Optimizing the Execution of Dynamic Robot Movements With Learning Control

Okan Koç, Guilherme Maeda, and Jan Peters

***Abstract*—High-speed robotics typically involves fast dynamic trajectories with large accelerations. Kinematic optimization using compact representations can lead to an efficient online computation of these dynamic movements, however successful execution requires accurate models or aggressive tracking with high-gain feedback. Learning to track such references in a safe and reliable way, whenever accurate models are not available, is an open problem. Stability issues surrounding the learning performance, in the iteration domain, can prevent the successful implementation of model-based learning approaches. To this end, in this paper we propose a new adaptive and *cautious* iterative learning control (ILC) algorithm where the stability of the control updates is analyzed probabilistically: the covariance estimates of the adapted local linear models are used to increase the probability of update monotonicity, exercising caution during learning. The resulting learning controller can be implemented efficiently using a recursive approach. We evaluate it extensively in simulations as well as in our robot table tennis setup for tracking dynamic hitting movements. Testing with two seven degree of freedom anthropomorphic robot arms, we show improved and more stable tracking performance over high-gain proportional and derivative (PD) control, model-free ILC (simple PD feedback type) and model-based ILC without cautious adaptation.**

***Index Terms*—Adaptive control, iterative learning control, recursive estimation, robot learning, robust control.**

I. Introduction

MOST dynamic tasks in robotics include a *tracking* component, where the system is controlled to follow a desired reference trajectory. Robot table tennis [1], [2], in particular involves the generation of fast dynamic trajectories with high accelerations. These trajectories can be optimized well on the kinematics level, but reaching the target state and returning the ball requires accurate tracking of these hitting movements. Computing the appropriate control inputs for tracking can be a challenging task, especially when using cable-driven arms, such as the Barrett WAM shown in Fig. 1, due to mechanical compliance and low bandwidth.

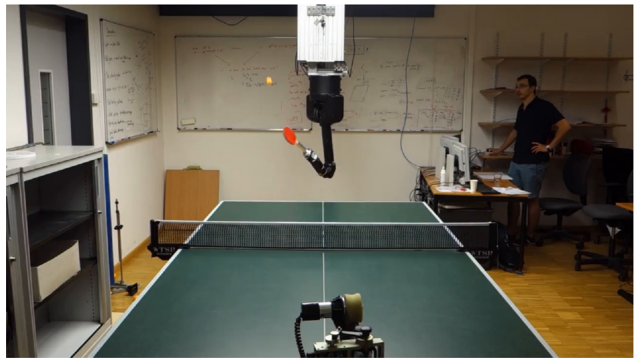

Fig. 1. Our robot table tennis platform where a seven degree of freedom Barrett WAM arm is shown facing a ball-launcher. The ball is tracked using four cameras on the ceiling. Whenever a ball is approaching the robot, reference trajectories are computed online in order to return the ball to a desired location on the opponent's court. Such trajectories can be optimized on the kinematics level [2], however it is hard to execute them accurately without having access to accurate dynamics models. Iterative learning control, using inaccurate models, can still lead to an efficient approach for learning to track these trajectories.

Iterative learning control (ILC) is a control theoretic learning framework restricted to tracking (time varying) reference trajectories [3]. In ILC, the goal is to improve the tracking performance, reducing the future deviations along the fixed trajectory, and ultimately driving them to the minimum possible. After observing the deviations from the reference trajectory at each iteration, the errors are fed back to the (feedforward) control inputs for the next iteration. Any available dynamics models can be incorporated easily during these updates, see e.g., [4] and [5]. ILC has been used successfully in several robotics tasks to improve trajectory tracking performance under unknown repeating disturbances and model mismatch [3].

While there have been many impressive applications of reinforcement learning (RL) [6] to learn robotic tasks [7], RL remains to be computationally and information-theoretically hard in general. Much of control, on the other hand, can be reduced to supervised learning with the appropriate reference trajectories. By making good use of existing, albeit imperfect, models

Manuscript received December 9, 2018; accepted March 3, 2019. Date of publication May 3, 2019; date of current version August 1, 2019. This paper was recommended for publication by Associate Editor S. Calinon and Editor A. Billard upon evaluation of the reviewers' comments. This work was supported in part by the European Community's Seventh Framework Programme (FP7/2007–2013) under Grant 610878 (3rd HAND), in part by the European Union's Horizon 2020 research and innovation programme under Grant 640554 (SKILLS4ROBOTS), and in part by a project commissioned by the New Energy and Industrial Technology Development Organization (NEDO). *(Corresponding author: Okan Koç.)*

O. Koç is with the Max Planck Institute for Intelligent Systems, 72076 Tuebingen, Germany (e-mail: okan.koc@tuebingen.mpg.de).

G. Maeda is with the Department of Brain Robot Interface, ATR Computational Neuroscience Laboratories, Kyoto 619-0288, Japan (e-mail: g.maeda@atr.jp).

J. Peters is with the Max Planck Institute for Intelligent Systems, 72076 Tuebingen, Germany, and also with the Technische Universitaet Darmstadt, 64289 Darmstadt, Germany (e-mail: jan.peters@tuebingen.mpg.de).

This paper has supplementary downloadable material available at http://ieeexplore.ieee.org, provided by the authors. The submitted video is in standard 'mp4' format and was run successfully on both Linux (Ubuntu 18.04) and Mac OS (Mojave), using default video players.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TRO.2019.2906558

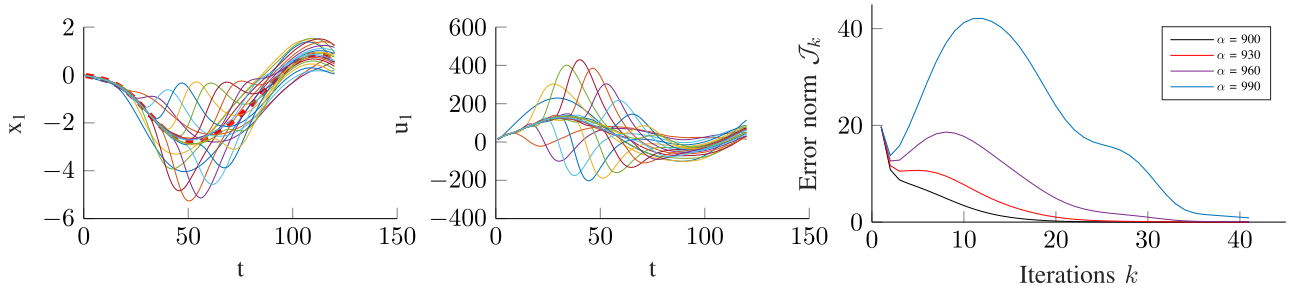

Fig. 2. Learning performance of ILC, using inaccurate models without incorporating a notion of uncertainty, may not be monotonic in practice. One can observe ripples that move through the trajectory, which can cause instability or damage the robot. In simulations, we can create this effect easily by increasing the spectral norm of the difference between the nominal and the actual (lifted) dynamics matrices. The desired trajectory for the first state $\mathbf{x}_1$ is shown in dashed red on the LHS for a 2-D linear time invariant system. The second plot shows the ILC feedforward commands for this particular trajectory and state. The third plot shows the Frobenius norm of the trajectory deviations, $\mathcal{J}_k$, plotted over the iterations k. The nonmonotonicity of the learning performance is aggravated, as the mismatch scale α controlling the spectral norm of the difference is increased. Increasing α further can prevent even asymptotic stability. The curves were generated by direct inversion of the (lifted) model. Our proposed Bayesian approach, on the other hand, minimizing the expected cost throughout the iterations, uses the posterior over the dynamics model parameters to make more cautious decisions.

and smooth reference trajectories with ILC, learning efficiency in robotics tasks can be improved significantly. However, it is rather difficult to ensure a stable learning performance in practice, see Fig. 2 for an illustration.

In this paper, we introduce a new model-based learning approach for tracking a variety of fast, dynamic movements stably, while maintaining learning and computational efficiency. Stability of the updates, or the probability of update monotonicity, is increased by making use of dynamics model covariance estimates. We refer to this as *caution* throughout the text, and the resulting algorithm is cautious precisely in this sense. A cautious learning control algorithm can hence be defined as one that incorporates a probabilistic notion of stability (in the iteration domain) during decision making, for the control input updates. This property proves to be critical, as we show the learning performance for fast robot table tennis striking movements. The proposed Bayesian approach, using the posterior over the dynamics model parameters, maintains both adaptation and caution in model-based ILC, while being efficient in terms of learning performance and computational complexity.

Our contributions can be stated succinctly as follows: We propose a new adaptive and cautious model-based ILC algorithm, that is implemented efficiently using a recursive formulation. More specifically, the existing model-based recursive ILC approach of Amann *et al.* [5], introduced briefly in Section II, is extended to include adaptation (by using Linear Bayes Regression on the errors) and caution (or in other terms, robustness to modeling errors, which shows itself as learning stability in the iteration domain). The proposed approach minimizes an expected quadratic cost term over the trajectory deviations, which still yields a closed-form solution, resulting in a cautious yet efficient learning performance. In the closed-form solution, the covariances of the learned local linear models are employed as adaptive regularizers.

The expected cost minimization distinguishes the framework from more conservative min–max approaches, such as the robustly convergent ILC proposed in the literature (using H_∞ and μ-synthesis techniques [8]). Related work in the theory and practice of ILC, as well as some more general applications of learning in robotics tasks, are briefly mentioned in Section I-A. Before introducing the expected cost minimization framework in Section IV, we discuss model adaptation in Section III with linear time-varying (LTV) models and show that Broyden's method [9] can be derived from linear Bayesian regression (LBR) as the forgetting factor goes to zero. Thus, the proposed approach belongs to the family of quasi-Newton ILC methods [10].

The resulting adaptive and cautious ILC algorithm, called *bayesILC*, is described in Section V, and extensions are discussed for additional robustness to nonrepetitive disturbances. Derivations for the recursive and cautious learning control update are given in Appendix A. We evaluate *bayesILC* first in extensive simulations in Section VI, showing that the proposed method is stable, efficient and can outperform other state-of-the-art learning approaches. We then present online learning results on our robot table tennis platform for tracking dynamic hitting movements. Appendix B briefly introduces the parameterization of these hitting movements. We discuss the real robot learning results in Section VII and conclude with brief mentions of promising future research directions.

A. Related Work

Since the 80s, there have been many different ILC update laws proposed, with the D-type update law of Arimoto *et al.* [11] being one of the first. See [3] and [4] for reviews and categorization of the different update laws. Theoretically, most ILC updates are *linear repetitive processes* that can be analyzed using two-dimensional (2-D) systems analysis [12], i.e., assuming the desired trajectory is fixed and the initial conditions can be reset perfectly, the error over the iterations has a (discrete) dynamics of its own. Stability of the ILC updates and monotonic convergence in particular can then be studied using dynamical systems theory. These notions also play an important role in the design of practical ILC algorithms. See [3] and [13] for a discussion and [14] for insight into convergence and stability issues appearing in an implementation.

Stability issues and the induced oscillations (see Fig. 2 for a simple simulation example) can easily damage the system to be controlled. For instance, joint limits can be exceeded in

a robotics application or other task-imposed state constraints can be violated. Such issues complicate the application of ILC in high-dimensional robotics problems. In practice, additional complications can occur, such as varying initial conditions, violating the assumptions made in most of the ILC literature. Robustness to varying initial conditions were considered, e.g., in [15]–[17]. For additional robustness to nonrepeating disturbances or noise, a robust feedback controller should be used alongside ILC, see e.g., [14] and [18].

Methods that learn to track (periodic or episodic) trajectories need to compensate for modeling uncertainties and other repetitive disturbances acting on the system to be controlled. However, methods that can efficiently learn the dynamics are model-based (e.g., most of optimization-based ILC [3], [5]) and at least require knowing the correct signs for the linearized dynamics of the system [19], [20].

When executing model-based learning algorithms on dynamical systems, it is essential for stability and safety to incorporate a notion of model uncertainty. Otherwise the learning algorithms can be overconfident and quickly go unstable [14]. One way to achieve a more stable performance in ILC is to filter the high-frequency updates. These robust methods are mostly known as Q-filtering [3] and typically incur a tradeoff between stability and performance: the system will often fail to converge to the minimal steady-state error. In this paper, we use a different (probabilistic) approach to increase the stability margins of model-based ILC that does not incur such a tradeoff. To that end, we expand on the previous work of Amann *et al.* [5], one of the first model-based ILC approaches introducing an optimal-control-based ILC design. The recursive implementation first introduced in this paper closely relates to numerically-stable plant-inversion approaches [21]. We extend the recursive formulation to include adaptation and caution: adaptation of the model parameter means and variances are performed at each iteration using LBR. The resulting Bayesian approach, minimizing the expected cost throughout the iterations, uses the posterior over the dynamics model parameters to make more cautious decisions.

Model adaptation in ILC can be studied in the context of solving nonlinear equations. Tracking a fixed reference perfectly corresponds to solving for the control inputs that drive the deviations to zero. Hence, quasi-Newton methods such as the Broyden's method [9] and generalized secant method [22] were proposed as adaptation methods in the ILC literature to update the plant dynamics. Broyden's method, without having access to the gradients of a black-box function $\mathbf{f}(\mathbf{x})=\mathbf{0}$, maintains a Jacobian matrix approximation $\mathbf{F}$. The matrix $\mathbf{F}$ is updated at each iteration k in order to satisfy the *secant equation*

$$\mathbf{f}_k - \mathbf{f}_{k-1} = \mathbf{F}_k(\mathbf{x}_k - \mathbf{x}_{k-1}) \tag{1}$$

which can then be inverted to yield[1]

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \mathbf{F}_k^{\dagger}\mathbf{f}_k. \tag{2}$$

Convergence under restrictive assumptions have been shown for Broyden's method. For solving systems of nonlinear equations, arguably efficiency rather than stability or monotonic convergence is of importance, and a simple trust-region approach (based on a merit function) suffices to improve stability. We will show how Broyden's method can be seen as a limiting case of LBR in Section III. The proposed method thus belongs to the family of quasi-Newton optimization methods [9], where the black-box nature of the quasi-Newton approaches is augmented to include caution during the ILC updates: monotonic convergence, or update stability in the iteration domain, is of paramount importance in robotics tasks.

An application of model-based ILC to reject repeating disturbances was shown in quadrocopter flight [23], where a constrained convex optimization with imposed control input limits was solved, rather than a direct inversion of the nominal model dynamics. An impressive application of ILC to a robotic surgical task was presented in [24] utilizing an EM-based ILC update law. ILC was also combined with robust observers to control a heavy-duty hydraulic arm in an excavation task [25].

Besides ILC, another learning framework that learns inaccurate models for control is model-based RL. Including variance fully in the decision-making process can result in efficient and stable learning [26]. However such involved procedures exhibit computational runtime difficulties and have not been implemented in high-dimensional real-time robotics tasks.

II. Problem Statement and Background

Most tasks in robotics can be learned more efficiently whenever feasible trajectories are available. Learning-based control approaches can then focus on tracking these trajectories without relying on accurate models. The goal in trajectory tracking is to track a given reference $\mathbf{r}(t),\ 0 \le t \le T$, by applying the control inputs $\mathbf{u}(t)$. In dynamic robotic tasks, the references are often in the combined state space of joint positions and velocities $(\mathbf{q}^{\mathrm{T}}, \dot{\mathbf{q}}^{\mathrm{T}})^{\mathrm{T}} \in \mathcal{Q} \subset \mathbb{R}^{2n}$, and the control inputs $\mathbf{u} \in \mathcal{U} \subset \mathbb{R}^m$ are applied for each joint of the robot, i.e., $m = n$. The reference trajectories in table tennis, for instance, enable the execution of hitting and striking motions, e.g., forehand and backhand strikes. Such trajectories can be generated online with nonlinear constrained optimization [2]. Finding the right control inputs to track them accurately is the focus of ILC.

1) Linearizing an Inaccurate Model: Consider a nonlinear robot dynamics model

$$\ddot{\mathbf{q}} = \mathbf{f}(\mathbf{q}, \dot{\mathbf{q}}, \mathbf{u}) \tag{3}$$

e.g., for rigid body dynamics of the form

$$\ddot{\mathbf{q}} = \mathbf{M}^{-1}(\mathbf{q})\{\mathbf{u} - \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} - \mathbf{G}(\mathbf{q})\} \tag{4}$$

where on the right-hand side (RHS) are the inverse of the inertia matrix $\mathbf{M}(\mathbf{q})$, the Coriolis and centrifugal forces $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}}$, and the vector of gravitational forces $\mathbf{G}(\mathbf{q})$. This nonlinear dynamics model can be linearized around a given joint space trajectory $\mathbf{r}(t),\ 0 \le t \le T$, with nominal inputs $\mathbf{u}_{\mathrm{IDM}}(t)$ calculated using the inverse dynamics model [27]. We then obtain the following LTV representation:

$$\dot{\mathbf{e}}(t) = \mathbf{A}(t)\mathbf{e}(t) + \mathbf{B}(t)\boldsymbol{\delta}\mathbf{u}(t) + \mathbf{d}(t, \mathbf{u}) \tag{5}$$

where the state vector is the joint angles and velocities $\mathbf{x} = [\mathbf{q}^{\intercal}, \dot{\mathbf{q}}^{\intercal}]^{\mathrm{T}}$, the state error is denoted as $\mathbf{e}(t) = \mathbf{x}(t) - \mathbf{r}(t)$, the deviations from the nominal inputs are $\boldsymbol{\delta}\mathbf{u}(t) = \mathbf{u}(t) - \mathbf{u}_{\mathrm{IDM}}(t)$

[1] Broyden's method can also directly update the inverse.

and the continuous time-varying matrices are

$$\mathbf{A}(t) = \left.\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\right|_{(\mathbf{r}(t),\mathbf{u}_{\mathrm{IDM}}(t))}, \mathbf{B}(t) = \left.\frac{\partial \mathbf{f}}{\partial \mathbf{u}}\right|_{(\mathbf{r}(t),\mathbf{u}_{\mathrm{IDM}}(t))}. \quad (6)$$

In the error dynamics (5), the additional (unknown) term $\mathbf{d}(t, \mathbf{u})$ accounts for the disturbances and the effects of the linearization (i.e., higher order terms). We can discretize (5)–(6) with step size δ, $N = T/\delta$ and step index $j = 1, \ldots, N$ to get the following discrete-time linear system:

$$\mathbf{e}_{j+1} = \mathbf{A}_j \mathbf{e}_j + \mathbf{B}_j \boldsymbol{\delta}\mathbf{u}_j + \mathbf{d}_{j+1} \quad (7)$$

where the matrices $\mathbf{A}_j, \mathbf{B}_j$ are the discretizations of (6). Conventional (discrete) ILC algorithms learn to compensate for the errors incurred along the trajectory by updating the control inputs $\boldsymbol{\delta}\mathbf{u}_j$ iteratively.

Whenever we refer to the outcome of a particular iteration k, we will use the first subindex for iterations and the second subindex will be used to denote the (discrete) time step, i.e., the vectors $\mathbf{e}_{k,j} \in \mathbb{R}^{2n}$, $\boldsymbol{\delta}\mathbf{u}_{k,j} \in \mathbb{R}^m$ denote the deviations and control input compensations at the time step j during iteration k, respectively. The control commands to be applied at iteration $k+1$ as

$$\mathbf{u}_{k+1,j} = \mathbf{u}_{k,j} + \boldsymbol{\delta}\mathbf{u}_{k,j} \quad (8)$$

are computed using the deviations $\mathbf{e}_{k,j}$ at iteration k.

2) Recursive Norm-Optimal ILC: Norm-optimal ILC uses the discrete LTV model in (7) to minimize the next iteration errors, where the computed control inputs are optimal with respect to some vector norm. These approaches based on optimality criteria can learn efficiently by taking advantage of the inaccurate models. Batch methods that compute the next iteration compensations stack the model matrices together to compute (a possibly weighted and dampened) pseudoinverse of this block lower-diagonal matrix. As an alternative, some methods use convex programming to compute these optimal compensations under additional constraints.

The condition of this *lifted* model matrix typically grows exponentially with the horizon size N and computing the pseudoinverse stably becomes very difficult. Downsampling trajectories restores the condition number and a stable inversion becomes much more manageable, at the cost of reduced tracking performance. As a better alternative, optimization-based approaches, depending on the particular optimizer, may avoid computing the pseudoinverse. However, such approaches can still be computationally intensive, and may not be suitable for online learning.

As an alternative, Amann *et al.* [5] have shown that the direct batch inversion of the lifted model matrix can be avoided by recursively computing the ILC compensations (in one pass) using the linear quadratic regulator (LQR) for disturbance rejection [28]. After estimating the disturbances $\mathbf{d}_{j+1}$ at the k'th trial, the optimal control problem for tracking a desired trajectory can be written as

$$\min_{\boldsymbol{\delta}\mathbf{u}} \sum_{j=1}^{N} \mathbf{e}_{k+1,j}^{\mathrm{T}} \mathbf{Q}_j \mathbf{e}_{k+1,j} + \boldsymbol{\delta}\mathbf{u}_{k,j}^{\mathrm{T}} \mathbf{R}_j \boldsymbol{\delta}\mathbf{u}_{k,j}$$
$$\text{s.t. } \mathbf{e}_{k+1,j+1} = \mathbf{A}_j \mathbf{e}_{k+1,j} + \mathbf{B}_j \mathbf{u}_{k+1,j} + \mathbf{d}_{j+1}. \quad (9)$$

Reduction of the ILC problem to the known LQR solution has not attracted much attention however from the control and learning communities, since it was not clear how to study stability and convergence in this formulation.

III. Model Adaptation

Whenever there is model-mismatch, the (linearized) models cannot be assumed to hold accurately around the reference trajectory. There is hence a risk that the learning process described in Section II will not be stable. As a remedy, in this section we propose a natural Bayesian adaptation of model matrices with LBR and discuss different alternatives in the context of robotics.

A. Recursive Estimation of Model Matrices

The observed deviations from the trajectory at iteration k, $\mathbf{e}_{k,j}$, can be used to update the discrete-time LTV model matrices $\mathbf{A}_{k,j}, \mathbf{B}_{k,j}$ that describe the nonlinear dynamics around the trajectory, to first order. Instead of estimating all the parameters together in a costly estimation procedure, the model matrices $\mathbf{A}_{k,j}, \mathbf{B}_{k,j}$ can rather be updated separately for each $j = 1, \ldots, N$, given the smoothened errors $\hat{\mathbf{e}}_{k,j}$

$$\hat{\mathbf{e}}_{k,j+1} = \mathbf{A}_{k,j} \hat{\mathbf{e}}_{k,j} + \mathbf{B}_{k,j} \mathbf{u}_{k,j} + \mathbf{d}_{j+1} \quad (10)$$

which can be rewritten using the Kronecker product and the vectorization operator as follows:

$$\hat{\mathbf{e}}_{k,j+1} - \hat{\mathbf{e}}_{k-1,j+1} \approx \mathbf{X}_{k,j} \operatorname{vec}[\mathbf{A}_{k,j}, \mathbf{B}_{k,j}]$$
$$\mathbf{X}_{k,j} = \operatorname{vec}[\hat{\mathbf{e}}_{k,j} - \hat{\mathbf{e}}_{k-1,j}, \boldsymbol{\delta}\mathbf{u}_{k,j}]^{\mathrm{T}} \otimes \mathbf{I}. \quad (11)$$

If we incorporate the belief (including the uncertainty) about the linear dynamics models as Gaussian priors in LBR

$$\boldsymbol{\theta}_{k,j} = \operatorname{vec}[\mathbf{A}_{k,j}, \mathbf{B}_{k,j}]$$
$$\mathbf{y}_{k,j} = \hat{\mathbf{e}}_{k,j+1} - \hat{\mathbf{e}}_{k-1,j+1}$$
$$\rho(\boldsymbol{\theta}_{k,j}|\mathbf{y}_{k,j}) \propto \rho(\mathbf{y}_{k,j}|\boldsymbol{\theta}_{k,j})\rho(\boldsymbol{\theta}_{k,j})$$
$$\rho(\boldsymbol{\theta}_{k,j}) = \mathcal{N}(\boldsymbol{\theta}_{k,j}|\boldsymbol{\mu}_{k,j}, \boldsymbol{\Sigma}_{k,j}) \quad (12)$$

with a Gaussian likelihood function

$$\rho(\mathbf{y}_{k,j}|\boldsymbol{\theta}_{k,j}) = \mathcal{N}(\mathbf{y}_{k,j}|\mathbf{X}_{k,j}\boldsymbol{\theta}_{k,j}, \sigma^2\mathbf{I}) \quad (13)$$

the models parameter means $\boldsymbol{\mu}_{k,j}$ and variances $\boldsymbol{\Sigma}_{k,j}$ can be updated as

$$\boldsymbol{\Sigma}_{k,j} = (\tfrac{1}{\sigma^2}\mathbf{X}_{k,j}^{\mathrm{T}}\mathbf{X}_{k,j} + \boldsymbol{\Sigma}_{k-1,j}^{-1})^{-1}$$
$$\boldsymbol{\mu}_{k,j} = \boldsymbol{\Sigma}_{k,j}\left(\boldsymbol{\Sigma}_{k-1,j}^{-1}\boldsymbol{\mu}_{k-1,j} + \tfrac{1}{\sigma^2}\mathbf{X}_{k,j}^{\mathrm{T}}\mathbf{y}_{k,j}\right). \quad (14)$$

Smoothened position and velocity error estimates can be obtained, for example, using a zero-phase Butterworth filter.

1) Relation to Broyden's Method: Broyden's method [9] can be seen as a limiting case of LBR. The mean estimates in (14) are also the solutions of the following linear ridge regression problem:

$$\min_{\boldsymbol{\theta}} \tfrac{1}{\sigma^2}\|\mathbf{y}_{k,j} - \mathbf{X}_{k,j}\boldsymbol{\theta}\|_2^2 + (\boldsymbol{\theta} - \boldsymbol{\theta}_{k,j})\boldsymbol{\Sigma}_{k,j}^{-1}(\boldsymbol{\theta} - \boldsymbol{\theta}_{k,j}) \quad (15)$$

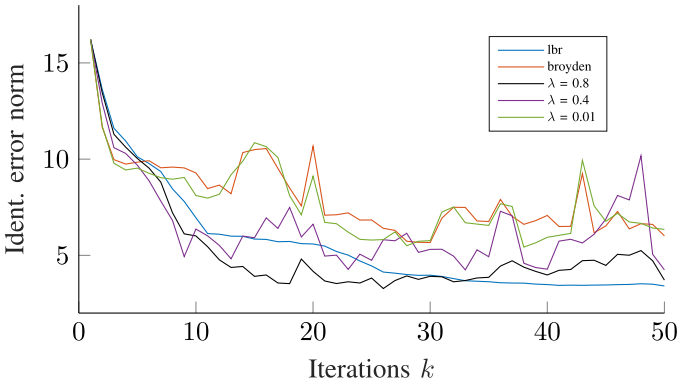

Fig. 3. Broyden's method [9], which can be considered as an adaptation framework within ILC, is a limiting case of LBR. As the forgetting factor λ of an exponentially weighted LBR model goes to zero, LBR transitions to Broyden's method. Broyden's method is very sensitive to noise and adapts very aggressively. Throughout the paper, we discuss and evaluate several adaptation laws, which are less sensitive to noise but are still flexible. The figure shows the evolution of the identification error norm for an unknown LTV system. The Frobenius norm of the difference between the adapted model matrices ($\mathbf{A}_{k,j}$ and $\mathbf{B}_{k,j}$) and the actual (fixed) matrices (denoted as identification error norm) are plotted for each iteration $k = 1, \ldots, 50$.

and as $\sigma^2 \to 0$ we get the (weighted) Broyden's update for one iteration, which, written in vectorized form, is solving independently for every time step

$$\min_{\boldsymbol{\theta}} \ (\boldsymbol{\theta} - \boldsymbol{\theta}_{k,j})\boldsymbol{\Sigma}_{k,j}^{-1}(\boldsymbol{\theta} - \boldsymbol{\theta}_{k,j}) \tag{16}$$

$$\text{s.t.} \ \ \mathbf{y}_{k,j} = \mathbf{X}_{k,j}\boldsymbol{\theta}. \tag{17}$$

Broyden's method is too sensitive to the sensor noise in robotics tasks as it satisfies the secant rule (17) exactly. On the other hand, LBR in (14) for fixed noise parameter σ^2, is using *all* of the past iteration data equally. The norm of the covariance decreases monotonically in each update. For unknown dynamic systems that are highly nonlinear but smooth, to prevent premature shrinking of the covariance matrix, a better alternative is to set an *exponential weighting* in the adaptation. For a fixed forgetting factor $\lambda \in [0, 1]$, the update in (14) becomes

$$\begin{aligned}\boldsymbol{\Sigma}_{k,j} &= (\tfrac{1}{\sigma^2}\mathbf{X}_{k,j}^{\mathrm{T}}\mathbf{X}_{k,j} + \lambda\boldsymbol{\Sigma}_{k-1,j}^{-1})^{-1},\\ \boldsymbol{\mu}_{k,j} &= \lambda\boldsymbol{\Sigma}_{k,j}\boldsymbol{\Sigma}_{k-1,j}^{-1}\boldsymbol{\mu}_{k-1,j} + \tfrac{1}{\sigma^2}\boldsymbol{\Sigma}_{k,j}\mathbf{X}_{k,j}^{\mathrm{T}}\mathbf{y}_{k,j}.\end{aligned} \tag{18}$$

The forgetting factor λ is used to perform exponential weighting of the previous iteration data. As $\lambda \to 0$, we get the (unweighted) Broyden's method,[2] and as $\lambda \to 1$, (18) reduces to (14). Hence, our proposed adaptation law (18) can be embedded within a one-parameter family of quasi-Newton ILC methods, where the forgetting factor parameter trades off adaptation flexibility and robustness to noise. At the one end of the spectrum, Broyden's method adapts flexibly and aggressively to the latest data at the cost of being very sensitive to noise. This can be alleviated with a judicious choice of the forgetting factor. See Fig. 3 for an illustration.

[2]Unlike the case where $\sigma^2 \to 0$, this equivalence is valid for all the subsequent iterations as well. It can be seen more easily from the filter form of (18).

B. Imposing Structure

The structure in the forward dynamics model (4) is not considered in the update rule (18): any change in the control inputs in this model directly affects the instantaneous joint accelerations, and only indirectly the joint velocities in the future time steps. By differentiating the smoothened joint velocities, one can instead impose the following regression model:

$$\ddot{\mathbf{q}}_{k,j} - \ddot{\mathbf{q}}_{k-1,j} \approx \mathbf{A}_k(\delta j)\mathbf{e}_{k,j} + \mathbf{B}_k(\delta j)\boldsymbol{\delta}\mathbf{u}_{k,j} \tag{19}$$

where we dropped the hat for notational simplicity. The *continuous* model matrices $\mathbf{A}_k(\delta j), \mathbf{B}_k(\delta j)$ are members of a reduced parameter space, i.e., $\mathbf{A}_k(\delta j) \in \mathbb{R}^{n\times 2n}, \mathbf{B}_k(\delta j) \in \mathbb{R}^{n\times m}$, $j = 1, \ldots, N$. After regressing on the continuous model matrices as in (14), they can be discretized (as discussed before) to form the discrete-time model parameter means $\mathbf{A}_{k,j} \in \mathbb{R}^{2n\times 2n}, \mathbf{B}_{k,j} \in \mathbb{R}^{2n\times m}$, and covariances $\boldsymbol{\Sigma}_{k,j}$.

As an alternative, note that the rigid body dynamics (3) is parameterized by the link masses, three link center of mass values, and six inertia parameters. A total of ten parameters are used for each link to fully parameterize the inverse dynamics model

$$\mathbf{u} = \mathbf{M}(\mathbf{q};\boldsymbol{\theta})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q},\dot{\mathbf{q}};\boldsymbol{\theta})\dot{\mathbf{q}} + \mathbf{G}(\mathbf{q};\boldsymbol{\theta}) \tag{20}$$

which can be stacked for each $j = 1, \ldots, N$ to form the regression model

$$\begin{aligned}\mathbf{U}_k &\approx \mathbf{Y}(\mathbf{Q}_k^{(0)}, \mathbf{Q}_k^{(1)}, \mathbf{Q}_k^{(2)})\boldsymbol{\theta}_k\\ \mathbf{U}_k &= \left(\mathbf{u}_{k,1}^{\mathrm{T}}, \mathbf{u}_{k,2}^{\mathrm{T}}, \ldots, \mathbf{u}_{k,N}^{\mathrm{T}}\right)^{\mathrm{T}}\\ \mathbf{Q}_k^{(l)} &= \left(\mathbf{q}_{k,1}^{(l)\mathrm{T}}, \mathbf{q}_{k,2}^{(l)\mathrm{T}}, \ldots, \mathbf{q}_{k,N}^{(l)\mathrm{T}}\right)^{\mathrm{T}}, \ l = 0, 1, 2\end{aligned} \tag{21}$$

where $\boldsymbol{\theta}_k \in \mathbb{R}^{10n}$ appears *linearly*. The index l denotes the degree of the derivatives of the smoothened joint angles, i.e., $l = 0, 1, 2$ are used to denote the joint position, velocity, and acceleration estimates in (21), respectively. Based on these joint estimates, only the link parameters are updated with LBR as in (14). The forward dynamics model[3] (3) can then be used to sample the means and variances of the continuous LTV matrices, e.g., using Monte Carlo sampling. Discretization as discussed earlier converts the continuous-time model parameter means and variances into their discrete-time form. An advantage of this approach is to *compress* learning to a lower dimensional space, reducing the variance of the updates at the cost of an introduced bias. Moreover, since the link parameters are invariant throughout the iterations, such an update avoids the flexible yet independent adaptation of the model matrices for each j, and the necessity of introducing a forgetting factor.

IV. Cautious Learning Control

The posterior model covariances $\boldsymbol{\Sigma}_{k,j}$ can be used to make more *cautious* decisions within a stochastic control framework. The uncertainty of the model parameters can be seen as a *multiplicative* noise model and the ILC optimality criterion (9) can

[3]The forward dynamics model (3), unlike the inverse dynamics (21), depends *nonlinearly* on the link parameters.

be extended to include expectations over them. The multiplicative noise model, unlike the additive noise case, does not lead to *certainty-equivalence*: the covariance estimates are incorporated in the decision rule. To see how the expected cost minimization leads to caution, note that

$$\mathbb{P}\left(\mathbf{e}_{k+1,j}^{\mathrm{T}}\mathbf{Q}_j\mathbf{e}_{k+1,j} \geq \hat{\mathbf{e}}_{k,j}^{\mathrm{T}}\mathbf{Q}_j\hat{\mathbf{e}}_{k,j}\right) \leq \frac{\mathbb{E}[\mathbf{e}_{k+1,j}^{\mathrm{T}}\mathbf{Q}_j\mathbf{e}_{k+1,j}]}{\hat{\mathbf{e}}_{k,j}^{\mathrm{T}}\mathbf{Q}_j\hat{\mathbf{e}}_{k,j}} \tag{22}$$

which follows from Markov's inequality. Minimizing the upper bound forces the probability of nonmonotonicity to be low as well.

1) Expected Cost Minimization: For the expected cost case, where the expectation is taken over the *random variables* $\mathbf{A}_{k,j}$ and $\mathbf{B}_{k,j}$, for each j, the optimality criterion

$$\begin{aligned}&\min_{\boldsymbol{\delta\mathbf{u}}} \sum_{j=1}^{N} \mathbb{E}_{\mathbf{A}_{k,j},\mathbf{B}_{k,j}}\left[\mathbf{e}_{k+1,j}^{\mathrm{T}}\mathbf{Q}_j\mathbf{e}_{k+1,j}+\boldsymbol{\delta}\mathbf{u}_{k,j}^{\mathrm{T}}\mathbf{R}_j\boldsymbol{\delta}\mathbf{u}_{k,j}\right] \\ &\text{s.t. } \mathbf{e}_{k+1,j+1} = \mathbf{A}_{k,j}\mathbf{e}_{k+1,j} + \mathbf{B}_{k,j}\mathbf{u}_{k+1,j} + \mathbf{d}_{j+1}\end{aligned} \tag{23}$$

can be solved recursively using dynamic programming [29]

$$\begin{aligned}\boldsymbol{\delta}\mathbf{u}_{k,j} &= \mathbf{K}_{k,j}\mathbf{e}_{k+1,j} - \boldsymbol{\Phi}_{k,j}^{-1}\boldsymbol{\ell}_{k,j} \\ \mathbf{K}_{k,j} &= -\boldsymbol{\Phi}_{k,j}^{-1}\boldsymbol{\Psi}_{k,j} \\ \boldsymbol{\Phi}_{k,j} &= \mathbb{E}_{\mathbf{B}_{k,j}}\left[\mathbf{B}_{k,j}^{\mathrm{T}}\mathbf{P}_{k,j+1}\mathbf{B}_{k,j}\right] + \mathbf{R}_j \\ \boldsymbol{\Psi}_{k,j} &= \mathbb{E}_{\mathbf{A}_{k,j},\mathbf{B}_{k,j}}\left[\mathbf{B}_{k,j}^{\mathrm{T}}\mathbf{P}_{k,j+1}\mathbf{A}_{k,j}\right] \\ \boldsymbol{\ell}_{k,j} &= \mathbb{E}_{\mathbf{B}_{k,j}}\left[\mathbf{B}_{k,j}^{\mathrm{T}}\mathbf{P}_{k,j+1}(\mathbf{B}_{k,j}\mathbf{u}_{k,j}+\mathbf{d}_{j+1})+\mathbf{B}_{k,j}^{\mathrm{T}}\mathbf{b}_{k,j+1}\right]\end{aligned} \tag{24}$$

where $\mathbf{b}_{k,j}$ and the Ricatti matrices $\mathbf{P}_{k,j}$ evolve backwards according to

$$\begin{aligned}\mathbf{P}_{k,j} &= \mathbf{Q}_j + \mathbf{M}_{k,j} - \boldsymbol{\Psi}_{k,j}^{\mathrm{T}}\boldsymbol{\Phi}_{k,j}^{-1}\boldsymbol{\Psi}_{k,j} \\ \mathbf{M}_{k,j} &= \mathbb{E}_{\mathbf{A}_{k,j}}\left[\mathbf{A}_{k,j}^{\mathrm{T}}\mathbf{P}_{k,j+1}\mathbf{A}_{k,j}\right] \\ \mathbf{b}_{k,j} &= \mathbb{E}_{\mathbf{A}_{k,j},\mathbf{B}_{k,j}}\left[\bar{\mathbf{A}}_{k,j}^{\mathrm{T}}(\mathbf{b}_{k,j+1}+\mathbf{P}_{k,j+1}(\mathbf{B}_{k,j}\mathbf{u}_{k,j}+\mathbf{d}_{j+1}))\right]\end{aligned} \tag{25}$$

starting from $\mathbf{P}_{k,N} = \mathbf{Q}_N$ and $\mathbf{b}_{k,N} = \mathbf{0}$. The random closed loop system dynamics is given by the matrices

$$\bar{\mathbf{A}}_{k,j} = \mathbf{A}_{k,j} + \mathbf{B}_{k,j}\mathbf{K}_{k,j}. \tag{26}$$

By a direct comparison to the LQR solution to (9), it can be seen that the control input compensations $\boldsymbol{\delta}\mathbf{u}_{k,j}$ in (24) are computed similarly, with the appropriate expectations added. The ILC update is decomposed into two components: a current-iteration feedback term $\mathbf{u}_{\mathrm{fb}} = \mathbf{K}_{k,j}\mathbf{e}_{k+1,j}$ calculated using the iteration dependent Riccati equations and a feedforward, purely predictive term $\mathbf{u}_{\mathrm{ff}} = -\boldsymbol{\Phi}_{k,j}^{-1}\boldsymbol{\ell}_{k,j}$, solved backwards for each $j = 1, \ldots, N$. The feedforward terms are responsible for compensating for the estimated *random* disturbances $\mathbf{d}_j$, calculated using (10).

Cautious update (24) can be implemented without explicitly calculating the disturbances. If the disturbances are taken as random variables defined via the filtered errors $\hat{\mathbf{e}}_{k,j}$ of the last iteration

$$\mathbf{d}_{j+1} = \hat{\mathbf{e}}_{k,j+1} - \mathbf{A}_{k,j}\hat{\mathbf{e}}_{k,j} - \mathbf{B}_{k,j}\mathbf{u}_{k,j} \tag{27}$$

the recursion can be simplified by introducing

$$\boldsymbol{\nu}_{k,j} = \mathbf{b}_{k,j} + \mathbf{P}_{k,j}\mathbf{e}_{k,j}. \tag{28}$$

The feedforward and feedback compensations $\boldsymbol{\delta}\mathbf{u}_{k,j}$ can then directly be computed as

$$\begin{aligned}\boldsymbol{\delta}\mathbf{u}_{k,j} &= \mathbf{K}_{k,j}(\mathbf{e}_{k+1,j} - \mathbf{e}_{k,j}) - \boldsymbol{\Phi}_{k,j}\mathbb{E}_{\mathbf{B}_{k,j}}\left[\mathbf{B}_{k,j}^{\mathrm{T}}\boldsymbol{\nu}_{k,j+1}\right] \\ \boldsymbol{\nu}_{k,j} &= \mathbb{E}_{\mathbf{A}_{k,j},\mathbf{B}_{k,j}}\left[\bar{\mathbf{A}}_{k,j}^{\mathrm{T}}\boldsymbol{\nu}_{k,j+1}\right] + \mathbf{Q}_j\mathbf{e}_{k,j}.\end{aligned} \tag{29}$$

See Appendix A for a detailed derivation. Equation (29) is easier to implement, since the disturbances do not need to be estimated explicitly. The compensations $\boldsymbol{\delta}\mathbf{u}_{k,j}$ are added to the total control inputs applied at iteration k. In an adaptive implementation, the feedback components of the update, $\mathbf{K}_{k,j}(\mathbf{e}_{k+1,j} - \mathbf{e}_{k,j})$, does not completely subtract the previous feedback controls $\mathbf{K}_{k-1,j}\mathbf{e}_{k,j}$ from the total control inputs, as the feedback matrices are also adapted over the iterations.

Typically ILC is used to feed the past errors along the trajectory (filtered and multiplied with a *learning* matrix) back to the system for the next trial as feedforward compensations. A well-designed feedback controller, whenever available, is only used to reject nonrepeating disturbances and to stabilize the system in the time domain. The recursive implementation (29), on the other hand, readily provides and updates a feedback controller based on past performance. From here on, we will refer to the feedforward part of (29) as $\boldsymbol{\delta}\mathbf{u}_{k,j}$, keeping the feedback control separate.

2) Computing the Expectations: The expectations appearing in (24) can be calculated given the covariances $\boldsymbol{\Sigma}_{k,j}$ of the parameters

$$\begin{aligned}\boldsymbol{\Phi}_{k,j} &= \tilde{\boldsymbol{\Phi}}_{k,j} + \mathbf{R}_j \\ \tilde{\boldsymbol{\Phi}}_{k,j}^{a,b} &= \sum_{c=1}^{n}\sum_{d=1}^{n}\mathbf{P}_{k,j+1}^{c,d}\left(\mathbb{E}\left[\mathbf{B}_{k,j}^{c,a}\right]\mathbb{E}\left[\mathbf{B}_{k,j}^{d,b}\right] + \sigma\left(\mathbf{B}_{k,j}^{c,a}, \mathbf{B}_{k,j}^{d,b}\right)\right) \\ \boldsymbol{\Psi}_{k,j}^{a,b} &= \sum_{c=1}^{n}\sum_{d=1}^{n}\mathbf{P}_{k,j+1}^{c,d}\left(\mathbb{E}\left[\mathbf{B}_{k,j}^{c,a}\right]\mathbb{E}\left[\mathbf{A}_{k,j}^{d,b}\right] + \sigma\left(\mathbf{B}_{k,j}^{c,a}, \mathbf{A}_{k,j}^{d,b}\right)\right) \\ \mathbf{M}_{k,j}^{a,b} &= \sum_{c=1}^{n}\sum_{d=1}^{n}\mathbf{P}_{k,j+1}^{c,d}\left(\mathbb{E}\left[\mathbf{A}_{k,j}^{c,a}\right]\mathbb{E}\left[\mathbf{A}_{k,j}^{d,b}\right] + \sigma\left(\mathbf{A}_{k,j}^{c,a}, \mathbf{A}_{k,j}^{d,b}\right)\right)\end{aligned} \tag{30}$$

where the upper indices a, b denote the corresponding entry of the matrix appearing on the left-hand side (LHS). The covariance matrices $\boldsymbol{\Sigma}_{k,j}$ contain the scalar covariance terms $\sigma(\cdot)$ on the relevant entries, i.e.,

$$\begin{aligned}\sigma\left(\mathbf{A}_{k,j}^{c,a}, \mathbf{A}_{k,j}^{d,b}\right) &= \left(\boldsymbol{\Sigma}_{k,j}\right)^{(a-1)n+c,(b-1)n+d} \\ \sigma\left(\mathbf{B}_{k,j}^{c,a}, \mathbf{A}_{k,j}^{d,b}\right) &= \left(\boldsymbol{\Sigma}_{k,j}\right)^{n^2+(a-1)n+c,(b-1)n+d} \\ \sigma\left(\mathbf{B}_{k,j}^{c,a}, \mathbf{B}_{k,j}^{d,b}\right) &= \left(\boldsymbol{\Sigma}_{k,j}\right)^{n^2+(a-1)n+c,n^2+(b-1)n+d}.\end{aligned} \tag{31}$$

The indexes of $\mathbf{B}_{k,j}$ covariances start from n^2 since the model matrix parameters in (12) are vectorized starting from $\mathbf{A}_{k,j}$.

Algorithm 1: Recursive, Adaptive, and Cautious *bayesILC*.

Require: $\mathbf{f}_{\text{nom}}$, $\mathbf{r}_j$, $\lambda, \epsilon > 0$, $\mathbf{Q}_j \succeq 0$, $\mathbf{R}_j \succ 0$, $\mathbf{\Sigma}_{0,j} \succ 0$

1: Move to initial posture $\mathbf{q}_0 = \mathbf{r}_0$, $\dot{\mathbf{q}}_0 = \mathbf{0}$.
2: Initialize $k = 1$, $\boldsymbol{\delta}\mathbf{u}_{0,j} = \mathbf{0}$, $j = 1, \ldots, N$
3: Compute mean dyn. parameters $\boldsymbol{\mu}_{0,j}$ by linearizing $\mathbf{f}_{\text{nom}}$
4: Compute feedback $\mathbf{K}_{0,j} = \text{LQR}(\mathbf{Q}_j, \mathbf{R}_j, \boldsymbol{\mu}_{0,j}, \mathbf{\Sigma}_{0,j})$
5: Execute with inv. dyn. $\mathbf{u}_{\text{IDM}}$ and feedback $\mathbf{K}_{0,j}$
6: Filter errors with a zero-phase filter (output: $\hat{\mathbf{e}}_{0,j}$)
7: **repeat** \\ ILC operation
8: Compute error norm $\mathcal{J}_k = \left(\sum_{j=1}^{N} \hat{\mathbf{e}}_{k,j}^{\mathrm{T}} \mathbf{Q}_j \hat{\mathbf{e}}_{k,j}\right)^{1/2}$
9: Compute $\boldsymbol{\delta}\mathbf{u}_{k,j}, \mathbf{K}_{k,j}$ recursively using (24)–(29)
10: Update feedforward controls $\mathbf{u}_{k+1,j} = \mathbf{u}_{k,j} + \boldsymbol{\delta}\mathbf{u}_{k,j}$
11: Execute with $\mathbf{u}_{\text{IDM},j} + \mathbf{u}_{k+1,j}$ and feedback $\mathbf{K}_{k,j}$
12: Observe errors $\mathbf{e}_{k,j} = \mathbf{x}_{k,j} - \mathbf{r}_j$
13: Filter errors with a zero-phase filter (output: $\hat{\mathbf{e}}_{k,j}$)
14: Update model $\boldsymbol{\mu}_{k,j}, \mathbf{\Sigma}_{k,j}$ using (18)
15: $k \leftarrow k + 1$
16: **until** $\mathcal{J}_k < \epsilon$

V. Online Implementation

In this section, we algorithmically describe the recursive, adaptive, and cautious *bayesILC* proposed in Sections III and IV in detail, with the extensions for an online robot learning application. We will consider tracking table tennis trajectories as our application of choice. The online learning algorithm is readily applicable to similar dynamic tasks with real-time constraints, such as throwing, catching skills in sports or fast, demanding manufacturing tasks.

The proposed ILC framework is summarized in Algorithm 1. Before entering the main loop (lines 7−16), the trajectory is executed with inverse dynamics and time-varying LQR feedback (line 5). The errors along the trajectory are filtered with a zero-phase filter (line 6). During the cautious ILC update the feedback control law as well as the feedforward control inputs are updated recursively (line 9). From the first iteration onwards, the means and the covariances of the model matrices are updated (line 14) before computing the feedforward input compensations $\boldsymbol{\delta}\mathbf{u}_{k,j}$ and the feedback matrices $\mathbf{K}_{k,j}$. If the variant adaptation laws discussed in Section III are employed, it will be enough to store the means and covariances of the relevant model parameters. These parameters can then be transformed, as discussed before, to form the discrete-time model matrix means and covariances, which are used in the cautious ILC update (line 9).

Based on the forgetting factor λ, the model adaptation strikes a balance between the prior model parameter distribution and the data observed in iteration k. For the discrete LTV model and the link parameter adaptation, the data used is $\mathbf{y}_{k,j} = \hat{\mathbf{e}}_{k,j+1} - \hat{\mathbf{e}}_{k-1,j+1}$. If continuous model matrix adaptation is performed, the data will instead be the smoothened joint acceleration differences, see (19). We discuss the effects of the forgetting factor and the different model adaptation strategies in more detail in Section VI.

The practitioner, wary of the model inaccuracies, can increase robustness and ensure stability by setting large diagonal terms for the initial covariance of model uncertainty, $\mathbf{\Sigma}_{0,j} = \gamma\mathbf{I}$, $\gamma \gg 1$, $j = 1, \ldots, N$. Moreover, setting large covariances initially helps to observe the inaccuracies of the model and the noise statistics. The covariance will be suitably decreased over the iterations, as adaptation (18) updates the linear models. Observing the noise statistics over the iterations can further help in the design of a good zero-phase filter to reject noise. Without accurate smoothing, adaptive ILC approaches run the risk of picking up noise in the adapted model matrices, which are then used in the control input update [in our case, in (29)]. This can hinder the control performance, hence we advise caution in the design of a smoothening filter.

The proposed update law takes advantage of the learning efficiency and computational advantages of model-based recursive ILC while being cautious with respect to model mismatch. The computational complexity of the recursive update is $\mathcal{O}(Nn^3)$ as opposed to batch norm-optimal ILC, where the batch pseudoinverse operation typically incurs $\mathcal{O}(N^3n^3)$ complexity. The batch model-based implementation using the *lifted-vector form* [3] inverts the input-to-output matrix $\mathbf{F}$

$$\begin{aligned} \mathbf{U}_{k+1} &= \mathbf{U}_k - \mathbf{F}^{\dagger}\mathbf{E}_k \\ \mathbf{E}_k &= \left(\mathbf{e}_{k,1}^{\mathrm{T}}, \mathbf{e}_{k,2}^{\mathrm{T}}, \ldots, \mathbf{e}_{k,N}^{\mathrm{T}}\right)^{\mathrm{T}} \end{aligned} \tag{32}$$

where the submatrices of the input-to-output matrix $\mathbf{F}$ are

$$\mathbf{F}_{(i,j)} = \begin{cases} \mathbf{A}_{i-1} \cdots \mathbf{A}_j \mathbf{B}_{j-1}, & j < i \\ \mathbf{B}_{j-1}, & j = i \\ \mathbf{0}, & j > i \end{cases}. \tag{33}$$

The condition of the lifted model matrix (33) grows exponentially with N and inverting it quickly becomes numerically unstable.

1) Implementation for Tracking Table Tennis Trajectories: The online learning framework for robot table tennis is described in Algorithm 2. Whenever a ball is initialized from a fixed ballgun with constant settings, located at $\mathbf{b}_0$, the trajectory generation framework will compute a particular striking trajectory (lines 2−3) to intercept and hit the ball in real time. See Appendix B for an overview of the trajectory generation pipeline. ILC can then be initialized (line 4) by linearizing the dynamics model $\mathbf{f}_{\text{nom}}$ around the computed trajectory points $\mathbf{r}_j, j = 1, \ldots, N$. ILC needs to be initialized only once, as long as the computed trajectory is capable of returning the ball to the opponent's court. The approximately 8 cm radius of the racket can cover for the inconstancy of the ballgun up to a certain degree.

Whenever the striking trajectory is executed (line 6), a returning trajectory will bring the arm back (line 7) from the current state to the fixed initial posture, $\mathbf{q}_0$. The returning trajectory can be as simple as a linear trajectory in the joint space. The *consistency* provided by the fixed ballgun in our setup, shown in Fig. 1, allows us to use ILC to track invariant trajectories over the iterations.

For a good performance in table tennis, the striking parts of these hitting movements need to be tracked accurately. The strikes are initially tracked with computed-torque inverse dynamics feedforward control commands and the additional LQR

Algorithm 2: ILC Improving Execution of Robot Table Tennis Hitting Movements Online.

Require: $\mathbf{q}_0$, $\mathbf{f}_{\text{ball}}$, *bayesILC*$(\dots)$ (see Algorithm1)
1: Move to initial posture $\mathbf{q}_0$, $\dot{\mathbf{q}}_0 = \mathbf{0}$.
2: Predict ball trajectory $\mathbf{b}_j$ using $\mathbf{f}_{\text{ball}}$
3: Compute trajectory $\mathbf{r}_j$ given $\mathbf{q}_0$ and $\mathbf{b}_j$, $j = 1, \dots, N$
4: Setup *bayesILC* (lines 2−4)
5: **repeat** \\ fixed ballgun throws balls at a constant rate
6: Execute strike with $\mathbf{u}_{\text{ILC}}$ and LQR feedback $\mathbf{K}$
7: Return to $\mathbf{q}_0$ with high-gain PD control and linear traj.
8: Update $\mathbf{u}_{\text{ILC}}$ and $\mathbf{K}$ with *bayesILC* (lines 9−14)
9: **until** ballgun is moved

feedback. The feedback law is computed for this purpose by linearizing the nominal dynamics model around the striking part of the reference trajectory. After a strike is completed, feedback will switch to proportional and derivative (PD) gains for the returning trajectory and the arm will come back close to $\mathbf{q}_0$. Learning with ILC can then take place (line 8) while waiting for another incoming table tennis ball.

The striking trajectory in table tennis is only an intermediary and does not need to be precisely tracked for a successful performance. In general, for hitting and catching tasks, the task performance depends critically on reaching the desired joint positions and velocities at the final time. A good performance along the trajectories is a means to this desired end: if feedback keeps the system stable around the trajectories, and the (linearized) models are reasonably valid around the trajectories, convergence to desirable performance levels can be rapid.

2) Coping With Varying Initial Conditions: Execution errors in tracking the reference trajectory (including the returning segment) prevents the robot from initializing in each iteration at the same state. Putting very high feedback gains on the returning trajectory or waiting long enough may suffice to initialize the system close to desired initial conditions, but in some occasions, none of these options may be desirable or available. For example, a robot *practicing* table tennis with a fixed ballgun running at a fixed rate, may not have time to initialize its desired posture accurately.

Starting from varying initial conditions $\mathbf{x}_{k,0} = [\mathbf{q}_{k,0}^\intercal, \dot{\mathbf{q}}_{k,0}^\intercal]^{\text{T}}$ one can consider updating the hitting movement $\mathbf{r}_j$ to take the robot to the same hitting state. For such online updating of trajectories, the invariant trajectory parameters $\mathbf{p}$ can be used to generate the trajectory from the current joint values. The reference control inputs $\mathbf{u}_{\text{IDM}}$ can then be recomputed based on the nominal inverse dynamics model. With this correction the total feedforward control commands $\mathbf{u}_{\text{ILC}}$ at iteration $k+1$ are recomputed as

$$\mathbf{u}_{\text{ILC},j} = \mathbf{u}_{k+1,j} + \mathbf{u}_{\text{IDM},j}(\tilde{\mathbf{r}}_j) - \mathbf{u}_{\text{IDM},j}(\mathbf{r}_j) \qquad (34)$$

where $\tilde{\mathbf{r}}_j$ is the updated trajectory starting from the perturbed initial state $\mathbf{x}_0 + \delta\mathbf{x}_{k,0}$. Using this simple adjustment (34), the stability of the learning performance can be greatly improved.

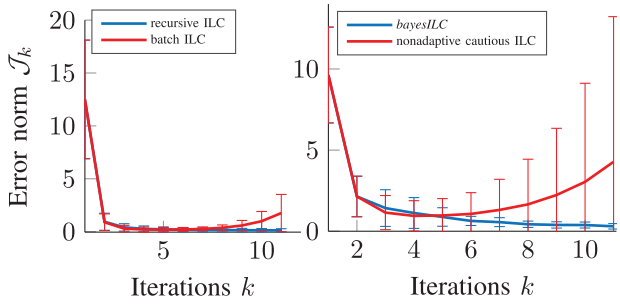

Fig. 4. ILC in recursive form is evaluated on random LTV systems. The Frobenius norm of the trajectory deviations, $\mathcal{J}_k$, is plotted over the iterations k. Results are averaged over ten experiments, where for each experiment, trajectories, nominal models, and actual models are drawn randomly from GPs. The performance of the batch pseudoinverse ILC (32) is shown in the red line. Numerical stability issues prevent it from stabilizing at steady state error, whereas recursive ILC (blue line) converges stably. If the model mismatch is increased, at some point, recursive ILC also diverges. Applying caution without adaptation is not enough to converge to steady state error. Cautious *and* adaptive *bayesILC*, on the other hand, applying the updates (14) and (29) iteratively, is very effective and shows a stably convergent behaviour.

VI. Evaluations and Experiments

In this section, we demonstrate the effectiveness of the ILC algorithm *bayesILC* presented in Algorithm 1 and described in detail in Section V in the context of tracking table tennis trajectories. We validate the proposed learning control law first in extensive simulations with linear and nonlinear models. In Section VI-B, we show real robot experiments with two seven degree of freedom (DoF) Barrett WAM arms for tracking table tennis striking movements.

A. Verification on Toy Problems

Stability is an important issue in the implementation of different learning controllers in real robot tasks. As a result, we setup extensive simulation experiments to validate the stability and robustness of our learning approach. We also discuss in detail the advantages of the recursive formulation over the batch pseudoinverse ILC (32).

1) Random Linear Models: We generate here random linear models and random trajectories drawn from Gaussian Processes (GPs) with squared exponential kernels. More specifically, the elements of the LTV model matrices $\mathbf{A}_j, \mathbf{B}_j$ are drawn from $(n+m)n$ uncorrelated GPs. The hyperparameters (scale, noise, and smoothness parameters) of these GPs are drawn independently from normal distributions with fixed means and variances. Moreover, random perturbations of these models (drawn the same way from $(n+m)n$ uncorrelated GP's) are generated to construct nominal models. Using the proposed random disturbance generation scheme, we can average the results and construct error bars for different ILC algorithms.

The performance of the recursive implementation [i.e., (29) with zero covariances and no adaptation] is shown in Fig. 4 on the LHS, where the results are averaged over ten different trajectories and models. The dimensions of the models are $n = 2, m = 2$, and the horizon size is set to $N = 120$. For the LQR and ILC calculations, $\mathbf{R} = 10^{-6}\mathbf{I}$ and the weighting matrix $\mathbf{Q}$ was set to the identity. In this case, the batch

model-based implementation using the pseudoinverse (32) is not stable at all without feedback. Applying LQR feedback and adding current iteration ILC in Fig. 4 improves the performance (red line in Fig. 4), but numerical issues (i.e., large condition number) in inverting the large model matrix $\mathbf{F}$ in lifted form (33) prevents it from stabilizing at steady state error. Tracking performance throughout the experiments is measured with respect to the Frobenius norm of the deviations $\mathbf{e}_{k,j}$, denoted as $\mathcal{J}_k$.

For the simulation results in Fig. 4, the spectral norm of the difference between the nominal and the actual models are each set to $\alpha\sigma_{\min}(\mathbf{F})$ where $\alpha = 100$. Increasing α further increases the probability that the model-based ILC is not monotonically convergent for some trial. For example, one can observe *asymptotically* but not *monotonically* convergent ILC behaviour when setting $\alpha = 990$ for a particular model and trajectory shown in Fig. 2. Increasing α further can prevent even asymptotic stability.

Especially in these cases of high model mismatch, the proposed adaptive and cautious *bayesILC* offers a stable and convergent ILC behaviour. In Fig. 4 on the RHS, we consider the case where $\alpha = 1000$. Recursive ILC that is also cautious does not show a stable convergent behaviour, whereas recursive ILC that is not cautious (i.e., covariance of the LTV matrices are zero) is not stable at all. Cautious *and* adaptive *bayesILC*, on the other hand, using LBR ($\lambda = 1.0$) to update the discrete-time LTV matrices $\mathbf{A}_{k,j}, \mathbf{B}_{k,j}$, shows a monotonic learning performance. The results are again averaged over ten different models and ten trajectories. For LBR, the initial covariances in (14) are set to $\mathbf{\Sigma}_{0,j} = \gamma\mathbf{I}$ for all $j = 1, \ldots, N$, where $\gamma = 10^4$ and the noise covariance is $\sigma^2 = 1$. Changing the exponent of the initial covariance, or reducing the forgetting factor λ in this case, can lead to a reduced or unstable learning performance.

2) GP Dynamics: The performance of the proposed algorithm *bayesILC* is evaluated next over random nonlinear models. In these set of experiments, we sample the states from n uncorrelated GPs with squared exponential kernels and random linear mean functions. The hyperparameters of these GPs are randomized as before. By sampling from such random nonlinear models, we can test the proposed algorithm under nonlinear uncertainties and noisy outputs. The *actual* model is simulated as follows:

1) random reference control inputs $\mathbf{v}_j \in \mathbb{R}^m, j = 1, \ldots, N$ are drawn K times from m *control* GPs;
2) n *oracle* GPs are used to sample $\mathbf{f}(\mathbf{x}_j, \mathbf{v}_j)$ and the generated dynamics is integrated (starting from zero initial conditions) using forward Euler, $dt = 1/N$, to form K trajectories. The GPs are conditioned during this process on the generated states $\mathbf{x}_j$ and inputs $\mathbf{v}_j$.

These n oracle GPs constitute the *actual* but unknown nonlinear dynamics model. Nominal models can be easily generated by using the predictions of the oracle GPs at a subset of the state space. The construction of a *nominal* model is described in detail as follows.

1) Another set of control inputs $\mathbf{u}_j,\ j = 1, \ldots, N$ are drawn from the *control* GPs, as before.
2) The mean predictions $\mathbf{f}(\mathbf{x}_j, \mathbf{u}_j)$ of the oracle GPs at $\mathbf{u}_j$ are used to evolve these control inputs (as in step 2 of the actual model).

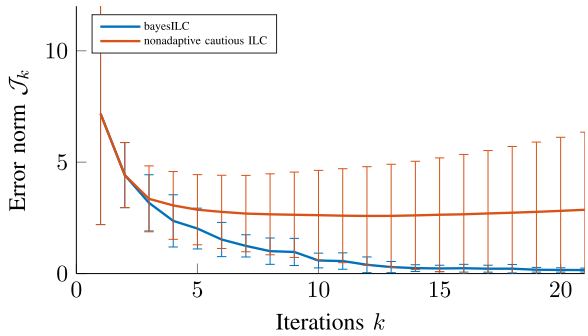

Fig. 5. Proposed ILC algorithm is evaluated on random nonlinear systems. The Frobenius norm of the trajectory deviations, $\mathcal{J}_k$, is plotted over the iterations k. Results are averaged over ten experiments, where for each experiment, trajectories and dynamics along these trajectories are drawn from GPs. Recursive ILC that is not cautious shows an unstable behaviour, and adding adaptation without caution is also not stable (both not shown in the figure). Purely cautious ILC (red line) is divergent for some of the trajectories. Cautious *and* adaptive *bayesILC*, on the other hand (blue line), shows a stable convergent learning performance.

3) The n separate *model* GPs (with same hyperparameters as the oracle) are conditioned on the resulting trajectory, i.e., the input pairs $(\mathbf{x}_j, \mathbf{u}_j)$ and the outputs $\mathbf{f}_j = (\mathbf{x}_{j+1} - \mathbf{x}_j)/dt$ for each time step $j = 1, \ldots, N$.
4) The mean derivative of the model GPs are calculated analytically (using the kernel derivatives). Discretized time-varying matrices $\mathbf{A}_j, \mathbf{B}_j$ and their variances $\mathbf{\Sigma}_{0,j}$ are constructed for each $j = 1, \ldots, N$, based on the mean and variance of the GP derivatives.

By sampling $K = 20$ trajectories for the conditioning of oracle GPs, we can cover a significant part of the state space in $n = 2$ dimensions. For each ILC iteration thereafter, the mean predictions are used as in step 2 to evolve the trajectory, but without further conditioning of the model GPs. Instead, adaptation is performed as before with LBR, replacing the steps 3 and 4. We can thus avoid the expensive online GP training.

Fig. 5 shows the learning performance for a horizon size of $N = 20$. The dimensions of the system is same as before, $n = 2, m = 2$ and $\mathbf{R} = 10^{-6}\mathbf{I},\ \mathbf{Q} = \mathbf{I}$. The results are averaged again over ten experiments. In this nonlinear setting, the recursive ILC that is not cautious shows an unstable behaviour (not shown in Fig. 5). Adaptive but not cautious ILC is also unstable (also not shown). Cautious but not adaptive ILC is not stable for some trajectories and can diverge (red line). Cautious *and* adaptive *bayesILC*, on the other hand (blue line), using LBR to update the discrete-time LTV matrices, shows again a stable convergent learning performance, improving over the purely cautious ILC. For LBR, the initial covariances in (14) are again set to $\gamma = 10^4$ times the identity and the noise covariance is $\sigma^2 = 1$. The best performance is reached when the forgetting factor is set to $\lambda = 0.9$. As before, changing the exponent of the initial covariance, or the forgetting factor, can lead to a reduced or unstable learning performance.

3) Barrett WAM Model: We next test ILC on striking movements (60) for a seven DoF Barrett WAM simulation model. In the simulations, the robot is started from a fixed initial state $\mathbf{q}_0$. The initial posture is chosen from one of the center, RHS or LHS

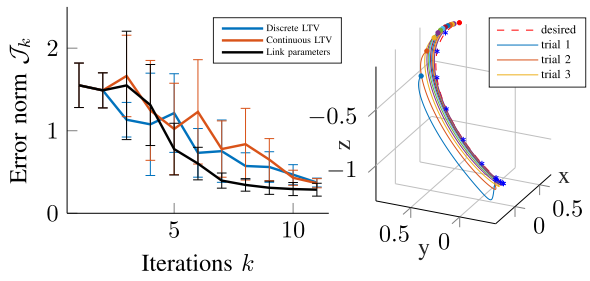

Fig. 6. Performance of the adaptive and cautious ILC algorithm *bayesILC* on the simulated Barrett WAM model is shown on the LHS. The Frobenius norm of the trajectory deviations, $\mathcal{J}_k$, is plotted over the iterations k. The results are averaged over ten different strikes and three different initial postures. Three different adaptation laws are considered, adaptation of discrete-time and continuous-time LTV models are shown in blue and red, respectively, while the adaptation of link parameters is shown in black. Forgetting factor was set to $\lambda = 0.8$ for all of the adaptation laws. One of the desired trajectories, shown in dashed red on the RHS, is tracked very closely in the final iteration. The blue markers correspond to the time profile of the motion, which are drawn uniformly spaced, one for each 80 ms. The final hitting positions reached are shown as filled circles.

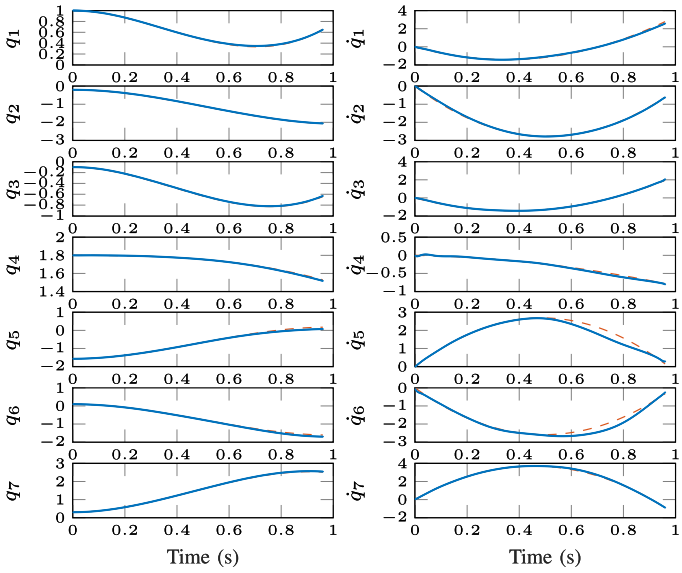

Fig. 7. Joint trajectories for a hitting movement on the Barrett WAM model. The reference trajectories, shown in dashed red, are tracked very closely with ILC in the final iteration, shown in blue.

resting postures of the robot. The striking parameters (61) are then optimized, based on an incoming table tennis ball with a randomly chosen incoming position and velocity. The link parameters of the Barrett WAM forward dynamics model used to simulate actual trajectories are perturbed randomly to construct nominal models for ILC. The linearization procedure described in Section II produces LTV nominal models that can be used by ILC to reduce the deviations from the desired (fixed) striking movement over the iterations.

The randomization during the optimization guarantees that a variety of hitting movements are tracked throughout the experiments. The performance of the proposed ILC approach *bayesILC* with three different adaptation laws is then evaluated over the striking segment of the optimized (striking and returning) trajectories. The convergence results are averaged over ten such striking movements, as shown in Fig. 6. The adaptation of discrete-time and continuous-time LTV models are shown in blue and red, respectively, while the adaptation of link parameters is shown in black. Forgetting factor was set to $\lambda = 0.8$ for all of the adaptation laws. Initial covariances are set to $\mathbf{\Sigma}_{0,j} = 10^4\mathbf{I}$ for continuous and discrete-time LTV model adaptation laws, while for link parameters, the initial covariances are $\mathbf{\Sigma}_{0,j} = 10^{10}\mathbf{I}$. The weights of the cautious ILC update (29) is set to $\mathbf{R} = 10^{-2}\mathbf{I}$, $\mathbf{Q} = \mathbf{I}$.

After updating the link parameter means and variances, we use an auto-differentiation tool (ADOL-C library in C ++) together with sampling to approximate the distribution of forward dynamics (3) derivatives $\mathbf{A}_{k,j}, \mathbf{B}_{k,j}$. More specifically, the forward dynamics is differentiated (with respect to joint positions, velocities and control inputs) at 100 link parameter samples drawn from the posterior distribution [i.e., normal distribution with means and variances given by (14)] online. This sampling procedure generates a reasonable approximation of posterior derivative means and variances.

In table tennis, if the robot arm follows the assigned reference trajectory precisely it will hit the ball with a desired velocity at the desired time. We can see on the RHS of Fig. 6 that an initial attempt (blue curve) falls short of the reference trajectories (dashed curve). The percentage of the balls that are returned to the opponent's court are close to zero. ILC then modifies the control inputs to compensate for the modeling errors. In the last attempt, the reference trajectory is executed almost perfectly. The accuracy of the table tennis task increases to 95% on average. Fig. 7 shows the adjusted control inputs for one striking movement.

The recursive ILC (without adaptation or caution) is convergent for some of the hitting movements in Fig. 6. However, similar to the previous simulation examples, the recursive form of the ILC update, depending on the accuracy of the model along the trajectories, can fail to converge for some trajectories (not shown in the figure). The proposed recursive, adaptive, and cautious algorithm *bayesILC*, with the three adaptation laws shown in Fig. 6, shows a better and faster convergence for a variety of trajectories.

The ILC experiments shown in Figs. 6 and 7 reset the initial posture always to the same desired posture $\mathbf{q}_0$. Next, we consider nonrepetitive disturbances around the desired initial posture. This would mean, physically, that the robot is not initialized accurately around the resting posture.

Comparisons to the baseline (black line) in Fig. 8 illustrate the additional robustness whenever the trajectory adaptation (34) is employed. We adapt the metric for this comparison according to the task: the costs indicated are the *final* costs (for hitting the incoming ball at the desired joint positions with desired joint velocities), not the full costs incurred along the reference trajectory. Note especially the faster convergence and increased accuracy of the proposed method with the reference trajectory and input adaptation (blue line). More robust performance is obtained by adapting the trajectories $\mathbf{r}_j$ and $\mathbf{u}_{\mathrm{IDM},j}$, which, in addition to performing better, shows much lower variance compared to the baseline.

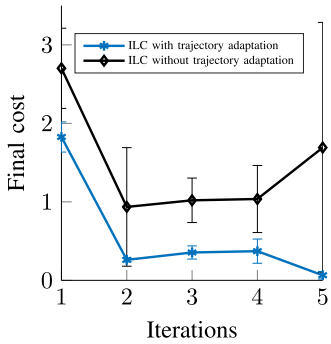

Fig. 8. Simulation results illustrating the additional robustness to varying initial conditions whenever the trajectories (states and control references) are adapted according to (34) (blue line). Note the unstable performance of ILC without such adaptation (black line), which keeps the references $\mathbf{r}_j$ and the inverse dynamics inputs $\mathbf{u}_{\mathrm{IDM},j}$ fixed.

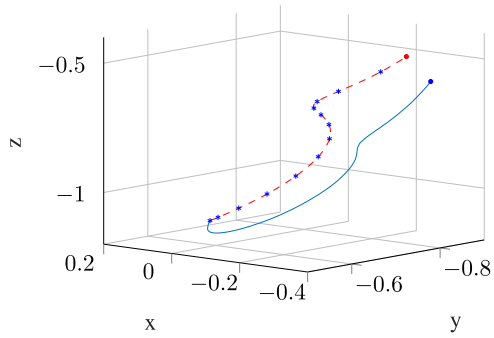

Fig. 9. Example of a striking movement for real robot table tennis is shown in red. The blue markers correspond to the time profile of the motion, which are drawn uniformly, one for each 80 ms. Executing this movement well with the Barrett WAM will lead to a good hit. Control errors in tracking lead to a poor hitting performance, shown in blue. The filled circles are the final reached hitting positions. High-gain PD feedback was used to track the reference in this real robot example. The tracking errors can be decreased efficiently and stably by applying the proposed recursive, cautious, and adaptive ILC algorithm *bayesILC*.

In practice, trusting the model too much at the beginning of the trajectory leads to the amplification of initial errors. Nonrepetitive starting postures violate the initial condition assumption typical of standard ILC updates. In this case, the feedback matrices $\mathbf{K}_{k,j}$, as opposed to the feedforward input updates $\boldsymbol{\delta}\mathbf{u}_{k,j}$, play a bigger role in the learning stability at the beginning of the trajectories, $j \ll N$.

B. Real Robot Table Tennis

Finally, we perform experiments on our robotic table tennis platform, see Fig. 10, where two seven DoF cable-driven, torque-controlled Barrett WAM arms (*Ping* and *Pong*) are hanging from the ceiling. The custom made Barrett WAM arms are capable of high speeds and accelerations (approx. up to 10 m/s^2 in task space). Standard size rackets (16 cm diameter) are mounted on the end-effector of the arms as can be seen in Fig. 10. A vision system consisting of four cameras hanging from the ceiling around each corner of the table is used for tracking the ball [30]. A ball launcher (see Fig. 1) is available to throw balls accurately to a fixed position inside the workspace of the robots. The incoming ball arrives with low-variability in desired positions and higher-variability in ball velocities. The whole area to be covered amounts to about 1 m^2 circular region surrounding a centered posture of the robots.

The realistic simulation environment SL [31] acts as both a simulator and as a real-time interface to the Barrett WAMs in our experiments. The initial positioning is given by a PD controller with high gains on the shoulder joints, which is then toggled off during the experiments with the striking movements, as summarized in Algorithm 2. The high-gain PD controller used to initialize the robots was also tested for tracking the striking movements, see Fig. 9. When ILC is applied on top of the PD controller, the learning quickly stagnates, leading to oscillations in some of the joints. Instead, a low-gain LQR feedback law is computed for the striking part of the movement with a linearized nominal dynamics model (7). The weighting matrices for this purpose are set to identity, $\mathbf{Q} = \mathbf{I}$, and the constant penalty matrix is chosen as $\mathbf{R} = 0.05\mathbf{I}$. Decreasing the scaling of the penalty matrix to 0.03 causes oscillations in the elbow joint, indicating that the nominal model is not very accurate. At the cost of larger initial error, we suggest increasing the input penalties $\mathbf{R}$ to improve the stability of ILC in other high DoF robotics applications.

After the visual system outputs a ball estimate, a ball model can be used along with an extended Kalman filter to predict a ball trajectory. The ball model accounts for some of the bouncing behavior of the ball and air drag effects. If the predicted ball trajectory coincides with the workspace of the robot, the motion planning system has to then come up with a trajectory that specifies how, where, and when to intercept the incoming ball. Desired Cartesian position, velocity, and orientations of the racket at the hitting time T impose constraints on the seven joint angles and seven joint velocities of the robot arm at T. Along with the desired hitting time T (or the time until impact), these 15 parameters are used to generate third-order joint space polynomials. These movements can be optimized online in 20−30 ms [2], or loaded from a lookup table. In the ILC experiments, the parameters in the lookup table are used without interpolation, to make sure that the same trajectory can be used for balls deviating slightly from their stored values. We make sure that the lookup table is dense enough and that the ballgun is fixed.

Some examples of the generated trajectories are shown in Fig. 10. After a strike, a linear joint trajectory is computed that will take the robots from the current state to the resting posture in $T_{\mathrm{rest}} = 1.0$ s. PD feedback control is turned ON again for this returning part of the trajectory. When the returning trajectory is executed, SL main thread running the inverse dynamics computations will continue to keep the arms stable around the resting posture, while another thread is detached to run the ILC update.[4] The ILC loop terminates successfully whenever the computed feedforward updates are within the respective torque limits. After a successful termination, if the actual posture is within

[4]Code is available in the public repository. [Online]. Available: https://gitlab.tuebingen.mpg.de/okoc/learning-control along with the test scripts used to generate the plots in the previous subsections.

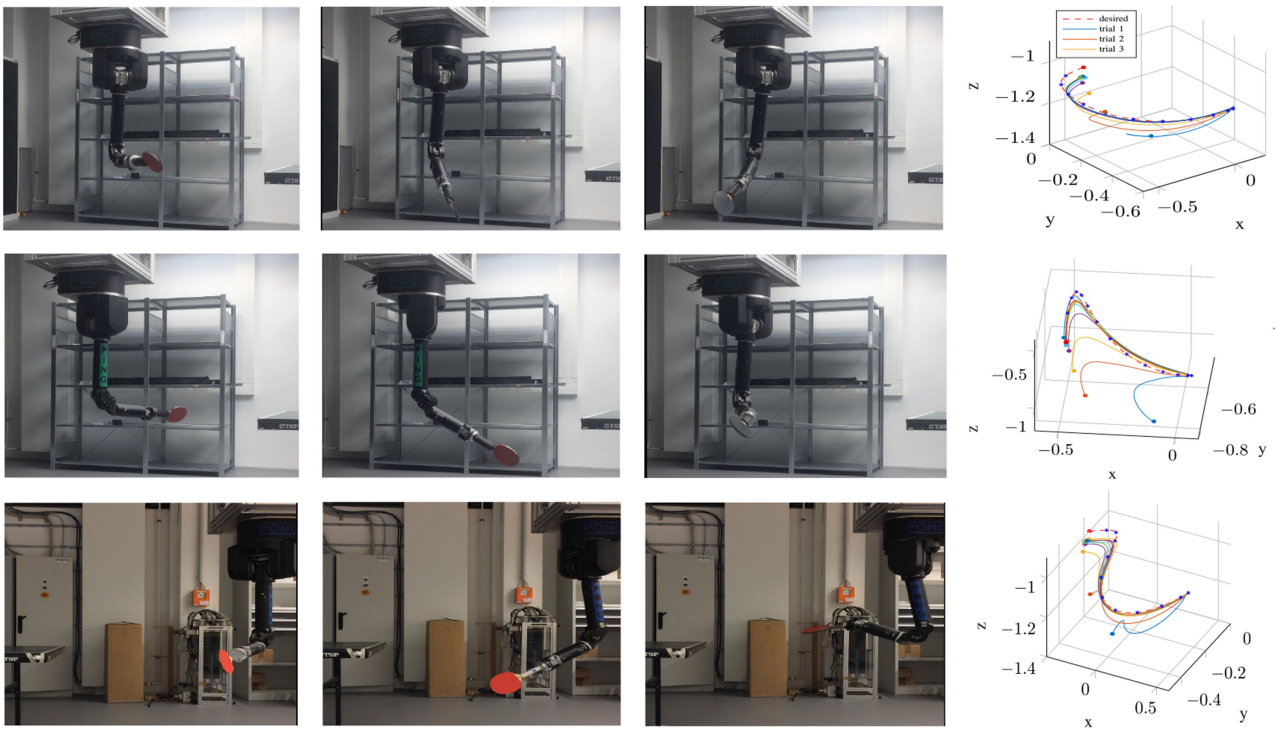

Fig. 10. Two Barrett WAMs (a.k.a. *Ping* and *Pong*) are initialized in our experiments in three different starting postures. We make controlled experiments with a simulated ballgun and generate many different hitting movements; three of them are shown in the above images. The proposed algorithm *bayesILC* leads to an efficient and stable learning approach for tracking these hitting movements. The RHS starting posture for the robot Ping can be seen on the upper left image. Initially, before learning with ILC starts, Ping performs poorly, and the hitting posture of the robot is shown in the upper central image. After five iterations, the hitting posture is corrected significantly as shown in the upper right image. Similarly, the central images show the operation of the ILC for another trajectory, where the starting posture for Ping is fixed on the LHS of the robot. On the bottom images, an ILC performance is shown for the robot Pong. The three plots on the RHS show the Cartesian trajectories corresponding to the ILC iterations. The reference trajectories are shown in dashed red, and the final hitting positions reached are shown as filled circles.

0.1 rad distance of the resting posture, the LQR feedback will be turned ON again and the robots will start moving to track the same striking motion.

We use a simulated ball to make more controlled experiments, focusing on the control aspect in more detail. If the striking robot movements are executed accurately, then the ball in simulation will be returned close to a desired position on the opponent's court. At different points in time we have identified three different sets of link parameters for rigid body dynamics. We can use these parameterizations of rigid body dynamics as potential nominal models to kick-start the learning process. We tested these nominal models first in slowed down hitting movements, where a slowdown rate of two means that the number of trajectory points double while the hitting time is held fixed. Cutting down the trajectories to an initial subset of the movement to restrict potential instabilities, or initial masking of some of the joints during ILC updates, are other techniques that we have employed to evaluate these nominal models in a careful manner. Of the three models, only one of them was suitable for the *local* learning that ILC provides. This model is further adapted with the proposed *bayesILC* algorithm in order to improve the tracking of the striking movements. Adaptation of the trajectories $\mathbf{r}_j$ and the nominal inputs $\mathbf{u}_{\mathrm{IDM},j}$ was additionally performed on top of ILC, to stabilize the learning process, since an accurate initialization of the joints (especially on the wrist and the elbow) was not possible with the Barrett WAMs.

We have compared *bayesILC* to two other ILC methods: batch ILC (32) and ILC with PD feedback (with constant p, d values). PD type ILC with constant p or d values is often too simplistic, and did not yield any improvement in our setup, even after tuning the p, d values. Batch ILC was tested with ten times downsampled trajectories, with adjustable learning rates. We have found batch ILC to be inferior to the recursive ILC when tested over multiple trajectories (slowed down and cut versions included).[5] Recursive ILC without any adaptation is monotonically convergent on average for about five iterations, bringing the root mean squared tracking error from about 0.80 to 0.40 on average. Repeating the trajectories for five more iterations, we note that the tracking error starts increasing slightly due to introduced oscillations in some of the joints. Introducing adaptation with recursive and cautious ILC (i.e., the proposed approach *bayesILC*) we can

[5]For batch pseduoinverse-based ILC, inversion of the model matrices (7) around the *unstable* hitting trajectory causes instability, which is alleviated by providing an additional *current iteration ILC* (CI) [3]. CI adds the current iteration k's feedback errors to the feedforward compensations for the next iteration $k+1$. As in our preliminary experiments with the Barrett WAM [32], we have applied CI in addition to stabilize a downsampled version of batch model-based ILC.

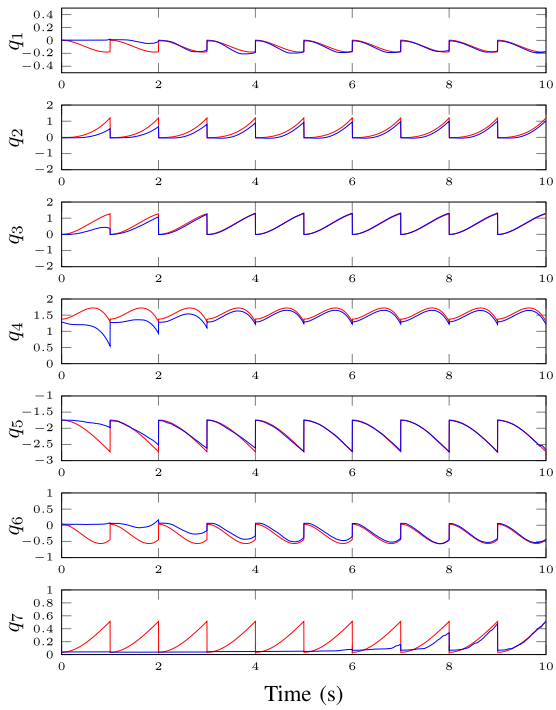

Fig. 11. Robot experiment results for cautious and adaptive *bayesILC*, shown for a particular reference trajectory. The ten iteration results are concatenated for convenience. The desired joint trajectories correspond to a hitting movement on the Barrett WAM. The reference trajectories, shown in red, are tracked very closely with ILC in the final iteration, shown in blue. Final cost goes down to 0.20 in the last iteration.

decrease the tracking error further, to about 0.20 monotonically in five more iterations. This enables a return accuracy of 40% of the simulated balls to the opponent's court. See Fig. 11 for the performance of *bayesILC* on a particular reference trajectory, evaluated over ten iterations.

The proposed update law *bayesILC* evaluated earlier adapts the discrete-time LTV models with a forgetting factor of $\lambda = 0.8$. This value was chosen experimentally, and could be optimized, e.g., using a dataset of previous ILC performances. The same parameter values are chosen for the initial covariances as in the simulation experiments with the Barrett WAM. Adapting the continuous LTV models, when the trajectories are smoothened suitably with a zero-phase filter, leads to faster updates with similar improvements in tracking performance. Using the online adaptation of the link parameters on the other hand, leads to poorer convergence in tracking for some of the joints (most notably, the elbow). This fact leads us to suspect that the rigid body dynamics model underfits, i.e., the mismatch for our Barrett WAMs is not purely parametric in nature. We see that the final cost (as 2-norm of deviations from desired joint hitting positions and velocities) drops down from 1.70 to 0.20 for *bayesILC* when the LTV model matrices are adapted directly. After performing ten more iterations, the percentage of balls successfully returned to the opponent's court increases from 40% to about 60% on average.[6]

[6]A video showing some example ILC performances for the two robots is [Online]. Available: https://youtu.be/27vHoLBwLoM.

VII. Conclusion

In this paper, we presented a novel ILC algorithm that is recursive, cautious, and adaptive at the same time. The closed-form update law (24) that was presented derives from the adaptive dual control literature and is sometimes referred to as *passive learning* [29]. The algorithm was then recast in a more efficient form (derived in Appendix A), which does not require the estimation of disturbances and can be implemented as a recursive ILC update. The update law makes it easy to introduce caution with respect to modeling uncertainties and online adaptation of the linearized model matrices. Unlike typical ILC updates, feedback matrices for the tracking of striking trajectories are adapted as well, which are useful for rejecting noise and varying initial conditions. We believe that the introduced ILC update yields a principled approach to adapt the models, as well as their regularizer, based on data.

The proposed algorithm *bayesILC* was evaluated in different simulations of increasing complexity. Finally in Section VI-B, we have presented real robot experiments on our robotic table tennis setup with two Barrett WAMs, see Fig. 10. It was shown that the proposed approach leads to an efficient way to learn to track hitting movements online. Hitting movements throughout the experiments are generated in the joint space of the robots and enable them to execute optimal striking motions. Control inputs, as well as a time-varying feedback law, are updated after each trial by using the model-based update rule that considers the deviations from the striking trajectory. After the trajectories are executed, the deviations can be used to adapt the model parameter means and variances using LBR. A forgetting factor was considered in addition to make adaptation more flexible. An adaptation of the reference trajectories as well as the nominal inputs was considered on top of *bayesILC* to render the method more effective and stable for initial posture stabilization errors.

Although we showed a stable and efficient way to learn to track references with ILC, we have not analyzed its generalization to arbitrary trajectories. In our table tennis setup, we are making progress to having the two robots play against each other. Generalization capacity would play an important role in extending the average game duration between the robots, as the trajectories during the table tennis matches would be generated online [2] according to the state of the game. We believe that in the case where the trajectories are changing, generalizing the learned control commands can be achieved by compressing them to a lower-dimensional input space (i.e., parameters). Learned feedforward commands could be projected to a parameterized feedback matrix, the parameters of which could represent the invariants between the trajectories. An efficient and stable implementation of such parameterizations will be the focus of our future work.

Appendix A
Cautious ILC Derivations

We provide in this section self-contained derivations of the cautious ILC update rule, given in (24) and simplified in (29).

Consider the following optimal control problem:

$$\min_{\boldsymbol{\delta u}} \sum_{j=1}^{N} \mathbb{E}_{\mathbf{A}_j,\mathbf{B}_j}\left[\mathbf{e}_{k+1,j}^{\mathrm{T}}\mathbf{Q}_j\mathbf{e}_{k+1,j}+\boldsymbol{\delta u}_{k+1,j}^{\mathrm{T}}\mathbf{R}_j\boldsymbol{\delta u}_{k+1,j}\right] \tag{35}$$

$$\text{s.t.}\quad \mathbf{e}_{k+1,j+1} = \mathbf{A}_j\mathbf{e}_{k+1,j} + \mathbf{B}_j\mathbf{u}_{k+1,j} + \mathbf{d}_{j+1} \tag{36}$$

where the LTV system matrices $\mathbf{A}_j, \mathbf{B}_j$ are random variables with known means and variances. Since $\mathbf{u}_{k+1,j} = \mathbf{u}_{k,j} + \boldsymbol{\delta u}_{k,j}$, we can rewrite (36) as

$$\mathbf{e}_{k+1,j+1} = \mathbf{A}_j\mathbf{e}_{k+1,j} + \mathbf{B}_j\boldsymbol{\delta u}_{k,j} + \bar{\mathbf{d}}_{j+1} \tag{37}$$

$$\bar{\mathbf{d}}_{j+1} = \mathbf{B}_j\mathbf{u}_{k,j} + \mathbf{d}_{j+1}. \tag{38}$$

The iteration index k will be removed in the last section due to space constraints. Notice that the value function for the optimal control problem (36) is a quadratic function of the errors along the trajectory

$$V(\mathbf{e}, j) = \mathbf{e}^{\mathrm{T}}\mathbf{P}_j\mathbf{e} + 2\mathbf{e}^{\mathrm{T}}\mathbf{b}_j + c_j \tag{39}$$

for time-varying matrices $\mathbf{P}_j \in \mathbb{R}^{2n\times 2n}$, vectors $\mathbf{b}_j \in \mathbb{R}^{2n}$ and $c_j \in \mathbb{R}$. We can then apply dynamic programming to compute the optimal solution recursively

$$V(\mathbf{e}_j, j) = \min_{\boldsymbol{\delta u}_j}\left(\mathbf{e}_j^{\mathrm{T}}\mathbf{Q}_j\mathbf{e}_j + \boldsymbol{\delta u}_j^{\mathrm{T}}\mathbf{R}_j\boldsymbol{\delta u}_j + V(\mathbf{e}_{j+1}, j+1)\right)$$

$$\begin{aligned} V(\mathbf{e}_{j+1}, j+1) = \mathbb{E}_{\mathbf{A}_j,\mathbf{B}_j}\big[&2\mathbf{b}_{j+1}^{\mathrm{T}}(\mathbf{A}_j\mathbf{e}_j + \mathbf{B}_j\boldsymbol{\delta u}_j + \bar{\mathbf{d}}_{j+1}) \\ &+ c_{j+1} + (\mathbf{A}_j\mathbf{e}_j + \mathbf{B}_j\boldsymbol{\delta u}_j + \bar{\mathbf{d}}_{j+1})^{\mathrm{T}} \\ &\times \mathbf{P}_{j+1}(\mathbf{A}_j\mathbf{e}_j + \mathbf{B}_j\boldsymbol{\delta u}_j + \bar{\mathbf{d}}_{j+1})\big]. \end{aligned} \tag{40}$$

The recursion starts from $\mathbf{P}_N = \mathbf{Q}_N$. Taking derivative w.r.t. $\boldsymbol{\delta u}_j$ of the RHS, we get

$$\begin{aligned} \mathbf{R}_j\boldsymbol{\delta u}_j + \big(&\mathbb{E}_{\mathbf{A}_j,\mathbf{B}_j}\left[\mathbf{B}_j^{\mathrm{T}}\mathbf{P}_{j+1}\mathbf{A}_j\right]\mathbf{e}_j + \mathbb{E}_{\mathbf{B}_j}\left[\mathbf{B}_j^{\mathrm{T}}\mathbf{P}_{j+1}\mathbf{B}_j\right]\boldsymbol{\delta u}_j \\ &+ \mathbb{E}_{\mathbf{B}_j}\left[\mathbf{B}_j^{\mathrm{T}}\left(\mathbf{P}_{j+1}\bar{\mathbf{d}}_{j+1} + \mathbf{b}_{j+1}\right)\right]\big) = 0. \end{aligned} \tag{41}$$

Solving (41) for the optimal control input compensations, and arranging using the notation in (24) we get

$$\begin{aligned} \boldsymbol{\delta u}_j &= \mathbf{K}_j\mathbf{e}_j - \boldsymbol{\Phi}_j^{-1}\boldsymbol{\ell}_j \\ \mathbf{K}_j &= -\boldsymbol{\Phi}_j^{-1}\boldsymbol{\Psi}_j \\ \boldsymbol{\Phi}_j &= \mathbf{R}_j + \mathbb{E}_{\mathbf{B}_j}\left[\mathbf{B}_j^{\mathrm{T}}\mathbf{P}_{j+1}\mathbf{B}_j\right] \\ \boldsymbol{\Psi}_j &= \mathbb{E}_{\mathbf{A}_j,\mathbf{B}_j}\left[\mathbf{B}_j^{\mathrm{T}}\mathbf{P}_{j+1}\mathbf{A}_j\right] \\ \boldsymbol{\ell}_j &= \mathbb{E}_{\mathbf{B}_j}\left[\mathbf{B}_j^{\mathrm{T}}(\mathbf{P}_{j+1}\bar{\mathbf{d}}_{j+1} + \mathbf{b}_{j+1})\right]. \end{aligned} \tag{42}$$

In order to derive a Riccati-like equation, we plug (42) into (40), and using (39) get

$$\begin{aligned} \mathbf{e}^{\mathrm{T}}\mathbf{P}_j\mathbf{e} + 2\mathbf{e}^{\mathrm{T}}\mathbf{b}_j + c_j = \;&\mathbf{e}_j^{\mathrm{T}}\mathbf{Q}_j\mathbf{e}_j + \mathbf{e}_j^{\mathrm{T}}\left(\boldsymbol{\Psi}_j^{\mathrm{T}}\boldsymbol{\Phi}_j^{-1}\mathbf{R}_j\boldsymbol{\Phi}_j^{-1}\boldsymbol{\Psi}_j\right)\mathbf{e}_j \\ &+ 2\boldsymbol{\ell}_j^{\mathrm{T}}\boldsymbol{\Phi}_j^{-1}\mathbf{R}_j\boldsymbol{\Phi}_j^{-1}\boldsymbol{\Psi}_j\mathbf{e}_j + \boldsymbol{\ell}_j^{\mathrm{T}}\boldsymbol{\Phi}_j^{-1}\mathbf{R}_j\boldsymbol{\Phi}_j^{-1}\boldsymbol{\ell}_j \\ &+ \mathbb{E}_{\mathbf{A}_j,\mathbf{B}_j}\left[\left(\bar{\mathbf{A}}_j\mathbf{e}_j + \mathbf{m}_j\right)^{\mathrm{T}}\mathbf{P}_{j+1}\left(\bar{\mathbf{A}}_j\mathbf{e}_j + \mathbf{m}_j\right)\right] \\ &+ 2\mathbb{E}_{\mathbf{A}_j,\mathbf{B}_j}\left[\left(\bar{\mathbf{A}}_j\mathbf{e}_j + \mathbf{m}_j\right)^{\mathrm{T}}\mathbf{b}_{j+1}\right] + c_{j+1} \end{aligned} \tag{43}$$

where we have introduced the terms

$$\begin{aligned} \bar{\mathbf{A}}_j &= \mathbf{A}_j + \mathbf{B}_j\mathbf{K}_j \\ \mathbf{m}_j &= \bar{\mathbf{d}}_{j+1} - \mathbf{B}_j\boldsymbol{\Phi}_j^{-1}\boldsymbol{\ell}_j. \end{aligned} \tag{44}$$

Checking for the equality of the quadratic terms we get, after some cancelations

$$\begin{aligned} \mathbf{P}_j &= \mathbf{Q}_j + \mathbf{M}_j - \boldsymbol{\Psi}_j^{\mathrm{T}}\boldsymbol{\Phi}_j^{-1}\boldsymbol{\Psi}_j \\ \mathbf{M}_j &= \mathbb{E}_{\mathbf{A}_j}\left[\mathbf{A}_j^{\mathrm{T}}\mathbf{P}_{j+1}\mathbf{A}_j\right] \\ \mathbf{b}_j &= \boldsymbol{\Psi}_j^{\mathrm{T}}\boldsymbol{\Phi}_j^{-1}\mathbf{R}_j\boldsymbol{\Phi}_j^{-1}\boldsymbol{\ell}_j + \mathbb{E}_{\mathbf{A}_j,\mathbf{B}_j}\left[\bar{\mathbf{A}}_j^{\mathrm{T}}\left(\mathbf{P}_{j+1}\mathbf{m}_j + \mathbf{b}_{j+1}\right)\right]. \end{aligned} \tag{45}$$

1) Rewriting the Feedforward Recursion: The control input compensations calculated in (42) can be simplified significantly by noting that the last three terms in the feedforward recursion of (45)

$$\begin{aligned} \mathbf{b}_j = \;&\mathbb{E}\left[\bar{\mathbf{A}}_j^{\mathrm{T}}\left(\mathbf{b}_{j+1} + \mathbf{P}_{j+1}\bar{\mathbf{d}}_{j+1}\right)\right] - \mathbb{E}\left[\mathbf{A}_j^{\mathrm{T}}\mathbf{P}_{j+1}\mathbf{B}_j\right]\boldsymbol{\Phi}_j^{-1}\boldsymbol{\ell}_j \\ &- \mathbf{K}_j^{\mathrm{T}}\mathbb{E}\left[\mathbf{B}_j^{\mathrm{T}}\mathbf{P}_{j+1}\mathbf{B}_j\right]\boldsymbol{\Phi}_j^{-1}\boldsymbol{\ell}_j - \mathbf{K}_j^{\mathrm{T}}\mathbf{R}_j\boldsymbol{\Phi}_j^{-1}\boldsymbol{\ell}_j, \end{aligned} \tag{46}$$

cancel out, leaving

$$\mathbf{b}_j = \mathbb{E}_{\mathbf{A}_j,\mathbf{B}_j}\left[\bar{\mathbf{A}}_j^{\mathrm{T}}\left(\mathbf{b}_{j+1} + \mathbf{P}_{j+1}\bar{\mathbf{d}}_{j+1}\right)\right]. \tag{47}$$

The cancelations can be seen easily by rewriting the first term in terms of the feedback matrix and grouping the last two terms together

$$-\mathbf{K}_j^{\mathrm{T}}\boldsymbol{\Phi}_j\boldsymbol{\Phi}_j^{-1}\boldsymbol{\ell}_j + \mathbf{K}_j^{\mathrm{T}}\boldsymbol{\ell}_j = 0. \tag{48}$$

2) Simplifying the Feedforward Recursion: The feedforward recursion in (47) still requires the explicit estimation of disturbances. This equation can be simplified further by rewriting the disturbances in terms of the previous trial errors

$$\begin{aligned} \bar{\mathbf{d}}_{j+1} &= \mathbf{e}_{k,j+1} - \mathbf{A}_j\mathbf{e}_{k,j} \\ \boldsymbol{\ell}_j &= \mathbb{E}\left[\mathbf{B}_j^{\mathrm{T}}(\mathbf{P}_{j+1}\mathbf{e}_{k,j+1} + \mathbf{b}_{j+1})\right] - \boldsymbol{\Psi}_j\mathbf{e}_{k,j}. \end{aligned} \tag{49}$$

Introducing $\boldsymbol{\nu}_{j+1} = \mathbf{P}_{j+1}\mathbf{e}_{k,j+1} + \mathbf{b}_{j+1}$, we can rewrite the optimal control input compensations as

$$\boldsymbol{\delta u}_j = \mathbf{K}_j(\mathbf{e}_{k+1,j} - \mathbf{e}_{k,j}) - \boldsymbol{\Phi}_j^{-1}\mathbb{E}\left[\mathbf{B}_j^{\mathrm{T}}\boldsymbol{\nu}_{j+1}\right]. \tag{50}$$

Rewriting (47) in terms of $\boldsymbol{\nu}_j$, we get

$$\boldsymbol{\nu}_j = \mathbb{E}\left[\bar{\mathbf{A}}_j^{\mathrm{T}}\boldsymbol{\nu}_{j+1}\right] + \left(\mathbf{P}_j - \mathbb{E}\left[\left(\mathbf{A}_j + \mathbf{B}_j\mathbf{K}_j\right)^{\mathrm{T}}\mathbf{P}_{j+1}\mathbf{A}_j\right]\right)\mathbf{e}_{k,j} \tag{51}$$

since $\mathbf{P}_j = \mathbf{Q}_j + \mathbf{M}_j - \boldsymbol{\Psi}_j^{\mathrm{T}}\boldsymbol{\Phi}_j^{-1}\boldsymbol{\Psi}_j$, the last term becomes

$$\left(\mathbf{P}_j - \mathbf{M}_j - \mathbf{K}_j^{\mathrm{T}}\boldsymbol{\Psi}_j\right)\mathbf{e}_{k,j} = \mathbf{Q}_j\mathbf{e}_{k,j} \tag{52}$$

hence, the feedforward recursion defining (50) can be computed independently of disturbance estimates

$$\boldsymbol{\nu}_j = \mathbb{E}\left[\bar{\mathbf{A}}_j^{\mathrm{T}}\boldsymbol{\nu}_{j+1}\right] + \mathbf{Q}_j\mathbf{e}_{k,j},\ j = 1, \ldots, N-1 \tag{53}$$

starting from $\boldsymbol{\nu}_N = \mathbf{0}$.

APPENDIX B
MOVEMENT GENERATION FOR TABLE TENNIS

In a highly dynamic and complex task such as robot table tennis, one often needs to consider an extension of the standard trajectory tracking task. Based on the varying initial positions and velocities of the robot arm and the trajectory of the incoming ball, in each table tennis stroke the robot arm needs to track different trajectories that start from different initial conditions and end with different desired goal states of the arm. Moreover these trajectories need to be optimized in time to intercept the ball. The striking trajectories $\mathbf{r}(t) = [\mathbf{q}_{\text{des}}(t), \dot{\mathbf{q}}_{\text{des}}(t)]^{\mathrm{T}}$ are generated online and tracked using the proposed ILC approach.

Striking movement primitives suited to table tennis have been proposed in [1] and [33] as an extension of discrete dynamic movement primitives (DMP). Unlike the original formulation [34], these extensions allow for an arbitrary velocity profile to be attached to the primitives around hitting time. However, these approaches are heavily structured for the problem at hand, introducing and tuning additional domain parameters. In [32], we instead proposed to use rhythmic movement primitives that allow for a limit cycle attractor, which is desirable if we want to maintain the striking motion through goal state. After the striking is completed the DMP can be used to return back to initial state or it can be terminated by setting the forcing terms to zero. An example is shown in Fig. 9.

One of the problems with such (kinesthetic) teach-in based approaches is that it is difficult to train heavy robots well for successful performance. For example, the shoulder of the Barrett WAM arm shown in Fig. 1 weighs 10 kg alone. It is rather difficult for humans to move the links with heavy inertia. The slower movements of the heavier links are typically compensated with faster movements of the lighter links (such as the wrist). However, tracking these trajectories can also be harder for more demanding wrist movements. An additional difficulty with cable-driven robots such as the Barrett WAM is that the wrists are harder to control.

Based on these considerations, we have worked on a free-final time optimal control based approach to generate minimum acceleration hitting movements for table tennis [2]. In the experiments section, we focus on learning to track these hitting movements. These trajectories are third order polynomials for each DoF of the robot.

We will briefly introduce here the trajectory generation framework introduced in [2]. Consider the following *free-time* optimal control problem [35]:

$$\min_{\ddot{\mathbf{q}},T} \int_0^T \ddot{\mathbf{q}}(t)^{\mathrm{T}} \mathbf{R} \ddot{\mathbf{q}}(t)\, \mathrm{d}t \tag{54}$$

$$\text{s.t.} \quad \boldsymbol{\Psi}_{\text{hit}}\big(\mathbf{q}(T), T\big) \in \mathcal{H} \tag{55}$$

$$\boldsymbol{\Psi}_{\text{net}}\big(\mathbf{q}(T), \dot{\mathbf{q}}(T), T\big) \in \mathcal{N} \tag{56}$$

$$\boldsymbol{\Psi}_{\text{land}}\big(\mathbf{q}(T), \dot{\mathbf{q}}(T), T\big) \in \mathcal{L} \tag{57}$$

$$\mathbf{q}(0) = \mathbf{q}_0 \tag{58}$$

$$\dot{\mathbf{q}}(0) = \dot{\mathbf{q}}_0 \tag{59}$$

where the final hitting time T is an additional variable to be optimized along with the joint accelerations $\ddot{\mathbf{q}}(t) : [0, T] \to \mathbb{R}^n$. The weighting matrix $\mathbf{R}$ for the accelerations is positive definite. Initial conditions for the robot are the joint positions $\mathbf{q}_0$ and joint velocities $\dot{\mathbf{q}}_0$. The inequality constraints (55)–(57) ensure that the task requirements for table tennis are satisfied, namely, hitting the ball, passing the net, and landing on the opponent's court.

Solutions of (54)–(59) can be found using Pontryagin's minimum principle [36]. The optimal $\mathbf{q}(t)$ in both cases is a third degree polynomial for each DoF. The striking time T, the joint position and velocity values at striking time $\mathbf{q}_f$ and $\dot{\mathbf{q}}_f$ fully parameterize this problem. The time it takes to return to the starting posture, T_{rest} can be chosen suitably, e.g., based on the speed of the ballgun. The polynomial coefficients for the striking trajectory

$$\mathbf{q}_{\text{strike}}(t) = \mathbf{a}_3 t^3 + \mathbf{a}_2 t^2 + \dot{\mathbf{q}}_0 t + \mathbf{q}_0 \tag{60}$$

can then be fully determined in joint-space

$$\begin{aligned} \mathbf{a}_3 &= \frac{2}{T^3}(\mathbf{q}_0 - \mathbf{q}_f) + \frac{1}{T^2}(\dot{\mathbf{q}}_0 + \dot{\mathbf{q}}_f) \\ \mathbf{a}_2 &= \frac{3}{T^2}(\mathbf{q}_f - \mathbf{q}_0) - \frac{1}{T}(\dot{\mathbf{q}}_f + 2\dot{\mathbf{q}}_0) \end{aligned} \tag{61}$$

for each DoF of the robot.

REFERENCES

[1] K. Muelling, J. Kober, O. Kroemer, and J. Peters, "Learning to select and generalize striking movements in robot table tennis," *Int. J. Robot. Res.*, no. 3, pp. 263–279, 2013. [Online]. Available: http://www.ias.informatik.tu-darmstadt.de/uploads/Publications/Muelling_IJRR_2013.pdf

[2] O. Koc, G. Maeda, and J. Peters, "Online optimal trajectory generation for robot table tennis," *Robot. Auton. Syst.*, vol. 105, pp. 121–137, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0921889017306164

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

[4] H.-S. Ahn, Y. Q. Chen, and K. Moore, "Iterative learning control: Brief survey and categorization," *IEEE Trans. Syst., Man, Cybernet., Part C, Appl. Rev.*, vol. 37, no. 6, pp. 1099–1121, Nov. 2007.

[5] N. Amann, D. H. Owens, and E. Rogers, "Iterative learning control for discrete time systems with exponential rate of convergence," 1995.

[6] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*, 1st ed. Cambridge, MA, USA: MIT Press, 1998.

[7] J. Kober and J. Peters, "Policy search for motor primitives in robotics," in *Proc. 21st Int. Conf. Neural Inf. Process. Syst.*, Cambridge, MA, USA: MIT Press, 2009. [Online]. Available: http://www-clmc.usc.edu/publications/K/kober_NIPS2008.pdf

[8] T. D. Son, G. Pipeleers, and J. Swevers, "Robust monotonic convergent iterative learning control," *IEEE Trans. Autom. Control*, vol. 61, no. 4, pp. 1063–1068, Apr. 2016.

[9] J. Nocedal and S. J. Wright, *Numerical Optimization*. New York, NY, USA: Springer, 1999.

[10] K. E. Avrachenkov, "Iterative learning control based on quasi-Newton methods," in *Proc. 37th IEEE Conf. Decis. Control*, 1998, pp. 170–174.

[11] S. Arimoto, S. Kawamura, and F. Miyazaki, "Bettering operation of robots by learning," *J. Robot. Syst.*, vol. 1, no. 2, pp. 123–140, 1984. [Online]. Available: http://dx.doi.org/10.1002/rob.4620010203

[12] E. Rogers, K. Galkowski, and D. H. Owens, *Control Systems Theory and Applications for Linear Repetitive Processes*. New York, NY, USA: Springer, 2007.

[13] M. Norrlöf and S. Gunnarsson, "Time and frequency domain convergence properties in iterative learning control," *Int. J. Control*, vol. 75, no. 14, pp. 1114–1126, 2002.

[14] R. W. Longman, "Iterative learning control and repetitive control for engineering practice," *Int. J. Control*, vol. 73, no. 10, pp. 930–954, 2000.

[15] S. Hillenbrand and M. Pandit, "An iterative learning controller with reduced sampling rate for plants with variations of initial states," *Int. J. Control*, vol. 73, no. 10, pp. 882–889, 2000.
[16] K.-H. Park and Z. Bien, "A generalized iterative learning controller against initial state error," *Int. J. Control*, vol. 73, no. 10, pp. 871–881, 2000.
[17] Y. Fang and T. Chow, "2-D analysis for iterative learning controller for discrete-time systems with variable initial conditions," *IEEE Trans. Circuits Syst. I, Fundam. Theory Appl.*, vol. 50, no. 5, pp. 722–727, May 2003.
[18] I. Chin, S. Qin, K. S. Lee, and M. Cho, "A two-stage iterative learning control technique combined with real-time feedback for independent disturbance rejection," *Automatica*, vol. 40, no. 11, pp. 1913–1922, Nov. 2004. [Online]. Available: http://dx.doi.org/10.1016/j.automatica.2004.05.011
[19] J. Z. Kolter and A. Y. Ng, "Policy search via the signed derivative," in *Robotics: Science and Systems*, J. Trinkle, Y. Matsuoka, and J. A. Castellanos, Eds. Cambridge, MA, USA: MIT Press, 2009. [Online]. Available: http://dblp.uni-trier.de/db/conf/rss/rss2009.html#KolterN09
[20] D. Nguyen-Tuong and J. Peters, "Model learning for robot control: A survey," *Cognitive Process.*, vol. 12, no. 4, pp. 319–340, Apr. 2011. [Online]. Available: http://www.is.tuebingen.mpg.de/fileadmin/user_upload/files/publications/2011/Cognitive-Science-2011-ModelLearning.pdf
[21] J. Ghosh and B. Paden, "Pseudo-inverse based iterative learning control for nonlinear plants with disturbances," in *Proc. 38th IEEE Conf. Decis. Control*, vol. 5, 1999, pp. 5206–5212.
[22] J. G. P. Barnes, "An algorithm for solving non-linear equations based on the secant method," *Comput. J.*, vol. 8, no. 1, pp. 66–72, 1965. [Online]. Available: http://dx.doi.org/10.1093/comjnl/8.1.66
[23] A. P. Schoellig, F. L. Mueller, and R. D'Andrea, " Optimization-based iterative learning for precise quadrocopter trajectory tracking," *Auton. Robots*, vol. 33, pp. 103–127, 2012. [Online]. Available: http://dx.doi.org/10.1007/s10514-012-9283-2
[24] J. van den Berg *et al.*, "Superhuman performance of surgical tasks by robots using iterative learning from human-guided demonstrations," in *Proc. IEEE Int. Conf. Robot. Autom.*, May 2010, pp. 2074–2081.
[25] G. J. Maeda, I. Manchester, and D. Rye, "Combined ILC and disturbance observer for the rejection of near-repetitive disturbances, with application to excavation," *IEEE Trans. Control Syst. Technol.*, vol. 23, no. 5, pp. 1754–1769, Sep. 2015.
[26] M. Deisenroth and C. Rasmussen, "PILCO: A model-based and data-efficient approach to policy search," in *Proc. Int. Conf. Mach. Learn.*, 2011. [Online]. Available: http://www.ias.informatik.tu-darmstadt.de/uploads/Publications/Deisenroth_ICML_2011.pdf
[27] M. W. Spong, S. Hutchinson, and M. Vidyasagar, *Robot Modeling and Control*. Hoboken, NJ, USA: Wiley, 2006. [Online]. Available: http://opac.inria.fr/record=b1119287
[28] B. D. O. Anderson and J. B. Moore, *Optimal Control: Linear Quadratic Methods*. New York, NY, USA: Dover, 1989.
[29] D. A. Kendrick, *Stochastic Control for Economic Models*. New York, NY, USA: McGraw-Hill, 2002.
[30] C. H. Lampert and J. Peters, "Real-time detection of colored objects in multiple camera streams with off-the-shelf hardware components," *J. Real-Time Image Process.*, vol. 7, no. 1, pp. 31–41, Mar. 2012. [Online]. Available: http://dx.doi.org/10.1007/s11554-010-0168-3
[31] S. Schaal, "The SL simulation and real-time control software package," Univ. Southern California, Los Angeles, CA, USA, 2009. [Online]. Available: http://www-clmc.usc.edu/publications/S/schaal-TRSL.pdf
[32] O. Koc, G. Maeda, G. Neumann, and J. Peters, "Optimizing robot striking movement primitives with iterative learning control," in *Proc. IEEE-RAS 15th Int. Conf. Humanoid Robots (Humanoids)*, 2015, pp. 80–87.
[33] J. Kober, K. Muelling, O. Kroemer, C. Lampert, B. Schoelkopf, and J. Peters, "Movement templates for learning of hitting and batting," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2010, pp. 853–858.
[34] A. Ijspeert, J. Nakanishi, and S. Schaal, "Movement imitation with nonlinear dynamical systems in humanoid robots," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2002, pp. 1398–1403.
[35] D. Liberzon, *Calculus of Variations and Optimal Control Theory: A Concise Introduction*. Princeton, NJ, USA: Princeton Univ. Press, 2011.
[36] L. S. Pontryagin, V. G. Boltyanskii, R. V. Gamkrelidze, and E. F. Mishchenko, *The Mathematical Theory of Optimal Processes*. New York, NY, USA: Interscience, 1962.

Okan Koç received the B.S. degree in electrical engineering and mathematics from Boğaziçi University, Istanbul, Turkey, in 2009, the M.S. degree in applied mathematics from ETH Zürich, Zürich, Switzerland, in 2013, and the Ph.D. degree in intelligent autonomous systems from Technische Universität Darmstadt, Darmstadt, Germany, in 2018.

From 2014 to 2018, he was with the Empirical Inference Department, Max Planck Institute for Intelligent Systems, Tüebingen, Germany. His research interests include optimal control, learning control, and more specifically robot learning.

Guilherme Maeda received the master's degree in control engineering from the Tokyo Institute of Technology, Tokyo, Japan, in 2007, and the Ph.D. degree in robotics from the Australian Centre for Field Robotics, Darlington, NSW, Australia, in 2013.

He is currently a Research Scientist with the Department of Brain Robot Interface, ATR Computational Neuroscience Laboratories, Kyoto, Japan. From 2013 to 2017, he was a Team Leader with the Intelligent Autonomous Systems Group, Technische Universität Darmstadt, Darmstadt, Germany. His research interests include robotics, specifically in control, learning methods for control, and their applications to physical human–robot interaction and collaboration.

Jan Peters received the M.Sc. degree in computer science from FernUni Hagen, Hagen, Germany, in 2000, the M.Sc. degree in electrical engineering from TU Munich, Munich, Germany, in 2001, and the M.Sc. degrees in computer science and mechanical engineering and the Ph.D. degree in computer science from the University of Southern California, Los Angeles, CA, USA, in 2002, 2006, and 2007, respectively.

He is currently a Full Professor (W3) of Intelligent Autonomous Systems with the Department of Computer Science, Technische Universität Darmstadt, Darmstadt, Germany, and at the same time an Adjunct Senior Research Scientist with the Max-Planck Institute for Intelligent Systems, Tüebingen, Germany, where he heads the Robot Learning Group in the Empirical Inference Department.

Mr. Peters has been the recipient of awards such as the Dick Volz Best U.S. Ph.D. Thesis Runner Up Award, in 2007, the Robotics: Science and Systems - Early Career Spotlight, in 2012, the IEEE Robotics and Automation Society's Early Career Award, in 2013, and the INNS Young Investigator Award, in 2013.