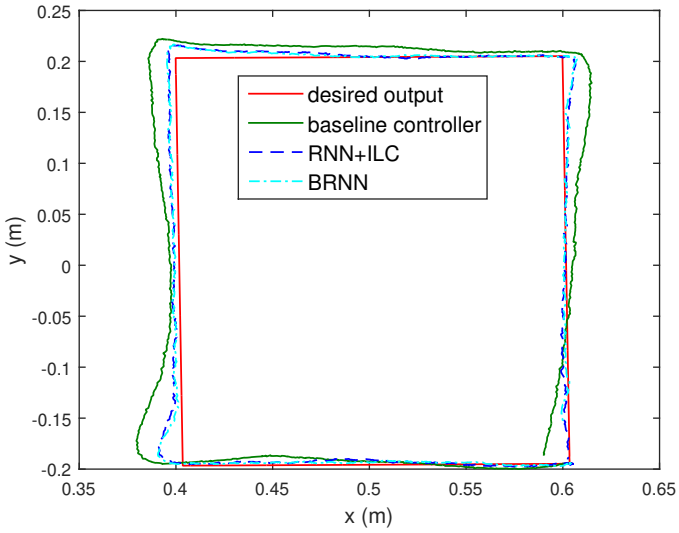


Figure 4.21: Comparison of tracking performance using the baseline controller and the RNN feedforward controllers for a Cartesian square trajectory in $x-y$ plane with constant z. Both URNN and BRNN feedforward controllers achieve better tracking than the baseline controller.

improved by implementing additional ILC iterations on the physical robot, with using the feedforward control trajectory generated by URNN/BRNN as the initial guess in (3.7). One example is illustrated in Figure 4.22. The blue curve denotes the average tracking error over 7 joints of the sinusoidal joint trajectory in Figure 4.19, by using the optimal input u_{opt} generated from running ILC with the URNN. The red curve denotes the error using u_{opt} as the initial guess for (3.7), and implement two extra ILC iterations on Baxter.

Human-Teleoperated Motion Tracking

We may encounter the situation that the desired trajectory is not entirely given and the feedforward compensation must be conducted in a real-time manner. To verify the feasibility of the BRNN for real-time feedforward dynamics compensation, we use an Xbox joystick to teleoperate the left arm of Baxter using the resolved velocity controller in (2.3). In this case, the desired trajectory is randomly planned online. Note that at each time step t, we need to collect future 24 user inputs (as from (4.8)) to collect the required 50 ($T = 50$) input joint setpoints to the BRNN, which causes about 0.24 s delay with the 100 Hz communication rate. The delay can be reduced by using a smaller T. However, if T is too small, BRNN may not capture the inverse dynamics of the manipulator. Figure 4.23 compares the results of tracking one randomly teleoperated joint trajectory using the BRNN feedforward controller and the baseline controller, with corresponding

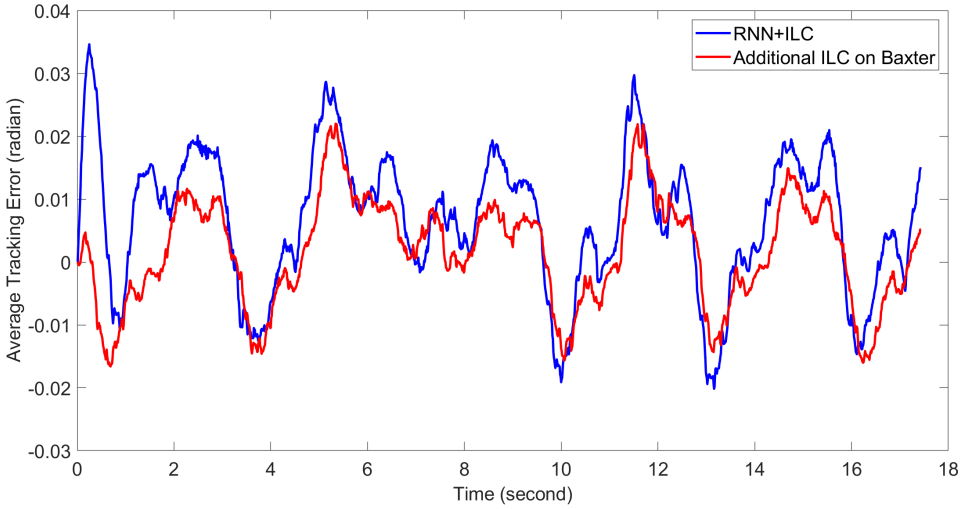


Figure 4.22: Comparison of average tracking errors over all 7 joints of the sinusoidal joint trajectory in Figure 4.19. The blue curved denotes the average tracking error using the ILC+URNN approach. The red curved denotes the average tracking error with implementing 2 more ILC runs on Baxter, using the feedforward input trajectory generated by ILC+URNN approach as the initial guess for (3.7)**. The tracking performance is further improved by running extra ILC iterations.**

tracking errors summarized in Table 4.10. The figure and the table highlight that the feedforward control input generated by the BRNN effectively improves the tracking performance for the randomly generated trajectory compared with the baseline controller (in average by over 30 % of improvement).

4.3.4 Discussion

This section presents two approaches for trajectory tracking control of a flexible-joint robot manipulator with feedforward dynamics compensation using recurrent neural networks. The first approach combines the gradient-based ILC with a trained URNN, which approximates the causal forward dynamics of the robot inner loop. Instead of using an MNN as described in Section 4.2, the second approach explores the use of a BRNN to capture the non-causal behavior of the stable inverse of robot inner loop dynamics. Both approaches find the feedforward control input for an entirely specified desired trajectory, whereas the second neural-learning approach also works for a real-time dynamics compensation scenario. Although the forward dynamics approximation using the URNN may not be perfect, it still provides a feasible initial guess for implementing extra ILC iterations on the physical robot to further improve the tracking. On the other hand, the

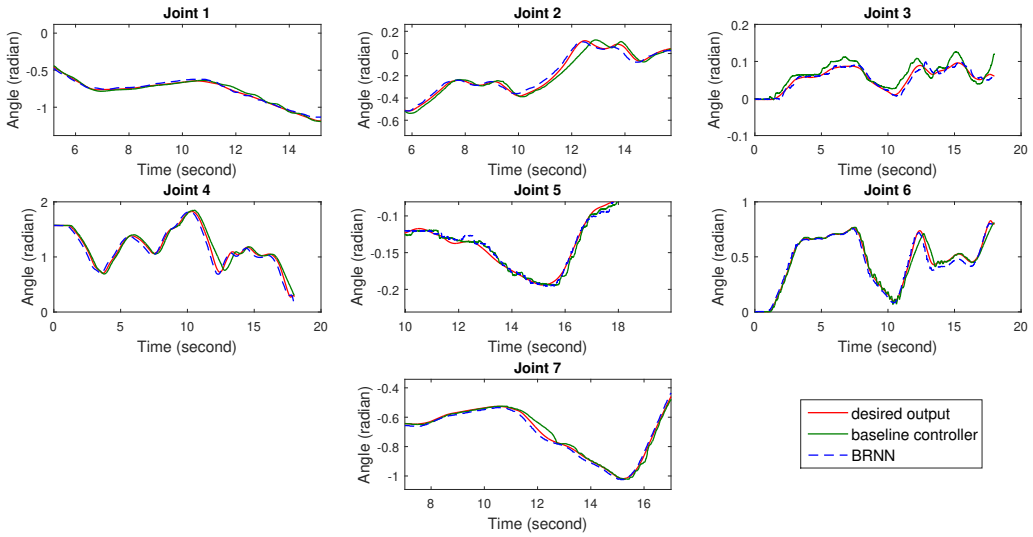


Figure 4.23: Comparison of tracking performance with using the baseline controller and the BRNN feedforward controller of a teleoperated multi-joint trajectory. The plots of joint 1, 2, 5 and 7 are zoomed in to better visualize the difference. The BRNN feedforward controller achieves better tracking than the baseline controller.

BRNN is a more generic network and is especially useful for teleoperation. However, it requires a large amount of data for the network training. Also, the dynamics compensation using RNNs works better for some joints (joints 1 - 4) than others (joints 5 - 7), which is demonstrated from Tables 4.7, 4.8, and 4.10. The difference is possibly caused by the stronger structural flexibility of joints 5-7.

These data-driven approaches described in this section and in Section 4.2 have demonstrated significantly improved tracking performance, but there exist several drawbacks. First of all, the training of the NNs rely on a large amount of data to model the robot dynamics accurately. In addition, these approaches lack a tracking error convergence and stability analysis. In next section, we propose a neural adaptive controller to address these issues.

4.4 Neural Adaptive Control

If the robot dynamics is expressed into a linear-in-parameter form, adaptive controller has been developed to achieve globally asymptotic tracking [10]. However, knowledge of robot dynamics is rarely available, particularly for flexible-joint manipulators. This section presents a neural adaptive controller for the tracking control of a flexible-joint robot under the outer loop

Table 4.10: The BRNN feedforward controller achieves real-time dynamics compensation and produces better tracking of the 7-dimensional user teleoperated joint trajectory than the baseline controller, in terms of ℓ_2 and ℓ_∞ norm of tracking errors.

Joint	Baseline Controller Error (rad)		BRNN + Feedback Error (rad)	
	ℓ_2	ℓ_∞	ℓ_2	ℓ_∞
1	1.4939	0.1826	1.2083	0.0954
2	3.0560	0.3547	1.2055	0.0797
3	0.6279	0.0196	0.3023	0.0599
4	5.4097	0.6381	2.5719	0.1529
5	0.2238	0.0265	0.1946	0.0140
6	1.4105	0.1241	1.3121	0.0932
7	0.7065	0.0592	0.6404	0.0344

joint position control. This controller is based on a neural network, which may be viewed as a linear-in-parameter representation of the robot dynamics.

4.4.1 Approach Overview

Given the structure of the robot dynamics, adaptive control essentially linearizes the system through controller online adaptation to cancel the nonlinear dynamics. Neural network, as a universal function approximator, is a candidate to model the robot dynamics for adaptive controllers development. A popular choice is the radial basis function (RBF) NN [26] since it naturally expresses the approximation linearly in the parameters.

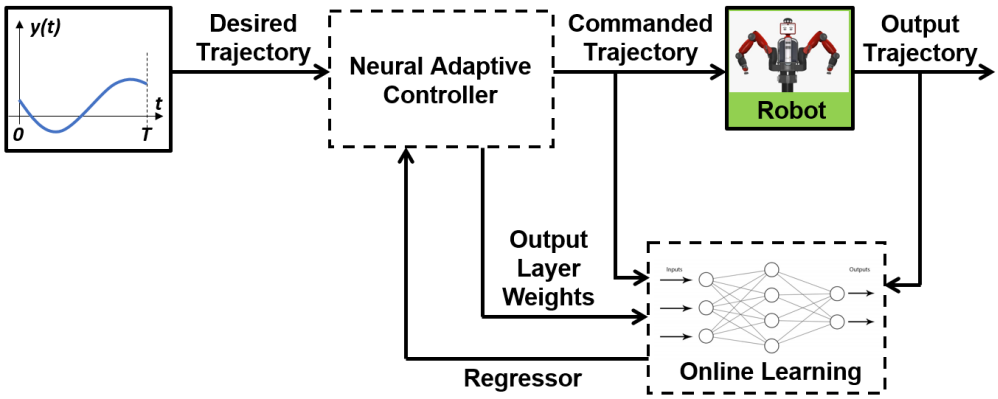


Figure 4.24: Overall neural adaptive control architecture.

Figure 4.24 shows the schematics of the control architecture. We propose a network to represent the linear-in-parameter form of the robot inner loop dynamics. The network consists of a regressor network and an output layer. Thus, the NN parameters are classified into two categories: the internal weights of the regressor, and the output layer weights. The updates of the regressor and output layer weights happen at different time scales. The output layer weights are updated at a fast rate by the adaptation law developed from the stability analysis. On the other hand, the internal weights are updated slowly via backpropagation using online collected robot input/output data. During the backpropagation, the output layer weights remain fixed. The updated regressor is then involved in the output layer weights adaptation. This time scale separation of the regressor and output layer parameters ensures the parameters update decoupling and is like the singular perturbation theory.

4.4.2 Problem Statement

We consider the trajectory tracking control problem for a flexible-joint robot manipulator modeled as [26], [27]:

$$\begin{aligned} M(q)\ddot{q} + C(q,\dot{q})\dot{q} + G(q) + K\cdot(q - q_m) = 0 \\ J_m\ddot{q}_m + K\cdot(q_m - q) = \tau, \end{aligned} \tag{4.9}$$

where q, q_m are the link position and motor position in radian. $M(q), C(q,\dot{q})\dot{q}, G(q)$ denote the manipulator inertia matrix, Coriolis and centrifugal torques, and gravitational torques, respectively. M is symmetric and positive definite and $\dot{M} - 2C$ is skew-symmetric. τ is the motor torque input, and J_m and K are constant matrices representing the motor inertia and joint stiffness matrix, which are typically diagonal and positive definite. To simplify the analysis, we make the following assumptions:

- The motor position q_m is perfectly controlled, i.e., we can ignore the motor dynamics and use q_m as an input to drive q.
- Joint stiffness is the same for all joints, i.e., $K = k_p I$, $k_p > 0$.

Then the robot dynamics can be rewritten as

$$M(q)\ddot{q} + C(q,\dot{q})\dot{q} + H(q) = k_p q_m, \tag{4.10}$$

where $H(q) = G(q) + k_p q$. Based on a neural network, we then express the dynamics into a linear-in-parameter form as follows:

$$Y(\dot{q}_1, q, \dot{q}_2, \ddot{q})a := k_p^{-1}(M(q)\ddot{q} + C(q, \dot{q}_2)\dot{q}_1 + H(q)), \tag{4.11}$$

where Y is the regressor and a is a vector. Our goal is to design an adaptive controller for q_m that achieves globally asymptotically tracking of a desired trajectory q_d with bounded first and second-order time derivatives. The controller is first tested in simulation with a single pendulum, for which the assumption that K is a scalar naturally holds. Then it is applied to a multi-joint Baxter robot and we compare its performance with a baseline proportional-derivative (PD) controller (with gains K_1 and K_2) as below:

$$q_m = q_d - K_1(q - q_d) - K_2(\dot{q} - \dot{q}_d). \tag{4.12}$$

Finally, the neural adaptive controller is implemented with the online regressor backpropagation to demonstrate the feasibility of the proposed regressor online learning scheme.

4.4.3 Methodology

In this section, we propose a network architecture and design an adaptive controller that achieves asymptotically trajectory tracking for the robot manipulators modeled in (4.10).

4.4.3.1 Controller Design

For the controller development, we adopt the following Lyapunov-like function for stability analysis. The design of NN architecture naturally arises during the stability analysis.

$$V = \frac{1}{2}s^T M s + \frac{1}{2}\widetilde{a}^T k_p P^{-1}\widetilde{a}, \tag{4.13}$$

where M and k_p are the robot inertia matrix and joint stiffness in (4.10). s is a sliding vector where $s = \dot{e}(t) + \Lambda e(t)$ with tracking error $e(t) = q(t) - q_d(t)$. $\widetilde{a} = \hat{a} - a$ and $\hat{a}$ is an estimate of the constant vector a. The designed weighting matrix P controls the parameter adaptation rate and is symmetric and positive definite. The Hurwitz matrix $-\Lambda$ makes $e(t), \dot{e}(t)$ converge to zero exponentially as s converges to zero. q_r is defined so that $s = \dot{q} - \dot{q}_r$ (i.e., $\dot{q}_r = \dot{q}_d - \Lambda e$).

By taking the time derivative of V and from (4.10) and (4.11), we have:

$$\begin{aligned}
\dot{V} &= s^T M\dot{s} + \frac{1}{2}s^T \dot{M}s + \dot{\hat{a}}^T k_p P^{-1}\tilde{a} \\
&= s^T (M\ddot{q} - M\ddot{q}_r) + \frac{1}{2}s^T \dot{M}s + \dot{\hat{a}}^T k_p P^{-1}\tilde{a} \\
&= s^T (k_p q_m - H - C\dot{q} - M\ddot{q}_r) + \frac{1}{2}s^T \dot{M}s + \dot{\hat{a}}^T k_p P^{-1}\tilde{a} \\
&= s^T (k_p q_m - H - C\dot{q}_r - M\ddot{q}_r) + \frac{1}{2}s^T (\dot{M} - 2C)s + \dot{\hat{a}}^T k_p P^{-1}\tilde{a} \\
&= s^T (k_p q_m - H - C\dot{q}_r - M\ddot{q}_r) + \dot{\hat{a}}^T k_p P^{-1}\tilde{a} \\
&= s^T k_p (q_m - Y(\dot{q}_r, q, \dot{q}, \ddot{q}_r)a) + \dot{\hat{a}}^T k_p P^{-1}\tilde{a}.
\end{aligned} \tag{4.14}$$

If Y is known, then we may design q_m to cancel Ya using the estimate $\hat{a}$ as

$$q_m = Y(\dot{q}_r, q, \dot{q}, \ddot{q}_r)\hat{a} - K_s s - k\mathbf{sgn}(s). \tag{4.15}$$

The $K_s s$ term with $K_s > 0$ ensures the tracking error convergence. The additional high gain feedback term $k\mathbf{sgn}(s)$ provides robustness with respect to the network modeling errors and noises. It follows

$$\dot{V} = -s^T k_p K_s s - k k_p \|s\|_1 + k_p s^T Y\tilde{a} + k_p \dot{\hat{a}}^T P^{-1}\tilde{a}, \tag{4.16}$$

where $\|\cdot\|_1$ denotes the vector 1-norm. By choosing the adaptation law of $\hat{a}$ as

$$\dot{\hat{a}} = -PY^T(\dot{q}_r, q, \dot{q}, \ddot{q}_r)s, \tag{4.17}$$

we have

$$\dot{V} = -s^T k_p K_s s - k k_p \|s\|_1 . \tag{4.18}$$

This implies $s \to 0$ asymptotically which in turn implies $e \to 0$ exponentially. In the next section, we design a neural network architecture to approximate $Y(\dot{q}_r, q, \dot{q}, \ddot{q}_r)a$.

4.4.3.2 Neural Network Design

Network Architecture

We design a network as a directed acyclic graph that consists of different layers, as illustrated in Figure 4.25. Table 4.11 summarizes the connections between neighboring layers. For one arm of the Baxter robot (with 7 joints), the input to the network is a 28×1 vector $(\dot{q}_1, q, \dot{q}_2, \ddot{q})$, and

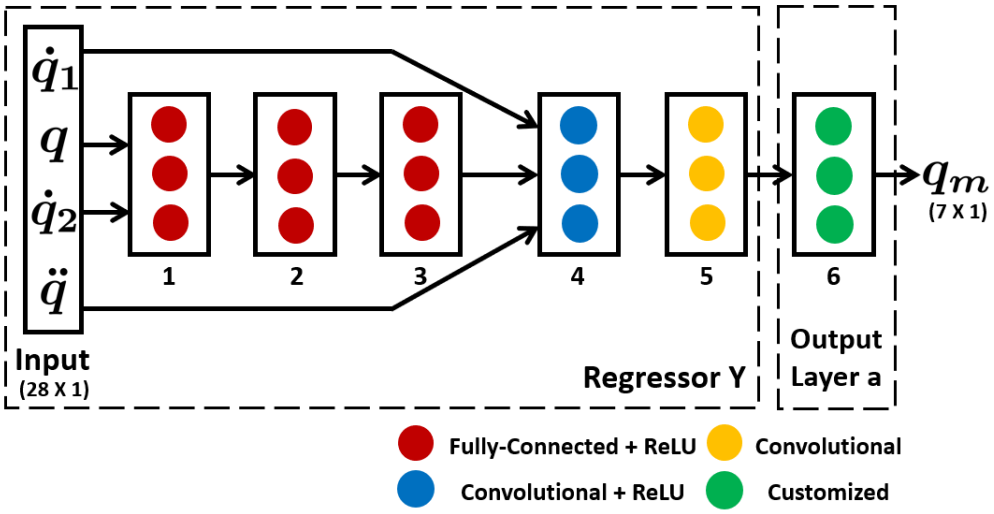


Figure 4.25: Network architecture that expresses robot dynamics as a linear-in-parameter form for the neural adaptive controller development.

Table 4.11: Connection between neighboring network layers.

Layer Number	Learnable Size	Output Size
1	Weights: 100×14 Bias: 100×1	100×1
2	Weights: 100×100 Bias: 100×1	100×1
3	Weights: 7×100 Bias: 7×1	7×1
4	Weights: 1×1×3×100 Bias: 1×1×100	7×1×100
5	Weights: 1×1×100×200 Bias: 1×1×200	7×1×200
6	Weights: 200×1	7×1

the output is a 7×1 vector q_m. The regressor network $Y(\dot{q}_1, q, \dot{q}_2, \ddot{q})$ has its own internal weights from the fully-connected layers and convolutional layers, and the output layer a is a vector with dimension compatible with the output of Y. Note that we need one convolutional layer (the yellow block in Figure 4.25) instead of a fully-connected layer as the final layer of Y, as the size of output of Y is 2-dimensional (7 by N) for a 7-joint robot, whereas a fully-connected layer only provides a 1-dimensional output. In contrast, for an SISO system such as a single pendulum, the last layer of Y is a fully-connected layer.

Training Data Collection

We collect the output trajectory from the left arm of the Baxter robot by commanding 30 sinusoidal trajectories q_m (in comparison, for the RNNs approach described in Section 4.3 we have commanded 500 trajectories) with different magnitudes and frequencies. These commanded trajectories respect the Baxter joint limits and also cover a feasible portion of its joint workspace. Each commanded trajectory contains 2500 joint position setpoints for all 7 joints (thus a 7 by 2500 matrix) and is sent to Baxter at 100 Hz. The output joint position trajectory q as well as the joint velocity trajectory $\dot{q}$ are collected at the same rate. The joint acceleration trajectory $\ddot{q}$ is generated by cubic spline interpolation of the joint velocity trajectory.

Network Offline Training for Initialization

The input to the network is $(\dot{q}(t), q(t), \dot{q}(t), \ddot{q}(t))$, and the output of the network is the commanded position $q_m(t)$. We use 80 % of the collected data for network training and 20 % for testing. As we did before, the training is conducted on a NVIDIA GeForce GTX 1050 Ti GPU card, and the main python file runs on a desktop with an Intel Core i7-7700 CPU with 4 cores and 8 logical processors. We use a per-GPU batch size of 256, and use L_2 norm regularization to avoid overfitting, and the rectified linear unit function to introduce nonlinearity for the NNs. A fixed learning rate of 10^{-3} is employed for the training using `AdamOptimizer`. The training process converges after 5 epochs. After the initial network training, we use the output of the neural network with input of $(\dot{q}_r, q, \dot{q}, \ddot{q}_r)$ as $Y\hat{a}$ to implement the controller in (4.15), and the output layer weights are updated using (4.17). On a slower time scale, the regressor Y is updated using online backpropagation with L_2 norm regularization. Note that the network provides an estimate of the complete Y matrix, so we may use Y^T directly in the update law (4.17).

4.4.3.3 Online Backpropagation for Regressor Update

When the system is subject to dynamics variation (e.g., attaching a load and friction change) and external disturbances, pure adaptation of output layer weights by (4.17) may take a long time to make the tracking error converge. Here, we explore the online learning of regressor Y with the collected input/output data of the system to speed up the error convergence. Unlike the output layer weights, the regressor weights are updated at a slow rate. When updating the regressor, we fetch and freeze the current value of the output layer weights and only update the regressor weights through backpropagation. Then the updated regressor is utilized in (4.15) and (4.17). An example of using this time scale separation for the regressor online learning is illustrated in Figure 4.26.

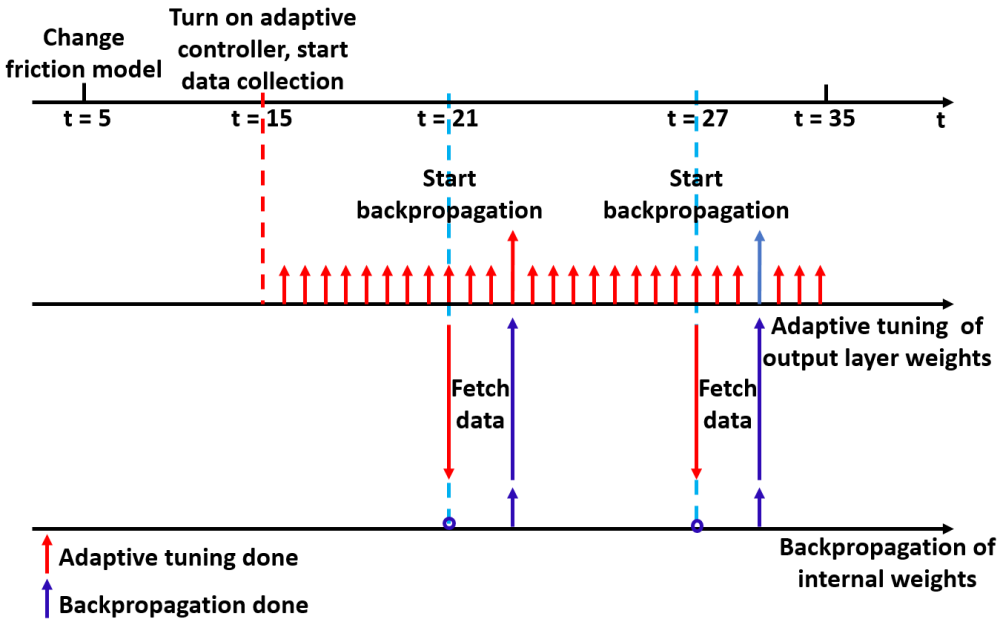


Figure 4.26: Schematics of the regressor online learning. The update of the regressor weights is at a slower rate compared with the output layer weights adaptation.

4.4.4 Simulation Results

To demonstrate the feasibility of the proposed controller with the regressor online learning scheme, we first simulate the scenario that the internal friction model changes from viscous and Coulomb friction to Stribeck friction for a single pendulum modeled in Simulink. The system dynamics is based on (4.9) and the assumption that K is a scalar naturally holds for this SISO system.

We collect 6000 training sets $(q_m, q, \dot{q}, \ddot{q})$ from the single pendulum in Simulink by commanding a chirp multisine signal to the system. Then, a network with architecture denoted in Figure 4.25 is trained using MATLAB Deep Learning Toolbox with the following variations: first, the input size of the network is 4 by 1 and the output size is 1 by 1; second, the last layer of the regressor is a fully-connected layer instead of a convolutional layer, considering that the single pendulum is an SISO system. We apply the trained network to two identical single pendulums, and compare their tracking performance of a sinusoidal trajectory. We implement the regressor online learning for one system (system I), and as a comparison in system II, no online learning is conducted. The simulation procedure is described as below:

- From $t = 0 \sim 5$ s, for both systems I and II, we implement the controller in (4.15) with the trained network but without the adaptation of $\hat{a}$ in (4.17). The value of $\hat{a}$ remains fixed and

is from the initial network training.

- At $t = 5$ s, for both systems we change the friction model and implement the same controller in (4.15) as above.
- At $t = 15$ s, we conduct the output layer adaptation in (4.17) for both systems. Also, we start collecting the input/output data of system I with the updated friction model.
- At $t = 21$ s, we start conducting the regressor online learning for system I using the collected input/output data every 5 seconds until end of the simulation. The first, second and third online updates complete at around $t = 22$ s, $t = 27$ s, and $t = 32$ s, respectively. Note that the controller in (4.15) keeps activated during the entire simulation for both systems.

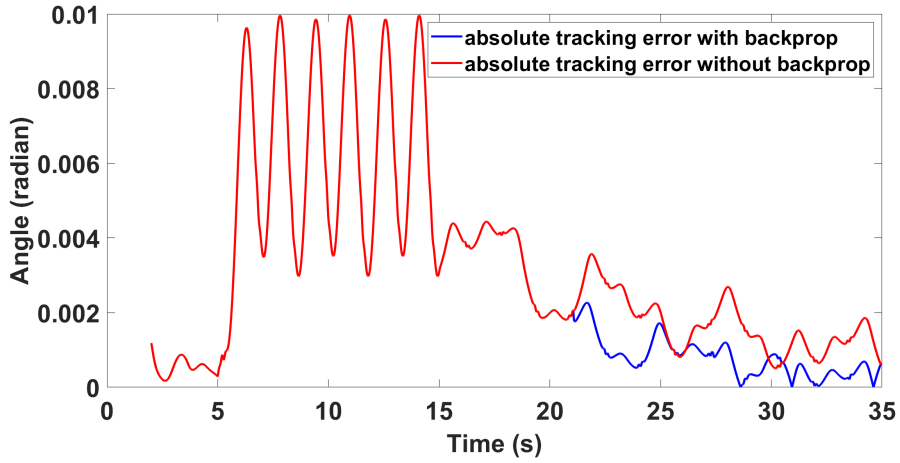


Figure 4.27: Comparison of absolute tracking errors of a sinusoidal trajectory for the single pendulum with and without online backpropagation of the regressor. The large errors from $t = 0 \sim 2$ s (the initial tracking error is 0.25 radian at $t = 0$ and the mean error is 0.0461 radian from $t = 0 \sim 2$ s) are not plotted to better visualize the later small errors. The regressor online learning speeds up the tracking error convergence.

Figure 4.26 illustrates the parameter update timeline of system I in the simulation. The comparison of absolute tracking error in radian of two systems over the entire process is presented in Figure 4.27. The large tracking errors during $t = 0 \sim 2$ s are ignored to better visualize the later smaller tracking errors.

The figure clearly demonstrates the following stages in the simulation:

- The tracking errors converge for both systems from $t = 0 \sim 5$ s, which means our trained network approximates the single pendulum dynamics accurately.

- From $t = 5 \sim 15$ s, the tracking errors of both systems increase due to the change of the friction model and the inability of the network to approximate the updated system dynamics.
- Once we implement the output layer weights adaptation at $t = 15$ s, the tracking errors decrease of both systems. Note that for system I, the tracking error is further reduced compared with system II with the complete of regressor update via online backpropagation.

4.4.5 Experimental Results with a Physical Baxter Robot

We then apply the neural adaptive controller to a Baxter robot for trajectory tracking control. We first evaluate the feasibility of our assumptions in Section 4.4.2. In this experiment, the Baxter is commanded to track an unseen multi-joint sinusoidal trajectory and the tracking performance of the neural adaptive controller is compared with the baseline PD controller in (4.12). Note that there is no regressor online update implemented for the first experiment. The second experiment is a comparative study, in which we compare the performance of the adaptive controller with and without the regressor online learning by tracking a multi-joint sinusoidal trajectory with a payload attached to the robot gripper.

4.4.5.1 Experiment I: Feasibility Study

Figure 4.28 compares the tracking of an unseen sinusoidal joint position trajectory using the baseline PD controller in (4.12) and the adaptive controller in (4.15) and (4.17). The controller parameters are chosen as

$$
\begin{aligned}
&(K_1, K_2) = (0.2, 0.1),\ k = 5,\ P = 0.05 \\
&\Lambda = \text{diag}(3, 6, 3, 20, 10, 10, 10) \\
&K_s = \text{diag}(0.1, 0.01, 0.01, 0.1, 0.01, 0.01, 0.01).
\end{aligned}
$$

Table 4.12 reports the corresponding tracking errors in terms of ℓ_2 and ℓ_∞ norms (using the error vectors at each sampling instant) for all 7 joints. The neural adaptive controller outperforms the PD controller (in average by over 60 % of improvement for all 7 joints), which demonstrates the feasibility of our assumptions in Section 4.4.2.

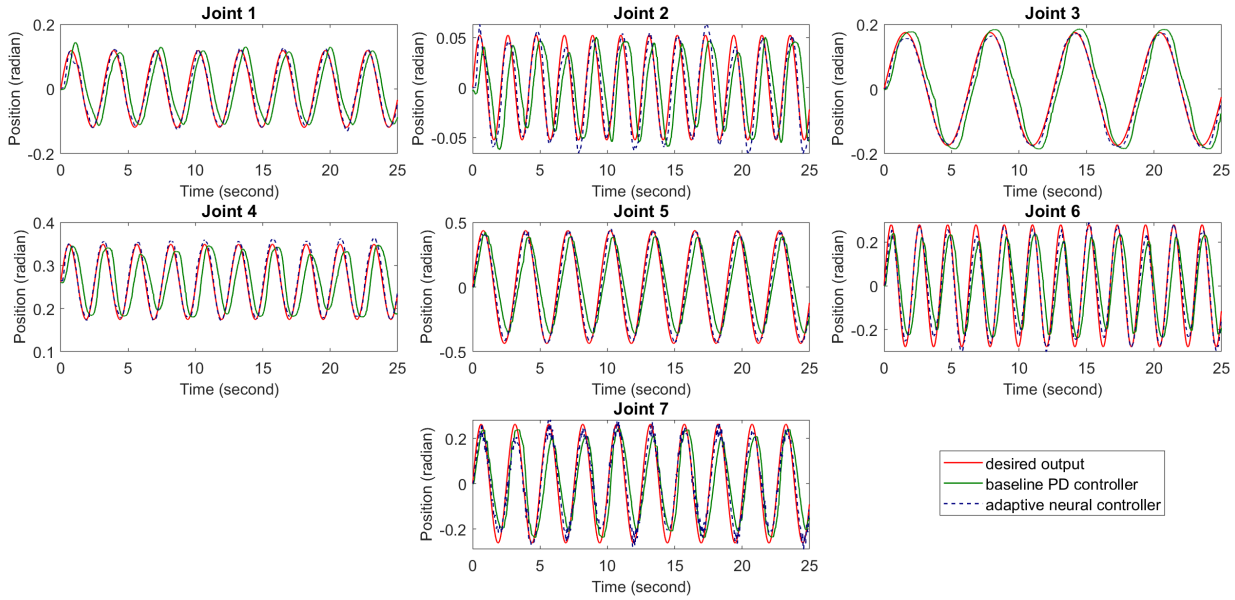


Figure 4.28: Comparison of tracking performance with the baseline PD controller and the neural adaptive controller for a multi-joint sinusoidal trajectory. The neural adaptive controller presents a better tracking than the baseline controller, which demonstrates the feasibility of the assumptions in Section 4.4.2.

4.4.5.2 Experiment II: Comparative Study

We then conduct a comparative study to demonstrate the effectiveness of the regressor online learning scheme by attaching a 0.6 kg payload to the robot gripper (the maximum payload of the gripper is around 5 pounds). We compare the tracking performance of two systems with (system I) and without (system II) the regressor online learning with the following procedure:

- From $t = 0 \sim 25$ s, for both systems I and II, we implement the controller in (4.15) but without the adaptation of $\hat{a}$ in (4.17). The value of $\hat{a}$ remains fixed and is from the initial network training.

- At $t = 25$ s, for both systems we attach the load to the robot gripper and implement the same controller as above.

- At $t = 50$ s, we implement the output layer weights adaptation in (4.17) for both systems, and we also start collecting the input/output data of system I.

- For system I, we implement two regressor online updates using the collected data every 25 seconds and the online updates complete at $t = 75$ s and $t = 100$ s, respectively.

Table 4.12: In the feasibility study, the neural adaptive controller presents a better tracking of the sinusoidal multi-joint trajectory in Figure 4.28 than the baseline controller, in terms of ℓ_2 and ℓ_∞ norm of tracking errors.

Joint	Baseline Controller Tracking Error (rad)		Neural Adaptive Controller Tracking Error (rad)	
	ℓ_2	ℓ_∞	ℓ_2	ℓ_∞
1	2.7368	0.0885	0.5717	0.0473
2	1.8055	0.0715	0.5204	0.0311
3	2.0676	0.0706	0.5034	0.0275
4	2.6028	0.0955	0.4335	0.0290
5	7.4070	0.2534	2.3770	0.1064
6	5.9475	0.2274	1.6294	0.0803
7	4.7206	0.1873	2.2645	0.1278

Figure 4.29 illustrates the comparison of the absolute tracking errors of two systems for all 7 joints during the experiment. The error increases between $t = 25\ s \sim 50\ s$ after attaching the load at $t = 25\ s$. With the implementation of the output layer adaptation starting from $t = 50\ s$, the tracking errors of both systems decrease but the system I with online learning (dashed blue curve in Figure 4.29) reduces more than system II (red curve in Figure 4.29). Table 4.13 compares the Frobenius norm of tracking errors of all 7 joints at different stages for two systems. The table indicates that both systems have similar tracking performance from $t = 0 \sim 75\ s$ with the same condition. However, starting from $t = 75\ s$, system I presents a better tracking performance than system II due to the regressor online learning. The comparison also shows that the adaptive controller is able to compensate for the unknown load without the regressor online update. However, the online learning of regressor speeds up the tracking error convergence with a better model approximation accuracy.

It remains an open problem to choose the proper update rate ratio between the regressor online learning and output layer weights adaptation. In the comparative experiment, we implement the regressor online update every 25 seconds whereas the output layer weights adapt at a rate around 100 Hz (thus a ratio of 2500:1).

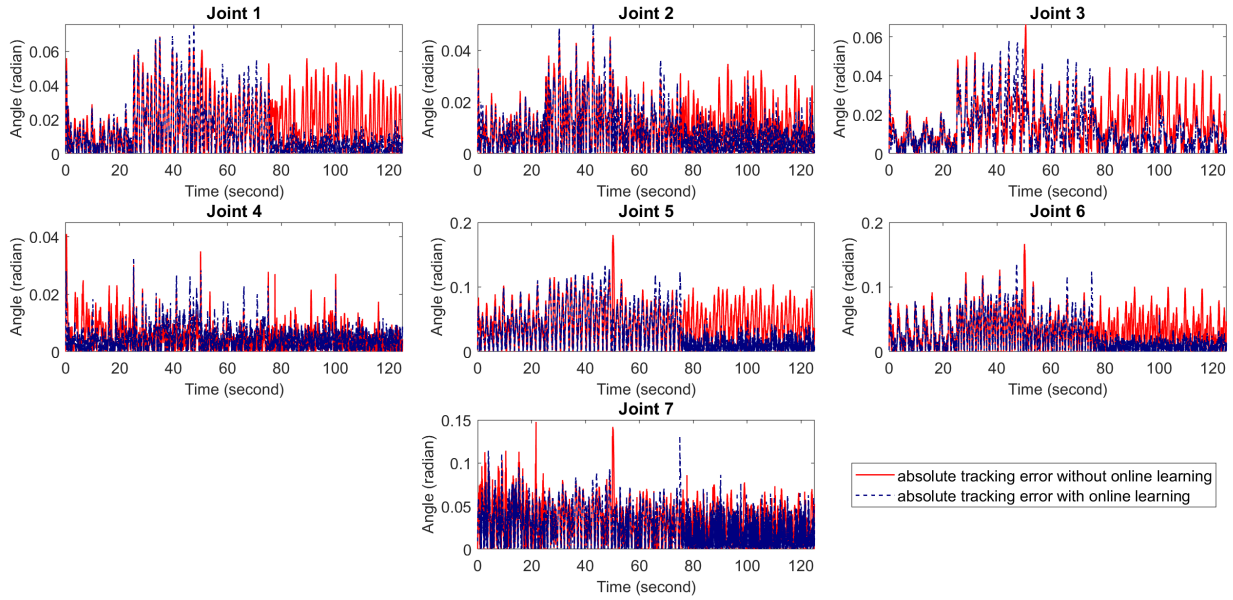


Figure 4.29: Comparison of absolute tracking errors using the developed neural adaptive controller with (blue curve) and without (red curve) the regressor online learning for tracking a multi-joint sinusoidal trajectory, with an unknown load attached at the robot end effector. The regressor online learning finishes at $t = 75$ s and $t = 100$ s, which speeds up the tracking error convergence.

Table 4.13: Frobenius norm of tracking errors (in radian) for all 7 joints of the multi-joint sinusoidal trajectory in Figure 4.29 using the neural adaptive controller with (system I) and without (system II) online learning of the regressor. The regressor online learning scheme speeds up the tracking error convergence starting from $t = 75$ s.

System	$[0,25]s$	$[25,50]s$	$[50,75]s$	$[75,100]s$	$[100,125]s$
I	2.1449	3.745	2.830	1.577	1.199
II	2.376	3.494	3.045	2.748	2.689

4.4.6 Neural Adaptive Controller for the ABB Robot

The same neural adaptive control framework may also apply to a rigid-joint robot such as the ABB industrial robot. The dynamics of a rigid-joint robot has a similar form to (4.10):

$$M(q)\ddot{q} + C(q,\dot{q})\dot{q} + G(q) = \tau, \tag{4.19}$$

Table 4.14: The neural adaptive controller presents a better tracking of a sinusoidal multi-joint trajectory for the ABB robot EGM interface than the baseline PD controller, in terms of RMS and ℓ_∞ norm of tracking errors.

Joint	Original Error (rad)		PD Control Error (rad)		Neural Adaptive Error (rad)	
	RMSE	ℓ_∞	RMSE	ℓ_∞	RMSE	ℓ_∞
1	0.0083	0.0109	0.0055	0.0065	0.0014	0.0026
2	0.0109	0.0143	0.0070	0.0077	0.0027	0.0046
3	0.0341	0.0469	0.0253	0.0337	0.0068	0.0118
4	0.0221	0.0296	0.0154	0.0192	0.0050	0.0106
5	0.0124	0.0163	0.0083	0.0098	0.0030	0.0053
6	0.0825	0.1089	0.0553	0.0651	0.0192	0.0306

where τ is the joint torque vector. We assume the inner-loop torque controller as

$$\tau = -K_p \cdot (q - q_c) - K_d \dot{q} + G(q), \tag{4.20}$$

where q_c is the commanded joint position. It follows

$$M(q)\ddot{q} + (C(q,\dot{q}) + K_d)\dot{q} + K_p q = K_p q_c. \tag{4.21}$$

Then a neural adaptive controller is developed in Appendix D for ABB's EGM interface using the similar stability analysis and neural network architecture as described in Section 4.4.3. Table 4.14 compares the tracking of a sinusoidal joint position trajectory using a PD controller and the neural adaptive controller in RobotStudio.

4.5 Summary

In this chapter, four neural-learning controllers for robot outer loop trajectory tracking control are presented in a progressive style. These controllers focus on inner loop dynamics compensation rather than torque level control, which makes them more readily applicable to current industrial robots. The first controller combines iterative learning control with MNNs, with training data generated rapidly by implementing the gradient-based ILC in a dynamics simulator offered by the vendor. The second controller adopts a URNN to approximate the causal forward robot inner loop dynamics, and the URNN is used to generate the gradient descent direction for ILC. The third

controller explores the use of a BRNN to directly approximate the non-causal stable dynamical inversion. The development of these three controllers is essentially a simulator-building process, and a large amount of data are required for network training. The forth controller uses a DAG network to emulate the linear-in-parameter expression of robot dynamics. The output layer adaptation is based on stability analysis and regressor online learning is implemented via backpropagation to speed up the tracking error convergence. Compared with the first three controllers, this neural adaptive controller demonstrates asymptotic trajectory tracking and is more data-efficient in terms of off-line network training. The neural adaptive controller has been applied to both flexible-joint and rigid-joint robot manipulators. In next chapter, we tackle a more challenging control problem when contact happens between the robot and the environment.

CHAPTER 5
ROBOT MANIPULATORS HYBRID FORCE AND MOTION CONTROL

This chapter considers the robotic hybrid force and motion control problem and is motivated by a robotic deep rolling task. Deep rolling is a mechanical surface treatment method that applies specified pressure to the metal part surface to enhance its fatigue life. Traditional deep rolling uses computer numerical controlled (CNC) machines to repeatably impart the required high pressure. Such implementation frequently requires modification of the CNC platform to accommodate the high spindle load and is challenging to program for complex geometries. Compared with CNC machines, industrial robots have advantages including lower cost, higher throughput and higher flexibility. Here, we demonstrate the use of an industrial robot as an alternative to CNC machines for deep rolling. The model-free iterative learning control algorithm described in Chapter 3 is adapted to deep rolling with flat and curved surfaces.

5.1 Introduction

Surface treatment processes, including roller burnishing, deep rolling, and laser peening are commonly utilized to improve performance of a wide variety of commercial and aerospace products. Of all these methods, the deep rolling process is of particular interest because it may be customized according to part geometry and the required compressive residual stress profile to enhance part fatigue life and strength [68]. During deep rolling, a workpiece surface is exposed to high local mechanical load using a spherical or cylinder-like tool to induce work hardening and compressive residual stress in the near-surface material. CNC platforms are typically used in industry for deep rolling due to their rigidity, but they are typically expensive with low throughput. Furthermore, many CNC platforms are not designed to accommodate the high spindle loads required for rolling/burnishing. Large robotic systems could provide an excellent alternative to traditional CNC platforms for deep rolling (as illustrated in Figure 5.1) with the advantages of lower cost, higher throughput, and process flexibility for parts.

Industrial robots typically only allow users to access the external position and velocity con-

Portions of this chapter previously appeared in: S. Chen, Z. Wang, A. Chakraborty, M. Klecka, G. Saunders, and J. T. Wen, "Robotic deep rolling with iterative learning motion and force control," *IEEE Robot. Autom. Lett.*, vol. 5, no. 4, pp. 5581-5588, Oct. 2020.

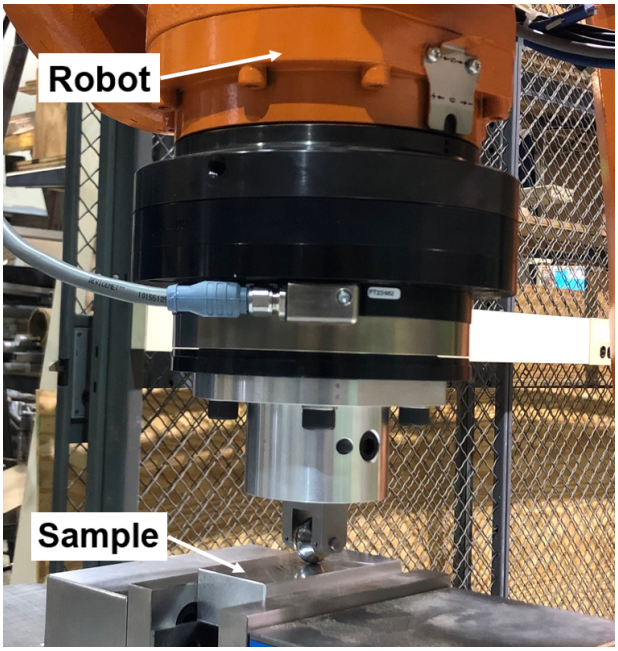


Figure 5.1: Robotic deep rolling with a flat sample.

trol interfaces. In this chapter, we first present and compare three strategies to implement deep rolling on a flat sample using the ABB IRB6640 robot with the EGM position control interface. We then extend the rolling scenario to a curved sample. For flat sample rolling, the first approach is position-based rolling with open-loop force control. In this case, a user commands a pre-computed position trajectory that produces the desired contact force and motion in orthogonal directions. However, this approach does not reject force disturbances and is difficult to be applied to non-constant force profiles or curved parts. The second method uses force feedback to implement impedance control [50] to ensure force profile tracking. Such schemes are offered as part of a proprietary add-on feature for some industrial robots, e.g., the ABB force control package [90]. Compared with the open-loop control, the impedance control is robust with respect to the robot and material properties and can reject force disturbances. However, the tracking performance for time-varying or position-dependent force profiles is frequently compromised by the robot dynamics, joint servo controllers, and motion-induced inertial forces. The third approach is based on the model-free gradient-based iterative learning control as introduced in Chapter 3, which requires a nearly diagonal system.

In this chapter, we first apply the model-free gradient-based ILC algorithm in (3.7) to robotic deep rolling on a flat surface by designing an MIMO system with a symmetric impulse response kernel. This system directly takes the desired Cartesian force and motion as input and the measured

Cartesian force and motion as output. We have implemented all three approaches mentioned above for flat-surface rolling and evaluated the tracking performance for a trapezoidal force and motion profile using the ABB industrial robot. The open loop controller is not applicable for such a time-varying force profile. The ILC controller improves the RMS tracking error by over 70 % compared with impedance control. We then apply the ILC algorithm to deep rolling on a curved surface, by designing an SISO system, which takes the commanded force along surface normal as input and the measured surface normal force as output. This scalar system naturally has a symmetric impulse response kernel so that the model-free ILC is applicable.

To our best knowledge, this is the first theoretical and experimental work that applies ILC to robotic deep rolling. The contributions of this study include:

1. Introduction of an ILC algorithm for motion and force trajectory tracking that uses the physical process itself to generate the gradient descent direction and does not require explicit model information.

2. Demonstration of the applicability of the proposed ILC algorithm to the commanded position control interface available in many industrial robots.

3. Experimental demonstration of the feasibility of the approach for deep rolling on a flat surface with a desired time-varying force profile, as well as rolling on a curved surface with a constant target force along the surface normal.

A preliminary investigation of using an industrial robot as an alternative to CNC machines for deep rolling process has been reported in [91], but it has not addressed the force control problem. The goal of this study is to lower the barrier of using industrial robots for surface treatment processes by improving the robot force control accuracy while retaining the benefits of flexibility and lower cost.

5.2 MIMO ILC for Robotic Deep Rolling on a Flat Sample

Robotic deep rolling involves moving a rigid roller along a pre-determined trajectory while applying a desired force normal to the surface tangent plane containing the path. The task specification is therefore motion along the surface of the part and force orthogonal to the motion direction. However, the control input is the commanded joint position q_c and the measured outputs are the actual joint position q and joint velocity $\dot{q}$. In this section, we will transform the input/output

variables in the physical system to the Cartesian Space suitable for motion/force control using the gradient ILC algorithm in (3.7).

5.2.1 System Input/Output Map

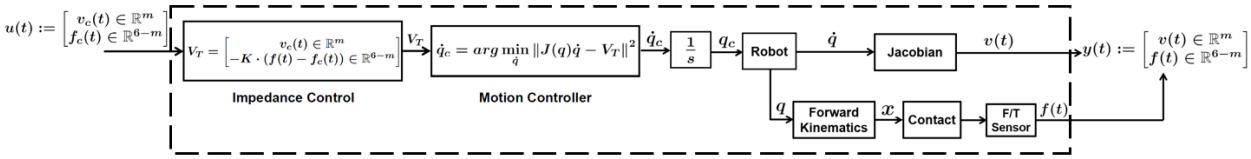


Figure 5.2: Structure of the input/output map for ILC implementation.

As the desired hybrid force and motion profile are described in the Cartesian Space, we will use the resolved motion controller as shown in Figure 5.2. The input is the commanded motion and force, (v_c, f_c), in the orthogonal directions in the robot world frame. The commanded force is converted to a velocity command using impedance control (generalized damper):

$$v_s = -K \cdot (f - f_c). \tag{5.1}$$

The combined desired task space velocity, V_T, is converted to the commanded joint velocity and then joint position, q_c, which in turn is fed into the robot controller. The measured joint velocity is mapped to the task velocity in the motion direction. The contact force is directly measured using the force/torque sensor. We consider the transformed input/output map as $y = \mathcal{G}(u)$ where

$$u(t) = \begin{bmatrix} v_c(t) \\ f_c(t) \end{bmatrix}, \quad y(t) = \begin{bmatrix} v(t) \\ f(t) \end{bmatrix}. \tag{5.2}$$

Similarly, the desired motion and force profile is denoted by $y_d(t)$. Note that since the robot joint controller is stable, $\mathcal{G}$ is a stable (nonlinear) dynamical system.

5.2.2 System Characterization

As discussed in Chapter 3, a sufficient condition for applying the model-free gradient-based ILC is the nearly diagonal structure of the map $\mathcal{G}$. We will consider the condition for the specific robot that we use, but the methodology is applicable to other industrial robots. We employ the ABB IRB6640-180 articulated robot with the EGM module, as introduced in Chapter 2.

The joint servo controllers for industrial robots are typically well tuned so the joint motion follows the commanded motion closely, and there is almost no cross-coupling. Denote the mapping from the commanded joint angle q_c to the actual joint angle q by:

$$q = H(q_c). \tag{5.3}$$

By applying a chirp signal to q_c, we may obtain the local frequency response of the linearization of H at various robot configurations, as shown in Figure 4.3. Since deep rolling does not involve large changes of the robot configuration, H is approximately a linear time invariant system. Furthermore, the joint servo controllers are tuned to achieve almost the same closed loop behavior. Therefore, H is nearly identical for all joints. This means H may be replaced by a stable scalar transfer function.

Away from singularities, the resolved motion controller is equivalent to the J^{-1} controller. So the joint angle commanded is

$$\begin{aligned} \dot{q}_c &= J^{-1} \cdot V_T = J^{-1} \cdot \begin{bmatrix} v_c \\ -K \cdot (f - f_c) \end{bmatrix} \\ &= J^{-1} \left(\begin{bmatrix} I & 0 \\ 0 & K \end{bmatrix} \begin{bmatrix} v_c \\ f_c \end{bmatrix} + \begin{bmatrix} 0 \\ -K \cdot f \end{bmatrix} \right). \end{aligned} \tag{5.4}$$

For the output y, we assume a linear spring model for the surface with the spring constant K_s. When the arm is in contact with the surface, we approximate the system output as:

$$y = \begin{bmatrix} v \\ f \end{bmatrix} \approx \begin{bmatrix} J_m \dot{q} \\ K_s (J_s \Delta q) \end{bmatrix} \approx \begin{bmatrix} t_s^{-1} \cdot I & 0 \\ 0 & K_s \end{bmatrix} J \Delta q, \tag{5.5}$$

where $\dot{q}$ is approximated by $\Delta q / t_s$. t_s is the sampling period (4 ms for the ABB EGM), and J_m and J_s are the Jacobians in the motion and force directions, respectively. Combining $\Delta q = H(\Delta q_c)$ with (5.4), we have

$$y \approx \begin{bmatrix} I & 0 \\ 0 & K_s t_s \end{bmatrix} J H J^{-1} \left(\begin{bmatrix} I & 0 \\ 0 & K \end{bmatrix} u + \begin{bmatrix} 0 \\ -K \cdot f \end{bmatrix} \right). \tag{5.6}$$

As explained above, we approximate our industrial robot controller by a single linear transfer

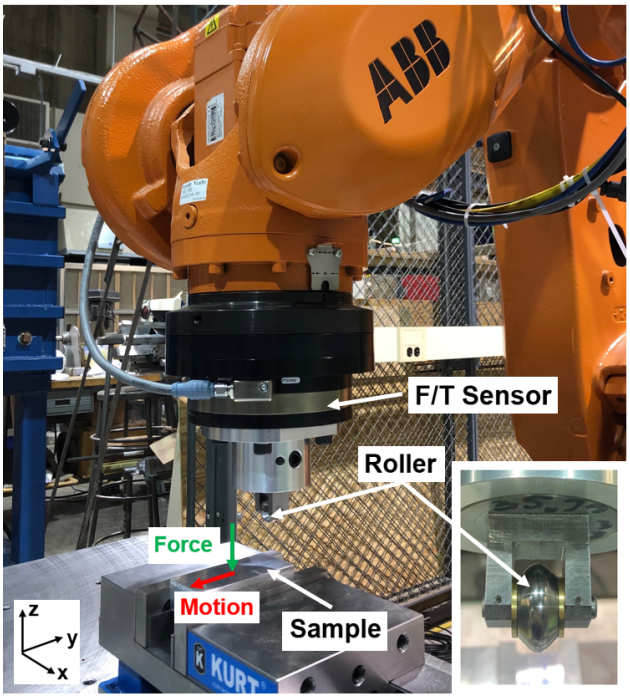


Figure 5.3: Experimental setup for robotic deep rolling with a flat sample.

function for all joints. In that case, H and J^{-1} commute, and we have

$$y \approx \begin{bmatrix} H & 0 \\ 0 & (I + t_s K_s H K)^{-1} t_s K_s H K \end{bmatrix} u. \tag{5.7}$$

Since H is a scalar and the surface spring is assumed decoupled, we choose the force control gain K to be diagonal, so that the input/output relationship from u to y is approximately diagonal. If the approximation is sufficiently close, then as shown in Chapter 3, the ILC algorithm, Algorithm 1, will ensure convergence of $\underline{y}$ to $\underline{y}_d$.

5.2.3 Experimental Results

5.2.3.1 Experimental Setup

With the ABB robot as described in Section 5.2.2, a 6-axis ATI Omega160 force/torque sensor capable of measuring forces in the range of $\pm$ 2500 N and with a measurement uncertainty of 1 % is mounted at the robot flange. A hardened-steel roller is connected to the force/torque sensor through a yoke and a roller bearing. The size of the flat sample is $102.7 \times 48.2 \times 25.4$ mm. The entire testbed is shown in Figure 5.3.

We have implemented three control approaches: position-based open-loop rolling, impedance

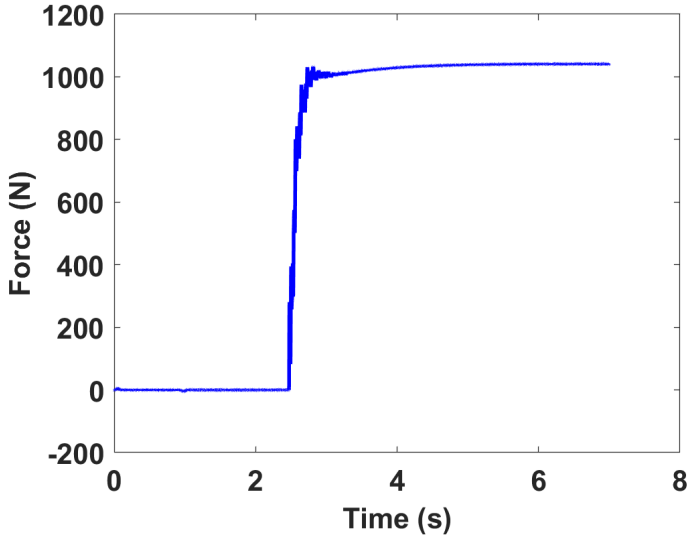


Figure 5.4: The open-loop position-based control achieves good force tracking for a constant 1000 N desired force in z.

control, and iterative learning control. As denoted in Figure 5.3, for all experiments, the desired force is in the z direction of the robot world frame, and the desired motion is along the y direction, i.e., y_d is in form of:

$$y_d := \begin{bmatrix} 0 & 0 & 0 & 0 & v_d^y & f_d^z \end{bmatrix}^T. \tag{5.8}$$

The total number of samples of the trajectory is $N = 2000$.

5.2.3.2 Implementations and Comparison

Open-Loop Position Control

We first evaluate the open-loop position-based force control for tracking a constant 1000 N force. With the attached force/torque sensor, we first determine the desired position of robot end effector that produces the contact force by manually jogging the robot to just establish contact with the material. Then, we command the pre-determined position trajectory using RAPID (the high-level programming language for ABB robots control). The force tracking performance is plotted in Figure 5.4.

This open-loop control approach is straightforward and easy to implement using the standard robot controller. However, it is difficult to determine the desired position trajectory if the desired force profile is time-varying or position-dependent, or the part has a curved geometry.

Impedance Control

For impedance control, we use the full generalized mass-spring-damper desired impedance driven by the measured force to generate the commanded robot joint position:

$$V_T = \dot{X}_T = K_p(F - F_{ref}) + K_i(X_T - X_{ref}) + K_d\ddot{X}_T, \tag{5.9}$$

where X_T is a parameterization of the task space position and orientation of the robot end effector. F_{ref}, X_{ref} are the referenced trajectories. We consider the desired force and motion profiles as trapezoidal in time with the peak force at 1000 N in the z direction. We have tested a group of different set of gains (K_p, K_i, K_d), and the tracking performance corresponding to the best set of gains ($K_p = 0.0002$, $K_d = 0.0008$, $K_i = 0.0004$) is shown in Figure 5.5a. The presence of a large delay in force tracking is evident, due to the dynamics in H.

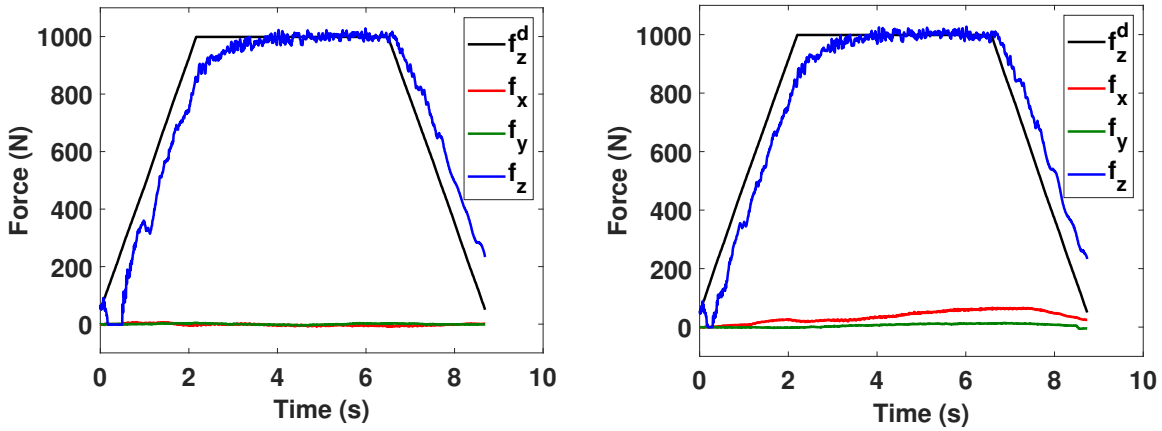


Figure 5.5: (a) Force tracking performance with a full generalized mass-spring-damper impedance controller. (b) Force tracking performance with a generalized damper controller. The tracking performance of using the generalized damper (only keeps K_p in (5.9)**) is similar but slightly worse than using a full generalized mass-spring-damper controller (keeps all gains in** (5.9)**).**

ILC Control

For the ILC implementation, we consider the same force and motion profile as in the impedance control case. We use the high-fidelity dynamic simulator RobotStudio to tune key parameters including the learning rate α and control gain for ILC update. Note that we add a step over of 0.25 mm in the x direction for each ILC iteration to ensure rolling is performed on the unrolled material. For the impedance control component as shown in Figure 5.2, we only keep the gain $K = 0.0002$ in (5.1), which is a generalized-damper controller as in the analysis in Section 5.2.

The force tracking performance (shown in Figure 5.5b) of the first roll (without any iterative refinement) is similar but slightly worse than the result in Figure 5.5a (more obvious in the force in x direction), where we tune and use all three gains K_p, K_i, K_d. With the desired force and motion profiles as initial input $\underline{u}_0$ in (3.7), we implement 9 ILC iterations thus 18 step overs, because for each update we need to roll twice due to the time reversal filtering procedure. For all ILC iterations, we use a fixed learning rate $\alpha = 0.25$.

Figure 5.6 demonstrates the force tracking performance in z with number of ILC iterations. The RMS and $\|.\|_\infty$ errors for force tracking in z are summarized in Table 5.1. The RMS of tracking error monotonically decreases with the increase of ILC runs. The $\|.\|_\infty$ of error, although not monotonically decreasing, demonstrates a clearly decrease trend. After 9 iterations, the force tracking error in terms of RMS error improves by over 78 %.

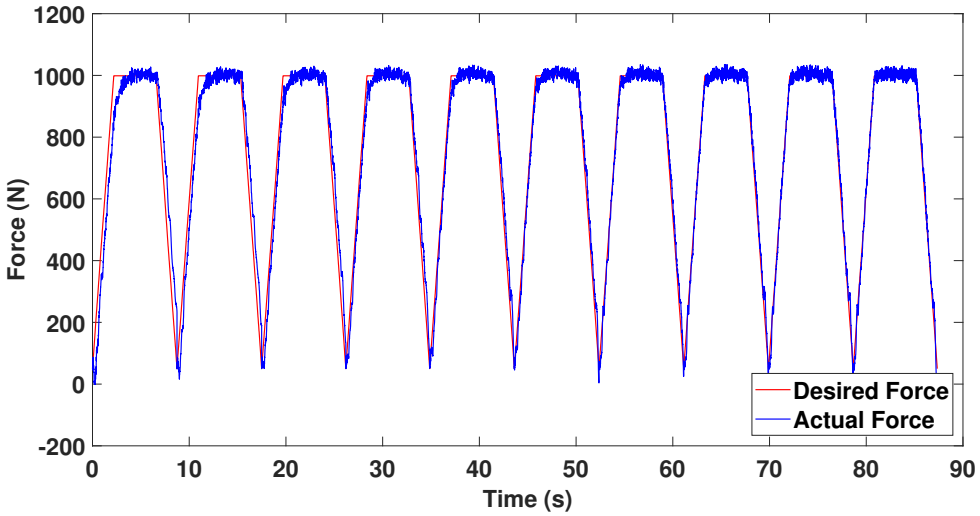


Figure 5.6: Force tracking performance of the same trapezoidal force profile in z as in Figure 5.5 over 9 ILC iterations. Results from the second roll in each iteration for gradient descent direction computation are omitted. The RMS force tracking error reduces by over 78 % after 9 ILC iterations, as summarized in Table 5.1.

The comparison of velocity tracking performance in y direction using the impedance controller, with 0 and 9 ILC iterations is illustrated in Figure 5.7. Note that as in the force tracking, the velocity tracking performance using the impedance controller is similar to using the generalized-damper control without any ILC iterations. The RMS and $\|.\|_\infty$ velocity tracking errors of all 9 ILC iterations are summarized in Table 5.2.

It needs to be mentioned that there is a mistake in coding the integral of $\dot{q}_c$ when we implement the ILC for motion control, which is corrected for the curved rolling.

Table 5.1: RMS and L_∞ norm of trapezoidal force tracking errors (1000 N peak force) in z with the ILC algorithm. The RMS force tracking error monotonically reduces by over 78 % after 9 iterations.

Iteration (No.)	RMSE (N)	L_∞ Norm Error (N)
0	98.3848	185.4981
1	77.6698	150.9880
2	62.0531	149.2758
3	49.3232	122.7392
4	40.4415	119.7123
5	32.9373	101.3830
6	27.1772	81.7024
7	24.3933	103.6499
8	22.2196	103.8649
9	20.7247	83.8624

Table 5.2: RMS and L_∞ norm of trapezoidal velocity tracking errors in y.

Iteration (No.)	RMSE (mm/s)	L_∞ Norm Error (mm/s)
0	3.1506	4.7358
1	2.6329	3.8745
2	2.2173	3.1618
3	1.8800	2.6507
4	1.6033	2.2064
5	1.3743	1.8320
6	1.1838	2.6493
7	1.0217	1.4880
8	0.8856	2.1225
9	0.7688	1.8215

The gradient update profiles for the force over the ILC iterations are shown in Figure 5.8. As the error decreases, the gradient also converges over the entire trajectory.

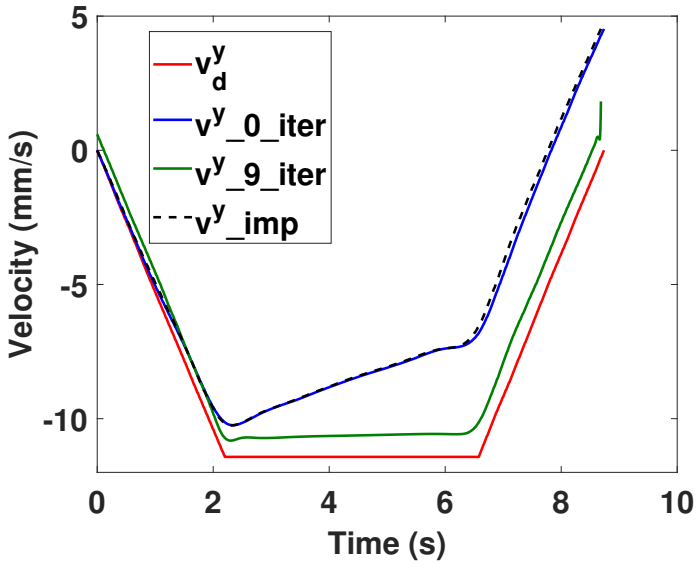


Figure 5.7: Comparison of tracking of the desired velocity profile (red) with the impedance controller (black), 0 ILC iteration (blue) and 9 ILC iterations (green).

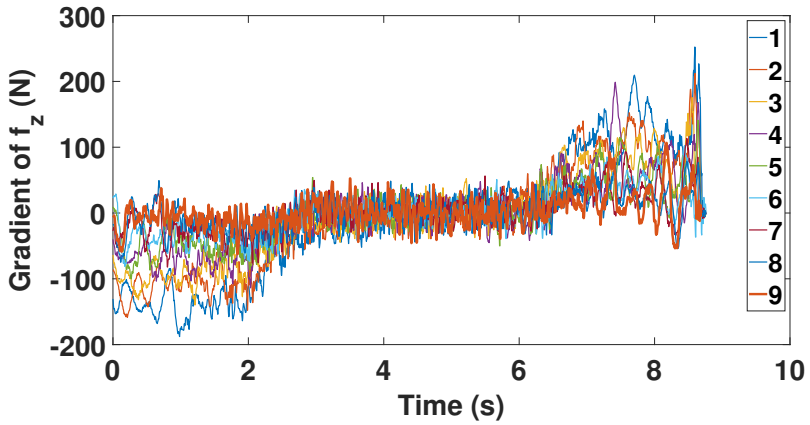


Figure 5.8: Gradient update for force over the nine ILC iterations. The gradient converges with increase number of ILC iterations.

We also evaluate the proposed ILC algorithm for a trapezoidal desired force with a peak of 2000 N. As summarized in Table 5.3, after 7 iterations, the tracking accuracy in terms of RMS error improves by over 73 %.

5.2.4 Discussion

In this section, we apply the gradient-based ILC approach to industrial robotic deep rolling on a flat surface with a nearly diagonal MIMO system. The proposed approach finds the gradient

Table 5.3: RMS and L_∞ norm of trapezoidal force tracking errors (2000 N peak force) in z with the ILC algorithm. The RMS force tracking error monotonically reduces by over 73 % after 7 iterations.

Iteration (No.)	RMSE (N)	L_∞ Norm Error (N)
0	183.2640	413.1678
1	149.0200	345.3046
2	120.6820	300.3394
3	96.4220	251.4267
4	77.9340	214.2127
5	63.9560	192.5337
6	56.0300	175.6275
7	49.2820	150.0793

descent directions based on experimental data without the need for system modeling information. We validate the approach through experiments using an ABB industrial robot. The proposed ILC method outperforms the impedance control and is applicable to other industrial robot platforms that allow external command inputs.

One limitation of this study is that the ILC is only applied to rolling on a flat sample, for which case the desired force and motion are both 1-dimensional. In next section we extend the ILC to rolling on a curved surface (as shown in Figure 5.9) by designing a SISO system, where the motion control is taken care by the robot controller and ILC is only responsible for force control.

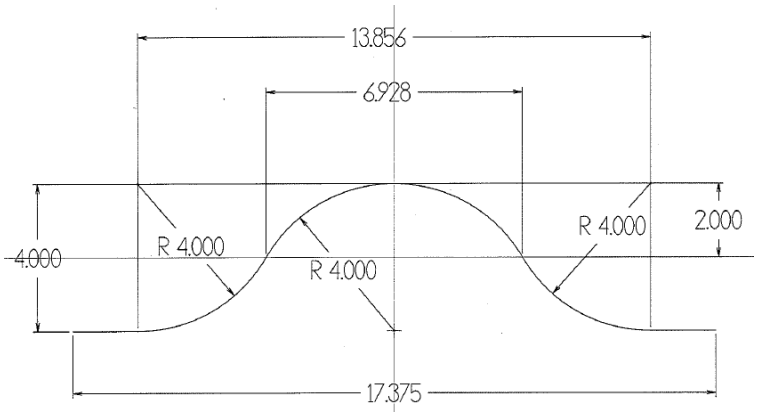


Figure 5.9: Sketch of a curved sample for robotic deep rolling (unit: inch). Image credit: Ken Myer.

5.3 SISO ILC for Robotic Deep Rolling on Surface with Complex Geometries

In this section, we apply the gradient-based ILC in (3.7) to rolling on a curved sample with known geometry by designing a SISO system as shown in Figure 5.10.

5.3.1 System Input/Output Map

The input $u(t) := f_c(t) \in \mathbb{R}^1$ and output $y(t) := f(t) \in \mathbb{R}^1$ are the commanded and measured force magnitude along the surface normal, respectively. The desired force profile along surface normal is denoted by $y_d(t) = f_d(t)$. We consider the transformed input/output map as $y = \mathcal{G}(u)$. Since the robot joint controller is stable, $\mathcal{G}$ is a stable (nonlinear) dynamical system. This scalar system has a symmetric structure and is approximately time-invariant, thus, we can directly apply the model-free ILC law in (3.7) to the system, and iteratively update the commanded force magnitude f_c along the surface normal.

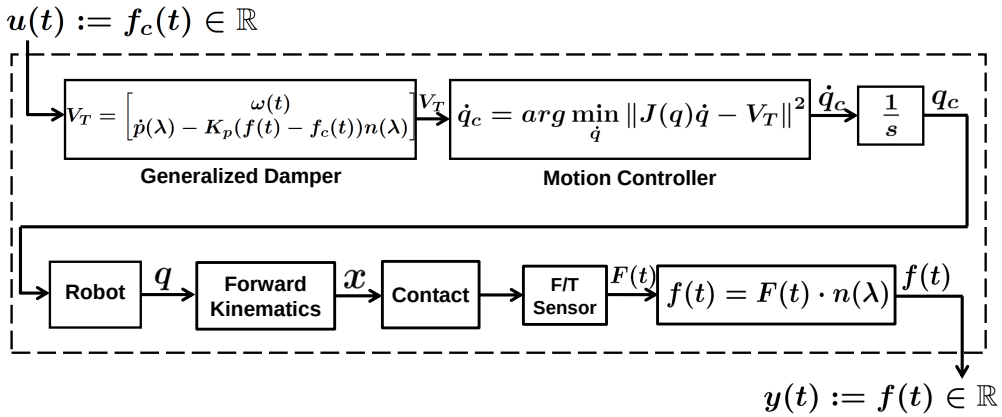


Figure 5.10: Structure of the SISO system for ILC implementation.

As shown in Figure 5.10, the desired task space velocity V_T consists of the motion V_f induced by the command force $f_c(t)$, and the motion V_m (including the rotational velocity and the tangent velocity) based on the surface geometry. As described in Section 5.3.3, the commanded force magnitude $f_c(t)$ is converted to the velocity command V_f along the surface normal using a generalized damper controller. V_m on the other hand, is based on a parametrization of the given surface geometry, as presented in Section 5.3.2. This combined velocity V_T is then converted into a commanded joint position setpoint q_c to EGM via the same resolved motion controller as in Figure 5.2.

5.3.2 Surface Parametrization

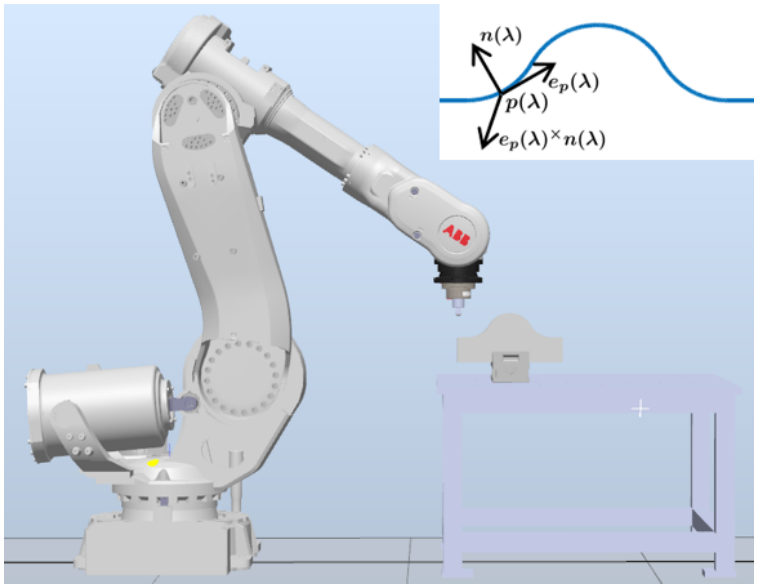


Figure 5.11: Experimental setup in RobotStudio for curved sample deep rolling.

Consider a parametrized path $p(\lambda)$ in the robot world frame, where λ is the path length on a curved surface. The outward surface normal is given by $n(\lambda)$. Then the linear velocity on the path in the robot world frame is given by

$$\dot{p}(\lambda) = p'(\lambda)\dot{\lambda}. \tag{5.10}$$

Choose an orthonormal frame on the path represented in the robot world frame (as denoted in Figure 5.11):

$$R_{0T} = \begin{bmatrix} e_p(\lambda) & e_p(\lambda)^{\times} n(\lambda) & -n(\lambda) \end{bmatrix}, \tag{5.11}$$

where $e_p(\lambda) := p'(\lambda)/||p'(\lambda)||$ and is along the surface tangent direction. $v^{\times}$ denotes the skew-symmetric cross-product matrix formed by $v = (v_1, v_2, v_3)^T$:

$$v^{\times} = \begin{bmatrix} 0 & -v_3 & v_2 \\ v_3 & 0 & -v_1 \\ -v_2 & v_1 & 0 \end{bmatrix}. \tag{5.12}$$

The derivative of R_{0T} is given by

$$\dot{R}_{0T} = \omega^{\times} R_{0T}, \tag{5.13}$$

where

$$\dot{R}_{0T} = \dot{\lambda} \begin{bmatrix} e_p'(\lambda) & e_p'(\lambda)^{\times} n(\lambda) + e_p(\lambda)^{\times} n'(\lambda) & -n'(\lambda) \end{bmatrix}. \tag{5.14}$$

Then angular velocity ω in the robot world frame may be calculated from

$$\omega^{\times} = \dot{R}_{0T} R_{0T}^T, \tag{5.15}$$

where

$$R_{0T}^T = \begin{bmatrix} e_p^T(\lambda) \\ -n^T(\lambda) e_p(\lambda)^{\times} \\ -n^T(\lambda) \end{bmatrix}. \tag{5.16}$$

The motion component of the spatial velocity V_T represented in the robot world frame is then given by

$$V_m = \begin{bmatrix} \omega \\ \dot{p}(\lambda) \end{bmatrix}, \tag{5.17}$$

which is linear in $\dot{\lambda}$.

5.3.3 Generalized Damper Force Controller

The command force magnitude f_c along the surface normal is converted to a velocity command V_f using a generalized damper force controller:

$$V_f = \begin{bmatrix} 0 \\ -K_p \cdot (f - f_c) n(\lambda) \end{bmatrix}, \tag{5.18}$$

where $K_p > 0$ is the control gain, and f is the measured force magnitude along the surface normal represented in the robot world frame and is calculated by the inner product of the measured force vector F and the surface normal $n(\lambda)$ (both represented in the robot world frame)

$$f = F \cdot n(\lambda). \tag{5.19}$$

The combined desired spatial velocity of robot end effector in the robot world frame is then given by:

$$V_T = V_m + V_f = \begin{bmatrix} \omega \\ \dot{p}(\lambda) - K_p \cdot (f - f_c) n(\lambda) \end{bmatrix}. \tag{5.20}$$

5.3.4 Experimental Results

With the experimental setup as shown in Figure 5.12, we have implemented the model-free gradient-based ILC algorithm for rolling on a curved sample (Aluminum 6700) with a constant desired force of 1000 N along the surface normal. This desired force magnitude is used as the initial guess $\underline{u}_0$ for (3.7). We have conducted two experiments with different sample configurations to demonstrate the feasibility of our approach. Similarly, we use the high-fidelity dynamic simulator RobotStudio to tune key parameters including the learning rate α and control gain K_p for ILC update.

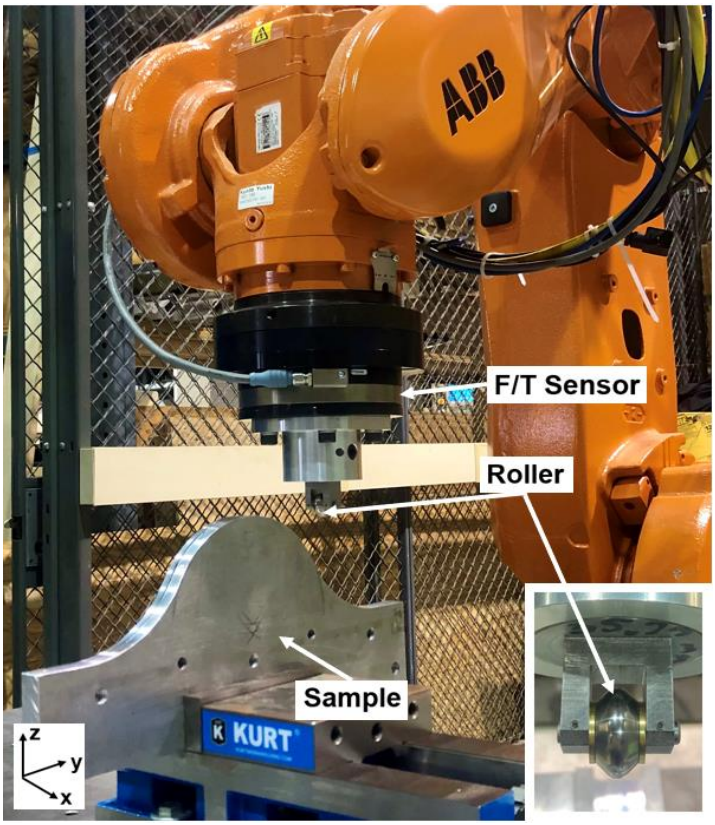


Figure 5.12: Experimental setup for robotic deep rolling with a curved sample.

5.3.4.1 Rolling Under a Singular-Free Configuration

In the first experiment, the sample is placed (as shown in Figure 5.13) such that robot rolls on the sample normally without suffering from any singularity issue. We have implemented 30 ILC iterations with a constant learning rate $\alpha = 0.2$. Figure 5.14 demonstrates the force tracking performance along the surface normal with increase number of ILC iterations. The RMS and $\|.\|_\infty$ errors for force tracking are summarized in Table 5.4. The RMS of tracking error monotonically

decreases with the ILC runs. The $\|.\|_\infty$ of error, although not monotonically decreasing, demonstrates a clearly decrease trend. After 30 iterations, the force tracking error in terms of RMS error improves by over 71 %. The gradient update profiles for the commanded force over the ILC iterations are shown in Figure 5.15. As the error decreases, the gradient update also converges over the entire trajectory.

Figure 5.13: Experimental setup for robotic deep rolling with a curved sample without singularity issues.

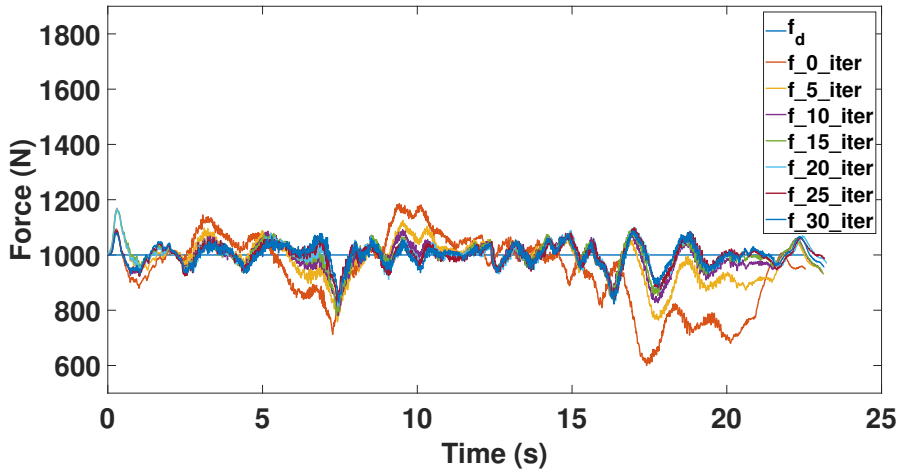


Figure 5.14: Force tracking performance in surface normal (1000 N desired force) over 30 ILC iterations. The RMS force tracking error reduces by over 71 % after 30 iterations.

Table 5.4: RMS and L_∞ norm of force tracking errors along the curved surface normal of Figure 5.14. The RMS force tracking error monotonically reduces by over 71 % after 30 iterations.

Iteration (No.)	RMSE (N)	L_∞ Norm Error (N)
0	140.8367	400.6209
5	72.3622	242.3042
10	50.8356	219.7953
15	46.5687	215.8822
20	44.0902	184.5822
25	40.7411	164.9601
30	40.5328	177.5347

5.3.4.2 Rolling Under a Near-Singular Configuration

In the second experiment, we place the sample at a configuration as shown in Figure 5.12. At this configuration, the robot suffers from a singularity issue when rolling at the high curvature part of the sample. To deal with this issue, we scale down the commanded angular velocity ω in (5.20) so the robot can pass through the singularity and finish rolling for the entire sample. In this case, we have implemented 49 ILC iterations, and Figure 5.16 demonstrates the force tracking performance with increase number of ILC iterations. However, due to the inaccurate angular velocity, the roller is not always perpendicular to the surface during rolling. This results in more oscillations and larger

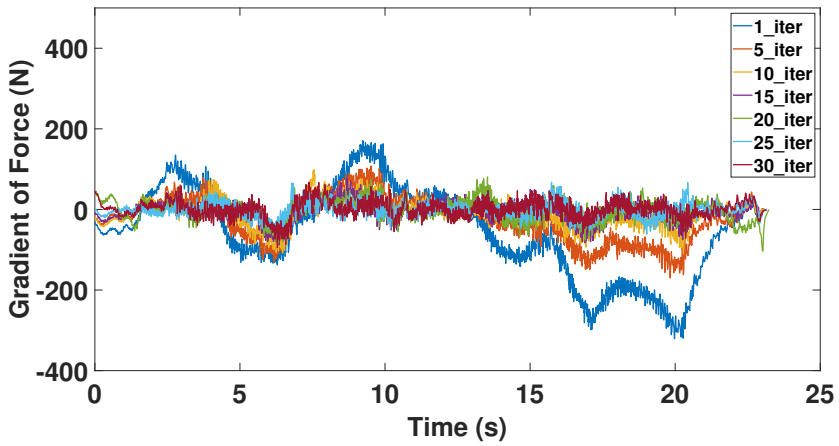


Figure 5.15: Gradient update for commanded force magnitude along surface normal over 30 iterations. The gradient converges with increase number of ILC iterations.

Table 5.5: RMS and L_∞ norm of force tracking errors of Figure 5.16. The RMS force tracking error monotonically reduces by over 90 % after 49 iterations.

Iteration (No.)	RMSE (N)	L_∞ Norm Error (N)
0	302.2104	921.7808
10	157.4218	488.1036
20	85.1536	300.1616
30	51.8709	204.5088
40	35.6210	136.6816
49	30.3840	137.3311

tracking errors compared with the first experiment, which is clearly demonstrated by comparing Figure 5.16 with Figure 5.14. We adopt a time-varying learning rate α as shown in Figure 5.17 to suppress the oscillations at the high curvature portion of the sample.

The RMS and $\|.\|_\infty$ errors for force tracking are summarized in Table 5.5. Similar to the first experiment, the RMS of tracking error monotonically decreases with the ILC runs. The $\|.\|_\infty$ of error, although not monotonically decreasing, demonstrates a clearly decrease trend. After 49 iterations, the force tracking error in terms of RMS error improves by over 90 %. The number of ILC iterations may be reduced by adopting a larger learning rate α, which requires the application of a low-pass filter to the output force signal to remove the high-frequency components. The gradient update profiles for the commanded force over the iterations are shown in Figure 5.18. Similarly, as the error decreases, the gradient update converges over the entire trajectory.

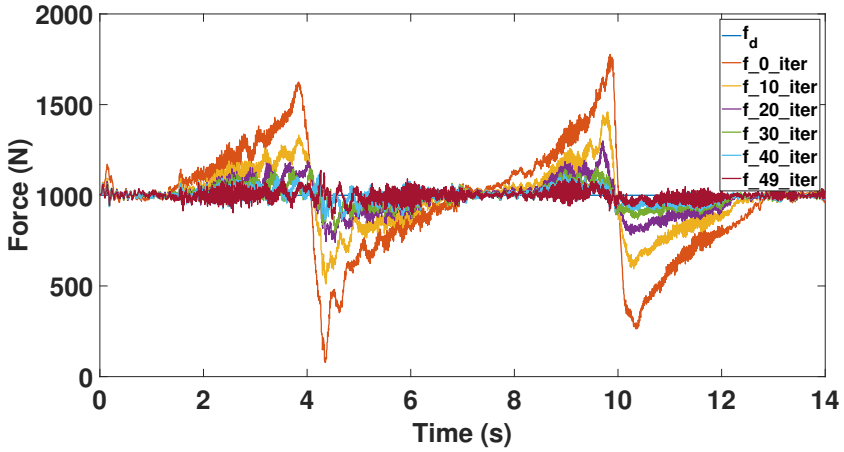


Figure 5.16: Force tracking performance in surface normal (1000 N desired force) over 49 ILC iterations. The RMS force tracking error reduces by over 90 % after 49 iterations.

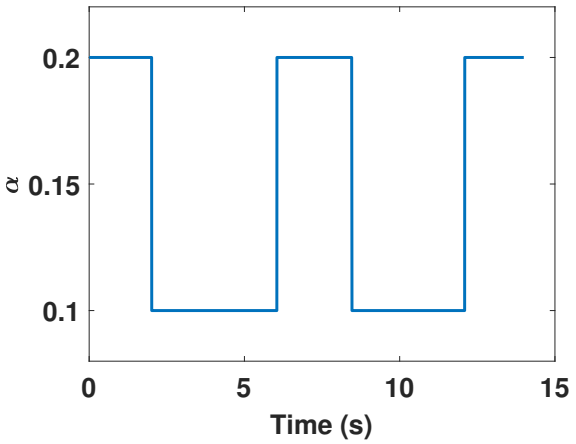


Figure 5.17: The time-varying learning rate α for gradient-based ILC. α is small at high curvature part of the curved sample to suppress oscillations.

5.4 Summary

This chapter presents the application of the model-free gradient-based ILC algorithm to robotic deep rolling tasks based on the design of a system with a nearly diagonal structure, and the assumption that the system is approximately time-invariant. This assumption can be verified experimentally by demonstrating that time-shift of the input trajectory creates an equal time-shift of the output trajectory. One sufficient condition under which this assumption is valid may be when the desired motion and force trajectories are sufficiently slowly time-varying. In addition, as presented in Chapter 4, it is interesting to explore the use of neural networks to approximate the

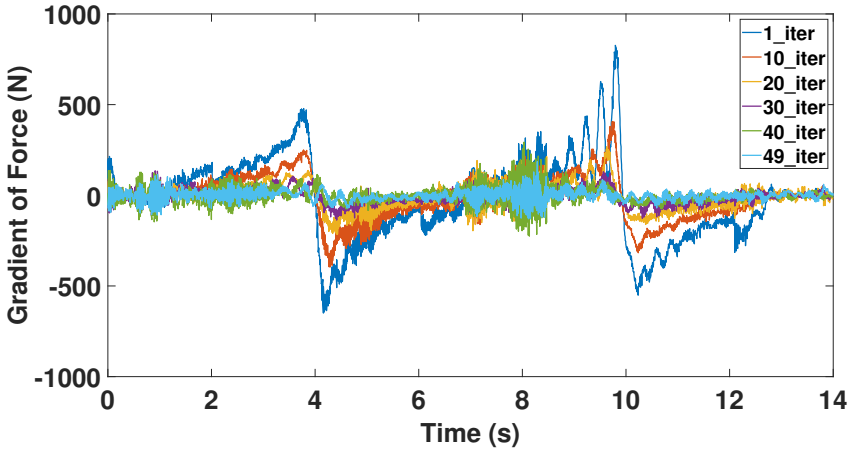


Figure 5.18: Gradient update for commanded force magnitude along surface normal over 49 iterations. The gradient converges with increase number of ILC iterations.

system dynamics for hybrid force and motion control.

CHAPTER 6
CONCLUSIONS AND FUTURE WORK

6.1 Conclusions

This thesis has developed advanced control algorithms for robot motion control as well as hybrid force/motion control with an outer-loop position control interface. Numerous control algorithms from literature have been investigated for robotics control. However, many of these control algorithms require exact knowledge of robot dynamics and a torque control interface, which are rarely available for industrial robots. A model-free gradient-based ILC algorithm has been developed to address these issues. We have demonstrated that the ILC algorithm is applicable to both robot motion control and force control. In addition, inspired by the ability of neural networks as a universal approximator, this thesis also presents a learning-based motion control framework that applies deep learning to classical robotics control algorithms, including adaptive control, iterative learning control and hybrid force/motion control. The main contributions of this thesis include:

- We have developed a resolved velocity controller based on quadratic programming. This kinematic controller is used in teleoperation as well as system design in this thesis. In contrast to the potential field and navigation function approaches, the resolved velocity controller allows us to do motion planning in the task space rather than the configuration space.

- We have presented a robust gradient-based iterative learning control algorithm. The gradient information may be either generated from an identified model using neural networks, or is directly produced through the physical system itself given that the system is nearly diagonal and time-invariant. This thesis explores the first option for pure motion control and the second one for hybrid force/motion control.

- We have employed various neural network architectures for robot dynamics compensation to achieve improved trajectory tracking accuracy. The choice of neural network architecture may depend on the system properties, such as the causality. A unidirectional recurrent network has been trained to approximate the causal robot forward inner loop dynamics. The network may be used as G_k for rapidly implementing iterative learning control (3.7) to find the initial guess for running additional iterative refinements on the physical system. On the

other hand, we have explored the use of an MNN and a BRNN to represent the possibly non-causal robot inverse inner loop dynamics, which achieve real-time dynamics compensation for a teleoperation task.

- We have developed a neural adaptive controller based on a directed acyclic graph network, which expresses the dynamics of a flexible-joint robot into a linear-in-parameter form. We have demonstrated that the tracking error convergence may be accelerated through the regressor online learning using backpropagation. Compared with other aforementioned neural-learning control algorithms, the DAG network requires much less training data for network offline initialization.

- Finally, we have applied the model-free iterative learning control algorithm to robotic deep rolling tasks. We have designed a symmetric multi-input and multi-output system for rolling on a flat sample, and a single-input and single-output system for curved sample rolling. For both cases, the gradient is generated through the system itself without the requirement of explicit system information.

6.2 Future Work

This thesis reports our preliminary works that apply machine learning and optimization techniques to robotics control. However, we should always push the boundary of our research. Here, we list some interesting future research directions:

- The resolved velocity controller discussed in Chapter 2 only solves for the instantaneous joint velocity. However, a mismatch may exist between the rate of collision checking and the external motion command interface. We may address this by adding look-ahead features into the optimization problem and solve for future velocity setpoints.

- The resolved velocity controller assumes a full knowledge of the constant environment for collision checking, which will fail if a new object is added into the environment and the object is movable. This can be potentially solved by using a group of cameras to monitor the workspace and use the updated environment knowledge (e.g., in the form of point cloud) for collision checking. However, it may be challenging to increase the collision checking rate.

- The gradient-based ILC is a trajectory-based gradient descent algorithm. Consider that implementing ILC with complex trajectories may be time-consuming, we can apply ILC to

primitive motion patterns that could be composed together for a complex motion.

- The training of networks for robot dynamics compensation requires a large amount of training data due to the highly nonlinear and complex robot dynamics. In this thesis we train every neural network from scratch for each individual robot. It would be worthwhile to develop a generic network as a universal robot dynamics approximator to speed up the training process for different robots. The development of this generic network requires specific domain knowledge of robot dynamics, which may be obtained from Newton-Euler or Lagrangian dynamics.

- The gradient-based ILC is applied to rolling on a curved and convex sample with known geometry by designing a stable SISO system. The next step is to evaluate the approach with a general concave surface. However, due to the concavity, the robot may suffer from singularity issues. In this case, a redundant robot manipulator can be adopted. It is also interesting to evaluate the rolling using robots with joint flexibility, which may be non-minimum-phase due to actuator/sensor non-collocation.

- We have explored the use of neural networks to improve trajectory tracking accuracy through robot dynamics compensation. The similar idea may be applicable to robot hybrid force and motion control problem. On one hand, a neural network that represents the causal forward dynamics of a stable physical system (e.g., Figures 5.2 and 5.10) may be trained by collecting a set of input and output trajectories (u, y) of the physical system. The trained neural network can be adopted to rapidly generate the gradient directions for ILC. On the other hand, a neural network that models the non-causal stable inverse dynamics of the same physical system may be trained by using the same data set but flipped input/output pairs (y, u).

REFERENCES

[1] A. Wan, J. Xu, S. Zhang, Z. Zhang, and K. Chen, "Learning optimal measurement and control of assembly robot for large-scale heavy-weight parts," in *IEEE ROBIO*, Dec. 2015, pp. 1240–1246.

[2] M. H. Ang, W. Lin, and S.-Y. Lim, "A walk-through programmed robot for welding in shipyards," *Ind. Rob.*, vol. 26, no. 5, pp. 377–388, Jul. 1999.

[3] G. Bone, M. Elbestawi, L. Liu, and N. Sinha, "Adaptive robotic deburring," in *IFAC Symp. on Adapt. Sys. in Control and Signal Processing*, vol. 23, Apr. 1990, pp. 313–318.

[4] N. Asakawa and Y. Takeuchi, "Teachingless spray-painting of sculptured surface by an industrial robot," in *IEEE ICRA*, vol. 3, Apr. 1997, pp. 1875–1879.

[5] V. N. Dubey and J. S. Dai, "A packaging robot for complex cartons," *Ind. Rob.*, vol. 33, no. 2, pp. 82–87, Mar. 2006.

[6] Y.-J. Lee and Y.-B. Lee, "An implementation of the automatic labeling rolling-coil using robot vision system," *J. Inst. Control. Robot. Syst.*, vol. 3, no. 5, pp. 497–502, Oct. 1997.

[7] M. Schöpfer, F. Schmidt, M. Pardowitz, and H. Ritter, "Open source real-time control software for the kuka light weight robot," in *WCICA*, Jul. 2010, pp. 444–449.

[8] ABB, *Application manual: Controller software IRC5: Robotware 6.08*, (2018). Accessed: Apr. 01, 2021. [Online]. Available: `https://abb.sluzba.cz/Pages/Public/IRC5UserDocumentationRW6/en/3HAC050798%5C%20AM%5C%20Controller%5C%20software%5C%20IRC5-en.pdf`.

[9] Z. Qu and D. M. Dawson, *Robust Tracking Control of Robot Manipulators*. Piscataway, NJ, USA: IEEE Press, 1995.

[10] J.-J. E. Slotine and W. Li, "On the adaptive control of robot manipulators," *Int. J. Rob. Res.*, vol. 6, no. 3, pp. 49–59, Sep. 1987.

[11] M. Spong, K. Khorasani, and P. Kokotovic, "An integral manifold approach to the feedback control of flexible joint robots," *IEEE J. Robot. Autom.*, vol. 3, no. 4, pp. 291–300, Aug. 1987.

[12] S. Arimoto, "Learning control theory for robotic motion," *Int. J. Adapt. Control.*, vol. 4, no. 6, pp. 543–564, Nov. 1990.

[13] C. Lin, "Nonsingular terminal sliding mode control of robot manipulators using fuzzy wavelet networks," *IEEE Trans. Fuzzy Syst.*, vol. 14, no. 6, pp. 849–859, Dec. 2006.

[14] C. Williams, S. Klanke, S. Vijayakumar, and K. Chai, "Multi-task gaussian process learning of robot inverse dynamics," in *Advances in NIPS*, Dec. 2009, pp. 265–272.

[15] H. Ho, Y. Wong, and A. Rad, "Robust fuzzy tracking control for robotic manipulators," *Simul. Model. Pract. Theory*, vol. 15, no. 7, pp. 801–816, Aug. 2007.

[16] Y. Jiang, C. Yang, J. Na, G. Li, Y. Li, and J. Zhong, "A brief review of neural networks based learning and control and their applications for robots," *Complexity*, vol. 2017, pp. 1–14, Oct. 2017. DOI: `10.1155/2017/1895897`.

[17] C. Lu, "Wavelet fuzzy neural networks for identification and predictive control of dynamic systems," *IEEE Trans. Ind. Electron.*, vol. 58, no. 7, pp. 3046–3058, Sep. 2011.

[18] W. He and Y. Dong, "Adaptive fuzzy neural network control for a constrained robot using impedance learning," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 4, pp. 1174–1186, Feb. 2018.

[19] S. Chen and J. T. Wen, "Adaptive neural trajectory tracking control for flexible-joint robots with online learning," in *IEEE ICRA*, May 2020, pp. 2358–2364.

[20] Y. Ouyang, L. Dong, Y. Wei, and C. Sun, "Neural network based tracking control for an elastic joint robot with input constraint via actor-critic design," *Neurocomputing*, vol. 409, pp. 286–295, Oct. 2020.

[21] W. Lou and X. Guo, "Adaptive trajectory tracking control using reinforcement learning for quadrotor," *Int. J. Adv. Robot. Syst.*, vol. 13, no. 1, Jan. 2016, Art. no. 38.

[22] Y. Jiang, Y. Wang, Z. Miao, J. Na, Z. Zhao, and C. Yang, "Composite-learning-based adaptive neural control for dual-arm robots with relative motion," *IEEE Trans. Neural Netw. Learn. Syst.*, to be published. DOI: 10.1109/TNNLS.2020.3037795.

[23] L. Kong, W. He, C. Yang, and C. Sun, "Robust neurooptimal control for a robot via adaptive dynamic programming," *IEEE Trans. Neural Netw. Learn. Syst.*, to be published. DOI: 10.1109/TNNLS.2020.3006850.

[24] J. Fei and C. Lu, "Adaptive sliding mode control of dynamic systems using double loop recurrent neural network structure," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 4, pp. 1275–1286, Mar. 2018. DOI: 10.1109/TNNLS.2017.2672998.

[25] Y. Liu, S. Lu, and S. Tong, "Neural network controller design for an uncertain robot with time-varying output constraint," *IEEE Trans. Syst., Man, Cybern. Syst.*, vol. 47, no. 8, pp. 2060–2068, Nov. 2017. DOI: 10.1109/TSMC.2016.2606159.

[26] M. Wang, H. Ye, and Z. Chen, "Neural learning control of flexible joint manipulator with predefined tracking performance and application to baxter robot," *Complexity*, vol. 2017, pp. 1–14, Oct. 2017.

[27] W. He, Z. Yan, Y. Sun, Y. Ou, and C. Sun, "Neural-learning-based control for a constrained robotic manipulator with flexible joints," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 12, pp. 5993–6003, Dec. 2018.

[28] L. Peng and P. Woo, "Neural-fuzzy control system for robotic manipulators," *IEEE Contr. Syst. Mag.*, vol. 22, no. 1, pp. 53–63, Aug. 2002.

[29] C. Chen, "Dynamic structure neural-fuzzy networks for robust adaptive control of robot manipulators," *IEEE Trans. Ind. Electron.*, vol. 55, no. 9, pp. 3402–3414, Aug. 2008. DOI: 10.1109/TIE.2008.926778.

[30] R. Wai and R. Muthusamy, "Fuzzy-neural-network inherited sliding-mode control for robot manipulator including actuator dynamics," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 24, no. 2, pp. 274–287, Feb. 2013. DOI: 10.1109/TNNLS.2012.2228230.

[31] W. He, Y. Dong, and C. Sun, "Adaptive neural impedance control of a robotic manipulator with input saturation," *IEEE Trans. Syst., Man, Cybern. Syst.*, vol. 46, no. 3, pp. 334–344, Mar. 2016.

[32] C. Yang, X. Wang, L. Cheng, and H. Ma, "Neural-learning-based telerobot control with guaranteed performance," *IEEE Trans. Cybern.*, vol. 47, no. 10, pp. 3148–3159, Oct. 2017.

[33] Y. Ji, D. Liu, and Y. Guo, "Adaptive neural network based position tracking control for dual-master/single-slave teleoperation system under communication constant time delays," *ISA Transactions*, vol. 93, pp. 80–92, Oct. 2019, Accessed: Apr. 01, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0019057819301387.

[34] T. Sun, H. Pei, Y. Pan, H. Zhou, and C. Zhang, "Neural network-based sliding mode adaptive control for robot manipulators," *Neurocomputing*, vol. 74, no. 14, pp. 2377–2384, Jul. 2011.

[35] F. L. Lewis, A. Yesildirek, and K. Liu, "Multilayer neural-net robot controller with guaranteed tracking performance," *IEEE Trans. Neural Netw.*, vol. 7, no. 2, pp. 388–399, Mar. 1996.

[36] I. Ranatunga, S. Cremer, F. L. Lewis, and D. O. Popa, "Neuroadaptive control for safe robots in human environments: A case study," in *IEEE CASE*, Aug. 2015, pp. 322–327.

[37] Z. Liu, C. Chen, Y. Zhang, and C. L. P. Chen, "Adaptive neural control for dual-arm coordination of humanoid robot with unknown nonlinearities in output mechanism," *IEEE Trans. Cybern.*, vol. 45, no. 3, pp. 507–518, Mar. 2015. DOI: 10.1109/TCYB.2014.2329931.

[38] L. Wang, Z. Liu, C. Chen, Y. Zhang, S. Lee, and X. Chen, "Energy-efficient svm learning control system for biped walking robots," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 24, no. 5, pp. 831–837, Feb. 2013. DOI: 10.1109/TNNLS.2013.2242486.

[39] G. Shi *et al.*, "Neural lander: stable drone landing control using learned dynamics," *arXiv:1811.08027*, 2018.

[40] A. Noormohammadi-Asl, M. Saffari, and M. Teshnehlab, "Neural control of mobile robot motion based on feedback error learning and mimetic structure," in *ICEE*, May 2018, pp. 778–783.

[41] J. Umlauft and S. Hirche, "Feedback linearization based on gaussian processes with event-triggered online learning," *IEEE Trans. Autom. Control*, vol. 65, no. 10, pp. 4154–4169, Dec. 2019.

[42] A. S. Polydoros, L. Nalpantidis, and V. Krüger, "Real-time deep learning of robotic manipulator inverse dynamics," in *IEEE/RSJ IROS*, 2015, pp. 3442–3448. DOI: 10.1109/IROS.2015.7353857.

[43] H. Chaoui, P. Sicard, and W. Gueaieb, "Ann-based adaptive control of robotic manipulators with friction and joint elasticity," *IEEE Trans. Ind. Electron.*, vol. 56, no. 8, pp. 3174–3187, Aug. 2009.

[44] H. D. Patino, R. Carelli, and B. R. Kuchen, "Neural networks for advanced control of robot manipulators," *IEEE Trans. Neural Netw.*, vol. 13, no. 2, pp. 343–354, Aug. 2002. DOI: 10.1109/72.991420.

[45] R. Selmic and F. Lewis, "Deadzone compensation in motion control systems using neural networks," *IEEE Trans. Automat. Contr.*, vol. 45, no. 4, pp. 602–613, Apr. 2000. DOI: 10.1109/9.847098.

[46] E. Rueckert, M. Nakatenus, S. Tosatto, and J. Peters, "Learning inverse dynamics models in o(n) time with lstm networks," in *IEEE-RAS Humanoids*, Nov. 2017, pp. 811–816.

[47] L. Jin, S. Li, J. Yu, and J. He, "Robot manipulator control using neural networks: A survey," *Neurocomputing*, vol. 285, pp. 23–34, Apr. 2018.

[48] O. Khatib, "A unified approach for motion and force control of robot manipulators: The operational space formulation," *IEEE J. Robot. Autom.*, vol. 3, no. 1, pp. 43–53, Feb. 1987.

[49] T. Yoshikawa, T. Sugie, and M. Tanaka, "Dynamic hybrid position/force control of robot manipulators-controller design and experiment," *IEEE J. Robot. Autom.*, vol. 4, no. 6, pp. 699–705, Dec. 1988.

[50] N. Hogan, "Impedance control: An approach to manipulation," in *ACC*, 1984, pp. 304–313.

[51] R. J. Anderson and M. W. Spong, "Hybrid impedance control of robotic manipulators," in *IEEE ICRA*, vol. 4, 1987, pp. 1073–1080.

[52] M. H. Raibert and J. J. Craig, "Hybrid position/force control of manipulators," *J. Dyn. Syst. Meas. Control*, vol. 103, no. 2, pp. 126–133, Jun. 1981.

[53] S. Jung and T. C. Hsia, "Neural network techniques for robust force control of robot manipulators," in *ISIC*, Aug. 1995, pp. 111–116.

[54] Y. Li, G. Wang, B. Dong, and B. Zhao, "Hybrid position–force control for constrained reconfigurable manipulators based on adaptive neural network," *Adv. Mech. Eng.*, vol. 7, no. 9, pp. 364–383, Sep. 2015.

[55] K. Rani and N. Kumar, "Design of intelligent hybrid force and position control of robot manipulator," in *ICSCC*, Dec. 2018, pp. 42–49.

[56] B. Potsaid, J. T. Wen, M. Unrath, D. Watt, and M. Alpay, "High performance motion tracking control for electronic manufacturing," *J. Dyn. Syst. Meas. Control*, vol. 129, pp. 767–776, Nov. 2007.

[57] M. Norrlöf, "An adaptive iterative learning control algorithm with experiments on an industrial robot," *IEEE Trans. Robot. Autom.*, vol. 18, no. 2, pp. 245–251, Aug. 2002.

[58] C.-C. Cheah and D. Wang, "Learning impedance control for robotic manipulators," *IEEE Trans. Robot. Autom.*, vol. 14, no. 3, pp. 452–465, Jun. 1998.

[59] Y. Li and S. S. Ge, "Impedance learning for robots interacting with unknown environments," *IEEE Trans. Control Syst. Technol.*, vol. 22, no. 4, pp. 1422–1432, Nov. 2013.

[60] B. Huynh, C. Wu, and Y. Kuo, "Force/position hybrid control for a hexa robot using gradient descent iterative learning control algorithm," *IEEE Access*, vol. 7, pp. 72 329–72 342, May 2019. DOI: 10.1109/ACCESS.2019.2920020.

[61] J. Bös, A. Wahrburg, and K. D. Listmann, "Iteratively learned and temporally scaled force control with application to robotic assembly in unstructured environments," in *IEEE ICRA*, May 2017, pp. 3000–3007.

[62] M. Aicardi, G. Cannata, and G. Casalino, "A learning procedure for position and force control of constrained manipulators," in *ICAR*, Jun. 1991, pp. 423–430.

[63] S. R. Pandian and S. Kawamura, "Hybrid force/position control for robot manipulators based on a d-type learning law," *Robotica*, vol. 14, no. 1, pp. 51–59, Jan. 1996. DOI: 10.1017/S0263574700018932.

[64] P. Lucibello, "A learning algorithm for improved hybrid force control of robot arms," *IEEE Trans. Syst., Man, Cybern.*, vol. 28, no. 2, pp. 241–244, Mar. 1998.

[65] D. Jeon and M. Tomizuka, "Learning hybrid force and position control of robot manipulators," in *IEEE ICRA*, May 1992, pp. 1455–1460.

[66] K. Tahara, S. Arimoto, M. Sekimoto, M. Yoshida, and Z. Luo, "On iterative learning control for simultaneous force/position trajectory tracking by using a 5 d.o.f. robotic thumb under non-holonomic rolling constraints," in *IEEE ICRA*, May 2008, pp. 2611–2616.

[67] B. Chen, Y. Wang, and P. Lin, "A hybrid control strategy for dual-arm object manipulation using fused force/position errors and iterative learning," in *IEEE/ASME AIM*, Jul. 2018, pp. 39–44.

[68] P. Delgado, I. Cuesta, J. Alegre, and A. Díaz, "State of the art of deep rolling," *Precision Engineering*, vol. 46, pp. 1–10, Oct. 2016.

[69] L. Lu and J. T. Wen, "Human-directed coordinated control of an assistive mobile manipulator," *IJIRA*, vol. 1, no. 1, pp. 104–120, Feb. 2017.

[70] ABB, *Product manual-irb6640*, (2010). Accessed: Apr. 01, 2021. [Online]. Available: http://nfs.gongkong.com/u_files/robot/zl/IRB6640.pdf.

[71] S. Chen *et al.*, "Software framework for robot-assisted large structure assembly," presented at the ASME 13th MSEC, College Station, TX, USA, Jun. 18-22, 2018.

[72] C. Connolly, "Technology and applications of abb robotstudio," *Ind. Rob.*, vol. 36, no. 6, pp. 540–545, Oct. 2009.

[73] S. Arimoto, S. Kawamura, and F. Miyazaki, "Bettering operation of dynamic systems by learning: A new control theory for servomechanism or mechatronics systems," in *IEEE CDC*, Dec. 1984, pp. 1064–1069.

[74] S. Arimoto, S. Kawamura, and F. Miyazaki, "Bettering operation of robots by learning," *J. Robot. Syst.*, vol. 1, no. 2, pp. 123–140, Jul. 1984, Accessed: Apr. 01, 2021. DOI: https://doi.org/10.1002/rob.4620010203. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/rob.4620010203. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.4620010203.

[75] Y. Chen and K. L. Moore, "Pi-type iterative learning control revisited," in *ACC*, May 2002, pp. 2138–2143. DOI: 10.1109/ACC.2002.1023953.

[76] Y. Chen and C. Wen, *Iterative Learning Control: Convergence, Robustness and Applications*. London, UK: Springer London, 1999.

[77] S. Arimoto and T. Naniwa, "Learnability and adaptability from the viewpoint of passivity analysis," *IASC*, vol. 8, no. 2, pp. 71–94, Mar. 2013. DOI: 10.1080/10798587.2002.10644208.

[78] H. Ahn, Y. Chen, and K. L. Moore, "Iterative learning control: Brief survey and categorization," *IEEE Trans. Syst., Man, Cybern.*, vol. 37, no. 6, pp. 1099–1121, Nov. 2007.

[79] D. A. Bristow, M. Tharayil, and A. G. Alleyne, "A survey of iterative learning control," *IEEE Contr. Syst. Mag.*, vol. 26, no. 3, pp. 96–114, May 2006.

[80] M. Norrlöf and S. Gunnarsson, "Experimental comparison of some classical iterative learning control algorithms," *IEEE Trans. Robot. Autom.*, vol. 18, no. 4, pp. 636–641, Dec. 2002.

[81] D. A. Bristow, M. Tharayil, and A. G. Alleyne, "A survey of iterative learning control," *IEEE Contr. Syst. Mag.*, vol. 26, no. 3, pp. 96–114, May 2006.

[82] J. Bolder, "Flexibility and robustness in iterative learning control : With applications to industrial printers," Ph.D. dissertation, Dept. Mech. Eng., Tech. Univ. Eindhoven, Eindhoven, Netherlands, 2015.

[83] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Trans. Knowl. Data Eng.*, vol. 22, no. 10, pp. 1345–1359, Oct. 2010.

[84] ABB, *Technical reference manual: Rapid instructions, functions and data types*, (2017). Accessed: Apr. 01, 2021. [Online]. Available: https://library.e.abb.com/public/b227fcd260204c4dbeb8a58f8002fe64/Rapid_instructions.pdf?x-sign=f79v/883X1nHGc8fqH+WAJ2F30y/M6TZfYUuPuQpP+jeMBygouyGg+WSj8A9Otry.

[85] M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks," *IEEE Trans. Signal Process.*, vol. 45, pp. 2673–2681, Nov. 1997.

[86] O. Ogunmolu, X. Gu, S. Jiang, and N. Gans, "Nonlinear systems identification using deep dynamic neural networks," *arXiv:1610.01439*, 2016.

[87] J. D. Wason and J. Wen, "Robot raconteur: A communication architecture and library for robotic and automation systems," in *IEEE CASE*, Aug. 2011, pp. 761–766. DOI: 10.1109/CASE.2011.6042513.

[88] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *JMLR*, vol. 15, pp. 1929–1958, Jun. 2014.

[89] 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 *IEEE ICRA*, May 2017, pp. 5183–5189.

[90] ABB, *Application manual - force control*, (2018). Accessed: Apr. 15, 2021. [Online]. Available: https://abb.sluzba.cz/Pages/Public/IRC5UserDocumentationRW6/en/3HAC050377%5C%20AM%5C%20Force%5C%20control%5C%20RW%5C%206-en.pdf.

[91] C. C. Wong, A. Hartawan, and W. K. Teo, "Deep cold rolling of features on aero-engine components," in *CSI*, Jul. 2014, pp. 350–354.

[92] D. S. Mitrinović and J. D. Kečkić, *The Cauchy Method of Residues: Theory and Applications*. Boston, MA, USA: Kluwer Academic Publishers, 1984.

APPENDIX A
EGM DYNAMICS CHARACTERIZATION

EGM sends the commanded joint angle vector $q_c \in \mathbb{R}^6$ to the low level servo controller as a set-point, and reads the actual joint angle vector, $q \in \mathbb{R}^6$. Because of the inner control loop, EGM itself has a dynamic response. Figure A.1 shows step responses of EGM at different joint configurations.

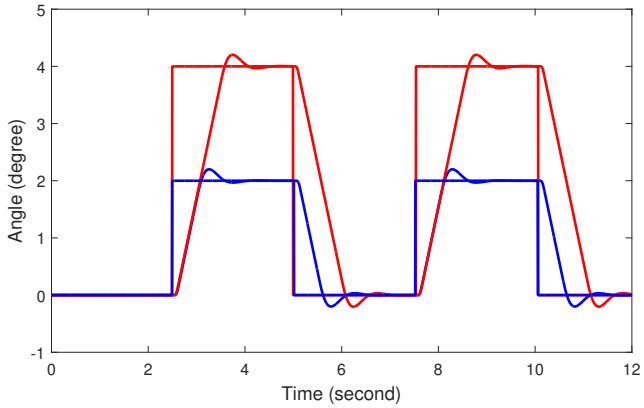


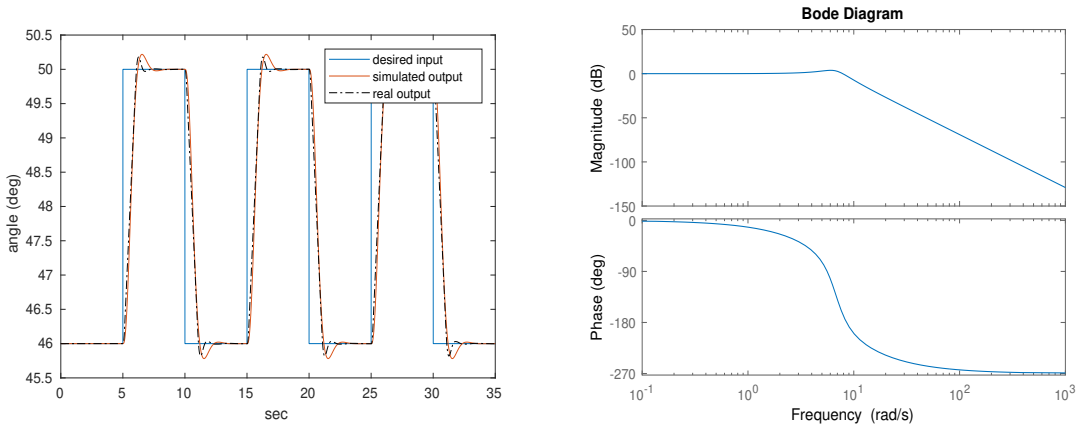
Figure A.1: EGM step responses with different input amplitudes (2 and 4 degrees) at two different robot configurations.

The responses display a classic under-damped behavior, but the nonlinearity (slower dynamics at higher input amplitude) and time delays are clearly visible. The nonlinear dynamics may be approximated by a first-order lag cascaded with a second-order under-damped system:

$$G(s) = \frac{a}{s+a} \cdot \frac{\omega^2}{s^2 + 2\zeta\omega s + \omega^2}, \tag{A.1}$$

where a is the gain of the low-pass filter, and $\zeta (0 < \zeta < 1), \omega$ are the damping ratio and the natural frequency of the second-order system, respectively. Step responses of various input steps are collected to fit this model, and Figure A.2a compares the step responses of the fitted linear model and the actual system. The corresponding Bode plot, as illustrated in Figure A.2b, has a bandwidth of about 0.5 Hz, and a significant phase lag. EGM dynamics is nonlinear due to the velocity and acceleration limits. In addition, these limits are configuration-dependent and the velocity limit dominates the acceleration limit when the robot is commanded with a slow motion profile. By setting the maximum commanded joint velocity to be 10 °/s, EGM stays inside a linear

region. And the linear model in (A.1) may be adopted, e.g., to speed up the ILC implementation.



(a) **(b)**

Figure A.2: The fitted linear model for approximation of the locally linear EGM dynamics. (a) Comparison of step responses between EGM and the identified linear model. (b) Bode plot of the fitted linear model has a limited bandwidth and a large phase lag.

APPENDIX B
ADJOINT OF A LINEAR TIME-INVARIANT SYSTEM

As described in Chapter 3, G_k is the linearization of $\mathcal{G}$ about the state trajectory, which can be written in a state space form as below:

$$\begin{aligned} \dot{\delta x} &= A(t)\delta x + B(t)\delta u \\ \delta y &= C(t)\delta x + D(t)\delta u, \end{aligned} \tag{B.1}$$

where $A(t) = \frac{\partial f}{\partial x}(x,u), B(t) = \frac{\partial f}{\partial u}(x,u), C(t) = \frac{\partial h}{\partial x}(x,u), D(t) = \frac{\partial h}{\partial u}(x,u)$. Note that both G_k and its adjoint G_k^* are linear time varying systems. From the definition of an adjoint of a linear operator, G_k^* is defined as follows:

$$\langle \delta y, G_k \delta u \rangle_{\mathcal{Y}} = \langle G_k^* \delta y, \delta u \rangle_{\mathcal{U}}, \tag{B.2}$$

where $\langle \cdot, \cdot \rangle_{\mathcal{X}}$ denotes the inner product defined on space $\mathcal{X}$.

With zero initial state, the response of (B.1) can be written as:

$$\delta y(t) = \int_0^t H(t,\tau)\delta u(\tau) d\tau, \tag{B.3}$$

where $H(t,\tau) = C(t)\Phi(t,\tau)B(\tau) + D(t)\delta(t-\tau)$ is the impulse response of system (B.1). $\Phi(t,\tau)$ is the state transition matrix associated with system (B.1), and $\delta(t)$ is the Dirac delta function. It can be easily verified by definition of adjoint in (B.2) that

$$(G_k^* \delta y)(t) = \int_t^T H^T(\tau,t)\delta y(\tau) d\tau, \tag{B.4}$$

where $H^T(\cdot,\cdot)$ denotes the transpose of the impulse response matrix $H(\cdot,\cdot)$. Let $\tau\prime = T - \tau$,

from (B.4) we have

$$\begin{aligned}(G_k^* e)(t) &= \int_t^T H^T(\tau,t)e(\tau)d\tau \\ &= \int_{T-t}^0 H^T(T-\tau\prime,t)e(T-\tau\prime)(-d\tau\prime) \\ &= \int_0^{T-t} H^T(T-\tau\prime,t)e(T-\tau\prime)d\tau\prime \\ \Rightarrow (G_k^* e)(t) &= \int_0^{T-t} H^T(T-\tau,t)e(T-\tau)d\tau.\end{aligned} \tag{B.5}$$

If the system (B.1) is time-invariant, we have

$$\begin{aligned}H^T(T-\tau,t) &= B^T e^{A^T(T-\tau-t)}C^T + D^T\delta(T-\tau-t) \\ &= B^T e^{A^T(T-t-\tau)}C^T + D^T\delta(T-t-\tau) \\ &= H^T(T-t,\tau).\end{aligned} \tag{B.6}$$

The adjoint system (B.5) is now

$$(G_k^* e)(t) = \int_0^{T-t} H^T(T-t,\tau)e(T-\tau)d\tau. \tag{B.7}$$

Let $t\prime = T-t$, it follows that

$$(G_k^* e)(T-t\prime) = \int_0^{t\prime} H^T(t\prime,\tau)e(T-\tau)d\tau. \tag{B.8}$$

Then from (B.3) and (B.8), we can see that G_k and its adjoint, G_k^*, are related through their impulse response kernels

$$G_k^* = \mathcal{T} G_k^{\mathsf{T}} \mathcal{T}, \tag{B.9}$$

where $\mathcal{T}$ is the time reversal operator [82], and G_k^{T} is the transpose of the impulse response of G_k.

APPENDIX C
INVERSE OF A NON-MINIMUM-PHASE SISO SYSTEM

Consider the following SISO and non-minimum-phase system:

$$G(s) = \frac{s-1}{s+1}, \tag{C.1}$$

which has an inversion of:

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

Given an output $y(t) = e^{-2t}$ of the system $G(s)$, the goal is to find the corresponding input $x(t)$. Take Laplace transform of $y(t)$, we have:

$$Y(s) = \frac{1}{s+2}. \tag{C.3}$$

Then the Laplace transform of $x(t)$ is:

$$X(s) = G^{-1}(s)Y(s) = \frac{s+1}{(s-1)(s+2)}. \tag{C.4}$$

The partial fraction expansion of $X(s)$ gives:

$$X(s) = \frac{2}{3(s-1)} + \frac{1}{3(s+2)}. \tag{C.5}$$

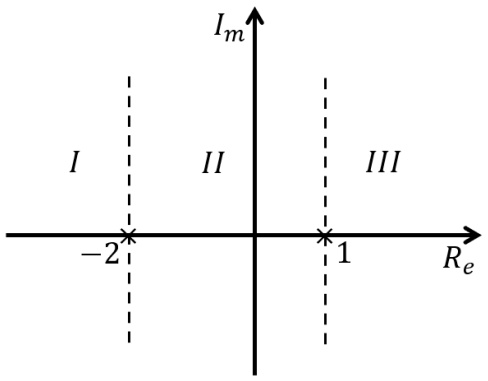


Figure C.1: Three regions of convergence of $X(s)$ in s plane.

There are three regions of convergence (ROC) of $X(s)$ as shown in Figure C.1, corresponding to different $x(t)$:

- In ROC $I(Re(s) < -2)$, the signal is non-causal and $x_1(t) = (-\frac{2}{3}e^{t} - \frac{1}{3}e^{-2t})u(-t)$, which is unbounded as $t \to -\infty$, and $u(t)$ is the unit step function.
- In ROC $II(-2 < Re(s) < 1)$, the signal is non-causal and $x_2(t) = -\frac{2}{3}e^{t}u(-t) + \frac{1}{3}e^{-2t}u(t)$, which is bounded.
- In ROC $III(Re(s) > 1)$, the signal is causal and $x_3(t) = \frac{2}{3}e^{t}u(t) + \frac{1}{3}e^{-2t}u(t)$, which is unbounded as $t \to \infty$.

These $x(t)$ may be computed by solving the inverse Laplace Transform formula:

$$x(t) = \frac{1}{2\pi j}\int_{\sigma - j\infty}^{\sigma + j\infty} X(s)e^{st}ds. \tag{C.6}$$

This integration is performed along the vertical line $s = \sigma$ in the complex plane. The real number σ is chosen so that the line $s = \sigma$ lies in different regions of convergence. It can be shown that when $X(s)$ is strictly proper, the complex integral in (C.6) equals a contour integral if the curve C is chosen as a Bromwich contour [92]:

$$\frac{1}{2\pi j}\int_{\sigma - j\infty}^{\sigma + j\infty} X(s)e^{st}ds = \frac{1}{2\pi j}\oint_{C} X(s)e^{st}\,\mathrm{d}s. \tag{C.7}$$

The contour integral can be solved by using the Cauchy's residue theorem [92]:

$$\frac{1}{2\pi j}\oint_{C} f(s)\,\mathrm{d}s = \sum_{k} R_k, \tag{C.8}$$

where C is a positively oriented (counterclockwise) closed curve, and $f(s)$ is analytic with finite number of poles inside C. R_k are the residues of the poles inside C and can be computed by:

$$R_k = \lim_{s \to p_k}(s - p_k)f(s), \tag{C.9}$$

where p_k is the k^{th} pole of $f(s)$. The choice of straight line $s = \sigma$ depends on different ROC:

- In ROC $I(Re(s) < -2)$, σ is chosen to the left of the left-most pole of $X(s)$ (i.e., $\sigma < -2$), and this results in a Bromwich contour enclosing all poles in the negative clockwise direction,

thus the sign is flipped:

$$x_1(t) = \frac{1}{2\pi j}\int_{\sigma-j\infty}^{\sigma+j\infty} X(s)e^{st}ds = -\sum_k R_k, \tag{C.10}$$

where R_k are the residues corresponding to the poles of $X(s)e^{st}$. Consequently, we have the non-causal solution $x_1(t) = (-\frac{2}{3}e^{t} - \frac{1}{3}e^{-2t})u(-t), t < 0$, which is unbounded as $t \to -\infty$.

- In ROC $II(-2 < Re(s) < 1)$, σ is chosen within the strip of $-2 < Re(s) < 1$, and the contour C includes two curves to enclose all poles of $X(s)$. The first curve encloses the pole of $s = -2$ and is positively oriented (counterclockwise). The second curve encloses the pole of $s = 1$ and is negatively oriented (clockwise). This results in a sign change for the second curve:

$$x_2(t) = \frac{1}{2\pi j}\int_{\sigma-j\infty}^{\sigma+j\infty} X(s)e^{st}ds = \begin{cases} R_1 & t > 0, \\ -R_2 & otherwise, \end{cases} \tag{C.11}$$

where R_1, R_2 correspond to the residues of the pole of $-2, 1$, respectively. We can then compute the non-causal but bounded solution $x_2(t) = -\frac{2}{3}e^{t}u(-t) + \frac{1}{3}e^{-2t}u(t)$.

- In ROC $III(Re(s) > 1)$, σ is chosen to the right of the right-most pole (i.e., $\sigma > 1$) such that the Bromwich contour encloses all poles and is positively oriented. In this case, we have:

$$x_3(t) = \frac{1}{2\pi j}\int_{\sigma-j\infty}^{\sigma+j\infty} X(s)e^{st}ds = \sum_k R_k, \tag{C.12}$$

and this results in the causal but unbounded solution $x_3(t) = \frac{2}{3}e^{t}u(t) + \frac{1}{3}e^{-2t}u(t), t > 0$.

In summary, the causal inverse $x_3(t)$ of this non-minimum-phase system is unstable, and a stable inverse $x_2(t)$ exists but it is non-causal. This motivates the use of neural networks to capture the non-causal behavior of dynamical inversion of a non-minimum-phase plant, as described in Chapter 4.

APPENDIX D
NEURAL ADAPTIVE CONTROL FOR ABB EGM

The dynamics model of rigid-joint robot manipulators can be written in the following form:

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

where $q, \dot{q}, \ddot{q}$ represent joint position, velocity, and acceleration. $M(q), C(q,\dot{q})\dot{q}, G(q)$ denote the manipulator inertia matrix, Coriolis and centrifugal torques, and gravitational torques, respectively. τ is the torque on link side. M is symmetric and positive definite, and $\dot{M} - 2C$ is skew symmetric.

We make the following assumptions for the neural adaptive controller design:

1. The inner-loop joint servo controller is in form of: $\tau = -K_p(q - q_c) - K_d\dot{q} + G(q)$, where q_c is the commanded joint setpoint, and K_p, K_d are positive definite control gains.

2. The gain K_p is unknown but is the same for every joint, i.e., $K_p = k_p \cdot I$, where $k_p > 0$. K_d is a diagonal and positive definite matrix.

Based on the above assumptions, we can rewrite the robot dynamics model as:

$$M(q)\ddot{q} + (C(q,\dot{q}) + K_d)\dot{q} + k_p q = k_p q_c. \tag{D.2}$$

By using $D(q,\dot{q}) = C(q,\dot{q}) + K_d, H(q) = k_p q$, we can simplify the above equation as below:

$$M(q)\ddot{q} + D(q,\dot{q})\dot{q} + H(q) = k_p q_c. \tag{D.3}$$

Same as (4.11), we then express the robot dynamics into a linear-in-parameter form as follows:

$$Y(\dot{q}_1, q, \dot{q}_2, \ddot{q})a := k_p^{-1}(M(q)\ddot{q} + D(q,\dot{q}_2)\dot{q}_1 + H(q)), \tag{D.4}$$

where Y is the regressor and a is a constant vector.

We come up with the following Lyapunov-like function for stability analysis:

$$V = \frac{1}{2}s^T M s + \frac{1}{2}\widetilde{a}^T k_p P^{-1}\widetilde{a}, \tag{D.5}$$

where s is the sliding vector and $s = \dot{e}(t) + \Lambda e(t) = \dot{q} - \dot{q}_r$. The trajectory tracking error is $e(t) = q(t) - q_d(t)$. q_r is a nominal reference variable with $\dot{q}_r = \dot{q}_d - \Lambda e$. The Hurwitz matrix $-\Lambda$ makes $e(t), \dot{e}(t)$ converge to zero exponentially as s converges to zero. P is a designed diagonal positive-definite matrix to control the parameter adaptation rate. $\widetilde{a} = \hat{a} - a$ and $\hat{a}$ is an estimate of the constant vector a.

Take the time derivative of V, we have:

$$\begin{aligned}
\dot{V} &= s^T M\dot{s} + \frac{1}{2}s^T \dot{M}s + \dot{\hat{a}}^T k_p P^{-1}\tilde{a} \\
&= s^T(M\ddot{q} - M\ddot{q}_r) + \frac{1}{2}s^T \dot{M}s + \dot{\hat{a}}^T k_p P^{-1}\tilde{a} \\
&= s^T(k_p q_c - H - D\dot{q} - M\ddot{q}_r) + \frac{1}{2}s^T \dot{M}s + \dot{\hat{a}}^T k_p P^{-1}\tilde{a} \\
&= s^T(k_p q_c - H - Ds - D\dot{q}_r - M\ddot{q}_r) + \frac{1}{2}s^T \dot{M}s + \dot{\hat{a}}^T k_p P^{-1}\tilde{a} \\
&= s^T(k_p q_c - H - D\dot{q}_r - M\ddot{q}_r) + \frac{1}{2}s^T (\dot{M} - 2C - 2K_d)s + \dot{\hat{a}}^T k_p P^{-1}\tilde{a} \\
&= s^T(k_p q_c - H - D\dot{q}_r - M\ddot{q}_r) - s^T K_d s + \dot{\hat{a}}^T k_p P^{-1}\tilde{a} \\
&= s^T k_p(q_c - Y(\dot{q}_r, q, \dot{q}, \ddot{q}_r)a) - s^T K_d s + \dot{\hat{a}}^T k_p P^{-1}\tilde{a}.
\end{aligned} \tag{D.6}$$

If Y is known, then we may design q_c to cancel Ya using the estimate $\hat{a}$ as

$$q_c = Y(\dot{q}_r, q, \dot{q}, \ddot{q}_r)\hat{a} - K_s s - k\mathbf{sgn}(s). \tag{D.7}$$

The $K_s s$ term with $K_s > 0$ ensures the tracking error convergence. The additional high gain feedback term $k\mathbf{sgn}(s)$ provides robustness with respect to the network modeling errors and noises. It follows

$$\dot{V} = -s^T(k_p K_s + K_d)s - kk_p \|s\|_1 + k_p s^T Y\tilde{a} + k_p \dot{\hat{a}}^T P^{-1}\tilde{a}, \tag{D.8}$$

where $\|\cdot\|_1$ denotes the vector 1-norm. By choosing the adaptation law of $\hat{a}$ as

$$\dot{\hat{a}} = -PY^T(\dot{q}_r, q, \dot{q}, \ddot{q}_r)s, \tag{D.9}$$

we have

$$\dot{V} = -s^T(k_p K_s + K_d)s - kk_p \|s\|_1 . \tag{D.10}$$

This implies $s \to 0$ asymptotically which in turn implies $e \to 0$ exponentially. We may apply the same neural network architecture as shown in Figure 4.25 to approximate $Y(\dot{q}_r, q, \dot{q}, \ddot{q}_r)a$. For the neural network training, we have collected the output trajectories in RobotStudio for 100 sinusoidal trajectories with different magnitudes and frequencies. The joint velocity and acceleration are approximated by cubic spline interpolation of the joint position data.

APPENDIX E
SUPPLEMENTARY FILES

E.1 Permissions for Section 2.2

This file contains the permissions for the reproduction of material used in Section 2.2 of this dissertation.

File name: ASME_permission.pdf

File type: Portable Document Format (PDF)

File size: 92 KB

Required application software: Any standard PDF viewer

Special hardware requirements: None