From Model-Based to Data-Driven Discrete-Time Iterative Learning Control

Bing Song

Submitted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy
in the Graduate School of Arts and Sciences

COLUMBIA UNIVERSITY

2019

ABSTRACT

From Model-Based to Data-Driven Discrete-Time Iterative Learning Control

Bing Song

This dissertation presents a series of new results of iterative learning control (ILC) that progresses from model-based ILC algorithms to data-driven ILC algorithms. ILC is a type of trial-and-error algorithm to learn by repetitions in practice to follow a pre-defined finite-time maneuver with high tracking accuracy.

Mathematically ILC constructs a contraction mapping between the tracking errors of successive iterations, and aims to converge to a tracking accuracy approaching the reproducibility level of the hardware. It produces feedforward commands based on measurements from previous iterations to eliminates tracking errors from the bandwidth limitation of these feedback controllers, transient responses, model inaccuracies, unknown repeating disturbance, etc.

Generally, ILC uses an a priori model to form the contraction mapping that guarantees monotonic decay of the tracking error. However, un-modeled high frequency dynamics may destabilize the control system. The existing infinite impulse response filtering techniques to stop the learning at such frequencies, have initial condition issues that can cause an otherwise stable ILC law to become unstable. A circulant form of zero-phase filtering for finite-time trajectories is proposed here to avoid such issues. This work addresses the problem of possible lack of stability robustness when ILC uses an imperfect a prior model.

Besides the computation of feedforward commands, measurements from previous iterations can also be used to update the dynamic model. In other words, as the learning progresses, an iterative data-driven model development is made. This leads to adaptive ILC methods.

An indirect adaptive linear ILC method to speed up the desired maneuver is presented here. The updates of the system model are realized by embedding an observer in ILC to estimate the system Markov parameters. This method can be used to increase the productivity or to produce high tracking accuracy when the desired trajectory is too fast for feedback control to be effective.

When it comes to nonlinear ILC, data is used to update a progression of models along a homotopy, i.e., the ILC method presented in this thesis uses data to repeatedly create bilinear models in a homotopy approaching the desired trajectory. The improvement here makes use of Carleman bilinearized models to capture more nonlinear dynamics, with the potential for faster convergence when compared to existing methods based on linearized models.

The last work presented here finally uses model-free reinforcement learning (RL) to eliminate the need for an a priori model. It is analogous to direct adaptive control using data to directly produce the gains in the ILC law without use of a model. An off-policy RL method is first developed by extending a model-free model predictive control method and then applied in the trial domain for ILC. Adjustments of the ILC learning law and the RL recursion equation for state-value function updates allow the collection of enough data while improving the tracking accuracy without much safety concerns. This algorithm can be seen as the first step to bridge ILC and RL aiming to address nonlinear systems.

Contents

List of Figures

List of Tables

Acknowledgements

First and foremost, I would like to express my special appreciation and thanks to my advisor Professor Richard W. Longman. The longer I work with you, the more wisdom I see. The attitudes towards research and towards life are the best gifts I have ever received during my Columbia journey. This will benefit me for my whole life.

I would also like to thank Professor Minh Q. Phan, my co-advisor. Thank you for your sharpness and strictness in research. The combined mentorship from you and Professor Longman makes this doctoral dissertation possible. Also, I thank Professor Nicolas W. Chbat for the rewarding internship and for encouraging me to chase my dream.

I would especially thank Professor Jeffrey W. Kysar for giving me a desk in your lab. I thank all my lab members for the lovely atmosphere created in our office, both from the Control Group and the Kysar lab, especially Francesco and Ae. Thank you for being such supportive friends. Another thank you goes to Prof. Philip Thomas from U Mass Amherst, and my friends there. You helped me to think and understand Reinforcement Learning as a computer science person.

My deepest gratitude goes to my parents. The best thing that ever happened in my life is being your daughter. Thank you for your open minds. Thank you for supporting me going

through adversity. Thank you for your sacrifices on my behalf. Thank you for teaching me to stick to my principles under whatever pressure.

I need and want to say thank you to my grandma, who brought me up. My biggest regret in my life is that I gave up the chance to see you for the last time. I clearly remember the moment when I promised to myself that I would show you my love by my academic performance, after you left us forever in 2006. All my life changes started from that moment. I hope you would feel satisfied with my work.

I will say the last "thank you" to myself. Sometimes you lost your courage; sometimes you lost your direction; sometimes you lost your patience. But you always corrected yourself in a few days and never give up your dream. Please keep being yourself and walk me through the incredible life adventure.

Bing Song

August 26, 2018

To my parents,

Mr. Song Liqi, and Mrs. Li Xiulian.

This page intentionally left blank

Chapter 1

Introduction

1.1 The Concept of ILC

In 1984, motivated by robots doing repetitive tracking operations in manufacturing, the concept of iterative learning control (ILC) began to flourish [1, 2, 3, 4, 5]. ILC is one type of trial-and-error control algorithm to learn to follow a finite-time trajectory by repetitions in practice, aiming to achieve "zero" tracking error after a few runs.

The goal of ILC as presented here is to achieve "zero" tracking error, which refers to the tracking accuracy approaching the reproducibility level of the hardware. A classical example to demonstrate the effectiveness of ILC is experiments performed on a 7 degree-of-freedom robot at Nasa Langley Research Center [6].

In this experiment, a linear model was separately used for each link, while the desired path was designed to assure a high degree of dynamic coupling and maximization of nonlinear robot dynamic effects. When the base joints were asked to make a 90 degree turn at the maximum speed specified by the robot manufacturer, the tracking error was decreased by a

factor of nearly 1000 in about 12 iterations for learning. This is close to the reproducibility level of the robot, which is the limit of possible improvement.

This "zero-error" tracking is realized by using data (previous tracking errors) from direct interaction with dynamic systems to eliminate errors from the bandwidth limitation of feedback controllers, model inaccuracies, transient response, and unknown repeating disturbances. Simply put, ILC adjust feedforward commands for the current iteration according to measurements from previous iterations in order to improve the tracking accuracy.

Mathematically the trajectory tracking problem is an inverse problem and ILC can take many forms to solve this inverse problem, for example, based on integral control concepts from classical control theory but applied in repetitions, based on indirect adaptive control theory or model reference adaptive control theory operating in time or in repetitions or both, etc.

The underlying principle of these various forms is to construct a contraction mapping between the tracking errors of two successive iterations in practice. To form this contraction mapping, different numerical methods can be used, such as in linear least squares problems and for root finding. For example, Reference [7] uses Tikhonov regularization for ill-conditioned systems and Reference [8] applies Newton method for nonlinear systems.

The difference between ILC and numerical methods is that ILC iterates in the real world with the dynamic system while numerical methods iterate in simulations with the system model. Iterations in the world lead to (1) a solution to the real system instead of the system model and hence the tracking accuracy is not limited by the model accuracy, and (2) a cut-off filter is essential for the ILC implementation in practice due to safety concerns.

For nearly four decades, ILC has achieved fruitful results [9, 10, 11, 12, 13, 14, 15], suc-

cessfully applied to Chrysler manufacturing assembly line, stroke rehabilitation [16], quadrocopters controllers [17], surgical robots [18], mobile robots [19], traffic control [20], etc, to improve the tracking accuracy. In today's trend of learning from data, ILC (together with repetitive control and reinforcement learning) is categorized and studied as learning control for robotics.

1.2 Learning From Data: ILC, Repetitive Control (RC), and Reinforcement Learning (RL)

With the development of the ability to handle large data, the study of data-driven control methods attracts more attention [21]. How to enable a system to mimic the human's learning process is one of the popular topics.

Karl Popper, one of the 20th century's greatest philosophers of science, summarized the philosophy of human's learning as "our knowledge, our aims, and our standards grow through an unending process of trial and error" [22]. This trial-and-error principle is shared by three types of algorithms: ILC, RC, and RL. These are named as "learning control" for robotics in a review paper [23]. It is helpful to understand ILC in comparison to RC and RL.

RC controllers are feedback controllers that attenuate periodic disturbances with the help of the error signal from previous periods [24] while ILC control is an off-line controller producing feedforward signals to improve the tracking accuracy of a finite-time maneuver. Compared to ILC and RL, the distinct feature of RC is that RC only has one episode with a periodic trajectory, i.e., the dynamics is continuous between periods.

The relationship between ILC and RC is well studied by the researchers in control: Reference [9, 12] have deeply analyzed their difference and connections; Reference [24] develops the unified formulation for ILC and RC; and Reference [25] studies the cross-fertilization approaches of these two type of algorithms.

RL solves Markov Decision Processes (MDPs), which can also be seen as direct optimal control [26, 27]. Classical RL methods, such as Q-learning, SARSA, and the actor-critic approach, estimate a state-action value (or a state value) for all state-action pairs (or states) and accordingly choose the actions that maximize this value. Its relationship with linear quadratic regulator and model predictive control (MPC) has been actively studied [28, 29, 30, 31].

Unlike RC, the understanding of the relationship between ILC and RL is still limited: Reference [32] compared the basic concepts without analysis of algorithms; sometimes ILC was misrepresented as a successful RL example for robotics, such as in Reference [33]. This may be partly due to the barrier between two different fields, control theory and machine learning. And the interaction of control theory and RL has been focused on RL and optimal control instead of RL and ILC.

To compare ILC and RL on the higher level, ILC uses previous data to eliminate the part of error that cannot be eliminated by model-based controllers while RL uses data to implicitly map the dynamics and finds an optimal policy; ILC solves trajectory tracking problems while RL can simultaneously solve trajectory tracking problems and trajectory planning problems; the strength of ILC lies in the use of a not-so-accurate model in order to obtain the monotonic decay of tracking errors while the advantage of RL stays in using data to find the solutions for stochastic/nonlinear optimization problems that are hard to be analytically solved. An interesting and open question is if and how they can mutually benefit by taking advantages

of both model and data.

1.3 Research Motivation: From Model-based to Data-driven

The motivation of this thesis is using data in ILC to relax the restrictions imposed by models. Model and data represent two sources of information that one can use for control. Models are approximations of reality while data from direct interaction with dynamic systems carries information beyond these approximations. The utilization of data therefore has the potential to release limitations due to approximation and push the performance boundaries of model-based control methods.

Actually, the development of ILC is a history of how to utilize model and data to improve the finite-time-trajectory tracking accuracy. One of the earliest ILC learning laws is the Arimoto learning law proposed in 1984 [2, 34], which takes the form of

$$u_{j+1}(t) = u_j(t) + \alpha e_j(t) \tag{1.1}$$

where the subscript j denotes the iteration number. This model-free learning law may suffer from bad learning transients that prevent the practical application of ILC [9, 35].

To avoid the bad learning transients in the practical application, the use of not-so-accurate models in ILC was intensely studied [9, 10, 11] and those model-based ILC methods can produce monotonic decay of the tracking errors.

Since the last decade, the study of model-based ILC has focused on two directions: (1)

practical control problems and theoretical challenges that are associated with practical application for linear ILC, and (2) nonlinear ILC [36]. This thesis summarizes the author's work to address problems in those two directions, which includes a series of new results of iterative learning control (ILC) that progresses from model-based ILC algorithms to data-driven ILC algorithms.

1.4 Thesis Organization

Chapter 1 is the introduction. The series of new results of ILC is presented from Chapter 2 to Chapter 5, which has been published in References [37, 38, 39, 40], followed by the Conclusions and Future work in Chapter 6. The reference list is at the end of this dissertation.

The organization from Chapter 2 to Chapter 4 not only follows the chronological order, but also fits the logical flow, from model-based to data-driven:

- Chapter 2 treats model-based ILC where the model is given a priori. It presents an improved method of ensuring ILC convergence in spite of errors in this model. Specifically, Chapter 2 presents a circulant filter to improve the robustness of model-based ILC addressing the issue of high frequency model error.
- In the ILC algorithms presented in Chapter 3 and Chapter 4, there exists a data-driven model development. Chapter 3 presents an indirect adaptive linear ILC method with an embedded observer to increase the execution speed of the pre-defined maneuver. This can be used to increase the productivity or to produce high tracking accuracy when the desired trajectory is too fast for feedback control to be effective.
- Chapter 4 shifts attention to making ILC algorithms for nonlinear systems approxi-

mated by bilinear models. The approach uses data to repeatedly create bilinear models in a homotopy approaching the desired trajectory. This requires that one be able to measure the full state, not just the tracked variable. Again there is a data driven model development in the algorithm, and this time it is much more complex.

- Chapter 5 finally uses model-free RL to eliminate the need for an a priori model. It presents a data-driven model-free ILC method, which uses an off-policy RL approach in the trial domain to produce the gains in the ILC learning law. It is data-driven in order to learn what must be known about the system dynamics, and it simultaneously addresses the ILC problem aiming for perfect tracking in the world.

The chapter of Conclusions and Future work briefly summarizes those achievements during the author's Ph.D. period, and proposes a research plan to investigate learning methods in the intersection of ILC and RL for nonlinear systems.

This page intentionally left blank

Chapter 2

Improvement of Robustness of ILC by Circulant Filter

2.1 Introduction

In engineering, models of physical systems usually fail to include some high-frequency dynamics that may be hidden in the noise. For instance, Reference [6, 41] reported an experiment on a robot at NASA Langley Research Center whose control systems had a bandwidth of 1.4 Hz and the Nyquist frequency was 200 Hz. Frequency response tests were unable to create a model above about 20 Hz, much smaller than 200 Hz. In spacecraft applications, residual vibration modes are always left out of the model. Control terminology talks about parasitic poles to describe unmodeled high frequency dynamics. It is therefore natural to use a low-pass cutoff filter to stop the learning process above some frequency.

Reference [6, 41] introduced the use of zero-phase low-pass filtering in ILC. The cutoff filter in ILC must be a zero-phase filter, otherwise phase distortions introduced by the

filter will prevent convergence to zero error even below the cutoff, as well as aggravate the convergence with extra phase error. One can convert an IIR filter into zero-phase by forward-backward filtering. The basic forward-backward procedure is as follows. First, a basic filter, like a Butterworth filter or a Chebyshev filter, is chosen. Then one filters forward through the data, then reverses the time sequence of the filtered data and filters again, and then reverses the twice-filtered data back to the original time sequence. The backward filtering brings in a phase lead that cancels the phase lag of the first forward filtering, while the attenuation above the cutoff is doubled.

There are difficulties with the use of forward-backward zero-phase filters [42]. The chosen filter is necessarily a difference equation which needs to have initial conditions specified, with associated transients. Various approaches are given in the literature to pick initial conditions to minimize the size of the transients, aiming to get closer to the desired steady state frequency response behavior more quickly. It is important to recognize that both the beginning and the end of the data have transients because of filtering starting from both directions. Initial conditions need to be specified at both ends. Reference [43] presents various methods to choose initial conditions. MATLAB supplies a filtfilt function for forward-backward zero-phase filtering and we demonstrate here that using this filtfilt can destabilize an otherwise stable ILC law. This instability issue stems from the mismatch between frequency response based filter design and finite time application.

In this chapter, we present a circulant zero-phase filtering approach for ILC that addresses this mismatch. The chosen basic filter, for example, a Butterworth filter, is converted into a circulant matrix form representation with the size of the number of total time steps needed. The singular value decomposition (SVD) of this matrix has precisely the magnitude response

in the singular values. The input singular vectors are sines and cosines of the frequencies that one can see in the given finite number of time steps. And the output singular vectors are precisely the sine and cosine responses including the phase change. The discrete Fourier transform (DFT) of this circulant matrix produces a diagonal matrix with the steady state frequency response on the diagonal. This filter matrix is hence precisely implementing the desired steady state frequency response and avoiding the complications brought by the mismatch. This circulant filtering is extending the finite data sequence into an infinite periodic sequence by repeating the original data. This extended sequence is likely to have discontinuities since the original sequence does not necessarily start and end at the same point. Gibbs phenomenon will happen and cause wiggles. A pre-reflection method to eliminate this Gibbs phenomenon is also proposed and demonstrated by numerical experiments.

In all, this circulant zero-phase filter is bridging the mismatch described above, and eliminating the difficulties associated with the initial conditions at both the beginning and the end, and avoiding the potential instability produced by initial condition issues in other forward-backward IIR approaches. Numerical examples demonstrate the effectiveness of this approach.

2.2 Discrete-Time Linear ILC

2.2.1 Formulation of Discrete-Time Linear ILC

In ILC, a digital system repeats a p time-step operation and returns to its initial condition after each iteration. The time starts being counted again when the system starts a new run.

Consider a single-input-single-output (SISO) state space system model

$$\begin{aligned} x_j(k+1) &= Ax_j(k) + Bu_j(k) \\ y_j(k) &= Cx_j(k) + Du_j(k) + d(k) \end{aligned} \tag{2.1}$$

where j denotes the iteration number, k is the time-step number, x_j is the state variable, u_j is the input command, y_j is the output, d is a repeating disturbance represented as an equivalent output disturbance. Matrices A, B, C, and D are state matrices. When $D \neq 0$, by defining the command history for iteration j as $\underline{u}_j = \begin{bmatrix} u_j(0) & u_j(1) & u_j(2) & \cdots & u_j(p-1) \end{bmatrix}^T$, the output history as $\underline{y}_j = \begin{bmatrix} y_j(0) & y_j(1) & y_j(2) & \cdots & y_j(p-1) \end{bmatrix}^T$, and the repeated disturbance as $\underline{d} = \begin{bmatrix} d(0) & d(1) & d(2) & \cdots & d(p-1) \end{bmatrix}^T$, the system dynamics can be presented by

$$\underline{y}_j = P\underline{u}_j + O\underline{x}(0) + \underline{d} \tag{2.2}$$

where $\underline{x}(0)$ denotes the initial condition; P is a Toeplitz matrix of the system Markov parameters representing the unit pulse response history; and O is the system observability matrix:

$$P = \begin{bmatrix} D & & & & \\ CB & D & & & \\ & CB & \ddots & & \\ & & \ddots & \ddots & \\ CA^{p-3}B & & & \ddots & \ddots \\ CA^{p-2}B & CA^{p-3}B & & CB & D \end{bmatrix} \quad \text{and} \quad O = \begin{bmatrix} C \\ CA \\ \vdots \\ CA^{p-2} \\ CA^{p-1} \end{bmatrix} \tag{2.3}$$

Given a desired p time-step maneuver, the error history is defined as $\underline{e}_j = \underline{y}_d - \underline{y}_j$ where $\underline{y}_d$ is the desired output. Moreover, SISO system is assumed and the formulas can be extended to MIMO systems.

The above formulation handles systems with $D \neq 0$. For a system sampled by a zero-order-hold (ZOH), it is common that there is one time-step delay between input and output, i.e., $D = 0$. In that case, the output history is redefined as $\underline{y}_j = \begin{bmatrix} y_j(1) & y_j(2) & \cdots & y_j(p) \end{bmatrix}^T$ while the command history remains the same, resulting in a P by shifting one-step upward, that is,

$$P = \begin{bmatrix} CB & & & & & \\ CAB & CB & & & & \\ & CAB & \ddots & & & \\ & & \ddots & \ddots & & \\ CA^{p-2}B & & & \ddots & \ddots & \\ CA^{p-1}B & CA^{p-2}B & & & CAB & CB \end{bmatrix} \tag{2.4}$$

When a low-pass filter is applied to an input (command) history update, one uses a non-zero D term. To model a differential equation fed by a zero-order hold, D is usually zero.

For linear ILC, a general form of an ILC law with a filter is

$$\underline{u}_{j+1} = F(\underline{u}_j + L\underline{e}_j) \tag{2.5}$$

where L is the learning matrix and F is a zero-phase low-pass cut-off filter, used to cut off the learning at high frequencies where substantial model error can destabilize the learning process. The purpose of this paper is to develop an improved design of the cut-off filter.

Reference [44, 45] have discussed and summarized various model-based learning laws and a few examples of L are

$$\begin{aligned} L_{inverse} &= \alpha \hat{P}^{\dagger} \\ L_{contractionMapping} &= \alpha \hat{P}^{T} \\ L_{partialIsometry} &= \alpha \hat{V}_{svd}^{T} \hat{U}_{svd} \end{aligned} \tag{2.6}$$

where α is a scalar, † denotes the pseudo-inverse, and $\hat{V}_{svd}$ and $\hat{U}_{svd}$ are the singular vectors of $\hat{P}$.

2.2.2 Stability Criterion and Monotonic Convergence Criterion

Stability and Monotonic Convergence Conditions

We can present approaches to studying the convergence of an ILC law. Combining Eqs. (2.2) and (2.5) and using the definition of error, one can write

$$\begin{bmatrix} \underline{e}_{j+1} \\ \underline{u}_{j+1} \end{bmatrix} = M \begin{bmatrix} \underline{e}_{j} \\ \underline{u}_{j} \end{bmatrix} + \begin{bmatrix} \underline{y}_d - O\underline{x}(0) - \underline{d} \\ 0 \end{bmatrix} \quad \text{where} \quad M = \begin{bmatrix} -PFL & -PF \\ FL & F \end{bmatrix} \tag{2.7}$$

Difference equation, Eq. (2.7), is asymptotically stable if and only if all eigenvalues are less than one in magnitude

$$\left| \lambda_i(M) \right| < 1, \quad \forall i \tag{2.8}$$

Reference [9] demonstrates that a simple ILC law that is asymptotically stable can have prohibitively bad learning transients from iteration to iteration. In simulations of a robot

control system, the iterations reaches a root-mean-square (RMS) error of 1.1991×10^{51} radians but eventually converges to zero. A necessary and sufficient condition for monotonic decay of the RMS error and command histories is that all singular values of the matrix M are less than one

$$\left|\sigma_i(M)\right| < 1, \quad \forall i \tag{2.9}$$

The second approach produces a decoupled difference equation for the command history, by substituting $\underline{e}_j = \underline{y}_d - \underline{y}_j$ and Eq. (2.2) into Eq. (2.5)

$$\underline{u}_{j+1} = F(I - LP)\underline{u}_j + FL(\underline{y}_d - O\underline{x}(0) - \underline{d}) \tag{2.10}$$

If $\underline{u}_j$ converges to some $\underline{u}_\infty$, then $\underline{e}_j$, given by $\underline{e}_j = -P\underline{u}_j + \underline{y}_d - O\underline{x}(0) - \underline{d}$ converges to some $\underline{e}_\infty$. Therefore, a necessary and sufficient condition for stability in ILC is that all eigenvalues of the matrix $F(I - LP)$ are less than one in magnitude

$$\left|\lambda_i(F(I - LP))\right| < 1, \quad \forall i \tag{2.11}$$

Monotonic convergence of the command history RMS results if and only if

$$\sigma_i(F(I - LP)) < 1, \quad \forall i \tag{2.12}$$

where σ_i is the ith singular value.

Equivalent stability criterion in DFT domain.

The normalized DFT of the j^{th} command history is given by $\underline{u}_{e,j} = H\underline{u}_j$ where the (i,j) component of H is $H(i,j) = (1/\sqrt{p})z_0^{-(i-1)(j-1)}$, $z_0 = e^{2\pi\hat{i}/p}$ and $\hat{i} = \sqrt{-1}$. Using this normalization $H^{-1} = H^{*T}$, where the superscript $\star$ denotes the complex conjugate and the superscript T denotes transpose. Stability is a property of the homogeneous equation $\underline{u}_{j+1} = F(I-LP)\underline{u}_j$. Then $\underline{u}_{e,j+1} = H\underline{u}_{j+1} = HF(I-LP)\underline{u}_j = HFH^{-1}[H(I-LP)H^{-1}]H\underline{u}_j = F_e(I-L_eP_e)\underline{u}_{e,j}$ where $F_e = HFH^{-1}$, $L_e = HLH^{-1}$, and $P_e = HPH^{-1}$, relate to the input frequency to output frequency relationships of F, L, and P.

The $\underline{u}_{e,j}$ converges for all initial control histories, if and only if $\underline{u}_j$ converges because H is a full-rank and invertible matrix. Therefore

$$\left|\lambda_i(F_e(I-L_eP_e))\right| < 1, \quad \forall i \tag{2.13}$$

is a frequency based stability condition equivalent to Eq. (2.11).

Note that ${\underline{u}^*_{e,j}}^T\underline{u}_{e,j} = (H\underline{u}_j)^{*T}(H\underline{u}_j) = \underline{u}_j^T H^{*T}H\underline{u}_j = \underline{u}_j^T\underline{u}_j$. Therefore the RMS error of $\underline{u}_{e,j}$ decreases monotonically if and only if the RMS error of $\underline{u}_j$ decreases monotonically. Hence, one can write Eq. (2.12) in frequency form

$$\left|\sigma_i(F_e(I-L_eP_e))\right| < 1, \quad \forall i \tag{2.14}$$

where σ_i is the i^{th} singular value.

2.3 ILC Necessitates Zero-Phase Filters for Robustification

Iterative Learning Control initially asks for zero error at every time step. For a long enough trajectory that one can think in terms of steady state frequency response, this means one asks for zero error at every frequency component up to Nyquist frequency. When one makes a model of a physical system one must expect that the model deteriorates at high frequencies, with the model missing some high frequency residual modes or so-called parasitic poles, whose dynamics can be hidden in the noise level. If such missing high frequency dynamics is sufficiently badly modeled to have a 180 degree phase difference between the model and real world behavior at these frequencies, then the ILC law will add to the error rather than decrease the error. The ILC action will amplify the the error pulling it out of the noise level, instead of attenuating the error. Hence, for robustification to high frequency model error, it is standard practice to include the zero-phase low-pass filter F to stop the learning process for components of the error above some frequency [6, 41, 43]. Usually, the frequency cutoff is determined in hardware tests, because one does not know in advance at what frequency the model error becomes too large for convergence. The filter must be zero-phase, otherwise it distorts the signal in the pass band, making the ILC aim for zero error following the distorted signal.

Starting from an IIR filter such as a Butterworth low-pass filter, one can make it into a zero-phase filter by the following procedure. First filter the signal forward in time. Reverse the time in the resulting filtered history vector and filter again. Then reverse the time again. The second filtering in reverse time cancels the phase change produced by the initial forward

application of the filter, and doubles the attenuation. Using Toeplitz matrix representations of the filter forward and backward in time, one can write the result $\underline{y}$ of forward-backward filtering a vector $\underline{x}$ as follows

$$\begin{aligned}
\underline{y} &= J\{P_b[J(P_f\underline{x} + O_f\underline{x}_f(0)) + O_b\underline{x}_b(0)]\} \\
&= P_b^T P_f\underline{x} + JP_bJO_f\underline{x}_f(0) + JP_bO_b\underline{x}_b(0) \\
&= [P_b^T P_f + JP_bJO_f\mathcal{T}_1(\underline{x}) + JP_bO_b\mathcal{T}_2(\underline{x})]\underline{x} \\
&= F\underline{x}
\end{aligned} \qquad \text{where} \quad J = \begin{bmatrix} & & & 1 \\ & & 1 & \\ & 1 & & \\ \iddots & & & \\ 1 & & & \end{bmatrix} \tag{2.15}$$

where $\underline{x}$ is the signal to be filtered; $\underline{y}$ is the filtered signal; P_f, $\underline{x}_f(0)$, and O_f are correspondingly the Toeplitz matrix, the initial condition and the observability matrix for the forward filter; and similarly P_b, $\underline{x}_b(0)$, and O_b are for the backward filter. Matrix J is a matrix to invert the time history. Matrix $P_f = P_b$ to achieve zero-phase.

Matrices P_f and P_b come from the convolution sum solution of the filter difference equation and hence they produce zero initial condition filter solutions. The steady state response to sinusoids will not satisfy all zero initial conditions. Since frequency response is a steady state response concept, the initial condition effects in the convolution sum solution contaminate the filtered results within a settling time both at the beginning of the trajectory for the forward filtering and the end of the trajectory for the backward filtering. If one could know what initial conditions $\underline{x}_f(0)$ and $\underline{x}_b(0)$ would be on the steady state response, the contamination could be eliminated.

When the state variables in the state space model are defined from an autoregressive ex-

ogenous (ARX) model, then the initial conditions are the outputs at a number of preceding time steps. This choice of initial conditions can be thought of as making a finite extension in the desired trajectory. The extensions can be longer than the number of steps needed for initial conditions. The literature contains various approaches to making these finite extensions: 1. constant-padding or simply repeating the initial value [46, 42], 2. reflection, e.g. an odd reflection around the initial value [46, 42], and 3. minimization of the difference between forward-backward filtering and backward-forward filtering [47]. In methods 1 and 2 the $\underline{x}_f(0)$ and $\underline{x}_b(0)$ can be written as a linear function of $\underline{x}$ and hence there is a matrix F such that $\underline{y} = F\underline{x}$,

$$F = P_b^T P_f + JP_bJO_f\mathcal{T}_1(\underline{x}) + JP_bO_b\mathcal{T}_2(\underline{x}) \tag{2.16}$$

where now F represents the forward-backward filter including initial conditions. An earlier version of MATLAB filtfilt function used method 2 and Reference 8 shows that using this in ILC can produce instability in an otherwise stable learning law. The current version of filtfilt uses method 3 above which is based on Reference [47]. Equation 4 in that reference establishes that the initial conditions $\underline{x}_f(0)$ and $\underline{x}_b(0)$ are again a linear function of $\underline{x}$. Hence there is again a filter matrix F including the initial conditions for the current MATLAB algorithm. In the next section, we will show that this filtfilt can also produce instability in the ILC context. If method 1 uses zero initial conditions, then the zero-phase filter reduces to $P_f^T P_f$. Provided the singular values of $(I - LP)$ are less than 1, then introducing a zero-phase filter $P_f^T P_f$ with singular values less than 1, will preserve stability. But the filtered results contain transient contamination. We will develop a circulant filter approach which eliminates contamination from filter transients and when introduced into an ILC law, preserves its stability.

2.4 Example of Instability in ILC Produced by Filtfilt Zero-Phase Low-Pass Filtering

Consider the following transfer function

$$G(s) = \left(\frac{a}{s+a}\right)\left(\frac{\omega_0^2}{s^2 + 2\zeta\omega_0 s + \omega_0^2}\right) \tag{2.17}$$

which is a reasonably good model of the command to response of the control systems for each link of a Robotics Research Corporation robot [6]. The constants are $a = 8.8$, $\zeta = 0.5$ and $\omega_0 = 37$. The DC gain for a constant input is 1. The constants are $a = 8.8$, $\zeta = 0.5$ and $\omega_0 = 37$. The DC gain is 1. This continuous system is fed by a zero-order hold running at 200 Hz (sampling period $T = 0.005$ s). The total number of time steps in one iteration is 200. A contraction mapping law $L = P^T$ is applied for learning, where P is the system Toeplitz matrix. This learning law guarantees monotonic convergence without any filter. The singular values σ_i of P are all less than 1 and the singular values $I - PL$ satisfy $|1 - \sigma_2^2| < 1$ smaller than 1.

Identifying the DFT Matrix of MATLAB Filtfilt

Instead of developing an analytical expression for matrix F representing the MATLAB filtfilt algorithm, we identify the DFT matrix F_e from input-output data produced by the code. This approach could be used to study other linear extension methods as well. We apply the filtfilt algorithm to sine and cosine signals of each frequency, and from the input-output results, we identify the matrix F_e. Then using the stability condition in the frequency based form

Eq. (2.14) will establish that filtfilt can destabilize the otherwise stable iterative learning process.

Consider an even number p of time steps in a trajectory, which means one can observe Nyquist frequency in the data. Frequencies that can be seen are DC and f_1, f_2, $\cdots$, f_N where f_1 is the fundamental frequency, f_2 is the first harmonic, f_N is the Nyquist frequency, and $N = p/2$. The (m, n) component of F_e is denoted by $F_e(m, n)$.

The process is as follows:

1. For the first column in F_e corresponding to DC, use input as

$$\underline{u}_0 = \begin{bmatrix} \frac{1}{\sqrt{p}} & \frac{1}{\sqrt{p}} & \cdots & \frac{1}{\sqrt{p}} \end{bmatrix}^T \tag{2.18}$$

   and the DFT of filtfilt output is vector $Y_0 = H\underline{y}_0$. The first column is $F_e(m, 1) = Y_0(m)$.

2. For column $N + 1$ corresponding to the Nyquist frequency, the input is

$$\underline{u}_N(k) = \frac{1}{\sqrt{p}} \cos(2\pi f_N kT) \tag{2.19}$$

   and $F_e(m, N + 1) = Y_N(m)$ where vector $Y_N(m)$ is the DFT of the filtfilt output.

3. The inputs are chosen as $\underline{\hat{u}}_{s,n}$ and $\underline{\hat{u}}_{c,n}$, $i = 1, 2, 3, \cdots, N-1$, produced by normalizing to unit length $\underline{u}_{s,n}(k) = \sin(2\pi f_i kT)$ and $\underline{u}_{c,n}(k) = \cos(2\pi f_i kT)$, $k = 1, 2, 3, \cdots, p$.

4. Do filtfilt in the time domain giving p-component output vectors $\underline{\hat{y}}_{s,n}$ and $\underline{\hat{y}}_{c,n}$ and their DFT vectors are $Y_{s,n} = H\underline{\hat{y}}_{s,n}$ and $Y_{c,n} = H\underline{\hat{y}}_{c,n}$.

5. The components from the second column to column N , i.e. the left half of the

matrix without the first column, are $F_e(m, n+1) = Y_{c,n}(m) + \hat{i}Y_{s,n}(m)$, where $m = 1, 2, 3, \cdots, p$ and $n = 1, 2, 3, \cdots, N-1$. The components of the right half of the matrix without column $N+1$, i.e. from column $N+2$ to the last column, are $F_e(m, p-n+1) = Y_{c,n}(m) - \hat{i}Y_{s,n}(m)$.

This method is applied to estimate the DFT matrix F_e of Filtfilt based on a 6^{th} order MATLAB digital Butterworth filter with a cut-off frequency of 10 Hz providing a sampling frequency 200 Hz. To test the accuracy of F_e, the filtered results of the estimated F_e and the real filtfilt from the same original signals are compared. The procedure to filter a signal through F_e is as follows: do DFT of the to-be-filtered signal, multiply the DFT of the signal by F_e, and do inverse DFT of the product producing the filtered signal in time domain. An example comparing the filtered outputs by these two models of filtfilt is demonstrated in Figure 2.1(a). The dotted line plots the input white Gaussian signal. The dot-dash line and the dashed line are on top of each other, so the F_e model matches the filtfilt algorithm. To closer examine the accuracy of F_e, four white Gaussian signals randomly produced by MATLAB are used as inputs for both filter models and the output differences caused by the error of the estimated P_e are plotted in Figure 2.1(b). This figure shows the F_e produced by this method matches the filtfilt algorithm with 12 digits accuracy. In later ILC simulations, the error reaches a minimum around 4 digits and hence this F_e is accurate enough for stability analysis.

2.4.1 Stability Analysis of an ILC System Implemented with Filtfilt

Filtfilt F_e matrices are identified for cut-off frequencies from 1 Hz to 99 Hz in increments of 1 Hz. Then Toeplitz matrix P is computed for system Eq. (2.17) and the DFT version $P_e = HPH^{-1}$, which makes the learning law $L_e = P_e^T$. We study asymptotic stability condition Eq. (2.14). Figure 2.4 gives the maximum magnitude of eigenvalues, the spectral radius of $F_e(I - L_eP_e)$ as a function of cut-off frequency. The spectral radius goes above 1 between 8 and 20 Hz with a maximum value of 1.05755 at a 10 Hz, violating the stability condition. Closer examination reveals that it is also violated for higher cut-off frequencies between 25 Hz to 40 Hz and 76 Hz to 99 Hz. Thus, introducing a filtfilt zero-phase low-pass cut-off filter into a learning control law, for the purpose of robustifying the law to high-frequency errors, can produce its own instabilities.

2.4.2 Demonstration of Instability of the Learning Process using Filtfilt

Consider four different trajectories illustrated in Figure 2.5: *Trajectory 1:* When t is less than $t_{max} = 0.5pT$, it follows y_{d1}. When t is bigger than t_{max}, it returns to the initial position along the mirror-reflection of the incoming trajectory. The trajectory is continuous but has a discontinuous first derivative at the mid-point. *Trajectory 2:* When t is less than $t_{max} = 0.5pT$, it follows y_{d2}. When t is bigger than t_{max}, it returns to the initial position along the mirror-reflection of the incoming trajectory. This trajectory is continuous with a continuous first derivative. *Trajectory 3:* It is periodic as y_{d3} in Eq. (2.20). All derivatives of this trajectory are continuous, but if the system is at rest before zero, then it has a discontinuity in the acceleration at zero. *Trajectory 4:* It follows y_{d4} while $t_{max} = pT$. Hence it starts at zero with

zero velocity and non-zero acceleration and ends at different position.

$$\begin{aligned} y_{d1} &= \frac{t}{t_{max}} & y_{d2} &= \frac{3}{2t_{max}^2}t^2 - \frac{1}{2t_{max}^3}t^3 \\ y_{d3} &= 0.5 - 0.5cos(\frac{6\pi}{pT}t) & y_{d4} &= \frac{3}{2t_{max}^2}t^2 - \frac{1}{2t_{max}^3}t^3 \end{aligned} \tag{2.20}$$

A contraction mapping law $L = P^T$ is applied to the system Eq. (2.17) fed by a zero-order hold using filtfilt with a cut-off frequency of 10 Hz. Figure 2.2 plots the error history of the 200th iteration for each trajectory. The errors for all plots lie on top of each other except for the filtfilt error history, which in the case of trajectory 3 can reach 300 times the size of the desired trajectory. Figure 2.3 gives the RMS error history for each trajectory. The dotted curve corresponding to ILC learning without a filter converges in each case and the ILC law with filtfilt cutoff at 10 Hz diverges exponentially, linearly when seen on a log plot, in all cases.

2.4.3 Comments on the Stability Issues

We make several comments summarizing issues in evaluating stability of ILC systems with zero-phase low-pass filtering. If the ILC without the zero-phase filter satisfies the monotonic decay condition Eq. (2.12), so that $(I - LP)$ has all singular values less than unity, then a zero-phase filter design F has all singular values less than unity can be used without destabilizing the originally stable law. However, zero-phase filters normally have the initial condition terms $\underline{x}_f(0)$ and $\underline{x}_b(0)$. One must create an expression that incorporates these terms into the coefficient matrix F, making $\underline{y} = F\underline{x}$, before one can apply stability condition Eq. (2.8). Reference [42] showed that it is possible for extensions to create the situation where

the filtered error has the opposite sign to that of the actual error, and that this can produce instability. This instability is related to the initial and final transient portions of the filtered trajectory, and is not observed based solely on the $P^T P$ term. In addition, the objective of the filter design is to make a cutoff that is based exclusively on frequency. Reference [48] shows that when p is large the singular value decomposition (SVD) of P converges to the frequency response, but for a finite p, the singular vectors can be a mix of more than one frequency. We seek a method to address both of these issues.

2.5 A Circulant Zero-Phase Filter Can Produce Steady State Response From Finite-Time Input Signals

2.5.1 The Circulant Form of a Toeplitz Matrix of Markov Parameters

We investigate the use of the circulant form of a low-pass filter to design zero-phase filters for ILC. Hence D is nonzero. A circulant matrix is a special kind of Toeplitz matrix where the first column consists of the Markov parameters and other columns follow the pattern that each is rotated one element down relative to the preceding column while the element at the bottom goes to the top. For example, the system Toeplitz matrix with D term is Eq. (2.3) and its asymptotically equivalent circulant matrix is

$$\hat{P} = \begin{bmatrix} D & CA^{p-2}B & & & CAB & CB \\ CB & D & CA^{p-2}B & & & CAB \\ & CB & D & \ddots & & \\ & & \ddots & \ddots & \ddots & \\ CA^{p-3}B & & & \ddots & \ddots & CA^{p-2}B \\ CA^{p-2}B & CA^{p-3}B & & & CB & D \end{bmatrix} \tag{2.21}$$

For SISO systems, CA^rB becomes a scalar, i.e., the Markov parameter, and both Toeplitz P and circulant $\hat{P}$ can be written into a linear combination of some basic matrices multiplied by the Markov parameters as follows,

$$\begin{aligned} P &= DI + (CB)R + \cdots + (CA^rB)R^{r+1} + \cdots + (CA^{p-2}B)R^{p-1} \\ \hat{P} &= DI + (CB)\hat{R}_1 + \cdots + (CA^rB)\hat{R}_{r+1} + \cdots + (CA^{p-2}B)\hat{R}_{p-1} \end{aligned} \tag{2.22}$$

where

$$R = \begin{bmatrix} 0 & & & & \\ 1 & 0 & & & \\ & 1 & \ddots & & \\ & & \ddots & \ddots & \\ & & & 1 & 0 \end{bmatrix} \tag{2.23}$$

$$\hat{R}_1 = \begin{bmatrix} & & & 1 \\ 1 & & & \\ & 1 & & \\ & & \ddots & \\ & & & 1 \end{bmatrix} \tag{2.24}$$

$$(\hat{R}_r)_{i,j} = \begin{cases} 1 & \text{when } i-j=r \\ 1 & \text{when } i-j=r-p \\ 0 & \text{otherwise} \end{cases} \tag{2.25}$$

In equations above, R is a p by p matrix with 1's in the first sub-diagonal and 0's otherwise and $\hat{R}_r$ is a cyclic permutation p by p matrix with ones on the rth subdiagonal and $(p-r)$th super diagonal (all zeros elsewhere), where $i, j = 1, 2, 3, \cdots, p$ and $r = 1, 2, 3, \cdots, p-1$. The cyclic matrix $\hat{R}_r$ is the circulant form of the matrix R^r. For instance, $\hat{R}_1$ is the circulant form of R.

2.5.2 The Circulant Matrix of Markov Parameters is the Matrix Form of the Filter's Steady State Frequency Response

Reference [48] shows that SVD of the lower triangular Toeplitz matrix of Markov parameters asymptotically converges to the steady state frequency response as p tends to infinity.

The magnitude response is given by the singular values and the phase change is indicated by the input and output singular vectors. For a finite p, there is coupling between frequencies. This coupling is studied in a later section. As discussed below Eq. (2.15), the lower triangular Toeplitz matrix is associated with zero initial conditions, and hence includes a transient period before reaching steady state response. By studying the DFT form and the SVD of a circulant matrix form, this section shows that the steady state response can be obtained from this circulant matrix, provided the number of time steps in the matrix is larger than that of the settling time.

Calculate the DFT of $\hat{P}$ using Eq. (2.22),

$$\begin{aligned}\hat{P}_e &= H\hat{P}H^{-1} \\ &= DI + (CB)H\hat{R}_1H^{-1} + \cdots + (CA^{r-1}B)H\hat{R}_rH^{-1} + \cdots + (CA^{p-2}B)H\hat{R}_{p-1}H^{-1}\end{aligned} \tag{2.26}$$

and

$$H\hat{R}_rH^{-1} = diag(1, z_0^{-r}, z_0^{-2r}, \cdots, z_0^{-(p-1)r}) \tag{2.27}$$

Using the formula for the sum of a geometric series produces the (i,j) component of $\hat{P}_e$

$$\hat{P}_e(i,j) = \begin{cases} D + C(z_0^{j-1}I - A)^{-1}(I - A^{p-1}z_0^{j-1})B & \text{when } i = j \\ \\ 0 & \text{otherwise} \end{cases} \tag{2.28}$$

If A^{p-1} is negligible, then the diagonal elements of $\hat{P}_e$ corresponds to the z-transfer function

evaluated at the discrete frequencies that can be seen in p time steps,

$$[D + C(zI - A)^{-1}B]|_{z=z_0^{j-1}} \tag{2.29}$$

The same conclusion is obtained for MIMO systems by calculating the components of $\hat{P}H^{-1}$ first and then computing the matrix product $H\hat{P}H^{-1}$. The i^{th} row of $\hat{P}$ (except $i = 1$) is

$$\begin{bmatrix} CA^{i-2}B & CA^{i-3}B & \cdots & CB & D & CA^{p-2}B & CA^{p-3}B & \cdots & CA^{i-1}B \end{bmatrix} \tag{2.30}$$

Thus the i^{th} row components of $\hat{P}\hat{H}^{-1}$, where $i = 1, 2, 3, \cdots, p$, are

$$(\hat{P}H^{-1})_{i,j} = \frac{1}{\sqrt{p}} z_0^{(i-1)(j-1)} \left[D + C(z_0^{j-1}I - A)^{-1}(I - A^{p-1}z_0^{j-1})B\right] \tag{2.31}$$

Therefore, the product of $H\hat{P}H^{-1}$ can be computed resulting in Eq. (2.28).

The diagonal components of $\hat{P}_e$ contains the magnitude response σ_j and the phase response θ_j of the system where j indicates the discrete frequency. An SVD of the diagonal complex matrix $\hat{P}_e$ can be written as

$$\hat{P}_e = U_e S_e V_e^{*T} = \begin{bmatrix} e^{\hat{i}\theta_1} & & & \\ & e^{\hat{i}\theta_2} & & \\ & & \ddots & \\ & & & e^{\hat{i}\theta_p} \end{bmatrix} \begin{bmatrix} \sigma_1 & & & \\ & \sigma_2 & & \\ & & \ddots & \\ & & & \sigma_p \end{bmatrix} \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & \ddots & \\ & & & 1 \end{bmatrix} \tag{2.32}$$

The output singular vector matrix V_e gives the phase shifting when setting the input singular

vector matrix U_e to the identity matrix. Note that the SVD expression is not unique, because any $e^{\hat{i}\alpha}$ can be multiplied to each singular vector pair. Another feature is that the singular values come in pairs because z_0^{j-1} and z_0^{p-j+1} refer to the same frequency.

The DFT version of a matrix has the same singular values as the matrix itself, e.g.

$$(\hat{P}_e)^{*T}\hat{P}_e = (H\hat{P}H^{-1})^{*T}(H\hat{P}H^{-1}) = H\hat{P}^T H^{-1} H\hat{P}H^{-1} = H\hat{P}^T\hat{P}H^{-1} \tag{2.33}$$

the eigenvalues of $\hat{P}^T\hat{P}$ and $(\hat{P}_e)^{*T}\hat{P}_e$ are the same, and therefore the singular values of $\hat{P}_e$ and P are the same, which come in pairs presenting the magnitude response. The complex singular vector pairs of $\hat{P}$ are $H^{-1}U_e$ and $H^{-1}V_e$. One can make these vector pairs real by some rotations $e^{\hat{i}\alpha_j}$. Numerical experience indicates that MATLAB produces input singular vectors which are sine or cosine waves of the corresponding frequencies, and output singular vectors are the same sine or cosine waves with phase shifts equal to the phase response.

Figure 2.7 illustrates the relationship between the system magnitude frequency response of system Eq. (2.17) fed by a zero order hold sampling at 200 Hz, the singular values of a 200 time-step lower triangular P matrix and the singular values of the corresponding circulant P_e matrix. The DC gain of the system is unity, and the first singular value of P_e corresponding to DC gives this steady state value. The first singular value of P is smaller because the output of this matrix contains the build up of the response to a DC input when the system starts with zero initial conditions. The singular values of P_e come in pairs, two identical magnitude response values for the sine and cosine inputs at that frequency, and they lie on the steady state frequency response line. The singular values of the Toeplitz matrix P however, are all distinct. Reference [48] performs a frequency analysis of each input singular vector. Every

other singular vector input is observed to have a nearly pure frequency equal to a frequency that can be observed, and the associated singular value is the frequency response. The singular values between these have singular vectors that mix the frequency with its neighbor frequency, and are plotted half way between. Table 2.1 compares the singular values of P_e and the steady state frequency response for the first 5 frequencies visible in 200 time steps, and the values match to within 0.01%.

Table 2.1: Magnitude response (SV) and singular values (SV) at DC and first 5 frequencies

Frequency	0 Hz	1 Hz		2 Hz		3 Hz		4 Hz		5 Hz	
MR	1.0000	0.8255		0.6052		0.4705		0.3810		0.3015	
SV	0.9998	0.8253	0.8253	0.6051	0.6051	0.4704	0.4704	0.3809	0.3809	0.3015	0.3015

Singular vectors associated with the second harmonic frequency 3 Hz are displayed in Figure 2.7(b). The singular vector of the Toeplitz matrix is plotted as the dot-dash line, whose shape is a distorted sinusoidal wave, whose distortion is obviously at the end of the curve. The solid line corresponds to the left (or input) singular vector and the dashed line to the right (or the output). Computed from sampled system transfer function, the phase response at 3 Hz is around 102° or 1.7838 rad. Through curve fitting in Matlab, the left singular vector fits the sine wave $0.1\sin(6\pi t + 1.2^{-17})$ with a unity R-square and root mean squared error of 5.36^{-9}, where each row represents each time step starting from zero to the end of the trajectory. The curve fitting result of the right singular vector is $0.1\sin(6\pi t - 1.784)$ with a unity R-square and root mean squared error of 4.72^{-10}, which has a phase shift equal to 1.784 rad, the phase response at 3 Hz. Hence the SVD of the circulant matrix made of system Markov parameters can serve as an alternative to the transfer function for the frequency response computation. The circulant matrix comprised of system Markov parameters is actually the matrix form of the system frequency response.

2.5.3 Numerical Simulations Demonstrate the Steady State Behavior of a Circulant Zero-Phase Butterworth Filter

Toeplitz matrix P_f of an IIR filter is causal while the circulant form $\hat{P}_f$ is non-causal. Because of the relationship between a circulant matrix and the corresponding system frequency response, this non-causal filtering process produces a steady state behavior for a finite data sequence. That is, a single frequency input going through a circulant filter leads to an output signal of the same frequency with magnitude and phase adjustment according to the corresponding frequency response. Therefore, it avoids the complexities caused by initial conditions. In other word, $\underline{y} = \hat{P}_f \underline{x}$, where $\underline{x}$ is the data to be filtered and $\underline{y}$ is the filtered output. Actually, a circulant filter automatically extends the to-be-filtered data into an infinite data sequence by periodically repeating the original data.

The zero-phase requirement can be satisfied by forward-backward filtering for circulant filters as well. The general procedure of circulant filter design is: 1. design a basic IIR filter sampled the same way as the ILC system; 2. convert the transfer function into a state space model (A, B, C and D); 3. write down the circulant matrix $\hat{P}_f$, the first column of which is comprised of $D,\ CB,\ CAB, \cdots,\ CA^{p-2}B$;and 4. the circulant zero-phase filter by forward-backward filtering, $F_{fb} = \hat{P}_f^T \hat{P}_f$.

A circulant forward-backward Butterworth filter is numerically tested by basic sine and cosine waves and the result is compared with that from a filtfilt (forward-backward) Butterworth filter. The basic Butterworth filter is chosen as the same in the previous simulation for filtfilt (6th order and 10 Hz cutoff). The sampling frequency and total time steps of a trajectory are also the same (200 Hz and 200 time steps). The filtered sine and cosine signals are

sinusoids without phase shift. The filtfilt Butterworth filter distorts two ends due to transients, while it keeps the middle part of filtered signal as expected.

The simulation results are displayed in Figure 2.6. Here we name the ratio of the 2-norm of filtered data over that of the original single-frequency data as the "magnitude response", regardless of the shape of the output signal. Figure 2.6(a) compares the "magnitude responses" of circulant filter and filtfilt with the designed magnitude response, which is double that of the basic 6th other Butterworth filter. The circle denotes the circulant and all these circles lie on the dashed line which plots the designed magnitude response. The stars for filtfilt obviously deviate from the dash line in the stop- band. Closer examination shows that those stars in passband also lie a little bit beyond the dashed line. This deviation is caused by the distortion at both ends of the data sequence. Figure 2.6(b) presents the outputs of a 25 Hz sine wave, which belongs to the stop-band. The output of the circulant filter (solid line) stays at zero, which implies the high frequency signal is successfully stopped. Wiggles at both ends appear in the dot-dash line for filtfilt. These wiggles are responsible for the instabilities brought into ILC systems. Therefore, this simulation demonstrates that the circulant forward-backward Butterworth filter produces steady state behavior in the DFT domain for finite-time implementation.

2.6 The Stability and the Final Tracking Error after Introducing a Circulant Zero-Phase Filter

2.6.1 Introducing a Circulant Forward-Backward Filter Can Preserve the Monotonic Decay of an ILC System

Consider an ILC law without a filter, that satisfied the condition for monotonic decay of the 2- norm of the command history. This means that the maximum singular value of $(I - L_e P_e)$ is less than unity. Now consider any circulant zero-phase low-pass filter F_e having maximum singular value equal to unity or smaller. This can be the result of using a Butterworth filter to form the zero phase filter. Note that a typical Chebyshev filter can produce some amplification and does not satisfy this requirement. Then the singular value of the product $F_e(I - L_e P_e)$ must also be less than unity. Hence, we conclude that use of a circulant zero-phase low-pass filter will not destabilize an otherwise stable ILC law, provided the filter maximum singular value is one or less. Note that this process eliminates the need to analyze the effects of the initial condition terms in other zero phase filters. These terms have been eliminated, and the associated transient corruptions of the filter results at the beginning and the end of the trajectory being filtered have been eliminated.

2.6.2 Reflecting Data to Prevent Discontinuities that Cause the Gibbs Phenomenon

Since the underlying algorithm of circulant filters is periodically repeating the original data into an infinite sequence, discontinuities exist in the extended infinite sequence unless the

original signal starts and ends at the same point. These discontinuities lead to the Gibbs phenomenon when filter- ing. This will produce a deviation of the filtered desired trajectory from the real desired trajectory approaching the end points. Pre-reflection can be applied to decrease the deviation. That is, before filtering, the data is extended twice of the original length by reflecting the signal. After filtering, extract the first half as the final filtered result. The equivalent circulant filter with reflection is

$$\hat{F}_{cr} = [I \;\; 0]\hat{F}_c[I \;\; J^T]^T \tag{2.34}$$

This pre-filtering reflection eliminates jump discontinuities in the extended trajectory.

As for the circulant filtering with pre-reflection, it is not easy to write an explicit formula about the eigenvalues. We can numerically calculate the eigenvalues of $\hat{F}_{cr}(I - LP)$ to check the stability. The ILC system in previous simulations is also tested with pre-reflection circulant filtering. All parameters remain the same as before. The eigenvalues of $\hat{F}_{cr}(I-LP)$ are presented as the circles in Figure 2.4. These circles stay below the unity line for all cut-off frequencies.

2.6.3 Simulations of ILC System with Zero-Phase Circulant Butterworth Filtering

The same numerical simulation for filtfilt is repeated for circulant forward-backward Butterworth filter with or without pre-reflection. In Figure 2.2, the learning processes without a filter (black dot line), with a typical circulant filter (red dash line) and with a pre-reflected circulant filter (magenta solid line) lead to "zero" error at the 200^{th} run, tracking each trajec-

tory. No difference is observed because of tracking error in the learning process with filtfilt is bigger than 1000 times of others' error. Figure 2.3 about the RMS of the tracking error in log scale versus iterations unveils the difference between circulant filtering with and without pre-reflection. The magenta solid lines are below the red dash lines at the quasi-steady state in all four subfigures. (The quasi-steady state refers to the time when the learning process slowly improves the tracking, that is, the plot of RMS of tracking error seems staying horizontally.) Hence the pre-reflection successfully decreases the final tracking error inevitably introduced by filtering out high frequency components in desired finite trajectory. The greatest improvement happens at tracking the 4^{th} trajectory, the ends of which are mismatched, implying an extended infinite discontinuous curve. Comparing all lines including filtfilt, circulant without pre-reflection and circulant with reflection, two conclusions about this simulation can be achieved: 1. circulant filters can avoid the instability issue that filtfilt introduced; and 2. pre-reflection can decrease the tracking error between filtered desired trajectory and the real desired trajectory and the amount of improvement depends on the smoothness of the extended infinite trajectory.

The underlying algorithm of circulant filters can be understood as an infinite wraparound extensions, that is, periodically repeating the original trajectory into an infinite data sequence. Typical circulant forward-backward filters remove the high frequency components of this infinite signal. And pre-reflection first extends the p-step trajectory into double length by reflection and then periodically repeats this $2p$-step trajectory into an infinite sequence. The circulant filter than stops the high frequency components in this new infinite sequence. Both circulant filtering algorithms avoid transients as well as the need for choosing initial condition choices.

2.7 The Relationship Between Toeplitz Matrix of Markov parameters and System Frequency Response

As mentioned before, Reference [48] showed that SVD of the lower triangular Toeplitz matrix approaches the steady state frequency response as the dimension of the matrix tends to infinity. This section analyzes the deviation Δ of the DFT of a Toeplitz matrix, $P_e \triangleq DFT(P)$ from the steady state frequency response given by the DFT of the circulant matrix $\hat{P}_e \triangleq DFT(\hat{P})$.

2.7.1 A Formula for the Deviation from the Frequency Response

A trajectory longer than the settling time is assumed so that A^{p-1} can be set to zero. The deviation Δ is then given by

$$P_e = HPH^{-1} = \hat{P}_e - \Delta \tag{2.35}$$

where

$$\Delta_{i,j} = \frac{1}{p} C(I - z_0^{-(i-1)} A)^{-1} (z_0^{j-1} I - A)^{-1} B \tag{2.36}$$

This DFT matrix is no longer diagonal and hence there is cross-talk between different frequencies, that is, a single frequency input produces multiple frequencies out.

The calculation of P_e is as follows: First, the sum of a geometric series that is used in later computation is

$$\sum_{k=1}^{p-j} z_0^{-(i-1)k} A^{k-1} = (z_0^{i-1} I - A)^{-1} (I - z_0^{(i-1)j} A^{p-j}) \tag{2.37}$$

The (k, j) component of P is

$$P(k,j) = \begin{cases} 0 & k < j \\ D & k = j \\ CA^{k-j-1}B & k > j \end{cases} \tag{2.38}$$

Therefore the (i, j) produce of HP is

$$\begin{aligned} (HP)(i,j) &= \sum_{k=1}^{p} H(i,k)P(k,j) \\ &= \frac{1}{\sqrt{p}}\big(z_0^{-(i-1)(j-1)}D + \sum_{k=j+1}^{p} z_0^{-(i-1)(k-1)}CA^{k-j-1}B\big) \\ &= \frac{1}{\sqrt{p}}\big[z_0^{-(i-1)(j-1)}D + z_0^{-(i-1)(j-1)}C(\sum_{k=1}^{p-j} z_0^{-(i-1)k}A^{k-1})B\big] \end{aligned} \tag{2.39}$$

Plug Eq. (2.37) into Eq. (2.39)

$$(HP)(i,j) = \frac{1}{\sqrt{p}}z_0^{-(i-1)(j-1)}\big[D + C(z_0^{i-1}I - A)^{-1}B\big] - \frac{1}{\sqrt{p}}C(I - z_0^{-(i-1)}A)^{-1}A^{p-j}B \tag{2.40}$$

The (i, k) components of $P_e = HPH^{-1}$ becomes

$$\begin{aligned} P_e(i,k) &= \sum_{k=j+1}^{p} (HP)(i,j)H^{-1}(j,k) \\ &= \delta_{i,k}\big[D + C(z_0^{i-1}I - A)^{-1}B\big] + \frac{1}{p}C(I - z_0^{-(i-1)}A)^{-1}\sum_{j=1}^{p}(z_0^{(j-1)(k-1)}A^{p-j})B \\ &= \delta_{i,k}\big[D + C(z_0^{i-1}I - A)^{-1}B\big] + \frac{1}{p}C(I - z_0^{-(i-1)}A)^{-1}(z_0^{k-1}I - A)^{-1}(I - A^p)B \end{aligned} \tag{2.41}$$

where the Kronecker δ is $\delta_{i,k} = 1$ when $i = k$ and zero otherwise. Hence when $A^{p-1} = 0$,

Eq. (2.41) becomes $P_e = \hat{P}_e - \Delta$, that is, Eq. (2.35).

Moreover, $\Delta_{i,j}$ has an interesting property that the sum of its column components is zero,

$$\sum_{k=1}^{p} \Delta_{i,k} = \frac{1}{p} C(I - z_0^{-(i-1)} A)^{-1} \sum_{k=1}^{p} \sum_{j=1}^{p} (z_0^{(j-1)(k-1)} A^{p-j}) B = 0 \tag{2.42}$$

This explains the observation of Figure 5 and Figure 6 in Reference [48]. The observation is that when using test input $\underline{u}_e$ in DFT domain with all unity components, the output vector more closely approximates the frequency response than the diagonal elements of P_e. Indeed, the sum of row components

$$\sum_{j=1}^{p} P_e(i,j) = \hat{P}_e(i,i) - \sum_{j=1}^{p} \Delta_{i,j} = \hat{P}_e(i,i) = D + C(z_0^{i-1} I - A)^{-1} B = G(z)|_{z=z_0^{i-1}} \tag{2.43}$$

represents the frequency response.

2.7.2 The Relationship Between the Input and Output Singular Vectors of the Lower Triangular Toeplitz Matrix

Another interesting property discovered here is the singular vectors of Toeplitz P are related.

$$P = USV^T \tag{2.44}$$

where $S = diag(\lambda_1, \lambda_2, \cdots, \lambda_p)$ is the singular value matrix; U is the output singular vector matrix, and $V$the input. The time invariant Toeplitz nature of the P matrix makes it symmetric about the anti-diagonal from lower left to upper right, and this allows the identity $P = JP^TJ$.

Thanks to this property, $\underline{u}_i$ and $(JV)_i$ are eigenvectors of PP^T for the same eigenvalue, that is, $\lambda_i = \sigma_i^2$, where σ_i is a singular value of P. Vector $\underline{u}_i$ is column i of output singular vector matrix U , the i^{th} singular vector. Matrix V is the input singular vector matrix. And $(JV)^i$ is the i^{th} column of JV. If the eigenvalues are distinct, all eigenvectors associated with different eigenvalues are linearly independent and orthogonal for symmetric matrices, like PP^T. The only freedom in the choice of unit length eigenvectors for this eigenvalue is the possibility of having the vector be multiplied by -1. We conclude that the vector $\underline{u}_i$ and the vector $(JV)^i$ satisfy

$$\underline{u}_i = (JV)_i \quad \text{or} \quad \underline{u}_i = -(JV)_i \tag{2.45}$$

2.7.3 DFT of a Toeplitz Matrix without a D Term

Another analogous equation of Eq. (2.35) for a Teoplitiz P (Eq. (2.4)) without a D term is

$$P_e(i,j) = \begin{cases} C\Big(I - \frac{A}{z_0^{i-1}}\Big)^{-1} B - \Delta(i,j) & \text{when } i = j \\ \\ -\Delta(i,j) & \text{otherwise} \end{cases} \tag{2.46}$$

where

$$\Delta(i,j) = \frac{1}{p} C\Big(I - \frac{A}{z_0^{i-1}}\Big)^{-1} \frac{A}{z_0^{j-1}} \Big(I - \frac{A}{z_0^{j-1}}\Big)^{-1} B \tag{2.47}$$

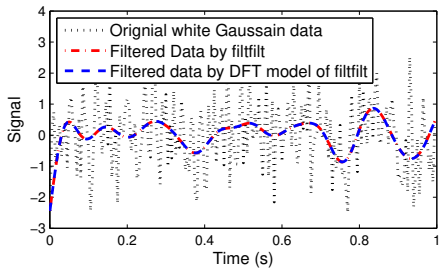
The extra z term in

$$\sum_{j=1}^{p} P_e(i,j) = C(I - \frac{A}{z_0^{i-1}})^{-1} B = zG(z)|_{z=z_0^{i-1}} \tag{2.48}$$

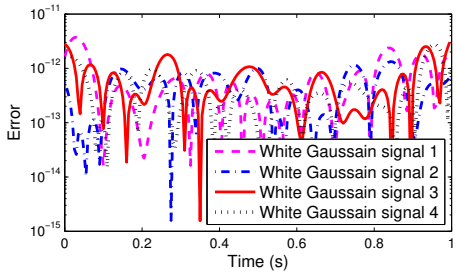
implies the one time step difference between $\underline{u}$ and $\underline{y}$. And all relationships described above remain the same.

2.8 Conclusion

Iterative Learning Control is unusual in control theory in that it asks to converge to zero error at every time step. To do so requires a system model that has relatively small phase error all the way to Nyquist frequency. Since such a model is not normally available, it is natural to ask for a cutoff of the learning process at high frequency. The cutoff needs to be done with a zero phase filter which normally is not a pure frequency cutoff because it requires assigning initial conditions both at the start and the end of the trajectory to start the filter forward and backward. Associated with these are transients that disturb the desired frequency response. Hence, there is a mismatch between the robustification objective given in the frequency domain, and the finite time filter implementation. Here it is shown that the zero phase filter algorithm in Matlab can produce instability of the learning process. This chapter presents a different approach to making a frequency cutoff in ILC, making use of the circulant form of a finite time filter that produces the infinite time frequency response behavior. The approach makes use of an extension of the signal to be filtered, by reflecting the signal, in order to eliminate the Gibbs phenomenon. The approach eliminates the mismatch, eliminates the choice of initial conditions, and eliminates the instability issues. The benefits and improvements from the approaches are demonstrated in examples.

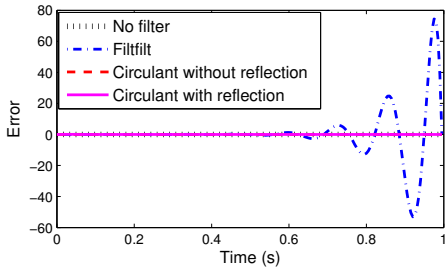


(a) Output of filtfilt and DFT model of filtfilt

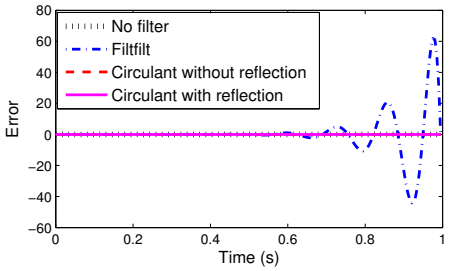


(b) Absolute error between filtered data by filtfilt and by DFT model of filtfilt

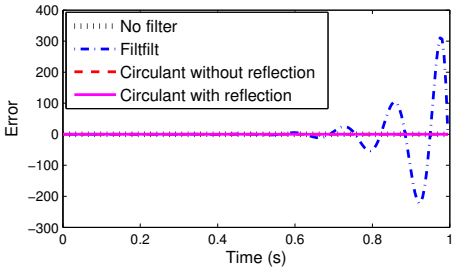
Figure 2.1: Accuracy test of DFT model of filtfilt



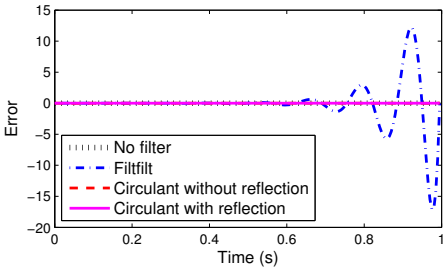
(a) Error in the 200th iteration for Trajectory 1



(b) Error in the 200th iteration for Trajectory 2

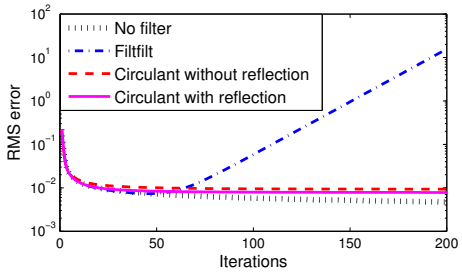


(c) Error in the 200th iteration for Trajectory 3

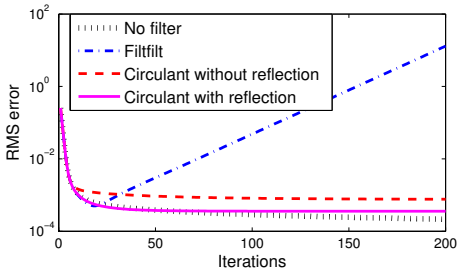


(d) Error in the 200th iteration for Trajectory 4

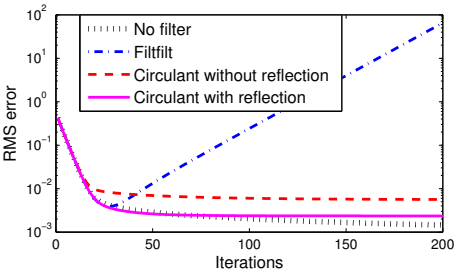
Figure 2.2: Error in the 200th iteration for 4 different trajectories



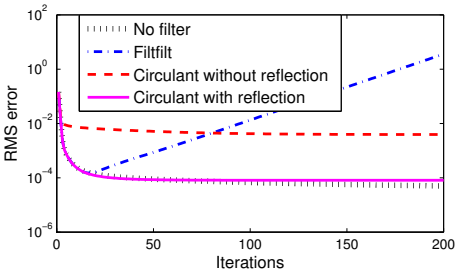
(a) RMS of error for Trajectory 1



(b) RMS of error for Trajectory 2



(c) RMS of error for Trajectory 3



(d) RMS of error for Trajectory 4

Figure 2.3: RMS of error for 200 iterations

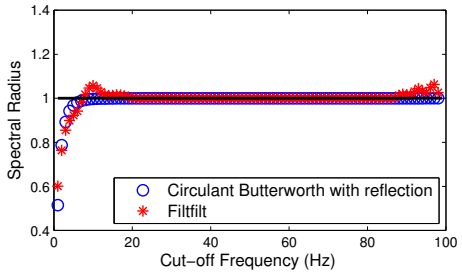


Figure 2.4: The maximum eigenvalues

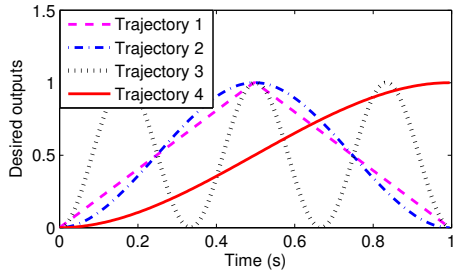
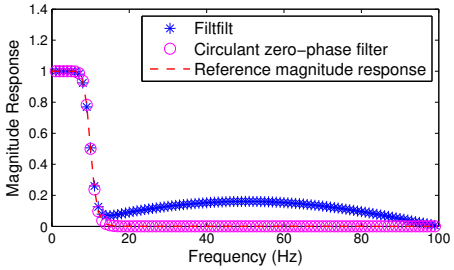
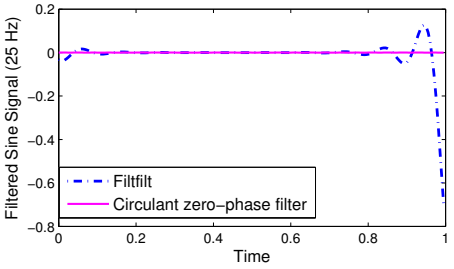


Figure 2.5: Desired trajectories

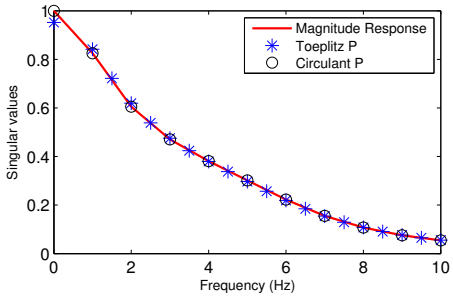


(a) "Magnitude Response" of zero-phase circulant Butterworth filter and Filtfilt Butterworth filter

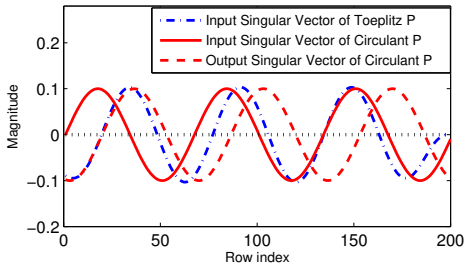


(b) Filtered sine signal in stop band

Figure 2.6: Zero-phase circulant Butterworth filter vs Filtfilt Butterworth filter



(a) Singular values of P and $\hat{P}$



(b) The singular vectors at the second harmonic frequency

Figure 2.7: The singular vectors at the second harmonic frequency

Chapter 3

Adaptive Model-Based Linear ILC to Increase Execution Speed

3.1 Introduction

Since the output data from previous iterations carries the information of system dynamics, it can be used to update the system model used in the ILC design. For linear ILC, this leads to an adaptive linear ILC method that allows different execution speeds of machines. The motivation to increase the execution speed of machines comes from the way humans learn a motion.

In 1983, a psychologist Howard Gardner categorized human intelligence into nine domains, among which is the bodily-kinesthetic intelligence. The core elements of bodily-kinesthetic intelligence are control of one's bodily motions and the capacity to handle objects skillfully [49]. Human beings learn how to make motions such as a serve in tennis through repetitions. A beginner's serve is slow, but the coach repeatedly corrects his motions until

the muscle memory is formed. In the repetition process he can simultaneously increase the speed and the accuracy.

Enormous efforts have been made to enable robots to have a similar learning ability via trial-and-error algorithms. There is currently little literature that aims to extend these algorithms for human-like speeding up during learning. One possible reason is that robots normally start at the desired speed. However, when tasks push robots to hardware limits, one may prefer starting at a slower speed and then tentatively increasing the speed, as is often done when training a robot and one is checking for possible collisions with the environment that might happen at the higher speeds. Also, in some cases a maneuver is initially learnt by demonstration, but then one wants to execute it faster, speeding up the machine. This paper proposes a trial-and-error algorithm for learning in iterations, and the algorithm includes methods that learn to execute the maneuver faster and faster.

Several publications have considered finding a way to apply ILC while successively speeding up the trajectory to be tracked. Prof. D'Andrea's group in ETH sped up the motion of a quadcopter following a figure eight trajectory [50]. The problem they addressed is not a standard ILC problem because it continuously performs the figure eight motion without stopping, and as a result they make use of steady state frequency response modeling. To handle the frequency response at higher speeds, they use linear extrapolation instead of adaptation based on the input-output data. The standard ILC problem performs a repeating maneuver always resetting to the same initial conditions before the next iteration. Thus, ILC asks for zero tracking error during the time-domain transients as well as during the steady state response portions of the trajectory that are characterized by steady state frequency response. Here we make use of the data obtained during ILC iterations to adaptively adjust the ILC law, and aim

to have zero error during the time-domain transients as well.

A second publication applies ILC to make a robot perform suturing [18]. Initially the surgical robot learns the suturing by human-guided demonstration, made it imitate the motions of a surgeon performing the task. This suturing process is then sped up, using ILC to maintain accuracy, resulting in "superhuman" performance. The approach uses the simplest form of ILC that can only work on very low order dynamic systems. In fact the dynamic model had no dynamics, the state in the next time step is the same as the current time step unless an input is applied. The suturing problem must however deal with geometric nonlinearities of robot kinematics. In more dynamic systems, the phase lags encountered at high frequency must be addressed by the ILC law in order to avoid instability. In this chapter an adaptive form of ILC is proposed can be applied to general LTI systems to speed up the trajectory.

The ILC approach proposed here is robust and adaptive in three respects. First, we can make ILC laws that are robustified to prescribed uncertainties in model parameters. The learning rate is lowered to improve robustness in any frequency range where extra robustness is needed, and the learning in the remaining parts of the error space can be left unaltered [45].

Second, if the model at high frequencies is sufficiently wrong, this model error will be revealed gradually, and instability in tracking will evolve slowly from the desired trajectory. This is because the magnitude frequency response of typical systems is small at high frequencies. And this allows time to apply a zero-phase low-pass cut-off filter to stabilize the learning law [51, 42, 48, 37] One can then progress to the third respect that is more than robust, and is truly adaptive. Because ILC can generate data that is specifically focused on what is wrong with the model, i.e. what in the model is sufficiently wrong that there is growth of error, one can use the data generated in this process to fix the model. Reference [52] investigates the use

of ILC as an alternative to the methods presented by the field of optimal experiment design for identification. Here we do Markov parameter estimation, making adaptive updates of the model, in order to stabilize the learning law.

Two approaches of Markov parameter estimation are discussed: the direct estimation for short trajectories and the observer method for long trajectories. The direct estimation method utilizes the Toeplitz matrix form of input-output mapping to calculate the system Markov parameters [53]. In the observer method, one identifies the Markov parameters of an asymptotically stable observer, and from these one can then develop the system Markov parameters [54]. This reference developed the method using deterministic observers. The OKID [55, 56] algorithm for system identification used the observer on stochastic models, in which case the observer Markov parameters become the steady state Kalman filter Markov parameters. Because the observer converges faster than the system, the process compresses the number of parameters that must be identified. From this small number of parameters one can obtain a full set of system Markov parameters.

In summary, this chapter analyzes the changes in the system model resulting from increasing the sampling rate, proposes methods to adjust learning rates, and discusses the implementation of Markov parameter estimation during learning. The result is a procedure for adaptive ILC to speed up the tracking along a desired trajectory.

3.2 Problem Formulation

In this chapter, because we will be repetitively identifying the system model from previous data, one may not need to use a cutoff filter, and thus the learning law can be represented by

$$\underline{u}_{j+1} = \underline{u}_j + L\underline{e}_j \tag{3.1}$$

The system dynamics is still described by Eq. (2.2). The product $P\underline{u}$ is the convolution sum particular solution of the system's difference equation. An asymptotic relationship between the singular value decomposition (SVD) of this Toeplitz P matrix and the system frequency response was presented in Reference [48]. The singular values of P correspond to the system magnitude Bode plot and the differences between input and output singular vectors of P give the Bode phase information. A recent publication, Reference [37] further unveils the underlying relationship of the Toeplitz P, the discrete Fourier transform (DFT) of P, and the system frequency response.

Reference [45] presents understanding of the learning law convergence rate associated with different singular values and their associated parts of the error space (i.e. the components of the error on the orthogonal unit vectors that form the columns of U). And then one can decide to specify the learning rate for each singular value. Based on the correspondence between the SVD of P and the frequency response of the sampled system, this adjustment of the learning rate for each singular vector becomes specifying the learning rate for different frequency components of the error. This interpretation is helpful when we speed up the desired trajectory, and hence raise its frequency content.

3.3 Choice of Sample Rate While Increasing the Execution Speed

ILC must be implemented in discrete time, because it must store the error history from the previous repetition. To increase the execution speed for discrete systems, two possible scenarios are considered.

Scenario 1 simply uses the same sample rate throughout the speeding up process. This implies that if the final trajectory is much faster than the original trajectory, the initial trajectory must have a large number of time steps in order to maintain fidelity of the trajectory after speeding up is completed. This could be a disadvantage because it may need large data sets and perhaps slower convergence as a result. The approach does have the advantage that the correct Markov parameters in the P matrix used to design L do not change. Note, however, that as the trajectory goes faster, the frequency content of the trajectory increases. Two situations might apply. One is that the desired trajectory is composed of frequencies all of which are low enough that one's model for P is good enough for convergence directly at the desired speed, and in this case there is no need to use a speeding up procedure. If a speeding up procedure is needed, we then need to re-evaluate the Markov parameters during the learning process, to correct the model at high frequencies.

Scenario 2 describes the desired trajectory in terms of a fixed number of equally spaced samples throughout the speeding up process. In order to make the digital control system perform the desired trajectory faster, one successively reduces the sampling time interval T. This approach maintains the same fidelity of representation of the desired trajectory throughout the speeding up process. It also has the advantage that one can directly use the command history

from the previous sample rate as a good starting point for the iterations at the faster sample rate. The disadvantage is that each time the sampling rate is increased, the sampled system has new dynamics. One needs to update the Markov parameters in the model P not only for the reasons described above, but because the Markov parameters are different for different sample rates. This paper studies the speeding up problem using this scenario, maintaining a constant number of sample times in the trajectory, but increasing the sampling rate.

3.4 Approaches to Updating the Model as Sample Rate is Increased

For the ILC laws that are based on the P matrix of Markov parameters, i.e. the pulse response history of the system, one wants to update these parameters based on data as execution speed is increased. This could be done using the data being generated during the iterative learning process, but to ensure richness one may desire to make some special tests for identification.

If one has a continuous time model, one can always re-compute the Markov parameters for any desired sample rate. As the sample rate increases, we need to compute better models of high frequency dynamics, so one would be re-identifying the system during the iterations. Note, however, that many system identification algorithms (e.g. Reference [57, 55]) use input-output data to identify the Markov parameters, and from them identify the associated state space difference equation model, and then the state space differential equation. The conversion to state space model involves some subtleties in making the choice of model order, and elimination of states, sometimes called noise modes, that are modeling the noise in the

data set instead of the system. Various tests are used to make such decisions, and it is perhaps not easy to automate. Hence, it seems natural to re-identify the Markov parameters.

If a relatively small change is made in the sample rate, one wonders if it is possible to make some modification to the Markov parameters without going back to the continuous time model, and not performing a new identification of Markov parameters from data. This could happen at several different levels: (1) Perhaps one could expand the Markov parameters in a Taylor series as a function of sample time interval. (2) Perhaps knowing the system matrices at one sample rate can allow one to know the matrices at a faster rate. (3) Another option could be using input-output data to find frequency response using discrete Fourier transforms of the input and output sequences, and invert it to find the unit pulse response, or Markov parameters.

Given the continuous LTI system model,

$$\begin{aligned} \dot{x}(t) &= A_c x(t) + B_c u(t) \\ y(t) &= C_c x(t) + D_c u(t) \end{aligned} \tag{3.2}$$

The discrete system model fed by a zero order hold (ZOH) with sampling period T is

$$\begin{aligned} x(k+1) &= Ax(k) + Bu(k) \\ y(k) &= Cx(k) + Du(k) \end{aligned} \tag{3.3}$$

where

$$
\begin{aligned}
A &= A(T) = e^{A_c T} = I + A_c T + \frac{1}{2!} A_c^2 T^2 + \cdots + \frac{1}{n!} A_c^n T^n + \cdots \\
B &= B(T) = \int_0^T e^{A_c \tau} d\tau B_c = A_c^{-1}(e^{A_c T} - I) B_c \\
C &= C_c \quad D = D_c
\end{aligned}
\tag{3.4}
$$

The Markov parameters $D,\ CB,\ CAB,\ \cdots, CA^nB$ of the sampled system can be calculated by Eq. (3.4). The expansion of the Markov parameters in a Taylor series through first order terms, i.e. a linear model of the change with sample time, needs the Markov parameter derivatives with sample time, for example,

$$
\frac{d(CA^kB)}{dT} = kCA_cA^kB + CA^{k+1}B_c \tag{3.5}
$$

Note that using this expression asks that one know the continuous time system matrices A_c and B_c, which is asking us to find the continuous time model, and this defeats the purpose of using the expansion. We comment that for the surgical robot the system matrix of a single motor, $A_c = I$, and then the equation simplifies to

$$
CA^kB\Big|_{A_c=I} = (\frac{1}{T} + 1)e^{(k+1)T} C_c B_c \tag{3.6}
$$

Now consider that we know the discrete time state space model A_1, B_1 for a given sample rate T_1. Can we modify these matrices to directly change the matrices to a different sample time T_2, A_2, B_2? Matrix A_2 can be estimated based on A_1 by

$$A_2 = e^{A_c T_2} = e^{A_c T_1 \frac{T2}{T1}} = A_1^{\frac{T2}{T1}} = M^{-1} \begin{bmatrix} \lambda_{1,1}^{\frac{T_2}{T_1}} & & & \\ & \lambda_{1,2}^{\frac{T_2}{T_1}} & & \\ & & \ddots & \\ & & & \lambda_{1,m}^{\frac{T_2}{T_1}} \end{bmatrix} M \tag{3.7}$$

where $\lambda_{1,j}$ is the j^{th} eigenvalues of A_1, M is the corresponding eigenvector matrix, and m is the dimension of A_1. However, to obtain B_2, one needs to go to Eq. (3.4). Again one goes back to the continuous model B_c, and again it appears that there is no direct way to change the matrices to a new sample rate.

The final consideration was to look at the frequency response. It is computed from

$$G(z) = (1 - z^{-1}) \mathscr{Z}\left(\frac{G(s)}{s}\right) \tag{3.8}$$

by substituting $z = e^{\hat{i}\omega T}$ where $\hat{i}$ is the imaginary unit. Experience with this conversion to the discrete time z-transfer function makes it clear that one cannot simply do some kind of interpolation or extrapolation to have the frequency response at a different sample rate. For example, the phase response is the same at both sample rates at the associated different Nyquist frequencies.

Because of all the complications discussed above, after the sample rate is changed, we choose to re-identify the Markov parameters using the data from that sampling rate generated during the learning process. The proposed trial-and-error algorithm is an adaptive ILC approach–an ILC control law robust to model errors implemented with Markov parameter estimation based on the data during learning. This idea is similar to the exploration-and-

exploitation in RL, which describes the tradeoff between acquiring new information (exploration) and capitalizing on the information available so far (exploitation). Here the difference is that the exploration done by Markov parameter estimation can be accomplished during the learning process, and may not require the use of special inputs solely for the purpose of parameter estimations/exploration.

3.5 Markov Parameters Estimation

Two Markov parameter estimation approaches are discussed, one direct estimation method for short trajectories and the other deadbeat observer method for long trajectories.

3.5.1 Direct Markov Parameter Estimation for Short Trajectories

The product $P\underline{u}$ is the convolution sum particular solution to the system difference equation and the order of the two sets of data in the convolution sum can be switched. To cancel the repeated disturbance from the input-output relationship, we need to difference two iterations, that is, $\delta_{j,M}\underline{y} = \underline{y}_{j+M} - \underline{y}_j = P(\underline{u}_{j+M} - \underline{u}_j) = P\delta_{j,M}\underline{u}$. One may wish to difference successive iterations or iterations that are more separated creating larger differences. By switching the order, the equation becomes a standard linear equation with the Markov parameters as the unknowns

$$\delta_{j,M}\underline{y} = \delta_{j,M}U\underline{h} \tag{3.9}$$

where

$$\delta_{j,M}U = \begin{bmatrix} \delta_{j,M}u(1) & & & & & \\ \delta_{j,M}u(2) & \delta_{j,M}u(1) & & & & \\ & \delta_{j,M}u(2) & \ddots & & & \\ & & \ddots & \ddots & & \\ \delta_{j,M}u(p-1) & & & \ddots & \ddots & \\ \delta_{j,M}u(p) & \delta_{j,M}u(p-1) & & & \delta_{j,M}u(2) & \delta_{j,M}u(1) \end{bmatrix}, \quad \underline{h} = \begin{bmatrix} D \\ CB \\ CAB \\ \cdots \\ CA^{p-3}B \\ CA^{p-2}B \end{bmatrix} \tag{3.10}$$

An unique solution of $\underline{h}$ exists if and only if $\delta_{j,M}U$ is of full rank, in other words, if and only if $\delta_{j,M}u(1) \neq 0$. The solution is $\underline{h} = \delta_{j,M}U^{-1}\delta_{j,M}\underline{y}$. One characteristic of this estimation method is that one first solves for the first Markov parameter since it is uncoupled, that is, the first Markov parameter satisfying $\delta_{j,M}y(1) = h(1)\delta_{j,M}u(1)$ and then this solution is used in the second which is uncoupled, etc. This chain of equations can have errors from the first equation propagating to the second, etc., making larger error in latter parameters. Latter parameters are hence less accurate.

There is a Markov parameter for every time step of the data. But the Markov parameters decay with time, and after one settling time of the system, may be small enough that one need not estimate them or use them in the learning gain matrix. Suppose first N Markov parameters are obviously above the noise-level. Eq. (3.9) can be modified to include only the first N Markov parameters in the column vector $\underline{h}$, but continue to use all of the equations. Then the matrix $\delta_{j,M}U$ can be replaced by the truncated $\delta_{j,M}U^{(N)}$

$$\delta_{j,M}U^{(N)}$$

$$= \begin{bmatrix} \delta_{j,M}u(1) & & & & & \\ \delta_{j,M}u(2) & \delta_{j,M}u(1) & & & & \\ \vdots & \delta_{j,M}u(2) & \ddots & & & \\ \vdots & \vdots & & \ddots & & \\ \vdots & \vdots & & & \delta_{j,M}u(1) & \\ \vdots & \vdots & & & \delta_{j,M}u(2) & \delta_{j,M}u(1) \\ \vdots & \vdots & & & \vdots & \vdots \\ \delta_{j,M}u(p-1) & \delta_{j,M}u(p-2) & \cdots & \cdots & \delta_{j,M}u(p-N+3) & \delta_{j,M}u(p-N+2) \\ \delta_{j,M}u(p) & \delta_{j,M}u(p-1) & \cdots & \cdots & \delta_{j,M}u(p-N+2) & \delta_{j,M}u(p-N+1) \end{bmatrix} \tag{3.11}$$

This is a set of equations with N unknowns, but more than N equations, and using the pseudo-inverse solution minimizes the equation error.

Reference [58] discusses designing ILC controllers from a limited number of Markov parameters. This reference suggests filling in other parameters in P by zeros, or making use of a window. This window aims to smooth the cliff in the Markov parameters from N^{th} to $N + 1^{th}$ parameters. For example, the accelerated exponential window produces some nonzero value, $exp[(-k)/(N - k + 1)]$, instead of zero for the k^{th} parameter when $k > N$. The reference also points out that increasing the sampling rate may imply a linear increase of the minimum number of parameters needed for stability and monotonic decay. Intuitively,

the system model we used would have more significant Markov parameters if the sampling rate is increased, since there are more time steps within the settling time of the system.

3.5.2 Markov Parameter Compression Through a Deadbeat Observer

System identification of lightly damped satellite structures requires many time steps in the matrix Eq. (3.11). Methods were developed in Reference [56, 55] to identify the Markov parameters of a deadbeat observer first and from these parameters one can compute the Markov parameters of the system. Since a deadbeat observer has Markov parameters to go to zero in the number of time steps equal to the number of state variable in the system, this very significantly compresses the number of parameters to identify. In the case of a stochastic model, the deadbeat observer in OKID becomes the steady state Kalman filter. Here we suggest the use of the deadbeat observer in Reference [56] for longer desired trajectories.

An interaction matrix is used in the development of the deadbeat observer, which is done by adding and subtracting a term containing this interaction matrix from the filter equations. Then there exists a choice of this matrix with the purpose of canceling the initial condition influence of the observer in N steps, where N is equal to or bigger than the system order n. Here we show the math of the observer propagating the state forward N steps for disturbance/noise-free SISO systems including a D term. In ILC, we need to use $\delta_{j,M}\underline{u}$ and $\delta_{j,M}\underline{y}$ instead of $\underline{u}$ and $\underline{y}$ shown below in order to eliminate any repeating disturbance from the equations.

First we package the outputs and commands into N-step vectors,

$$\begin{aligned} \underline{y}_N(k) &= \begin{bmatrix} y(k-N) & \cdots & y(k-2) & y(k-1) \end{bmatrix}^T \\ \underline{u}_N(k) &= \begin{bmatrix} u(k-N) & \cdots & u(k-2) & u(k-1) \end{bmatrix}^T \end{aligned} \tag{3.12}$$

Then adding the observer M into the system,

$$\begin{aligned} x(k+N) &= A^N x(k) + \bar{C}_N \underline{u}_N(k) - M[\underline{y}_N(k) - \underline{y}_N(k)] \\ &= A^N x(k) + \bar{C}_N \underline{u}_N(k) - M\underline{y}_N(k) - M(O_N x(k) + P_N \underline{u}_N(k))] \\ &= (A^N + MO_N)x(k) + (\bar{C}_N + MP_N)\underline{u}_N(k) - M\underline{y}_N(k) \\ y(k+N) &= Cx(k+N) + Du(k+N) \end{aligned} \tag{3.13}$$

where O_N and P_N are the N-step observability matrix O and N-step Toeplitz P matrix instead of p steps. The controllability matrix $\bar{C}_N$ is $[A^{N-1}B,\ \cdots,\ AB,\ B]$. As long as the system is observable and N is equal to or bigger than the system order n, there exists an M such that the sum $A^N + MO_N$ becomes a zero matrix. A is a n by n matrix; O_n is a N by n matrix; and M is a n by N matrix. Hence the output $y(k)$ ($k = N+1, N+2, \cdots, p$) can be treated as the output of an ARX model,

$$\begin{aligned} y(k+N) &= [C(\bar{C}_N + MP_N),\ -CM,\ D] \begin{bmatrix} \underline{u}_N(k) & \underline{y}_N(k) & u(k+N) \end{bmatrix}^T \\ &= [\beta_N,\ \cdots, \beta_2,\ \beta_1, \alpha_N,\ \cdots, \alpha_2,\ \alpha_1,\ D] \begin{bmatrix} \underline{u}_N(k) & \underline{y}_N(k) & u(k+N) \end{bmatrix}^T \\ &= \underline{\gamma}^T \underline{\Gamma}(k+N) \end{aligned} \tag{3.14}$$

The unknown parameters in the ARX model can be solved off-line directly from input-output

data and the least square solution is

$$\hat{\underline{\gamma}} = YW^{\dagger} \tag{3.15}$$

where

$$Y = \begin{bmatrix} y(N) & y(N+1) & \cdots & y(p) \end{bmatrix} \text{ and } W = \begin{bmatrix} \underline{u}_N(0) & \underline{u}_N(1) & \cdots & \underline{u}_N(p-N) \\ \underline{y}_N(0) & \underline{y}_N(1) & \cdots & \underline{y}_N(p-N) \\ u(N) & u(N+1) & \cdots & u(p) \end{bmatrix} \tag{3.16}$$

By plugging the $-CM = [\alpha_N,\ \cdots, \alpha_2,\ \alpha_1]$ matrix into

$$C(\bar{C}_N + MP_N) = C\bar{C}_N + (CM)P_N = [\beta_N,\ \cdots, \beta_2,\ \beta_1] \tag{3.17}$$

the first N Markov parameters which composes $C\bar{C}_N$ and P_N can be recovered. Rewrite the matrix M as $M = [M_1,\ M_2, \cdots,\ M_N]$ where $M_1,\ M_2, \cdots,\ M_N$ are the n by 1 partitions of matrix M. This M enables

$$A^N + MO_N = A^N + M_P CA^{N-1} + \cdots + M_1 C = 0 \tag{3.18}$$

Multiplied by C from left and then by B from right, we can compute $CA^N B$ from equation

$$CA^N B + (CM_p)CA^{N-1}B + \cdots + (CM_1)CB = 0 \tag{3.19}$$

where $CM_i = -\alpha_{N-i+1}$. Analogously an unlimited number of Markov parameters can be

recursively recovered

$$
\begin{aligned}
CB &= \beta_1 + \alpha_1 D \\
CAB &= \beta_2 + \alpha_2 D + \alpha_1 CB \\
&\vdots \\
CA^{N-1}B &= \beta_N + \alpha_N D + \alpha_{N-1} CB + \cdots + \alpha_1 CA^{N-2}B \\
CA^{N}B &= \alpha_N CB + \alpha_{N-1} CAB + \cdots + \alpha_1 CA^{N-1}B \\
CA^{N+1}B &= \alpha_N CAB + \alpha_{N-1} CA^2B + \cdots + \alpha_1 CA^{N}B \\
&\vdots
\end{aligned}
\tag{3.20}
$$

As in the direct estimation Eq. (3.9), Eq. (3.20) is a recursive computation and the error from the term CB can accumulate successively solving for the next parameter. But in that direct estimation approach, the CB is calculated from one time step of data, and in this deadbeat observer approach, the coefficients β_1 and α_1 used to calculate CB are computed from many time steps. This should give improved results.

As for the choice of N, one can choose based on the order n of the nominal model, or pick a larger value in order to be confident that one has made N larger than n.

3.5.3 Data Richness in Markov Parameter Estimation

As in system identification, identifying Markov parameters needs to have a rich input-output data set. Data from the start of the iterations is likely to have relatively large components at lower frequencies, with the result that the contribution to the data from high frequency parasitic poles is buried in the noise level and not reflected in the resulting Markov parameters.

On the other hand when the model is sufficiently wrong at high frequencies due to parasitic poles, ILC will amplify the error at these frequencies, helping to produce data in the later iterations where the parasitic pole contribution is evident. This suggests that one may need to retain information from early iterations along with later iterations in order to get good Markov parameters.

When the problem being addressed contains a deterministic disturbance $\underline{d}$ that repeats every iteration, then one must difference the data from two different iterations in order eliminate this disturbance from the data used to identify the Markov parameters. The ILC learning based on the Toeplitz matrix P will eliminate low frequency errors in early iterations relatively quickly, but the learning at high frequencies becomes slow. Therefore, taking a difference between later iterations that are not separated by many repetitions, can lose accuracy because one is taking differences of nearly equal numbers. And in addition the information content emphasizes high frequencies and the lower frequencies may not be well represented in the data. Thus, a decision process is needed to decide what runs need to be used when taking differences in the data. Note that in case the identification process is failing to produce convergence, one can decide to input test signals not resulting from the ILC law, to obtain rich data for identification.

Some other considerations include possible scaling of data. Data from early iterations containing relatively fast convergence of low frequency error $\underline{e}_j = \underline{y}_d - \underline{y}_j$ corresponding to larger signals when differenced. When used simultaneously with later runs where the error is mostly high frequency and smaller amplitude, may require scaling of the two signals to be of similar magnitude. The scaling is limited by the amplification of noise in the data it can produce.

3.6 Maintaining the Learning Rate as the Execution Speed is Increased

For the $L = P^T$ learning law, one can easily describe the learning speed of different parts of the error space in terms of the singular values of the P matrix. We set $F = I$ because we are doing model updates to produce robustness. Otherwise one might use a cutoff and increase it as sample rate increases. The law $L = \alpha P^T$ can be written as

$$\begin{aligned} \underline{e}_{j+1} &= (I - PL)\underline{e}_j = (I - \alpha PP^T)\underline{e}_j = (I - \alpha USV^T(USV^T)^T)\underline{e}_j \\ &= U(I - \alpha S^2)U^T\underline{e}_j = U \begin{bmatrix} 1-\alpha\sigma_1^2 & & & \\ & 1-\alpha\sigma_2^2 & & \\ & & \vdots & \\ & & & 1-\alpha\sigma_p^2 \end{bmatrix} U^T\underline{e}_j \end{aligned} \tag{3.21}$$

where $S = diag(\sigma_1, \sigma_2, \cdots, \sigma_p)$ is the singular value matrix of P; U and V are the singular vector matrices. We can convert the error into its components on the singular vectors in the orthogonal unit vectors that form the columns of U. The mapped error hence decreases according to $U^T\underline{e}_{j+1} = (1 - \alpha S^2)(U^T\underline{e}_j)$. Writing the error in these coordinates does not influence the square of the Euclidean norm error vector,

$$||U^T\underline{e}_{j+1}||^2 = (U^T\underline{e}_{j+1})^T(U^T\underline{e}_{j+1}) = \underline{e}_{j+1}^T(UU^T)\underline{e}_{j+1} = \underline{e}_{j+1}^T\underline{e}_{j+1} = ||\underline{e}_{j+1}||^2 \tag{3.22}$$

Therefore, one can use $1 - \alpha S^2$ to evaluate the uncoupled rate of convergence from one iteration to the next for each component of the error in this space.

Recalling that the singular values approximate the magnitude frequency response, the factor $1 - \alpha\sigma_i^2$ is the amount of decrease in one iteration of the associated frequency component. Because the magnitude response at high frequencies is usually low, i.e. some of the σ's are small, and the learning rate is correspondingly small. Small learning is beneficial for robustness to model error, but we are doing model updates to achieve robustness to learning. Hence we might want to adjust the learning law to maintain the learning rate for the fundamental, and for each harmonic, as the execution speed is increased.

The fastest learning law is the inverse of P, if P is invertible, which produces zero error immediately in the second iteration, i.e. $I - PL = I - PP^{-1} = 0$, provided the model is perfect. This law is not normally used, because of its lack of robustness, and because P is usually very ill-conditioned. One can use a quadratic learning law,

$$L = (P^T P + \gamma I)^{-1} P^T \tag{3.23}$$

which minimizes cost $J_{j+1} = \underline{e}_{j+1}^T \underline{e}_{j+1} + \gamma \delta_{j+1}\underline{u}^T \delta_{j+1}\underline{u}$ where $\delta_{j+1}\underline{u} = \underline{u}_{j+1} - \underline{u}_j$. For this quadratic learning law, the convergence of error components satisfies,

$$U^T \underline{e}_{j+1} = \left[I - S(S^2 + \gamma I)^{-1} S\right] U^T \underline{e}_j = diag(\frac{\gamma}{\sigma_i^2 + \gamma}) U^T \underline{e}_j \tag{3.24}$$

The convergence rate is similar for high frequencies. Details about the learning rate is discussed in Reference [45].

Each singular value approximately corresponds to a frequency that one can enumerate by index i, corresponding to DC, the fundamental frequency, and all harmonics up to Nyquist frequency associated with the fixed number of time steps in the trajectory. As the execution time is decreased, the sample time interval is decreased, the Nyquist frequency increases, and so do the frequencies of the fundamental and each harmonic. The total number of harmonics remains the same as does the number of singular values, but each singular value becomes associated with a higher frequency as the sample rate increases. Consider $L = P^T$ and denote the fundamental frequency by f_0, and the i^{th} harmonic frequency by f_i. When the sampling rate is increased, f_i becomes smaller and the related singular value becomes smaller. The convergence hence becomes slower. We can design a modified P transpose law that maintains the same learning rate, i.e. the same value of the singular value associated with any given harmonic, as the sample rate increase. Denote the Toeplitz matrix of the sampled system by P_s with the slower sampling rate, the SVD of P_s by $P_s = U_s S_s V_s^T$, and P_f with the faster sampling rate and $P_f = U_f S_f V_f^T$ respectively. When one uses $L_s = P_s^T$ and $L_f = P_f^T$ for these two learning rates, the faster sampled system needs more iterations to converge to zero error because $(1 - \sigma_{s,i}^2) < (1 - \sigma_{f,i}^2)$, where $\sigma_{s,i}$ is the i^{th} singular value of the slower system Toeplitz P_s, and $\sigma_{f,i}$ of P_f. To maintain the same learning rate, the adjustment of the contraction mapping law is

$$L = V_f S_s^2 S_f^{-1} U_f^T \tag{3.25}$$

The inverse cancels the singular values of the system, and the squared term supplies the desired singular values for the product of P and P^T. The convergence rate (with respect to the mapped error, $U^T \underline{e}_j$) is then kept at $(1 - \sigma_{s,i}^2)$ instead of $(1 - \sigma_{f,i}^2)$. One can substitute the

pseudo-inverse for the inverse so that bad singular values in S_f are ignored.

Theoretically, one can freely tune the singular values. The partial isometry law makes the learning rate correspond to one minus the magnitude frequency response, instead of its square as in the P transpose law above. Of course, one could make L have the inverse of the system singular values, and this produced the P inverse solution which has very poor robustness properties, and is too aggressive.

3.7 Procedural Issues While Increasing the Tracking Speed

Figure 3.1 presents a flow chart describing the procedure and decision points for the speeding up algorithm.

(1) The first item is to pick the initial command. When ILC is adjusting the command to a feedback control system, this should be the desired trajectory at the initial sample rate. Then one picks a learning rule for the first update. One might develop a model in advance from hardware tests for the purpose, and start using an L that employs this model. Alternatively one can apply the model free simplest learning law $L = \alpha I$ and then identify Markov parameters from the resulting data.

(2) One might want to do Markov parameter updates during the learning process at the initial rate. No matter where the model comes from, there are model uncertainties, and the possibility of parasitic poles that are still not represented in the model. Certainly, anytime the root mean square (RMS) of the error grows (or grow beyond the statistical fluctuation level),

one needs to do a Markov parameter update to correct what is sufficiently wrong to produce the growth. The growth produced by high frequency parasitic poles is a slow growth because the frequency response is small at high frequencies. And this allows both time to perform a Markov parameter update, and also input-output data is specifically created by what is wrong with the model, and therefore helps to correct it.

(3) A decision needs to be made of a threshold, that when the RMS error decays to this level, one increases the sample rate and start the learning process at the new rate. There will of course be enlarged error because the model is no longer appropriate. And one makes Markov parameter updates to find the parameters for this increased sample rate. The learning law is modified when the sample rate is increased as in Eq. (3.25) in order to maintain the speed of for each harmonic and the fundamental.

(4) When there is a repeating disturbance, one needs to take differences of the data from different runs. A decision must be made concerning which runs to use for this difference. A difference between two successive runs when the learning is slow, can made data with poor signal to noise ratio. When error is growing, it probably emphasizes high frequency error, and may need earlier runs to maintain the low frequency information in the model.

3.7.1 Numerical Experiments

A numerical experiment is first performed to illustrate the slow growth of the instability resulting from a parasitic pole. The system is then sped up by increasing the sampling rate, and adaptive ILC is applied to track a desired trajectory. A quadratic learning law with a relatively fast convergence rate is used, produced by a small penalty in the change of control in

the cost function. The RMS errors of each iteration during learning are plotted using Matlab, to demonstrate the learning process.

The System Model Used for Simulations

A 3rd order continuous system is used, where the real root is higher frequency and serves as a parasitic pole

$$G(s) = \frac{K_r}{(s+a)(s^2 + b_1 s + b_2)} \tag{3.26}$$

The $a = 18\pi$ corresponding to a high frequency (9 Hz) pole, $b_1 = 12.5664$ and $b_2 = 157.9137$ correspond to a mass-spring-damper system with 1 kg mass, 2 Hz natural frequency, and 0.5 damping ratio. The model used in the initial ILC design is the above equation without the parasitic pole, as if it was not observed in the identification data.

The initial sample rate is 20 Hz, and this is progressively increased to 40 Hz, increasing the sample time interval in units of 0.005 increments. Figure 3.2(a) shows the first 40 Markov parameters. The third Markov parameter at 20 Hz is the biggest, and forms the peak above 0.3. As the sampling rate is increased, this biggest Markov parameter becomes smaller and moves from the third one (20 Hz) to the six (40 Hz). The Markov parameters with higher sampling rate are on average smaller than the ones with the slower sampling rate, which implies a slower learning rate if $L = P^T$ is used. The desired trajectory (solid line) and the repeated disturbance (dashed line) in the simulation are plotted in Figure 3.2(b). When the sampling rate is 20 Hz, these two signals last for 5 s and after speeding up, the time needed is decreased to 2.5 s.

The Influence of the High Frequency Mode in Normal ILC

This simulation illustrates the behavior of ILC when a parasitic pole is missing in the model. The ILC first improves the error level and then starts to diverge with high frequency error growing, but growing relatively slowly. The quadratic learning law $L = (P^T P + 10^{-8} I)^{-1} P^T$ is applied. First, ILC is applied to system Eq. (3.26), sampled at 20 Hz aiming to track the solid line in Figure 3.2(b) in the presence of the disturbance, the dashed line in Figure 3.2(b). Two sets of Markov parameters are used to create the quadratic learning laws. When the law designed with the second order model is applied to the real world third order system, the result is the left plot of Figure 3.3 (dashed curve for 20 Hz, solid line for 40 Hz), which illustrates initial decay of the error followed by instability. When the law designed using the third order model is applied to the third order world, the results are in the right plot of Figure 3.3, and one observers monotonic decay to a numerical zero RMS error.

The dashed curve in the left figure shows the RMS error decreases initially and reaches a minimum error level above 10^{-3}, and then grows. Instability caused by the model error is seen after iteration 4 and grows to the same level (10^{-1}) at iteration 10 as that of the first iteration. Therefore, one can have at least 6 iterations to stabilize the system again before the instability may lead to any damage to the hardware or the system's environment. The RMS error when learning with the real system model (dashed line in the right figure) reaches 10^{-15} in 4 iterations. Similar observations are made when the sampling rate is increased to 40 Hz.

Comparing the solid lines and dashed lines in both figures, the slope of dashed lines (sampling rate 20 Hz) is bigger than that of the solid lines (sampling rate 40 Hz) when the RMS is decreasing. This bigger slope means the convergence rate of the system with the slower

sampling rate (20 Hz) is bigger than the system with the faster sampling rate. When the instability appears, i.e. the RMS error grows in the left figure, the slope of the dashed line is smaller than that of the solid line. Also, the minimum RMS error level at 40 Hz is bigger than that at 20 Hz. These differences show that the system when sampled at the faster rate suffers more from the model error at high frequency. Therefore, starting ILC from a slower sampling rate can be more robust to the system model errors at high frequencies.

Speeding Up ILC by Increasing Sampling Rate

This simulation demonstrates the adaptive ILC to speed up this 3rd order system following the trajectory in Figure. 3.2(b) with the presence of a repeated disturbance in the same figure. Simulations are made with and without measurement noise. The system sampling rate is increased from 20 Hz to 40 Hz, that is, the sampling period from 0.05 s to 0.025 s, either by step size $\delta T = 0.005$ s or $\delta T = 0.001$ s. No prior system knowledge is assumed and $L = I$ for the first two iterations is used to establish a nominal model. The quadratic learning law with the penalty of learning rates, $L = (P^T P + 10^{-8} I)^{-1} P^T$ using the estimated Markov parameters is applied otherwise.

The sampling rate is increased when the RMS error reaches a numerical zero or the measurement noise level. The last command fed to the system with the previous sampling rate is directly applied as the initial command to the system with the increased sampling rate. The Markov parameters of the slower system are used temporarily to design the L for the faster system until the Markov parameters are estimated again. Markov parameters are updated once using data from two iterations (the initial iteration and the other picked near the quasi-steady state) for each sampling rate by the deadbeat observer method.

Figure 3.4 plots the RMS error of each iteration during the two complete speeding up processes ($\delta T = 0.005$ s and $\delta T = 0.001$ s), without noise or with an uniformly distributed noise in the interval $\begin{bmatrix} -10^{-3} & 10^{-3} \end{bmatrix}$. The spikes in the figure are caused by the inconsistency of system dynamics after increasing the sampling rate, i.e., the same command produces different output after increasing the sampling rate. This difference will be different each time the sample rate is increased, so the spikes need not be the same height. The maximum RMS error level is around 0.5 from the initial iteration for all cases. In fact, the RMS error level of the spikes is around 0.05 or 10% of the maximum RMS error for $\delta T = 0.005$ s and 0.01 or 2% for $\delta T = 0.001$ s (the dashed line in Figure 3.4(b)). Therefore, by gradually increasing the sampling rate, one can avoid big RMS error during learning. Provided there is no noise, zero RMS tracking error (Matlab) is achieved as shown in Figure 3.4(a) and 3.4(b). Figure 3.4(c) and 3.4(d) demonstrates the cases with noise level 10^{-3}. The speeding up ILC successfully learns the trajectory and achieves the tracking accuracy around the noise level.

3.8 Conclusions

In this chapter, we propose an adaptive ILC method by incorporating Markov parameter estimation during the learning process, which can enable a human-like learning process, learning to execute a desired maneuver faster and faster. The faster execution of a specific maneuver is achieved via increasing the system sampling rate while maintaining the desired trajectory values at the sample times.

The model used by the ILC laws is based on the system model in the form of the Toeplitz matrix of Markov parameters. The nature of the changes in Markov parameters with sample

rate is studied. Two approaches to update Markov parameters during learning are presented, a direct estimation for short maneuvers and a deadbeat observer approach for long maneuvers. The Markov parameter estimation implemented in ILC serves the same purpose as exploration approaches in RL. The difference is that in RL, lack of exploration can result in a local optimal solution. But in ILC, the tracking error either diverges (often gradually) or it converges. In other words, the necessary exploration data will automatically appear during the learning process itself. One can then use Markov parameter estimation to finish the exploration and update the ILC law for exploitation again as needed. As for the exploitation, the method to maintain a desired learning rate is also discussed. Unlike RL, this adaptive ILC algorithm has control of the learning behavior or learning transients, showing its effectiveness as a safe exploration-and-exploitation for LTI systems.

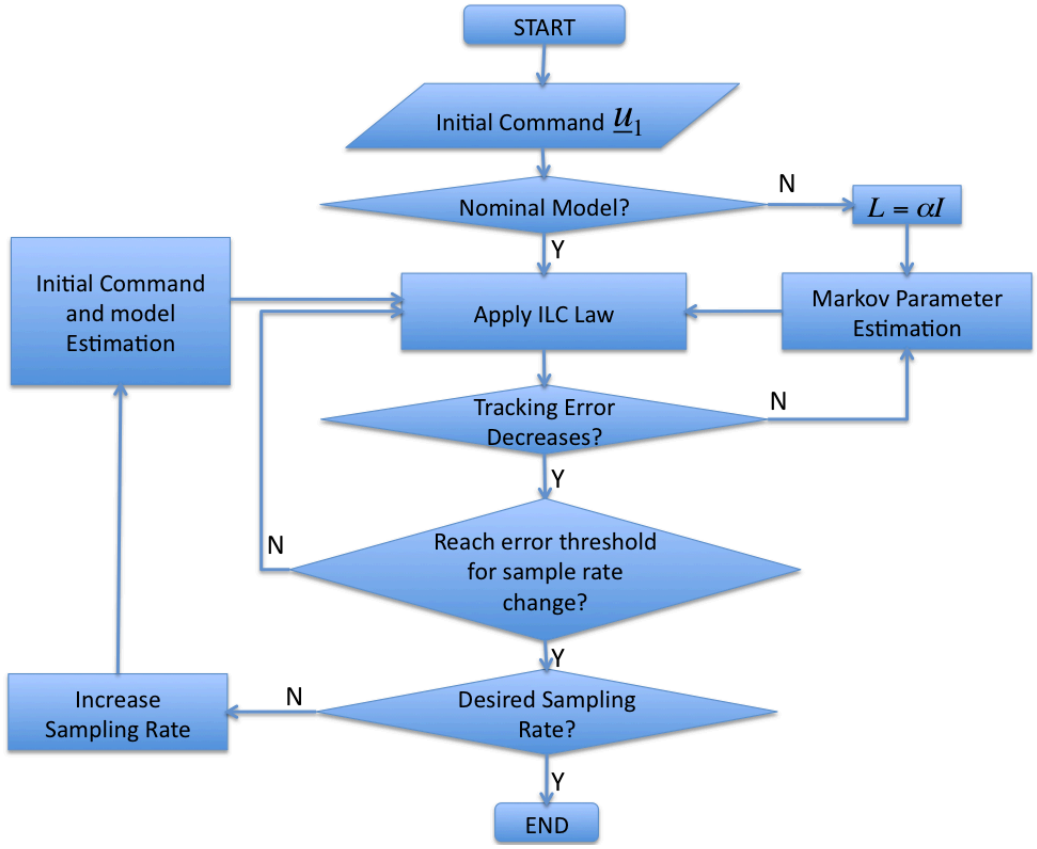
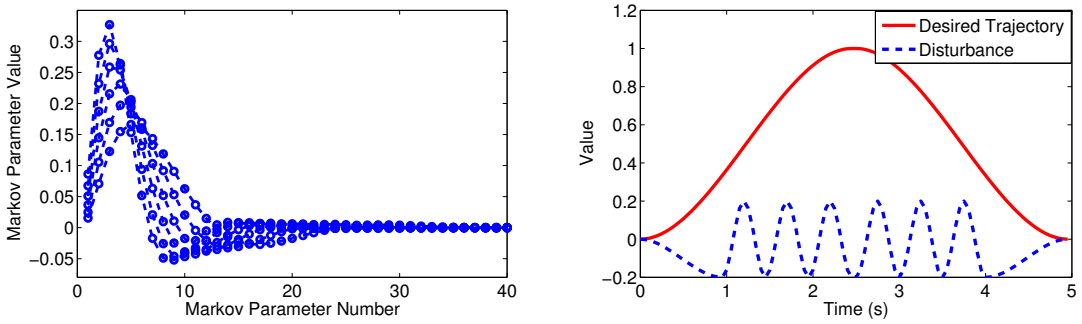


Figure 3.1: The flowchart of speeding up ILC by increasing sampling rate



(a) Markov parameters change as increasing sampling rate (step size $\delta T = 0.005$)

(b) Desired trajectory and repeated disturbance (sampling rate 20 Hz)

Figure 3.2: The system model, the desired trajectory and the repeated disturbance

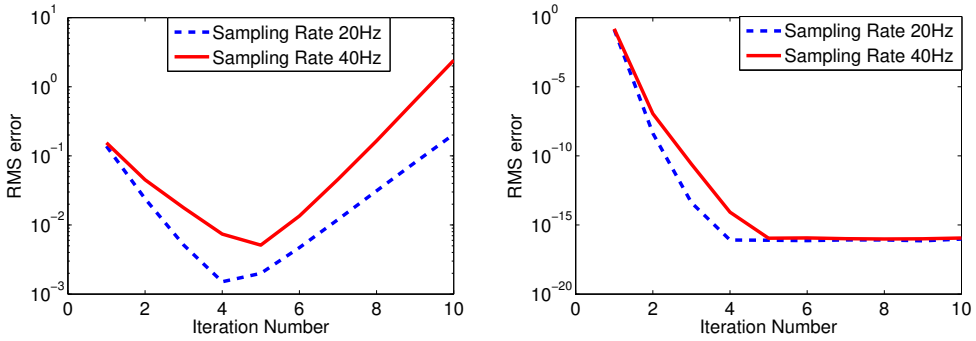
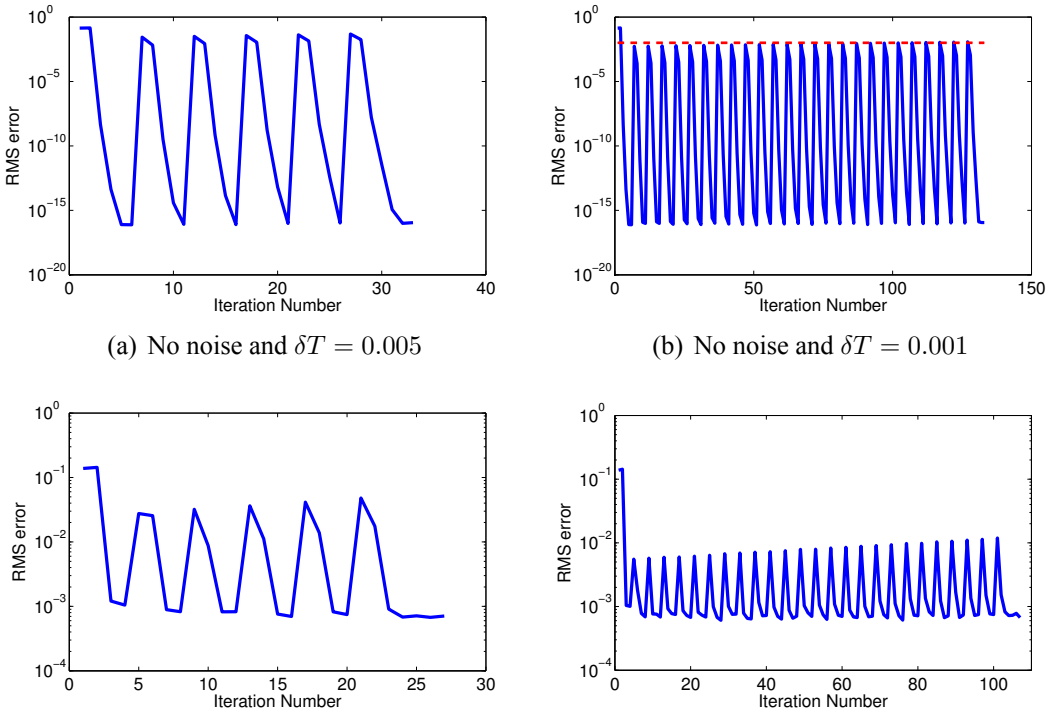


Figure 3.3: RMS error history of iterations during normal quadratic cost learning process



(a) No noise and $\delta T = 0.005$

(b) No noise and $\delta T = 0.001$

(c) Noise level 10^{-3} and $\delta T = 0.005$

(d) Noise level 10^{-3} and $\delta T = 0.001$

Figure 3.4: RMS error history during the speeding up ILC process

Chapter 4

Model-Based Nonlinear ILC through Carleman Bilinearization

4.1 Introduction

Given the measurements of all states, an ILC method based on bilinearization is developed for nonlinear systems in this chapter. For nonlinear ILC, until now, most publications focus on continuous nonlinear systems without sampling [59, 60, 36]. For discrete cases, the Arimoto-type ILC directly applied on nonlinear systems has been intensively investigated [61, 62, 63]. One application example is for freeway traffic control [20]. However, the issue of the bad learning transients of Arimoto-type ILC has not been discussed, which may be due to the fact that this phenomenon will not appear until the tracking error approaches zero. Other than the Arimoto-type approaches, Lin, Owens, and Hatonen have successfully developed a Newton-type ILC based on the Jacobian of nonlinear systems [8]. Longman, and Mombaur successfully apply model-based discrete linear ILC laws to linearized models of a nonlinear

system along a reference trajectory [64].

This chapter starts from ILC algorithms for a general system by minimizing a cost function, which results in an approach using Picard iteration that needs the analytical form of the Jacobian of a system model. However, there is difficulty having a general purpose model that applies to the large variety of nonlinear behaviors. Neural networks is one approach, but it often precludes development of analytical properties, the Jacobian for ILC design. ARX models have the potential to be applied here but with tedious computation needed. On the other hand, through Carleman bilinearization, a nonlinear system can be approximated by a bilinearized model to arbitrarily high accuracy, at the expense of increased dimensionality [65, 66].

Thus, in an effort to create ILC approaches to handle a large class of nonlinear systems, it is natural to develop ILC methods that can address bilinear models. These bilinear methods can then be applied to nonlinear systems through their bilinearized models. The bilinearized model can be a considerable improvement over the linearized model because a linearized model eliminates quadratic and higher order terms in an expansion, while the bilinear model remains bilinear as one makes the bilinearization more accurate by increasing the state dimension to capture higher order nonlinear effects.

The bilinearized model along the trajectory of the previous iteration is used in ILC, which is a good approximation of the nonlinear system in the close vicinity along the reference trajectory. The use of a bilinearized model can capture more nonlinear dynamics and hence leads to faster convergence. At the same time, the error in the bilinearized model is more sensitive to the distance with respect to the reference trajectory. For a desired trajectory outside the vicinity where the bilinearized model is valid, one can specify a series of trajectories, starting

with a feasible trajectory and converging to the desired trajectory to guide the ILC process.

By using this homotopy of the desired trajectory, the bilinear method can learn to follow the desired trajectory in a reasonable number of runs, which is a considerate improvement compared with the corresponding linear method. Numerical examples are performed to demonstrate the improvement. The material in this chapter has been presented in Reference [40] by the author, which takes an important step to fill the gap in the development of ILC laws specifically designed to handle nonlinear systems.

4.2 ILC Formulation for a General System

4.2.1 Problem Description and Input-Output Model used in ILC

The mathematical description of ILC for a general system is as follows. By representing the system dynamics by the input-output relationship for the entire trajectory,

$$\underline{y}_j = f(\underline{u}_j) + \underline{d} \tag{4.1}$$

where $\underline{d}$ represents the influence of the initial state and repeating disturbances, the tracking error propagation between two successive runs is

$$\underline{e}_j = \underline{e}_{j-1} + f(\underline{u}_{j-1}) - f(\underline{u}_j) \tag{4.2}$$

To construct a contraction mapping between $\underline{e}_{j-1}$ and $\underline{e}_j$, an ILC learning law is of the form

$$\underline{u}_j = h(\underline{u}_{j-1}, \underline{e}_{j-1}) \tag{4.3}$$

By using data from previous iterations, ILC can improve the tracking accuracy beyond the limit of the model accuracy used in the learning algorithm. The robustness of ILC towards the model error depends on the robustness of the contraction mapping, which is negatively correlated with the learning rate. That is, the faster an ILC law can learn, the less robust it is towards model error. For linear ILC, the monotonic decay of tracking errors and the learning rate of various ILC laws have been investigated [45]. These results for linear systems can be helpful to develop ILC algorithms for nonlinear systems. The most straightforward way is to develop ILC algorithms based on the linearized models of the nonlinear systems. Here in this paper, the ILC algorithms based on bilinearized models are proposed and the methods based on linearized models are used to form a homotopy of the desired trajectory to guide the ILC learning.

4.2.2 ILC for Time-Varying Linear Systems

Given a time-varying linear system of the form,

$$\begin{aligned} x(k+1) &= A_l(k)x(k) + B_l(k)u(k) \\ y(k) &= C_l(k)x(k) + D_l(k)u(k) \end{aligned} \tag{4.4}$$

the input-output relationship Eq. (4.1) for iteration j becomes

$$\underline{y}_j = P_l \underline{u}_j + \underline{d} \tag{4.5}$$

where P_l is made of the system Markov parameters,

$$P_l = \begin{bmatrix} D_l(0) & & & & \\ C_l(1)B_l(0) & D_l(1) & & & \\ C_l(2)A_l(1)B_l(0) & C_l(2)B_l(1) & D_l(2) & & \\ \vdots & & & \ddots & \\ C_l(p-1)A_l(p-2)\cdots A_l(1)B_l(0) & \cdots & \cdots & \cdots & D_l(p-1) \end{bmatrix} \tag{4.6}$$

Similar to ILC for LTI systems, some examples of L include

$$\begin{aligned} &\text{Transpose Law:} && L = \alpha \hat{P}_l^T \\ &\text{Partial Isometry Law:} && L = \alpha V_{svd}^T U_{svd} \\ &\text{Optimal Law:} && L = \alpha (\hat{P}_l^T Q \hat{P}_l + R)^{-1} \hat{P}_l^T Q \end{aligned} \tag{4.7}$$

The scalar α is a scaling factor to make the singular values of $I - P_l L$ less than unity, leading to monotonic decay of the tracking errors. The matrix $\hat{P}_l$ denotes the model used in ILC that contains model errors compared to the real system model P_l in Eq. (4.6). In the partial isomethry law, U_{svd} and V_{svd} are the singular vector matrices of $\hat{P}_l$. The optimal law, also

named as the quadratic learning law, is derived from minimizing a cost function

$$V_j = \frac{1}{2}\left[\underline{e}_j^T Q \underline{e}_j + (\underline{u}_j - \underline{u}_{j-1})^T R (\underline{u}_j - \underline{u}_{j-1})\right] \tag{4.8}$$

where Q and R are positive-definite weighting matrices. One advantage of the optimal learning law compared to other learning laws is that by adjusting Q and R, one can control the learning step size across iterations. This is essential for ILC algorithms for nonlinear systems based on linearized/bilinearized models because the linearized/bilinearized models are valid in the vicinity along the reference. However, when a big R is necessary, it will take an enormous number of iterations to learn the trajectory. In that case, a homotopy of desired trajectory is preferred instead to control the learning step size.

4.2.3 Optimization Formulation of ILC for a General System

The optimal learning law that minimizes the cost function Eq. (4.8) for a general system can be solved by the necessary condition for the optimality, i.e., $dV_j/d\underline{u}_j = 0$,

$$\begin{aligned}
\frac{dV_j}{d\underline{u}_j} &= \frac{1}{2}\left[\frac{d(\underline{e}_j^T Q \underline{e}_j)}{d\underline{e}_j}\frac{d\underline{e}_j}{d\underline{u}_j} + \frac{d(\underline{u}_j - \underline{u}_{j-1})^T R (\underline{u}_j - \underline{u}_{j-1})}{d\underline{u}_j}\right] \\
&= \underline{e}_j^T Q \left(\frac{d\underline{e}_j}{d\underline{u}_j}\right) + (\underline{u}_j - \underline{u}_{j-1})^T R \\
&= -(\underline{y}_d - f(\underline{u}_j) - \underline{d}_j)^T Q \nabla_{\underline{u}_j} f + (\underline{u}_j - \underline{u}_{j-1})^T R \\
&= -(\underline{e}_{j-1} + f(\underline{u}_{j-1}) - f(\underline{u}_j))^T Q \nabla_{\underline{u}_j} f + (\underline{u}_j - \underline{u}_{j-1})^T R \\
&= 0
\end{aligned} \tag{4.9}$$

where ∇f is the Jacobian of the vector function f and $\nabla_{\underline{u}_j} f$ denotes the Jacobian evaluated at $\underline{u}_j$,

$$\nabla_{\underline{u}_j} f = \frac{df}{d\underline{u}}(\underline{u}_j) = \begin{bmatrix} \frac{df(0)}{du(0)}(\underline{u}_j) & \frac{df(0)}{du(1)}(\underline{u}_j) & \cdots & \frac{df(0)}{du(p)}(\underline{u}_j) \\ \frac{df(1)}{du(0)}(\underline{u}_j) & \frac{df(1)}{du(1)}(\underline{u}_j) & \cdots & \frac{df(1)}{du(p)}(\underline{u}_j) \\ \vdots & \vdots & & \vdots \\ \frac{df(p)}{du(0)}(\underline{u}_j) & \frac{df(p)}{du(1)}(\underline{u}_j) & \cdots & \frac{df(p)}{du(p)}(\underline{u}_j) \end{bmatrix} \tag{4.10}$$

Therefore the optimal solution $\underline{u}_j^*$ should satisfy the equation

$$(\nabla_{\underline{u}_j^*} f)^T Q f(\underline{u}_j^*) + R\underline{u}_j^* = (\nabla_{\underline{u}_j^*} f)^T Q\Big(\underline{e}_{j-1} + f(\underline{u}_{j-1})\Big) + R\underline{u}_{j-1} \tag{4.11}$$

This is a nonlinear equation and can be numerically solved by Picard iteration. To summarize, the ILC algorithm by minimizing the cost function, Eq. (4.8), for a general system is presented in Table. 4.1.

Table 4.1: ILC algorithm for a General System

Initialize a feasible $\underline{u}_0$, Q, R, and $j = 0$. Record the output of the initial iteration, $\underline{y}_0$, and compute the error $\underline{e}_0$
ILC Iterations: Loop until the tracking error decreases below the desired threshold. For iteration j (starting from $j = 1$): Numerically compute $\underline{u}_j^*$ according to Eq. (4.11) Run the system with $\underline{u}_j^*$, measure the system output $\underline{y}_j$, and compute the tracking error $\underline{e}_j$. $j = j + 1$.

However, because of the complexity of the system dynamics, a general model for the entire trajectory of the form as in Eq. (4.1) may not be analytically computed, such as a nonlinear

system model in the form of neural networks. One method is to use a linearized model to approximate the nonlinear system along the previous iteration. The ILC algorithm can be simplied into Table 4.2.

Table 4.2: ILC algorithm for nonlinear systems based on linearized models

Initialize a feasible $\underline{u}_0$, Q, R, α, $j = 0$, and the threshold for the tracking error.
Loop until the tracking error decreases below the desired threshold. For iteration j (starting from $j = 0$):
(1) Run the system with $\underline{u}_j$, measure the system output $\underline{y}_j$, and compute the tracking error $\underline{e}_j$.
(2) Linearize the nonlinear model along $\underline{u}_j$ and $\underline{x}_{j,m}$ where m denotes the component of x; Compute the P_j^l based on the linearized model
(3) Update $\underline{u}_{j+1}$ according to some ILC law (e.g., Eq. (4.7)) based on the linearized model P_j^l.
The loop number of ILC iteration: $j = j + 1$.

However, a more accurate model can be obtained by Carleman bilinearization, which can approximate an nonlinear input-affine system to arbitrarily high accuracy at the cost of high dimensionality [65, 66]. This more accurate model including the nonlinear dynamics can lead to faster convergence. In this paper, the ILC methods based on this Carleman bilinearized model is developed and studied.

4.3 Discrete Bilinearized Model to Approximate Nonlinear Dynamics

To compute a bilinear model to approximate the nonlinear dynamics for ILC designs, it generally includes two steps: (1) rewriting the system in terms of the δ variables with respect to a reference, (2) Carleman bilinearization, and (3) discretization.

4.3.1 The Nonlinear δ-Model with Respect to a Reference Trajectory

The state in the δ space is the difference between the state $x_j(t)$ with respect to the state $x_r(t)$ of the reference; similarly defined are the control signal and the system output,

$$\begin{aligned} \delta x_j(t) &\triangleq x_j(t) - x_r(t) \\ \delta u_j(t) &\triangleq u_j(t) - u_r(t) \\ \delta y_j(t) &\triangleq y_j(t) - y_r(t) \end{aligned} \tag{4.12}$$

By plugging $x_j(t) = \delta x_j(t) + x_r(t)$, $u_j(t) = \delta u_j(t) + u_r(t)$, and $y_j(t) = \delta y_j(t) + y_r(t)$ into the nonlinear model, the δ-model of a nonlinear system can be obtained, of which $x_r(t)$, $u_r(t)$, and $y_r(t)$ are the parameters inside of the state matrices. In ILC, the previous trajectory is used as the reference trajectory, that is, $x_r(t) = x_{j-1}(t)$, $u_r(t) = u_{j-1}(t)$, and $y_r(t) = y_{j-1}(t)$. This leads to

$$\begin{aligned} \delta x_{j-1}(t) &= x_{j-1}(t) - x_r(t) = x_{j-1}(t) - x_{j-1}(t) = 0 \\ \delta u_{j-1}(t) &= u_{j-1}(t) - u_r(t) = u_{j-1}(t) - u_{j-1}(t) = 0 \\ \delta y_{j-1}(t) &= y_{j-1}(t) - y_r(t) = y_{j-1}(t) - y_{j-1}(t) = 0 \end{aligned} \tag{4.13}$$

4.3.2 Carleman Bilinearization

Carleman bilinearization is an approach that constructs a bilinear model to approximate nonlinear input-affine dynamics at the expense of high dimensionality, i.e., adding new states that are the higher order terms of the Taylor expansion of the input-output relationship. Bilinearization along equilibrium points is presented in Reference [65]. Carleman Bilinearization of the δ-model is presented through the following example, in which iteration $j-1$ is the

reference trajectory.

If the original states of the nonlinear system are $x(t) = \begin{bmatrix} x_1(t) & x_2(t) \end{bmatrix}^T$, the new states of the bilinearized model at j^{th} iteration are made from the terms in the Taylor expansion with respect to $x_{j-1}(t) = \begin{bmatrix} x_{j-1,1}(t) & x_{j-1,2}(t) \end{bmatrix}^T$, that is,

$$\begin{aligned} &\delta x_{j,1},\ \delta x_{j,2}, \\ &(\delta x_{j,1})^2,\ (\delta x_{j,2})^2,\ \delta x_{j,1}\delta x_{j,2}, \\ &(\delta x_{j,1})^3,\ (\delta x_{j,1})^2\delta x_{j,2},\ \delta x_{j,1}(\delta x_{j,2})^2,\ (\delta x_{j,2})^3, \\ &\vdots \end{aligned} \tag{4.14}$$

The bilinearized model is obtained by taking the derivatives of the new states and ignoring the higher order terms. For instance, to compute a second-order Carleman bilinearized model, all the third and higher order terms are assumed zero,

$$z_j(t) = \begin{bmatrix} \delta x_{j,1} & \delta x_{j,2} & (\delta x_{j,1})^2 & \delta x_{j,1}\delta x_{j,2} & (\delta x_{j,2})^2 \end{bmatrix}^T \tag{4.15}$$

The resultant continuous bilinearized model with $z_j(t)$ denoting the new state of the bilinearized model is of the form

$$\begin{aligned} \dot{z}_j(t) &= A_j^c z_j(t) + B_j^c \delta u_j(t) + \sum_{i=1}^{m} N_{j,i}^c z_j(t)\delta u_{j,i}(t) \\ y_j(t) &= C_j^c z_j(t) + D_j^c \delta u_j(t) \end{aligned} \tag{4.16}$$

where A_j^c, B_j^c, $N_{j,i}^c$, C_j^c, and D_j^c are iteration-dependent, i.e., functions of $u_{j-1}(t)$ and $x_{j-1}(t)$.

The vicinity where this bilinearized model is valid requires $|\delta x_{j,i}| < 1$. Compared with

linearized models, the model error in a bilinearized model is $O(\delta x_{j,i}^n)$, where $n > 2$ is the number of order one chosen to ignore. This implies that outside the area of $|\delta x_{j,i}| < 1$, the bilinearized model contains bigger model errors than the linearized model, while inside of the area of $|\delta x_{j,i}| < 1$, the bilinearized model is more accurate than the linearized model by including higher order dynamics.

4.3.3 Discretization

Because ILC needs to store the output measurements of the previous runs, discretization of the continuous model is essential. Reference [67] has discussed and compared different discretization methods. Here we use the forward Euler method which results in a discrete bilinear model of the form

$$\begin{aligned} z_j(k+1) &= A_j(k)z_j(k) + B_j(k)\delta u_j(k) + \sum_{i=1}^{m} N_{j,i}(k)z_j(k)\delta u_{j,i}(k) \\ y_j(k) &= C_j(k)z_j(k) + D_j(k)\delta u_j(k) \end{aligned} \tag{4.17}$$

where with the sampling period T_s

$$\begin{aligned} A_j(k) &= I + T_s A_j^c(kT_s) \\ B_j(k) &= T_s B_j^c(kT_s) \\ N_{j,i}(k) &= T_s N_{j,i}^c(kT_s) \\ C_j(k) &= C_j^c(kT_s) \\ D_j(k) &= D_j^c(kT_s) \end{aligned} \tag{4.18}$$

4.4 ILC for a Nonlinear System Based on Its Bilinearized Model

4.4.1 ILC Algorithms for Bilinear Systems

The Input-Output Relationship of the Bilinearized Model

For simplicity, the subscript of the iteration number is ignored and the bilinearized model Eq. (4.17) can be written as a time-varying bilinear system,

$$\begin{aligned} z(k+1) &= A(k)z(k) + B(k)\delta u(k) + \sum_{i=1}^{m} N_i(k)z(k)\delta u_i(k), \\ y(k) &= C(k)z(k) + D(k)\delta u(k) \end{aligned} \tag{4.19}$$

By defining,

$$N_z(k) = \begin{bmatrix} N_1 z(k) & N_2 z(k) & \cdots & N_m z(k) \end{bmatrix} \tag{4.20}$$

which is actually a function of $\delta \underline{u}$, the input-output relationship for one iteration, the vector function f in Eq. (4.1) can be written as

$$f(\delta \underline{u}) = P(\delta \underline{u})\delta \underline{u} \tag{4.21}$$

where the $(k+1, i+1)$ component of $P(\delta \underline{u})$ is

$$P_{k+1,i+1} = \begin{cases} C(k)A(k-1)\cdots A(i+1)(B(i)+N_z(i)), & p-1 \geq k > i+1 \\ C(i+1)(B(i)+N_z(i)), & k = i+1 \\ D(i), & k = i \\ 0, & 0 \leq k < i \end{cases} \tag{4.22}$$

and where $i = 0, 1, 2, \cdots, p-1$. The $(k+1, i+1)$ component of its Jacobian ∇f, i.e., $P_J(\delta \underline{u})$ can be calculated according to

$$P_{J,(k+1,i+1)} = \begin{cases} C(k)\tilde{A}(k-1)\cdots \tilde{A}(i+1)(B(i)+N_z(i)), & p-1 \geq k > i+1 \\ C(i+1)(B(i)+N_z(i)), & k = i+1 \\ D(i), & k = i \\ 0, & 0 \leq k < i \end{cases} \tag{4.23}$$

where $\tilde{A}_k = A(k) + \sum_{i=1}^{m} N_i(k)u_i(k)$ and $i = 0, 1, 2, \cdots, p-1$.

The Optimal Solution of Iteration j

In ILC, the bilinearized model is iteration-dependent (Eq. (4.17)), i.e., $P = P_j$ and $P_J = P_{J,j}$. By plugging $f = P_j$ and $\nabla f = P_{J,j}$ into Eq. (4.11), the optimal solution for iteration j is then

$$\delta \underline{u}_j^* = \Big((P_{J,j}^*)^T Q(P_j^*) + R\Big)^{-1} \Big[(P_{J,j}^*)^T Q \delta \underline{e}_{j-1} + \Big((P_{J,j}^*)^T Q P_{j-1} + R\Big)\delta \underline{u}_{j-1}\Big] \tag{4.24}$$

and this can be solved by Picard iteration, which is,

$$\delta \underline{u}_{j,k} = \Big((P_{J,j,k-1})^T Q(P_{j,k-1}) + R\Big)^{-1} \Big[(P_{J,j,k-1})^T Q \delta \underline{e}_{j-1} + \Big((P_{J,j,k-1})^T Q P_{j-1} + R\Big) \delta \underline{u}_{j-1}\Big] \tag{4.25}$$

When the previous trajectory is chosen as the reference trajectory, the tracking error in the δ space is the same as the original tracking error, that is,

$$\delta \underline{e}_j = (\underline{y}_d - \underline{y}^r) - (\underline{y}_j - \underline{y}^r) = \underline{y}_d - \underline{y}_j = \underline{e}_j \tag{4.26}$$

and $\delta u_{j-1} = 0$. Therefore, Eq. (4.24) becomes

$$\delta \underline{u}_j^* = \Big((P_{J,j}^*)^T Q(P_j^*) + R\Big)^{-1} (P_{J,j}^*)^T Q \underline{e}_{j-1} \tag{4.27}$$

which can be solved by Picard iteration, (corresponding to Eq. (4.25))

$$\delta \underline{u}_{j,k} = \Big((P_{J,j,k-1})^T Q(P_{j,k-1}) + R\Big)^{-1} (P_{J,j,k-1})^T Q \underline{e}_{j-1} \tag{4.28}$$

The input to the nonlinear system for the j^{th} iteration is

$$\underline{u}_j^* = \delta \underline{u}_j^* + \underline{u}_{j-1} \tag{4.29}$$

4.4.2 ILC Algorithms for Nonlinear Systems based on a Bilinearized Model

The ILC algorithm for nonlinear systems based on a bilinearized model is summarized in Table 4.3

Table 4.3: ILC algorithm for nonlinear systems based on a bilinearized model

Initialize a feasible $\underline{u}_0$, Q, R, α, $j = 0$, and the threshold for the tracking error.
Loop until the tracking error decreases below the desired threshold. For iteration j (starting from $j = 0$):
(1) Run the system with $\underline{u}_j$, measure the system output $\underline{y}_j$, and compute the tracking error $\underline{e}_j$.
(2) To compute the $\underline{u}^*_{j+1}$ for iteration $j + 1$ (Initialize the threshold for Picard iteration): (a) Bilinearize, discretize, and normalize the nonlinear model along $\underline{u}^r = \underline{u}_j$ and $\underline{x}^r_m = \underline{x}_{j,m}$ where m denotes the component of x: Compute the bilinearized model A_{j+1}, B_{j+1}, C_{j+1}, D_{j+1}, and N_{j+1}'s Compute the initial $P_{j+1,0}$ and $P_{J,j+1,0}$ with $\delta\underline{u}_{j+1,0} = 0$. (b) For Picard iteration k (starting from $k = 1$): Compute $\delta\underline{u}_{j+1,k}$ according to Eq. (4.28). Update $P_{j+1,k}$ and $P_{J,j+1,k}$ by using $\delta\underline{u}_{j+1,k}$. The loop number of Picard iteration: $k = k + 1$ until $\delta\underline{u}_{j+1,k} - \delta\underline{u}_{j+1,k-1}$ is smaller than some pre-defined threshold. the optimal solution $\delta\underline{u}^*_{j+1} = \delta\underline{u}_{j+1,k}$.
(3) Update $\underline{u}^*_{j+1} = \underline{u}_j + \delta\underline{u}^*_{j+1}$.
The loop number of ILC iteration: $j = j + 1$.

4.4.3 Homotopy of the Desired Trajectorie to Guide the Learning

Because the bilinearized model is a good approximation of the nonlinear dynamics only inside of the close vicinity ($|\delta x| < 1$) along the reference trajectory, the learning step size needs to be limited around the boundary of the vicinity. The choice of Q and R can adjust the learning step size, however, to a limited extent. Here we propose another method: applying a homotopy of

the desired trajectory.

The homotopy of the desired trajectory is a series of pseudo-desired trajectories that gradually leads to the desired trajectory. For iteration j, the pseudo-desired trajectory $\underline{y}_{d,j} = \underline{y}_d - (1-\alpha_j)\underline{e}_{j-1}$, where α_j is computed based on the linearized model as follows.

Given $\underline{e}_{j-1}$, one can compute $\delta\underline{u}_j^{<est>}$ based on the linearized model of the nonlinear system according to the optimal linear ILC law,

$$\delta\underline{u}_j^{<est>} = [(P_{l,j})^T Q P_{l,j} + R]^{-1}(P_{l,j})^T Q \underline{e}_{j-1} \tag{4.30}$$

where $P_{l,j}$ is the P_l matrix in Eq. (4.6) of the linearized model for the j^{th} iteration along the previous trajectory. The scaling factor α_j is chosen as the inverse of the maximum absolute value of $\delta\underline{u}_j^{<est>}$, for example, for a single input system

$$\alpha_j = \frac{1}{max(\delta\underline{u}_j^{<est>})} \tag{4.31}$$

Plugging $\underline{e}_j = \underline{y}_{d,j} - \underline{y}_j$ into the cost function Eq. (4.9), the learning law with homotopy based on bilinear systems becomes

$$\delta\underline{u}_j^* = \left((P_{J,j}^*)^T Q(P_j^*) + R\right)^{-1}(P_{J,j}^*)^T Q \alpha_j \underline{e}_{j-1} \tag{4.32}$$

and the learning law based on linear systems with homotopy becomes

$$\delta\underline{u}_j = \alpha_j L \underline{e}_{j-1} \tag{4.33}$$

4.5 Numerical Examples

4.5.1 Rigid Body Rotation Problem

The Nonlinear Model with Embedded Feedback Controllers:

The rotation of a rigid body can be described by Euler's rotation equations

$$\underline{I\dot{\omega}} + \underline{\omega} \times (\underline{I\omega}) = \underline{M} \tag{4.34}$$

where $\underline{M}$ is the applied torque vector, $\underline{I}$ is the inertia, and $\underline{\omega}$ is the angular velocity about the principal axes. In 3D principal orthogonal coordinates, $\underline{M} = \begin{bmatrix} M_1 & M_2 & M_3 \end{bmatrix}^T$, $\underline{I} = diag(I_1\ I_2\ I_3)$, and $\underline{\omega} = \begin{bmatrix} \omega_1 & \omega_2 & \omega_3 \end{bmatrix}^T$, the Euler equations become

$$\begin{aligned} I_1\dot{\omega}_1 + (I_3 - I_2)\omega_2\omega_3 &= M_1 \\ I_2\dot{\omega}_2 + (I_1 - I_3)\omega_3\omega_1 &= M_2 \\ I_3\dot{\omega}_3 + (I_2 - I_1)\omega_1\omega_2 &= M_3 \end{aligned} \tag{4.35}$$

The tracking problem to be solved is to design ILC controllers for all three axes applied outside given feedback controllers that enables the rigid body to rotate with the pre-defined angular velocities. The feedback controllers are needed to stabilize the system and ILC controllers can eliminate tracking errors from the bandwidth limitation of these feedback controllers.

The driving torques with feedback controllers and ILC controllers are

$$M_i = -b_i\omega_i + f_i \tag{4.36}$$

where the subcript $i = 1,\ 2,\ 3$ denotes the principal axis, b_i is pre-designed feedback gain and the feedforward signals f_i needs to be designed by ILC.

Plugging Eq. (4.36) into Eq. (4.35), the Euler equations then become

$$\begin{aligned}
\dot{\omega}_1 &= -a_1\omega_2\omega_3 - c_1\omega_1 + u_1 \\
\dot{\omega}_2 &= -a_2\omega_1\omega_3 - c_2\omega_2 + u_2 \\
\dot{\omega}_3 &= -a_3\omega_1\omega_2 - c_3\omega_3 + u_3
\end{aligned} \tag{4.37}$$

where

$$a_1 = \frac{I_3 - I_2}{I_1} \qquad a_2 = \frac{I_1 - I_3}{I_2} \qquad a_3 = \frac{I_2 - I_1}{I_3} \qquad c_i = \frac{b_i}{I_i} \qquad u_i = \frac{f_i}{I_i} \tag{4.38}$$

The Nonlinear Model in δ Space:

Given an input-output-history pair, $\underline{u}^r$ and $\underline{\omega}^r$, the δ variables are defined as

$$\delta\underline{\omega} = \begin{bmatrix} \delta\omega_1 \\ \delta\omega_2 \\ \delta\omega_3 \end{bmatrix} \triangleq \underline{\omega} - \underline{\omega}^r \qquad \delta\underline{u} \ = \begin{bmatrix} \delta u_1 \\ \delta u_2 \\ \delta u_3 \end{bmatrix} \triangleq \underline{u} - \underline{u}^r \tag{4.39}$$

The system dynamics Eq. (4.37) become

$$\begin{bmatrix} \delta\dot{\omega}_1 \\ \delta\dot{\omega}_2 \\ \delta\dot{\omega}_3 \end{bmatrix} = \begin{bmatrix} -c_1 & & \\ & -c_2 & \\ & & -c_3 \end{bmatrix} \begin{bmatrix} \delta\omega_1 \\ \delta\omega_2 \\ \delta\omega_3 \end{bmatrix} - \begin{bmatrix} a_1(\omega_2^r\delta\omega_3 + \omega_3^r\delta\omega_2 + \delta\omega_2\delta\omega_3) \\ a_2(\omega_1^r\delta\omega_3 + \omega_3^r\delta\omega_1 + \delta\omega_1\delta\omega_3) \\ a_3(\omega_1^r\delta\omega_2 + \omega_2^r\delta\omega_1 + \delta\omega_1\delta\omega_2) \end{bmatrix} + \begin{bmatrix} \delta u_1 \\ \delta u_2 \\ \delta u_3 \end{bmatrix} \tag{4.40}$$

4.5.2 Linearized Model and Bilinearized Model used in ILC

Discrete Linearized Model:

$$\begin{bmatrix} \delta\dot{\omega}_1 \\ \delta\dot{\omega}_2 \\ \delta\dot{\omega}_3 \end{bmatrix} = - \begin{bmatrix} c_1 & a_1\omega_3^r & a_1\omega_2^r \\ a_2\omega_3^r & c_2 & a_2\omega_1^r \\ a_3\omega_2^r & a_3\omega_1^r & c_3 \end{bmatrix} \begin{bmatrix} \delta\omega_1 \\ \delta\omega_2 \\ \delta\omega_3 \end{bmatrix} + \begin{bmatrix} \delta u_1 \\ \delta u_2 \\ \delta u_3 \end{bmatrix} \tag{4.41}$$

This linearized model then can be discretized by the forward Euler method

$$\begin{bmatrix} \delta\omega_1(k+1) \\ \delta\omega_2(k+1) \\ \delta\omega_3(k+1) \end{bmatrix} = (I - T_s) \begin{bmatrix} c_1 & a_1\omega_3^r(k) & a_1\omega_2^r(k) \\ a_2\omega_3^r(k) & c_2 & a_2\omega_1^r(k) \\ a_3\omega_2^r(k) & a_3\omega_1^r(k) & c_3 \end{bmatrix} \begin{bmatrix} \delta\omega_1(k) \\ \delta\omega_2(k) \\ \delta\omega_3(k) \end{bmatrix} + T_s \begin{bmatrix} \delta u_1(k) \\ \delta u_2(k) \\ \delta u_3(k) \end{bmatrix} \tag{4.42}$$

Discrete Carleman Bilinearized Model:

By defining the state variable $z(t)$ as

$$z(t) = \begin{bmatrix} \delta\omega_1 & \delta\omega_2 & \delta\omega_3 & (\delta\omega_1)^2 & (\delta\omega_2)^2 & (\delta\omega_3)^2 & (\delta\omega_1)(\delta\omega_2) & (\delta\omega_1)(\delta\omega_3) & (\delta\omega_2)(\delta\omega_3) \end{bmatrix}^T \tag{4.43}$$

the model obtained by the 2nd-order Carleman bilinearization (after discretization by the forward Euler method) is

$$\begin{aligned} z(k+1) &= (I + T_s A_c(k))z(k) + T_s B_c(k)\delta\underline{u}(k) + \sum_{i=1}^{3} T_s N_{ic}(k)z(k)\delta u_i(k) \\ \underline{\omega}(k) &= C_c z(k) + D_c \delta\underline{u}(k) + \underline{\omega}^r(k) \end{aligned} \tag{4.44}$$

where

$$B_c(1,1) = B_c(2,2) = B_c(3,3) = C_c(1,1) = C_c(2,2) = C_c(3,3) = 1,$$

$$N_{1,c}(4,1) = 2,\ N_{1,c}(7,2) = N_{1,c}(8,1) = 1,$$

$$N_{2,c}(5,2) = 2,\ N_{2,c}(7,1) = N_{2,c}(9,3) = 1,$$

$$N_{3,c}(6,3) = 2,\ N_{2,c}(8,1) = N_{3,c}(9,2) = 1,$$

$$A_c = \begin{bmatrix} -c_1 & -a_1\omega_3^r & -a_1\omega_2^r & 0 & 0 & 0 & 0 & 0 & -a_1 \\ -a_2\omega_3^r & -c_2 & -a_2\omega_1^r & 0 & 0 & 0 & 0 & -a_2 & 0 \\ -a_3\omega_2^r & -a_3\omega_1^r & -c_3 & 0 & 0 & 0 & -a_3 & 0 & 0 \\ 0 & 0 & 0 & -2c_1 & 0 & 0 & -2a_1\omega_3^r & -2a_1\omega_2^r & 0 \\ 0 & 0 & 0 & 0 & -2c_2 & 0 & -2a_2\omega_3^r & 0 & -2a_2\omega_1^r \\ 0 & 0 & 0 & 0 & 0 & -2c_3 & 0 & -2a_3\omega_2^r & -2a_3\omega_1^r \\ 0 & 0 & 0 & -a_2\omega_3^r & -a_1\omega_3^r & 0 & -(c_1+c_2) & -a_2\omega_1^r & -a_1\omega_2^r \\ 0 & 0 & 0 & -a_3\omega_2^r & 0 & -a_1\omega_2^r & -a_3\omega_1^r & -(c_1+c_3) & -a_1\omega_3^r \\ 0 & 0 & 0 & 0 & -a_3\omega_1^r & -a_2\omega_1^r & -a_3\omega_2^r & -a_2\omega_3^r & -(c_2+c_3) \end{bmatrix}$$

4.5.3 Simulations and Results

Optimal ILC methods based on the linearized model Eq. (4.42) and the bilinearized model Eq. (4.44) are tested in simulations. The first numerical experiment is to compare the linear and bilinear ILC methods without using the homotopy while increasing the amplitude of a sinusoidal desired trajectory. As the amplitude is increasing, one of these two methods should fail to improve the tracking accuracy, or even diverge. The second simulation is to track the trajectory that cannot be learned in Simulation 1, by applying a homotopy of the desired trajectory using the bilinear method and the linear method. The method of adjusting R to learn the same trajectory is also included to show the its limitation.

The numerical tests are performed in MATLAB and the nonlinear dynamics is simulated by ode113 for each time step. The learning methods tested are summarized in Table 4.4 and 4.5. Learning curves, i.e., the RMS error of each iteration as the iteration number increases, are plotted to compare the convergence rates of the tested ILC methods.

Table 4.4: ILC Methods Tested in Simulation 1

	Linear Method	Bilinear Method
Model in ILC	Eq. (4.42)	Eq. (4.44)
ILC Algorithm	Table 4.2	Table 4.3
ILC Learning Law	$L = [(P_{l,j})^T Q P_{l,j} + R]^{-1} (P_{l,j})^T Q$	Eq. (4.27)

Table 4.5: ILC Methods Tested in Simulation 2

	Homotopy Linear	Homotopy Bilinear	Bilinear with Big R
Model in ILC	Eq. (4.42)	Eq. (4.44)	Eq. (4.44)
ILC Algorithm	Table 4.2	Table 4.3	Table 4.3
ILC Learning Law	$\delta u_j = \alpha_j L \underline{e}_{j-1}$	Eq. (4.32)	Eq. (4.27)

Parameters in the simulations are $\underline{I} = \begin{bmatrix} 30 & 20 & 10 \end{bmatrix}^T$, $b_1 = b_2 = b_3 = 3$, $p = 200$, $T_s = 0.01$.

Simulation 1: Learning Desired Trajectories without Guidance by a Homotopy

The desired trajectories are chosen as

$$
\begin{aligned}
y_{d,1} &= 0.5 * [3 - cos(\frac{5.5\pi}{pT_s}kT) - cos(\frac{1.5\pi}{pT_s}kT) - cos(\frac{15\pi}{pT_s}kT)] + 2 \\
y_{d,2} &= k * [3 - cos(\frac{5.5\pi}{pT_s}kT) - cos(\frac{1.5\pi}{pT_s}kT) - cos(\frac{15\pi}{pT_s}kT)] + 2 \\
y_{d,3} &= 0.5 * [1 - cos(\frac{17\pi}{pT_s}kT)] + 2
\end{aligned}
\tag{4.45}
$$

where $y_{d,i}$ is the desired angular velocity for axis i and the amplitude of $\underline{y}_{d,2}$ for the second axis is increased gradually (i.e., the convergence region becomes more twisted when only considering the size of δx), while the other coefficients are fixed. A deviation from the origin (the equilibrium point for the system) is added to make the tracking harder. This set of trajectories can represent the problem when a rotating body needs to change its angular velocity. All the tests start with the same initial command history $\underline{u}_0$ created by randn. In the optimal learning law, $Q = I$, and $R = 0$ is used. Note that when $R = 0$, it is asking for the inverse solution in only one iteration, which refers to the fastest convergence rate the method could possibly achieve and the least robust to model errors.

Table 4.6: RMS error of the first 8 iterations by bilinear method with $k = 1.274$

Iteration Number	1	2	3	4	5	6	7	8
Axis 1	1.99	19.26	42.90	21.77	85.02	103.12	32.31	74.46
Axis 2	7.49	3.64	24.72	66.41	35.94	15.30	2.29	82.45
Axis 3	1.41	3.46	20.90	11.27	125.62	29.93	123.97	4.15

The amplitude k of $\underline{y}_{d,2}$ is increasing gradually from 1 with step size 0.001. At $k = 1.274$, Method 2 (the bilinear method) failed to decrease the tracking error and the RMS errors jumps approximately between 5 to 200, the first 8 iterations of which are listed in Table 4.6.

From $k = 1$ to $k = 1.274$, the bilinear method initially shows a faster convergence rate than the linear method. As k increases, the bilinear method is approaching the boundary of its convergence region where it needs more iterations to converge. The final tracking accuracy for all cases remains the same, between 10^{-14} to 10^{-13} as long as both methods converge. The learning curves of four values of $k \in (1,\ 1.1,\ 1.265,\ 1.27)$ are plotted to demonstrate this trend in the right figures of Figure 4.1 and their corresponding desired trajectories Eq. (4.45) are plotted in the left ones.

Simulation 2: Learning a Desired Trajectory that Necessitates Guidance by a Homotopy

The amplitude of the desired trajectories for all three axis are increased to

$$\begin{aligned} y_{d,1} &= [3 - cos(\frac{5.5\pi}{pT_s}kT) - cos(\frac{1.5\pi}{pT_s}kT) - cos(\frac{15\pi}{pT_s}kT)] + 5 \\ y_{d,2} &= 5 * [3 - cos(\frac{5.5\pi}{pT_s}kT) - cos(\frac{1.5\pi}{pT_s}kT) - cos(\frac{15\pi}{pT_s}kT)] + 4 \\ y_{d,3} &= 7 * [1 - cos(\frac{17\pi}{pT_s}kT)] + 6 \end{aligned} \tag{4.46}$$

which are plotted in Figure 4.2(a).

For the homotopy linear and bilinear methods, $Q = I$, and $R = 0$ is used. For the bilinear method with adjustment of R, the initial R is chosen to be 10 and decreases following $R_{j+1} = 0.99R_j$, which means at iteration 500, $R_{700} \approx 0.008$.

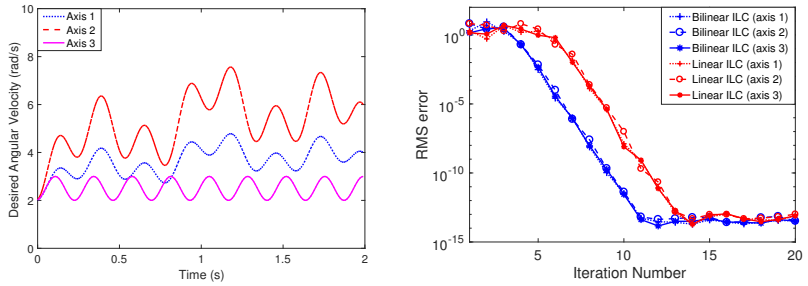
The learning curves of these three methods are plotted in Figure 4.2(b). The bilinear method with the homotopy (blue lines) converges to 10^{-13} around iteration 430, while the linear method with homotopy (red lines) needs 560 iterations. The bilinear method with ho-

motopty needs 130 iterations less to learn to follow the desired angular velocities. By adjusting R, the tracking error converges slightly to the magnitude level of 10^0 and then the Picard iteration failed to converge with $R_{380} \approx 0.2217$, which demonstrates the limitation of adjusting R to control the learning step size for nonlinear systems.

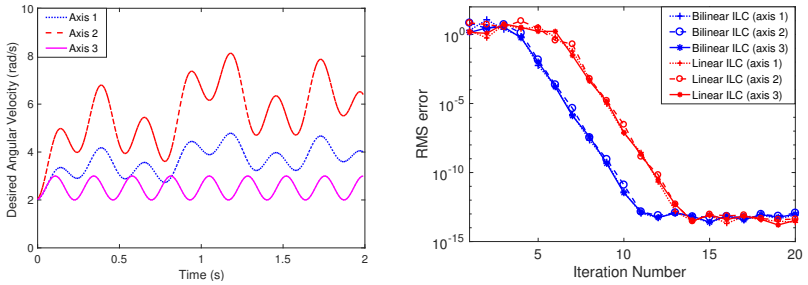
4.6 Conclusions

This section starts with a general ILC method by minimizing a cost function consisting of a quadratic term of the tracking error and a penalty of the difference of control actions between two successive iterations, which can be numerically solved by Picard iteration. To apply this general method to nonlinear systems, one needs to compute the input-output relationship for the entire trajectory for nonlinear systems, which is a tedious task. On the other hand, one can use a Carleman bilinearized model to approximate a nonlinear system to arbitrarily high accuracy, at the expense of increased dimensionality. Hence, the ILC method for bilinear systems is developed based on the general ILC method. The application of this bilinear ILC approach to nonlinear input-affine systems is then studied as well as its advantages and limitations when compared with the ILC method based on linearized models. Because of more nonlinear dynamics captured by the bilinear model, it converges slightly faster than the method using a linearized model. At the same time, the bilinear method is more sensitive to the model error caused by δx bigger than unity. Therefore, limitation of the learning step size is important. For a trajectory outside the convergence region, one can apply the homotopy technique, that is, by specifying a series of trajectories that eventually converges to the final desired trajectory to guide the ILC process. Using this technique, the bilinear method can

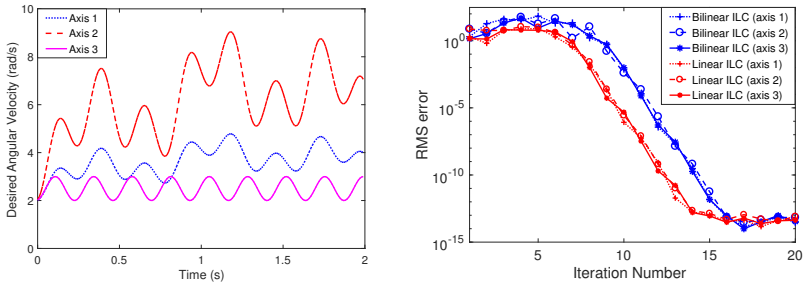
learn to follow the desired trajectory outside the convergence region with a faster learning rate compared with the correspond linear method.



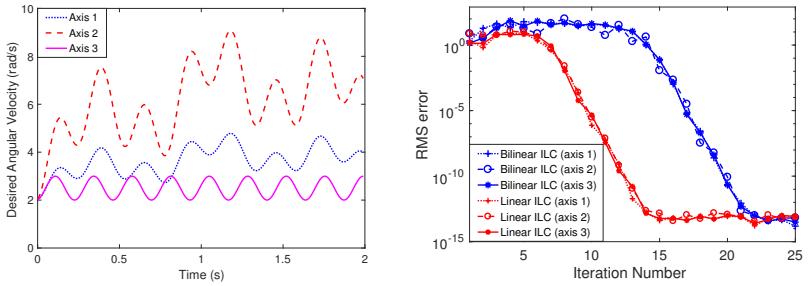
(a) Desired Trajectory (k=1) (b) Learning curves (k=1)



(c) Desired Trajectory (k=1.1) (d) Learning curves (k=1.1))

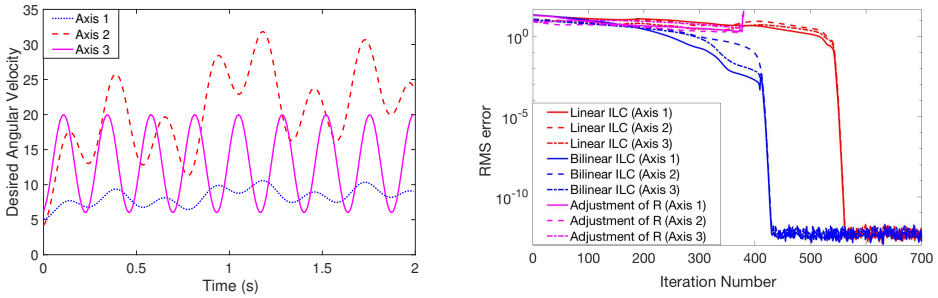


(e) Desired Trajectory (k=1.265) (f) Learning curves (k=1.265)



(g) Desired Trajectory (k=1.27) (h) Learning curves (k=1.27)

Figure 4.1: The learning curves as the amplitude of the desired trajectory for (i.e., k) Axis 2 increases.



(a) Desired Trajectories (b) Learning curves

Figure 4.2: The learning curves of methods with the homotpy.

Chapter 5

Data-Driven Linear ILC using Reinforcement Learning

5.1 Introduction

This chapter focuses on data-driven model-free ILC algorithms. The classical model-free ILC algorithm is the Arimoto-type learning law, which was originally proposed for continuous systems in [2, 34].

A possible version of this Arimoto-type ILC for discrete-time systems is of the form

$$\underline{u}_{j+1} = \underline{u}_j + \alpha \underline{e}_j \tag{5.1}$$

where

$$\underline{u}_j = \begin{bmatrix} u_j(0) & u_j(1) & \cdots & u_j(p-1) \end{bmatrix}^T \tag{5.2}$$

These Arimoto-type learning laws suffer from the bad learning transients as analyzed in [9,

35], which means asymptotical but not monotonic decay of the tracking error. Approaches to adjust the Arimoto-type learning law for the monotonic convergence include applying a linear quadratic tracker to condition the plant in [68], a time-varying learning gain [69] and using a zero-phase filter in [70]. However, these adjustments necessitate the knowledge of a system model.

Recently some other model-free ILC methods are proposed. For example, Reference [71, 72, 73] use a finite impulse response (FIR) filter to approximate the system output in terms of the outputs from previous iterations, i.e., the system output of the next iteration is written as the output of the current iteration added by the convolution summation of the FIR filter and the output of some previous iteration. The parameters of this FIR filter are estimated by minimizing the quadratic term of the difference between the estimated output and the desired output. However, the improvement of tracking accuracy by this method is limited because of the limited number of parameters of the FIR filter. The same group then developed another model-free method for LTI systems by estimating the system P matrix [74], which can be corrupted by noises and disturbances and thus lead to divergence.

References [75, 76, 77] used a cost function instead to evaluate the control command and the gradient descent method was used to update the command, the process of which needed numerical methods such as the Broyden-Fletcher-Goldfarb-Shanno method to approximate the Hessian matrix of the objective function (i.e., the cost function). This method is similar to the principle of reinforcement learning and without taking advantage of knowledge of model-based ILC algorithms, the convergence of this method is hard to be predicted; there is little control of the learning transients.

Generally speaking, model-free RL algorithms have little control of the learning process.

Directly applying RL for ILC problems is likely to be infeasible in practice due to safety concerns, not to mention the number of iterations needed.

On the other hand, a data-driven model predictive control (MPC) algorithm has been developed in [31], which can be seen as a Q-learning method in some extent. By rewriting and extending this MPC algorithm as an off-policy RL method, and adjusting for ILC problems, we develop a data-driven model-free ILC algorithm by RL in the trail domain for linear systems which can avoid bad learning transients. This method has the following distinct features.

(1) A quadratic parameterized value function: The value function for RL is defined as a cumulative cost function that consists of the quadratic costs of the current iteration and the future iterations. For linear systems, this can be of a quadratic form of the error and control differences for the current iteration and the future control actions together with the system dynamics which are implicitly included in the parameters.

(2) A ILC learning law that allows the improvement of tracking accuracy and collection of data simultaneously: As the goal is to improve the tracking accuracy instead of an accurate estimation of the value function, the tracking error can still converge to a desired level as long as there is a contraction mapping between two successive iterations.

(3) Batch update of the value function by least squares method: Least squares is chosen here instead of gradient descent or semi-gradient descent due to safety concerns.

(4) An off-policy RL method: "Off-policy" means that one can use an L that is safe for learning in practice while estimating the value function for a different L. This ensures that one can collect enough data for the estimation of the value function without safety concerns.

This chapter first introduces the notations in RL and ILC. The data-driven model-free ILC algorithm for linear systems is then presented in the RL language. A numerical example is

performed to demonstrate the feasibility of this method. One limitation of this method is the curse of dimensionality, which can be relaxed by function approximations.

5.2 Problem Formulation

For two successive trials, the error transmission can be described by

$$\underline{e}_{j+1} = \underline{e}_j - P\underline{\delta}_j \tag{5.3}$$

where

$$\underline{\delta}_j = \underline{u}_{j+1} - \underline{u}_j \tag{5.4}$$

and a general linear ILC learning law is of the form

$$\underline{\delta}_j = L_j\underline{e}_j \tag{5.5}$$

Because ILC approaches the inverse solution, it is possible that the inverse problem is ill-posed. Actually, Reference [78] states that for any zero-order-hold sampled systems with order higher than 3, P is ill-conditioned and the inverse problem is ill-posed if the sample rate is not particularly slow. Reference [79] suggests several approaches to find a stable inverse. One method is to not ask for zero tracking error at the first few time steps, the number of which is equal to the number of bad singular values of P. For example, if there is one bad singular value, the learning matrix L then becomes a p by $(p-1)$ matrix and Eq. (5.5) becomes

$$\underline{\delta}_j = L_j^{del}\underline{e}_j^{del} = L_j^{del}\begin{bmatrix} e_j(2) \\ \vdots \\ e_j(p) \end{bmatrix} \tag{5.6}$$

5.3 ILC Design in the Trial Domain by Optimization

By defining the cost of each iteration as

$$r_j = \underline{e}_{j+1}^T Q_e \underline{e}_{j+1} + \underline{\delta}_j^T R_\delta \underline{\delta}_j \tag{5.7}$$

where Q_e and R_δ are positive-definite diagonal matrices, a discounted cumulative cost is defined as

$$V_j = \sum_{i=0}^{\infty} \gamma^i r_{j+i} = \sum_{i=0}^{\infty} \gamma^i \left(\underline{e}_{j+i+1}^T Q_e \underline{e}_{j+i+1} + \underline{\delta}_{j+i}^T R_\delta \underline{\delta}_{j+i} \right) \tag{5.8}$$

where $\gamma \in (0, 1)$ is the forgetting factor (or the discount rate).

The ILC problem then can be rewritten as an optimization problem in the iteration domain, i.e., to find the optimal learning matrix L's in Eq. (5.5) that minimizes the discounted cumulative cost V_j for the system Eq. (5.3).

Given the system model, this optimization problem can be solved by Linear Quadratic Regulator (LQR) theory

$$L^* = (R_\delta + \gamma P^T K P)^{-1}(\gamma P^T K) \tag{5.9}$$

where K satisfies the Riccati equation

$$K = \gamma K - \gamma^2 K P (R_\delta + \gamma P^T K P)^{-1} P^T K + Q_e/\gamma \tag{5.10}$$

This optimal L^* can produce zero tracking error with enough iterations, adjust the learning rate by tuning R_δ, and avoid the bad learning transients. The goal of this paper is to develop an ILC approach that uses data to find the optimal L^* when the system model is unknown.

5.4 An Off-Policy RL Approach

One concern about applying RL algorithms for control problems is the safety issue during exploration. Here we rewrite and extend a model-free MPC method proposed in Reference [31] into an RL algorithm that has better control over the learning transients.

5.4.1 Notations and Assumptions

In the trial domain, the environment is depicted by Eq. (5.3), where $\underline{e}$ is the state s and $\underline{\delta}$ is the action a, which can be rewritten into the RL form,

$$S_{t+1} = S_t - PA_t \tag{5.11}$$

where $S_t = \underline{e}_j$, and $A_t = \delta_j$. The notations are summarized in Table. 5.1.

Table 5.1: Notations in ILC and RL

	state	action	environment
ILC notation	$\underline{e}$	$\underline{\delta}$	Eq. (5.3)
RL notation	$s \in \mathcal{S}$	$a \in \mathcal{A}$	Eq. (5.11)

Note that in the RL community, the capital letters are the symbols of variables while the lower case letters denote values in the set. For example, $s_t \in \mathcal{S}$ is the value that the state S_t

equals, at the "time" step t. Also, the definition of trajectory in RL is a sequence like

$$S_0, A_0, R_0, S_1, A_1, R_1, S_2, \cdots \tag{5.12}$$

The cost for each iteration in ILC, Eq. (5.7), is defined as the reward for each (s_t, a_t, s_{t+1}) transition, that is,

$$R_t = s_{t+1}^T Q_e s_{t+1} + a_t^T R_\delta a_t \tag{5.13}$$

The discounted cumulative cost in ILC, Eq. (5.8), is defined as the return G_t, that is,

$$G_t = \sum_{i=0}^{\infty} \gamma^i R_{t+i} \tag{5.14}$$

A state-action value function in RL is defined as

$$\begin{aligned} q_\pi(s,a) &\triangleq \mathbb{E}[G_t \mid S_t = s, A_t = a] \\ &= \sum_{s',r} \Pr(s', r \mid s, a)\{r + \gamma \mathbb{E}[G_{t+1} \mid S_{t+1} = s']\} \end{aligned} \tag{5.15}$$

5.4.2 The Recursion Equation

For deterministic cases, $\Pr(s', r \mid s, a) = 1$ for all (s, a) pairs and G_t is an unbiased sample. Given a deterministic policy $\pi(a \mid s)$ as $a = Ls$, for each (s, a, r, s') transition,

$$\begin{aligned}
\hat{q}_\pi(s, a, \Theta) &\triangleq \mathbb{E}[\hat{G}_t(S_t, A_t, \Theta) \mid S_t = s, A_t = a] \\
&= \sum_{s', r} \Pr(s', r \mid s, a)\{r + \gamma \mathbb{E}[\hat{G}_{t+1}(S_{t+1}, A_{t+1}, \Theta) \mid S_{t+1} = s']\} \\
&= r + \gamma \mathbb{E}[\hat{G}_{t+1}(S_{t+1}, A_{t+1}, \Theta) \mid S_{t+1} = s'] \\
&= r + \gamma \sum \pi(a' \mid s') \mathbb{E}[\hat{G}_{t+1}(S_{t+1}, A_{t+1}, \Theta) \mid S_{t+1} = s', A_{t+1} = a'] \\
&= r + \gamma \mathbb{E}[\hat{G}_{t+1}(S_{t+1}, A_{t+1}, \Theta) \mid S_{t+1} = s', A_{t+1} = Ls'] \\
&= r + \gamma \hat{q}_\pi(s', Ls', \Theta)
\end{aligned} \tag{5.16}$$

Since L is computed through $\partial \hat{q}_\pi(s', a', \Theta)/\partial a' = 0$, which is equivalent to $max_{a'} q(s', a')$ in the Q-learning algorithms, one can treat this algorithm as a Q-learning method.

5.4.3 Estimation of the State-Action Value

The $q_\pi(s, a)$ can be approximated by

$$\hat{q}_\pi(s, a, \Theta) = \Theta^T \phi(s, a) \tag{5.17}$$

where $\phi(s, a)$ are the basis functions.

For the linear ILC problem, we can prove that

$$\phi(s, a) = \phi(\underline{e}_j, \underline{\delta}_j) = \Big(\begin{bmatrix} \underline{e}_j & \underline{\delta}_j \end{bmatrix} \otimes \begin{bmatrix} \underline{e}_j & \underline{\delta}_j \end{bmatrix} \Big)^T \tag{5.18}$$

where $\otimes$ denotes the Kronecker product. The proof is presented in the appendix.

By using the estimated value, the recursion equation becomes

$$\hat{q}_\pi(s, a, \Theta) = r + \gamma \hat{q}_\pi(s', Ls', \Theta) \tag{5.19}$$

Plugging Eq. (5.17) into Eq. (5.19), the recursion equation that is valid for each transition (S_t, A_t, R_t, S_{t+1}) becomes

$$\Theta^T \phi(s, a) = r + \gamma \Theta^T \phi(s', Ls') \tag{5.20}$$

To compute Θ, here the least squares method is chosen. In RL, there exist other methods like the semi-gradient TD(0) and the gradient descent method to update Θ. However, these methods usually converge slower than the least squares method and have little control over the process before convergence. It is possible that the tracking error grows dramatically before it converges to a reasonable estimation of the real Θ. In control problems, safety is a priority and hence the least squares method is chosen to update the estimation of Θ.

One advantage of using Eq. (5.20) (or Eq. (5.19)) is that the learning matrix L used in those equations can be different from the L used for learning in practice, i.e., the L used to choose a according to s. This is named as "off-policy" in RL. The estimated Θ corresponds to the L used in Eq. (5.20) instead of the L used to create (s, a, s') pair. This allows one to use a feasible L in practice, such as with a much smaller learning rate to collect enough data for the estimation.

5.4.4 Improvement of the Policy

The policy (the learning law) needs to serve two purposes: exploitation and exploration. The exploitation role of the ILC law needs to form a contraction mapping between the tracking errors of two successive iterations and the exploration part needs the control actions to be able to do some probing of the system dynamics. Thanks to the robustness of ILC, these two roles do not contradict to each other, especially for linear systems. This means the stochastic learning law allows the improvement of the tracking accuracy and the collection of data simultaneously, which may decrease the number of iterations needed for convergence.

Instead of directly adding some randomness as $\delta_j = \alpha L\underline{e}_j + \beta\nu$, the learning law is chosen as

$$\delta_j = \begin{bmatrix} \alpha+\beta\nu_1 & & \\ & \ddots & \\ & & \alpha+\beta\nu_p \end{bmatrix} L\underline{e}_j \triangleq \tilde{L}\underline{e}_j \tag{5.21}$$

where ν_i is a random signal produced according to some stochastic distribution. The values of α and β in Eq. (5.21) determine the ratio between the exploitation term and the exploration term. This learning law can lead to monotonic convergence as long as $I - P\tilde{L}$ has all singular values less than unity. As the learning progresses, the weight of exploration can be decreased gradually until zero. If the tracking error suddenly increases, one can choose to use an earlier history of $\underline{e}$ with a different set of α and β to create the control action (the command history) again or one can use a safer L if any, for example, in some cases an Arimoto-type learning law with small learning gain.

The learning law can be written as the policy, that is,

$$a = \tilde{L}s \tag{5.22}$$

The improvement of policy is chosen by $\min_a \hat{q}_\pi$ that is, by solving the equation $\partial \hat{q}_\pi(s, a, \mathbf{\Theta})/\partial a = 0$, which leads to

$$L = -(\mathbf{\Theta}_{\delta,\delta})^{-1}(\mathbf{\Theta}_{e,\delta})^T \tag{5.23}$$

5.5 Data-Driven Model-Free ILC Algorithm

The model-free ILC algorithm is summarized in Table 5.2. This method can simultaneously lead to the optimal L and "zero" tracking error. It is possible that the tracking accuracy has been achieved before the optimal L is accurately learned.

Table 5.2: Model-Free ILC Algorithm

Initialize γ, Q_e, R_δ, α, β, L and u.
Loop until the tracking error is smaller than some threshold
In each batch $j = 1, 2, \cdots, N_{bat}$ where $N_{bat} \geq (length(\underline{e}) + length(\underline{\delta}))^2$ $\underline{u}_{j+1} = \underline{u}_j + \underline{\delta}_j$ where $\underline{\delta}_j$ is computed by Eq. (5.21) record the output $\underline{y}_j$ and compute the error $\underline{e}_j$.
Compute the estimation of Θ by least squares method. Compute the update of L according to Eq. (5.23).

Currently, the limitation of this algorithm is the tremendous number of iterations needed in one batch update, $(length(\underline{e}) + length(\underline{\delta}))^2$. This limitation is due to the high dimensionality used in the RL context, $S_t = \underline{e}_j$. The solution to this issue is to use function approximation,

that is,

$$\begin{aligned} \underline{\hat{e}}_j &= w_e^T \phi_e(\underline{e}_j) \\ \underline{\hat{\delta}}_j &= w_\delta^T \phi_\delta(\underline{\delta}_j) \end{aligned} \tag{5.24}$$

The number of iterations needed in one batch then becomes $(length(w_e) + length(w_\delta))^2$.

5.6 A Numerical Example

5.6.1 Problem Description

To demonstrate the effectiveness of the model-free ILC algorithm, we performed numerical experiments on a 4th-order system consisting of two masses, two springs, and only one damper (Figure 5.1). A desired trajectory is chosen, to keep the position of the second mass at 2 for 2 seconds and then go back to the origin, being stabilized there for 1 second (the red circled solid line in Figure 5.2). Given the lightly damped system, this trajectory is hard to followed, as the system tends to oscillate instead making it hard to hold steady.

The parameters of the dynamic system are $m_1 = m_2 = 1$, $c_1 = 0.01$, $k_1 = k_2 = 1$, sampled at 5 Hz, whose discrete state space model is

$$A = \begin{bmatrix} 0.9603 & 0.0198 & 0.1972 & 0.0013 \\ 0.0198 & 0.9801 & 0.0013 & 0.1987 \\ -0.3930 & 0.1958 & 0.9584 & 0.0198 \\ 0.1960 & -0.1974 & 0.0198 & 0.9801 \end{bmatrix} \qquad B = \begin{bmatrix} 0.0199 \\ 6.637e-5 \\ 0.1972 \\ 0.0013 \end{bmatrix} \tag{5.25}$$

$$C = \begin{bmatrix} 0 & 1 & 0 & 0 \end{bmatrix} \qquad D = 0$$

Other parameters are $\gamma = 0.9$, $Q_e = I$ and $R_\delta = 10^{-7} I$ where I is the p by p identity matrix.

5.6.2 Simulation Process

Simulations are performed in MATLAB as follows:

1. Create and store $L_{initial}^{del}$ by $0.001randn(p, p-1)$ and $\underline{u}_{initial}$ by $randn(p, 1)$.
2. Simulation 1: The learning curve of the optimal L_{opt} computed based on a model.

   According to Eq. (5.9), the optimal L_{opt} is computed and its deleted form (deleting the first row) L_{opt}^{del} is used to learn to follow the desired trajectory (the red circled solid line in Figure 5.2). The updating of commands follows

   $$\underline{u}_{j+1} = \underline{u}_j + L_{opt}^{del} \underline{e}_j^{del} \tag{5.26}$$

   The learning curve using the optimal L_{opt}^{del}, i.e., the RMS error of each iteration as the learning with L_{opt}^{del} progresses, serves as the standard to evaluate how close the learned L's are to the optimal L_{opt}.

   Note that because of the ill-conditioned P matrix of this 4^{th} order system, the deleted version of the learning matrix, L^{del} as in Eq. (5.6), is applied to replace the square matrix L. This implies that the tracking accuracy at the first time step is not considered and hence when computing RMS error of each iteration, the error at the first time step is not included.
3. Simulation 2: the learning curve of the Arimoto-type learning.

   An Arimoto-type learning law is applied as a comparison to the model-free method to

learn the desired trajectory. The updating of the control actions follows

$$\underline{u}_{j+1} = \underline{u}_j + 0.001\underline{e}_j^{del} \tag{5.27}$$

Its learning gain (0.001) is chosen to be on the same level with the gain of the initial learning matrix $L_{initial}^{del}$ for the model-free ILC methods and hence the bad learning transients caused by the Arimoto-type law are not exaggerated or diminished because of a different gain value.

4. Simulation 3: the learning curve of the data-driven model-free ILC, compared with three model-based learning laws.

   The model-free ILC algorithm (Table 5.2 with $L_{initial}^{del}$ and $\underline{u}_{initial}$) is used to learn the desired trajectory and α and β used for each batch are shown in the Table 5.3 After 4 batches, the updates of L are stopped. The learning curve of this process is plotted. Three model-based ILC methods, the transpose learning law $L = P^T$, the quadratic learning law $L = (PP^T + R_)^{-1}P^T$, and the optimal learning law Eq. (5.9) are also applied to learn the same trajectory.

Table 5.3: α and β

Batch	1	2	3	4	5
α	0.1	0.1	0.1	0.1	1
β	2	0.1	0.1	0.1	0

5.6.3 Results and Discussions

The desired trajectory (the red solid circled line) and the output (the blue dashed dotted line) after learning by the data-driven ILC (Simulation 3) are plotted in Figure 5.2. These two

plots are overlapping except for the first time step. The tracking errors are less than 1% with respect to the desired trajectory. The deviation at the first time step is caused by the deleted learning matrix as in Eq. (5.6) used to approach the stable inverse. In practice, the desired trajectory is supposed to be adjusted by adding one more time step at the beginning, which is to be compromised to achieve high tracking accuracy for the entire real desired trajectory.

The learning curve of the data-driven ILC using RL with 4 batch-updates of L are plotted in Figure 5.3. Each batch contains 2304 iterations. The minimum requirement of the number of iterations in each batch is $(23+24)^2 = 2209$. Here we used around 100 more, $(24+24)^2 = 2304$, and the more iterations included in one batch the more accurate the solution obtained by the least squares method.

In the first batch, without any prior information of the system dynamics and also with the purpose of exploration, $L^{del}_{initial}$ was created by $0.001randn(p, p-1)$, and hence the tracking error is slightly increased. Because of the small value of the $L^{del}_{initial}$, the RMS errors stay below 2 in the first batch.

During the second batch, the RMS error decreases by using the updated L based on the data from the first batch and the wiggles are caused by the exploration signal. (There are not any obvious wiggles in the first batch because of the already increasing error and the initial $L^{del}_{initial}$ created by randn.) The decreasing trend of the tracking error in the second batch is realized by the learning law, Eq. (5.21), which allows the contraction mapping while doing exploration at the same time. At the end of the first batch, the tracking error has reached the level of 10^{-2}, which is actually the first plateau of the learning curve of the optimal L, plotted by the red solid line in Figure 5.3.

For the sake of rich enough data, the tracking error of the initial run in the previous batch

is used as the tracking error to compute the command for the first run in the new batch. Hence the tracking error is increased when the updated L is used at the beginning of the third batch. And the tracking error is decreased again while data is collected for a better estimation of L. The same process is repeated for the fourth batch. From the fifth batch, the updating of L is stopped by resetting β to zero and α to 1. Figure 5.3 plots the RMS error for the first 15000 iterations to demonstrate the process of updating L. The tracking RMS error is decreased to the level of 10^{-14} after 25000 iterations.

This learning curve is compared with three model-based learning laws in Figure **??**. The learning curve of the L with 4 updates is plotted by black dashed line; the green dash-dot line represents the learning curve of the transpose law; the magenta dot line of the quadratic learning law, and the solid red of the optimal learning law. Because of the little damped dynamics, the transpose learning law and the quadratic learning law learn the high frequency motions a lot slower than the optimal learning laws (both model-based and model-free) as the green line stays around 0.7 and the magenta line around 10^{-3} even after 200000 runs. The model-free ILC by RL method (black) can follow the model-based optimal ILC (red) after the exploration stage (the four-batch updates of L but one can obtain the same results just by one batch depending on the richness of data), and the tracking error can be decreased to 10^{-13}, around the numerical zero after 200000 runs.

5.7 Conclusion

This chapter presents the data-driven model-free ILC by an off-policy RL in the trial domain for linear systems, which can be seen as the first step to bridge ILC and RL leading toward treatment of nonlinear systems. In this data-driven ILC method, the exploration of the dynamics can be limited in the action space where the contraction mapping of the errors between two successive iterations is still valid. Therefore, the improvement of the tracking accuracy can be achieved before the estimation of the parameters is accurate enough. Moreover, the off-policy method allows one to use a safer learning law to learn the trajectory and create data, while using the data to estimate the parameters of a value function for another learning law. This implies the collection of enough data while learning to follow the desired trajectory without safety concerns. Due to the extensive measurements needed, the number of iterations before convergence can be huge. Function approximation can relax this limitation. However, it probably cannot be completely avoided due to the model-free nature.

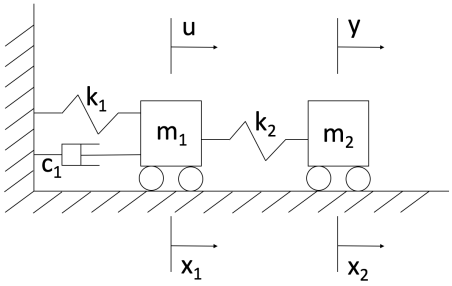


Figure 5.1: A spring-mass-damper system.

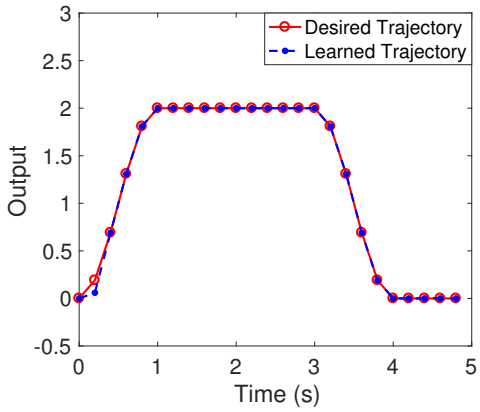


Figure 5.2: A 24-point desired trajectory and the learned trajectory by data-driven ILC.

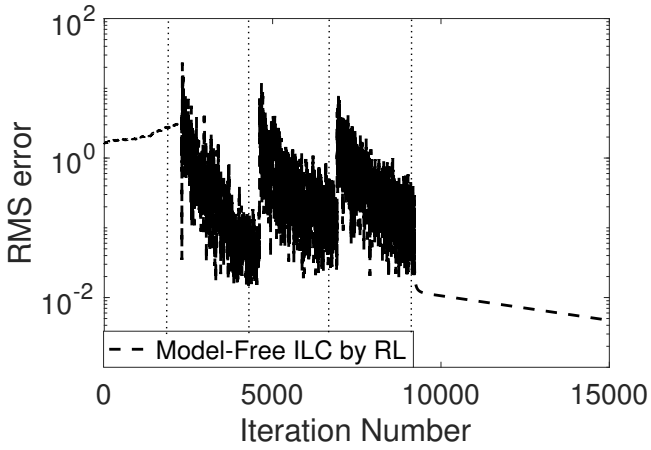


Figure 5.3: The learning curves of the data-driven ILC.

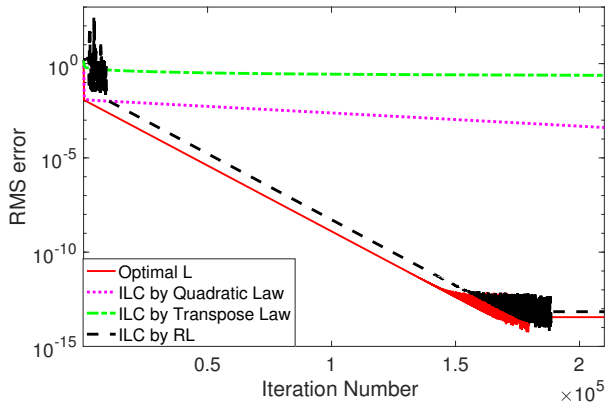


Figure 5.4: Learning Curves compared with model-based ILC.

5.8 Appendix

The proof in the control context, i.e., in terms of V_j, $\underline{e}_j$, and δ_j, is as follows: Because of the forgetting factor γ is less than 1, there exists a big enough $l \in \mathcal{Z}$ such that $\gamma^l \approx 0$. Therefore the finite-horizon cost

$$V_j = \sum_{i=0}^{l} \gamma^i r_{j+i} \tag{5.28}$$

is used to compute the infinite cost Eq. (5.8).

The following steps prove that the accumulative cost V_j can be written as a quadratic function with respect to $\underline{e}_j$ and $\underline{\delta}_j$.

1. Rewrite the cost V_j as a quadratic function of super-super vectors by stacking the error histories and control action differences for all iterations in the horizon.

By defining super-super vectors

$$\underline{\mathbf{e}}_{j+1}^{<l>} = \begin{bmatrix} \underline{e}_{j+1} \\ \underline{e}_{j+2} \\ \vdots \\ \underline{e}_{j+l} \end{bmatrix} \qquad \underline{\boldsymbol{\delta}}_{j}^{<l>} = \begin{bmatrix} \underline{\delta}_{j} \\ \underline{\delta}_{j+1} \\ \vdots \\ \underline{\delta}_{j+l-1} \end{bmatrix} \tag{5.29}$$

the finite-horizon value function Eq. (5.28) becomes

$$V_j = (\underline{\mathbf{e}}_{j+1}^{<l>})^T \mathbf{Q}_l \underline{\mathbf{e}}_{j+1}^{<l>} + (\underline{\boldsymbol{\delta}}_{j}^{<l>})^T \mathbf{R}_l \underline{\boldsymbol{\delta}}_{j}^{<l>} \tag{5.30}$$

where

$$\mathbf{Q}_l = \begin{bmatrix} Q_e & & & \\ & \gamma Q_e & & \\ & & \ddots & \\ & & & \gamma^{l-1} Q_e \end{bmatrix} \qquad \mathbf{R}_l = \begin{bmatrix} R_\delta & & & \\ & \gamma R_\delta & & \\ & & \ddots & \\ & & & \gamma^{l-1} R_\delta \end{bmatrix} \tag{5.31}$$

2. The accumulative cost V_j is a function with respect to $\underline{e}_j$ and $\underline{\delta}_j$ with future L's and the dynamics P implicitly included as parameters.

   Assuming a learning matrix L for future iterations ($j+1,\ j+2,\ \cdots,\ j+l-1$), the future error histories and future control action differences can be written as

$$
\begin{aligned}
\underline{e}_{j+1} &= \underline{e}_j - P\underline{\delta}_j \\
\underline{e}_{j+2} &= (I-PL)\underline{e}_{j+1} = (I-PL)\underline{e}_j - (I-PL)P\underline{\delta}_j \\
\underline{e}_{j+3} &= (I-PL)^2\underline{e}_{j+1} = (I-PL)^2\underline{e}_j - (I-PL)^2P\underline{\delta}_j \\
&\vdots \\
\underline{e}_{j+l} &= (I-PL)^{l-1}\underline{e}_{j+1} = (I-PL)^{l-1}\underline{e}_j - (I-PL)^{l-1}P\underline{\delta}_j
\end{aligned} \tag{5.32}
$$

$$
\begin{aligned}
\underline{\delta}_{j+1} &= L\underline{e}_{j+1} = L\underline{e}_j - LP\underline{\delta}_j \\
\underline{\delta}_{j+2} &= L\underline{e}_{j+2} = L(I-PL)\underline{e}_j - L(I-PL)P\underline{\delta}_j \\
&\vdots \\
\underline{\delta}_{j+l-1} &= L\underline{e}_{j+l-1} = L(I-PL)^{l-2}\underline{e}_j - L(I-PL)^{l-2}P\underline{\delta}_j
\end{aligned}
$$

Package the above future data as super-super-vectors,

$$
\underline{\mathbf{e}}_{j+1}^{<l>} = \begin{bmatrix} \underline{e}_{j+1} \\ \underline{e}_{j+2} \\ \underline{e}_{j+3} \\ \vdots \\ \underline{e}_{j+l} \end{bmatrix} = \mathbf{P}_e\underline{e}_j + \mathbf{P}_\delta\underline{\delta}_j \qquad \underline{\boldsymbol{\delta}}_j^{<l>} = \begin{bmatrix} \underline{\delta}_j \\ \underline{\delta}_{j+1} \\ \underline{\delta}_{j+2} \\ \vdots \\ \underline{\delta}_{j+l-1} \end{bmatrix} = \mathbf{L}_e\underline{e}_j + \mathbf{L}_\delta\underline{\delta}_j \tag{5.33}
$$

where

$$\mathbf{P}_e = \begin{bmatrix} I \\ I - PL \\ (I - PL)^2 \\ \vdots \\ (I - PL)^{l-1} \end{bmatrix} \qquad \mathbf{P}_\delta = \begin{bmatrix} -P \\ -(I - PL)P \\ -(I - PL)^2 P \\ \vdots \\ -(I - PL)^{l-1} P \end{bmatrix}$$

$$\mathbf{L}_e = \begin{bmatrix} 0 \\ L \\ L(I - PL) \\ \vdots \\ L(I - PL)^{l-2} \end{bmatrix} \qquad \mathbf{L}_\delta = \begin{bmatrix} I \\ -LP \\ -L(I - PL)P \\ \vdots \\ -L(I - PL)^{l-2} P \end{bmatrix} \tag{5.34}$$

The cost V_j then becomes

$$V_j(\underline{e}_j, \underline{\delta}_j) = \begin{bmatrix} \underline{e}_j \\ \underline{\delta}_j \end{bmatrix}^T \begin{bmatrix} \mathbf{\Theta}_{e,e} & \mathbf{\Theta}_{e,\delta} \\ \mathbf{\Theta}_{\delta,e} & \mathbf{\Theta}_{\delta,\delta} \end{bmatrix} \begin{bmatrix} \underline{e}_j \\ \underline{\delta}_j \end{bmatrix} \tag{5.35}$$

where

$$\begin{bmatrix} \mathbf{\Theta}_{e,e} & \mathbf{\Theta}_{e,\delta} \\ \mathbf{\Theta}_{\delta,e} & \mathbf{\Theta}_{\delta,\delta} \end{bmatrix} = \begin{bmatrix} \mathbf{P}_e^T \mathbf{Q}_l \mathbf{P}_e + \mathbf{L}_e^T \mathbf{R}_l \mathbf{L}_e & \mathbf{P}_e^T \mathbf{Q}_l \mathbf{P}_\delta + \mathbf{L}_e^T \mathbf{R}_l \mathbf{L}_\delta \\ \mathbf{P}_\delta^T \mathbf{Q}_l \mathbf{P}_e + \mathbf{L}_\delta^T \mathbf{R}_l \mathbf{L}_e & \mathbf{P}_\delta^T \mathbf{Q}_l \mathbf{P}_\delta + \mathbf{L}_\delta^T \mathbf{R}_l \mathbf{L}_\delta \end{bmatrix} \tag{5.36}$$

For convenience, we define

$$\mathbf{\Theta} = \begin{bmatrix} \mathbf{\Theta}_{e,e} & \mathbf{\Theta}_{e,\delta} \\ \mathbf{\Theta}_{\delta,e} & \mathbf{\Theta}_{\delta,\delta} \end{bmatrix} \tag{5.37}$$

and Eq. (5.35) becomes

$$V_j = \begin{bmatrix} \underline{e}_j \\ \underline{\delta}_j \end{bmatrix}^T \boldsymbol{\Theta} \begin{bmatrix} \underline{e}_j \\ \underline{\delta}_j \end{bmatrix} \tag{5.38}$$

Hence, V_j is a quadratic function with the parametric matrix $\boldsymbol{\Theta}$ in terms of $\begin{bmatrix} \underline{e}_j & \underline{\delta}_j \end{bmatrix}$, where $\boldsymbol{\Theta}$ includes the influence of future control inputs and the system dynamics. The state-value function then can be parameterized as

$$q_\pi(s,a) = q_\pi(\underline{e}_j, \underline{\delta}_j) = V_j = \begin{bmatrix} \underline{e}_j \\ \underline{\delta}_j \end{bmatrix}^T \boldsymbol{\Theta} \begin{bmatrix} \underline{e}_j \\ \underline{\delta}_j \end{bmatrix} \tag{5.39}$$

3. An alternative form that fits the function approximation

   By using the Kronecker product, Eq. (5.39) can be rewritten as

$$V_j = \Theta^T \Big(\begin{bmatrix} \underline{e}_j \\ \underline{\delta}_j \end{bmatrix}^T \otimes \begin{bmatrix} \underline{e}_j \\ \underline{\delta}_j \end{bmatrix}^T \Big)^T \tag{5.40}$$

   where Θ is a column vector by stacking the rows of the $\boldsymbol{\Theta}$ matrix. Therefore, the basis function is of the quadratic form.

This page intentionally left blank

Chapter 6

Conclusions and Future Work

An ILC algorithm uses the tracking error from previous iterations to compute a feedforward signal that can compensate a feedback controller and aims to increase the tracking accuracy approaching the reproducibility level of the hardware. Generally speaking, the use of models in ILC can lead to algorithms that are less computational demanding, have faster convergence, and have better stability and performance guarantees. The use of data can relax the restrictions brought by the model.

Because ILC asks for zero error at all frequencies, model-based ILC can be destabilized by high frequency model errors from residual modes or parasitic poles. Infinite impulse response zero-phase low-pass filtering is used to robustify to such model inaccuracy. These filters fail to produce the intended steady state frequency cutoff objective and can cause an otherwise stable ILC law to become unstable. Here we consider the circulant form of a zero-phase filter for ILC and show that it represents the desired steady state response cutoff and can guarantee that the circulant filter design will not destabilize a stable ILC law. This is the first time to introduce a circulant filter into ILC and take advantage of the off-line property of ILC.

Besides the computation of a feedforward control signal, the measurements from previous repetitions can also be used to update the system model used in ILC design. For linear systems, this leads to an indirect adaptive ILC method with an embedded observer to update the system Markov parameters. This adaptive ILC approach can produce high tracking accuracy when the trajectory is too fast for feedback control to be effective. It can also keep learning transients small so that constraints are not violated during the learning process.

For nonlinear systems, when the measurements of all system states are available, one can use a Carlemen bilinearized model to approximate the nonlinear dynamics, to arbitrarily high accuracy at the expense of increased dimensionality. Hence here an ILC approach for nonlinear systems based on Carleman bilinearization is proposed. This ILC approach uses data to repeatedly create bilinear models in a homotopy approaching the desired trajectory, and can converge considerably faster when compared to the corresponding method based on a linearized model. The ILC algorithm for a bilinear system is developed for the first time. It is also an important step to fill the gap in the development of ILC laws specifically designed to handle nonlinear systems.

At last, a data-driven model-free ILC for linear systems is developed by applying an off-policy RL method in the trial domain. This data-driven method has two important features: the improvement of tracking accuracy before accurate estimation of state-action values and the collection of enough data while learning the desired trajectory without safety concerns. The limitation of this method is the number of iterations needed, which can be relaxed by using function approximation. This algorithm can be seen as the first step to bridge ILC and RL aiming to address nonlinear systems.

As noted before, the research contributions in this thesis can be viewed as a progression

from model-based ILC algorithms to data-driven ILC algorithms. Chapter 2 is totally a priori model based, and seeks ways to robustify ILC convergence in the presence of modeling errors. Chapters 3 and 4 use data to update models, in the first case updating the system Markov parameters, and in the second case updating a progression of models along a homotopy. Chapter 5 finally eliminates the use the a priori model and performs ILC totally model free using RL.

The next step and the first step of future work is to (1) extend this data-driven ILC by RL method to nonlinear dynamics, and (2) directly use RL in the time domain for ILC problems to compute the feedforward signal. Then four ILC methods can be compared while increasing the nonlinearity of a system: (1) nonlinear ILC based on linearization, (2) nonlinear ILC based on Carleman bilinearization, (3) data-driven ILC by RL, and (4) model-free RL. This may result in a qualitative answer to the question when to use data-driven ILC and when to use system identification and a model-based ILC algorithm. The ultimate goal is to mutually benefit from both data and model for nonlinear ILC problems.

This page intentionally left blank

References

[1] M. Uchiyama, "Formation of high-speed motion pattern of a mechanical arm by trial (in japanese)," *Transactions of the Society of Instrument and Control Engineers*, vol. 14, pp. 706–712, 1978.

[2] S. Arimoto, S. Kawamura, and F. Miyazaki, "Bettering operation of robots by learning," *Journal of Robotic Systems*, vol. 1, no. 2, pp. 123–140, 1984.

[3] J. J. Craig, "Adaptive control of manipulators through repeated trials," in *Proceedings of the American Control Conference*, vol. 3, pp. 1566–1573, 1984.

[4] G. Casalino and G. Bartolini, "A learning procedure for the control of movements of robotic manipulators," in *IASTED Symposium on Robotics and Automation*, (The Netherlands), pp. 108–111, 1984.

[5] T. Mita and E. Kato, "Iterative control and its application to motion control of robot arm - a direct approach to servo-problems," in *IEEE Conference on Decision and Control*, vol. 24, pp. 1393–1398, IEEE, 1985.

[6] H. Elci, R. W. Longman, M. Phan, J.-N. Juang, and R. Ugoletti, "Discrete frequency based learning control for precision motion control," in *IEEE International Conference on Systems, Man, and Cybernetics*, vol. 3, pp. 2767–2773, IEEE, 1994.

[7] K. E. Avrachenkov and R. W. Longman, "Iterative learning control for over-determined, under-determined, and ill-conditioned systems," *International Journal of Applied Mathematics and Computer Science*, vol. 13, pp. 113–122, 2003.

[8] T. Lin, D. Owens, and J. Hätönen, "Newton method based iterative learning control for discrete non-linear systems," *International Journal of Control*, vol. 79, no. 10, pp. 1263–1276, 2006.

[9] R. W. Longman, "Iterative learning control and repetitive control for engineering practice," *International Journal of Control*, vol. 73, no. 10, pp. 930–954, 2000.

[10] D. A. Bristow, M. Tharayil, and A. G. Alleyne, "A survey of iterative learning control," *IEEE Control Systems*, vol. 26, no. 3, pp. 96–114, 2006.

[11] H. S. Ahn, Y. Chen, and K. L. Moore, "Iterative learning control: Brief survey and categorization," *IEEE Transactions on Systems Man and Cybernetics part C Applications and Reviews*, vol. 37, no. 6, p. 1099, 2007.

[12] Y. Wang, F. Gao, and F. J. Doyle III, "Survey on iterative learning control, repetitive control, and run-to-run control," *Journal of Process Control*, vol. 19, no. 10, pp. 1589–1600, 2009.

[13] Z. Bien and J.-X. Xu, *Iterative Learning Control: Analysis, Design, Integration and Applications*. Springer Science & Business Media, 2012.

[14] K. L. Moore, *Iterative Learning Control for Deterministic Systems*. Springer Science & Business Media, 2012.

[15] D. H. Owens, *Iterative Learning Control: An Optimization Paradigm*. Springer, 2015.

[16] C. T. Freeman, E. Rogers, J. H. Burridge, A.-M. Hughes, and K. L. Meadmore, *Iterative Learning Control for Electrical Stimulation and Stroke Rehabilitation*. Springer, 2015.

[17] A. P. Schoellig, F. L. Mueller, and R. D'Andrea, "Optimization-based iterative learning for precise quadrocopter trajectory tracking," *Autonomous Robots*, vol. 33, no. 1-2, pp. 103–127, 2012.

[18] J. Van Den Berg, S. Miller, D. Duckworth, H. Hu, A. Wan, X.-Y. Fu, K. Goldberg, and P. Abbeel, "Superhuman performance of surgical tasks by robots using iterative learning from human-guided demonstrations," in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2074–2081, IEEE, 2010.

[19] C. J. Ostafew, A. P. Schoellig, and T. D. Barfoot, "Visual teach and repeat, repeat, repeat: Iterative learning control to improve mobile robot path tracking in challenging outdoor environments," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 176–181, IEEE, 2013.

[20] Z. Hou, J.-X. Xu, and H. Zhong, "Freeway traffic control using iterative learning control-based ramp metering and speed signaling," *IEEE Transactions on Vehicular Technology*, vol. 56, no. 2, pp. 466–477, 2007.

[21] Z.-S. Hou and Z. Wang, "From model-based control to data-driven control: Survey, classification and perspective," *Information Sciences*, vol. 235, pp. 3–35, 2013.

[22] K. Popper, *Conjectures and Refutations: The Growth of Scientific Knowledge*. routledge, 2014.

[23] D. Zhang and B. Wei, "On the development of learning control for robotic manipulators," *Robotics*, vol. 6, no. 4, p. 23, 2017.

[24] G. Pipeleers and K. L. Moore, "Unified analysis of iterative learning and repetitive controllers in trial domain," *IEEE Transactions on Automatic Control*, vol. 59, no. 4, pp. 953–965, 2014.

[25] J. Zhu and R. W. Longman, "Cross fertilization between iterative learning control and repetitive control," *Advances in The Astronautical Sciences*, vol. 158, pp. 2287–2305, 2016.

[26] R. S. Sutton, A. G. Barto, and R. J. Williams, "Reinforcement learning is direct adaptive optimal control," *IEEE Control Systems*, vol. 12, no. 2, pp. 19–22, 1992.

[27] R. S. Sutton, A. G. Barto, *et al.*, *Reinforcement Learning: An Introduction*. MIT press, 1998.

[28] D. P. Bertsekas, "Dynamic programming and suboptimal control: A survey from adp to mpc," *European Journal of Control*, vol. 11, no. 4-5, pp. 310–334, 2005.

[29] F. L. Lewis, D. Vrabie, and K. G. Vamvoudakis, "Reinforcement learning and feedback control: Using natural decision methods to design optimal adaptive controllers," *IEEE Control Systems*, vol. 32, no. 6, pp. 76–105, 2012.

[30] D. Görges, "Relations between model predictive control and reinforcement learning," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 4920–4928, 2017.

[31] M. Q. Phan and S. M. B. Azad, "Model predictive control and model predictive q-learning for structural vibration control," *The Astronautical Sciences*, pp. 17–36, 2017.

[32] H.-S. Ahn, "Reinforcement learning and iterative learning control: Similarity and difference," in *Proceedings of the International Conference on Mechatronics and Information Technology, Gwangju, Korea*, pp. 3–5, 2009.

[33] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, 2013.

[34] 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 Conference on Decision and Control*, vol. 23, pp. 1064–1069, IEEE, 1984.

[35] J. J. Hätönen, D. H. Owens, and K. L. Moore, "An algebraic approach to iterative learning control," *International Journal of Control*, vol. 77, no. 1, pp. 45–54, 2004.

[36] J.-X. Xu, "A survey on iterative learning control for nonlinear systems," *International Journal of Control*, vol. 84, no. 7, pp. 1275–1294, 2011.

[37] B. Song and R. W. Longman, "Circulant zero-phase low pass filter design for improved robustification of iterative learning control," *Advances in The Astronautical Sciences*, vol. 156, pp. 2161–2180, 2015.

[38] B. Song and R. W. Longman, "Modifying iterative learning control to increase tracking speed by markov parameter updates," *Advances in The Astronautical Sciences*, vol. 158, pp. 2307–2326, 2016.

[39] B. Song and R. W. Longman, "Bilinearized model-based discrete-time iterative learning control for nonlinear systems (in press)," *Advances in The Astronautical Sciences*, 2018.

[40] B. Song and R. W. Longman, "Data-driven model-free iterative learning control using reinforcement learning (in press)," *Advances in The Astronautical Sciences*, 2018.

[41] H. Elci, R. W. Longman, M. Q. Phan, J.-N. Juang, and R. Ugoletti, "Simple learning control made practical by zero-phase filtering: Applications to robotics," *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications,*, vol. 49, no. 6, pp. 753–767, 2002.

[42] A. M. Plotnik and R. W. Longman, "Subtleties in the use of zero-phase low-pass filtering and cliff filtering in learning control," *Astrodynamics 1999*, pp. 673–692, 2000.

[43] J. Wallén, S. Gunnarsson, and M. Norrlöf, *Some Implementation Aspects of Iterative Learning Control*. Linköping University Electronic Press, 2010.

[44] M. Q. Phan, R. W. Longman, and K. L. Morre, "Unified formulation of linear iterative learning control," *Spaceflight Mechanics 2000*, pp. 93–111, 2000.

[45] J. Bao and R. W. Longman, "Unification and robustification of iterative learning control laws," *Advances in the Astronautical Sciences*, vol. 136, pp. 727–745, 2010.

[46] G. Strang and T. Nguyen, *Wavelets and Filter Banks*. SIAM, 1996.

[47] F. Gustafsson, "Determining the initial states in forward-backward filtering," *Signal Processing, IEEE Transactions on*, vol. 44, pp. 988–992, 1994.

[48] K. Chen and R. W. Longman, "Creating a short time equivalent of frequency cutoff for robustness in learning control," *Advances in the Astronautical Sciences*, vol. 114, pp. 95–114, 2003.

[49] H. Gardner and H. Thomas, "Educational implications of the theory of multiple intelligences," *Educational Researcher*, vol. 18, no. 8, pp. 4–10, 1989.

[50] M. Hehn and R. D'Andrea, "A frequency domain iterative learning algorithm for high-performance, periodic quadrocopter maneuvers," *Mechatronics*, vol. 24, no. 8, pp. 954–965, 2014.

[51] R. W. Longman and T. Songchon, "Trade-offs in designing learning/repetitive controllers using zero-phase filtering for long term stabilization," *Spaceflight Mechanics 1999*, pp. 243–262, 1999.

[52] R. W. Longman, K. Xu, and B. Panomruttanarug, "Designing learning control that is close to instability for improved parameter identification," in *Modeling, Simulation and Optimization of Complex Processes*, pp. 359–370, Springer, 2008.

[53] R. Longman, Y.-t. Peng, T. Kwon, H. Lus, R. Betti, and J.-N. Juang, "Adaptive inverse iterative learning control," *Advances in the Astronautical Sciences*, vol. 114, pp. 115–134, 2003.

[54] M. Q. Phan, L. G. Horta, J.-N. Juang, and R. Longman, "Identification of linear systems by an asymptotically stable observer," *NASA Langley Technical Report-3164*, 1992.

[55] J.-N. Juang, M. Phan, L. G. Horta, and R. W. Longman, "Identification of observer/kalman filter markov parameters: Theory and experiments," *Journal of Guidance, Control, and Dynamics*, vol. 16, no. 2, pp. 320–329, 1993.

[56] M. Phan, L. G. Horta, J.-N. Juang, and R. W. Longman, "Linear system identification via an asymptotically stable observer," *Journal of Optimization Theory and Applications*, vol. 79, no. 1, pp. 59–86, 1993.

[57] J.-N. Juang, J. E. Cooper, and J. R. Wright, "An eigensystem realization algorithm using data correlations (era/dc) for modal parameter identification," *Control, Theory and Advanced Technology*, vol. 4, no. 1, pp. 5–14, 1988.

[58] A. Chinnan and R. Longman, "Designing iterative learning controllers from limited pulse response data," *Advances in the Astronautical Sciences*, vol. 130, pp. 283–302, 2008.

[59] K. Smolders, M. Volckaert, and J. Swevers, "Tracking control of nonlinear lumped mechanical continuous-time systems: A model-based iterative learning approach," *Mechanical Systems and Signal Processing*, vol. 22, no. 8, pp. 1896–1916, 2008.

[60] A. Tayebi and C.-J. Chien, "A unified adaptive iterative learning control framework for uncertain nonlinear systems," *IEEE Transactions on Automatic Control*, vol. 52, no. 10, pp. 1907–1913, 2007.

[61] R. LONGMAN, C.-K. Chang, and M. Phan, "Discrete time learning control in nonlinear systems," in *A Collection of Technical Papers, 1992 AIAA/AAS Astrodynamics Specialist Conference*, pp. 501–511, 1992.

[62] J.-X. Xu, "Analysis of iterative learning control for a class of nonlinear discrete-time systems," *Automatica*, vol. 33, no. 10, pp. 1905–1907, 1997.

[63] M. Yu, S.-L. Jämsä-Jounela, *et al.*, "Adaptive iterative learning control for discrete-time nonlinear systems without knowing the control gain signs," 2013.

[64] R. Longman and K. Mombaur, "Implementing linear iterative learning control laws in nonlinear systems," *Advances in the Astronautical Sciences*, vol. 130, pp. 303–324, 2008.

[65] S. Svoronos, G. Stephanopoulos, and R. Aris, "Bilinear approximation of general nonlinear dynamic systems with linear inputs," *International Journal of Control*, vol. 31, no. 1, pp. 109–126, 1980.

[66] K. Kowalski and W.-H. Steeb, *Nonlinear Dynamical Systems and Carleman Linearization*. World Scientific, 1991.

[67] M. Q. Phan, Y. Shi, R. Betti, and R. W. Longman, "Discrete-time bilinear representation of continuous-time bilinear state-space models," *Advances in the Astronautical Sciences*, vol. 143, pp. 571–589, 2012.

[68] J. Hätönen, T. Harte, D. Owens, J. Ratcliffe, P. Lewin, and E. Rogers, "Discrete-time arimoto ilc-algorithm revisited," *IFAC Proceedings Volumes*, vol. 37, no. 12, pp. 541–546, 2004.

[69] K. L. Moore, Y. Chen, and V. Bahl, "Monotonically convergent iterative learning control for linear discrete-time systems," *Automatica*, vol. 41, no. 9, pp. 1529–1537, 2005.

[70] H. Elci, R. W. Longman, M. Q. Phan, J.-N. Juang, and R. Ugoletti, "Simple learning control made practical by zero-phase filtering: Applications to robotics," *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, vol. 49, no. 6, pp. 753–767, 2002.

[71] P. Janssens, G. Pipeleers, and J. Swevers, "Model-free iterative learning control for lti systems with actuator constraints," in *Proceedings of the 18th IFAC World Congress*, pp. 11556–11561, 2011.

[72] P. Janssens, G. Pipeleers, and J. Swevers, "Model-free iterative learning of time-optimal point-to-point motions for lti systems," in *IEEE Conference on Decision and Control and European Control Conference*, pp. 6031–6036, IEEE, 2011.

[73] P. Janssens, G. Pipeleers, and J. Swevers, "Model-free iterative learning control for lti systems and experimental validation on a linear motor test setup," in *American Control Conference*, pp. 4287–4292, IEEE, 2011.

[74] P. Janssens, G. Pipeleers, and J. Swevers, "A data-driven constrained norm-optimal iterative learning control framework for lti systems," *IEEE Transactions on Control Systems Technology*, vol. 21, no. 2, pp. 546–551, 2013.

[75] M.-B. Radac and R.-E. Precup, "Optimal behaviour prediction using a primitive-based data-driven model-free iterative learning control approach," *Computers in Industry*, vol. 74, pp. 95–109, 2015.

[76] M.-B. Radac, R.-E. Precup, and E. M. Petriu, "Model-free primitive-based iterative learning control approach to trajectory tracking of mimo systems with experimental validation," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 26, no. 11, pp. 2925–2938, 2015.

[77] M.-B. Radac, R.-E. Precup, and E. M. Petriu, "Constrained data-driven model-free ilc-based reference input tuning algorithm," *Acta Polytechnica Hungarica*, vol. 12, no. 1, pp. 137–160, 2015.

[78] Y. Li and R. W. Longman, "Characterizing and addressing the instability of the control action in iterative learning control," *Advances in The Astronautical Sciences*, vol. 136, pp. 1967–1985, 2010.

[79] X. Ji, T. Li, and R. W. Longman, "Proof of two stable inverses of discrete time systems," *Advances in The Astronautical Sciences*, vol. 162, pp. 123–136, 2018.