Autonomous Robots (2023) 47:1387–1403
https://doi.org/10.1007/s10514-023-10140-6

Reinforcement learning with model-based feedforward inputs for robotic table tennis

Hao Ma[1] · **Dieter Büchler**[2] · **Bernhard Schölkopf**[2] · **Michael Muehlebach**[1]

Received: 31 January 2023 / Accepted: 18 September 2023 / Published online: 17 October 2023

Abstract
We rethink the traditional reinforcement learning approach, which is based on optimizing over feedback policies, and propose a new framework that optimizes over feedforward inputs instead. This not only mitigates the risk of destabilizing the system during training but also reduces the bulk of the learning to a supervised learning task. As a result, efficient and well-understood supervised learning techniques can be applied and are tuned using a validation data set. The labels are generated with a variant of iterative learning control, which also includes prior knowledge about the underlying dynamics. Our framework is applied for intercepting and returning ping-pong balls that are played to a four-degrees-of-freedom robotic arm in real-world experiments. The robot arm is driven by pneumatic artificial muscles, which makes the control and learning tasks challenging. We highlight the potential of our framework by comparing it to a reinforcement learning approach that optimizes over feedback policies. We find that our framework achieves a higher success rate for the returns (100% vs. 96%, on 107 consecutive trials, see https://youtu.be/kR9jowEH7PY) while requiring only about one tenth of the samples during training. We also find that our approach is able to deal with a variant of different incoming trajectories.

Keywords Reinforcement learning · Iterative learning control · Supervised learning · Table tennis robot · Pneumatic artificial muscle · Soft robotics

1 Introduction

Reinforcement learning has been proven to be highly effective in a variety of contexts. An important example is `AlphaGo Zero` (Silver et al., 2016, 2017), which managed to completely overpower all human players in the game of Go. Other examples include the work of Oh et al. (2016), Tessler et al. (2017), Firoiu et al. (2017), Kansky et al. (2017) that focused on video games, the work of Yogatama et al. (2016), Paulus et al. (2017), Zhang and Lapata (2017) that focused on natural language processing, and the work of Liu et al. (2017), Devrim Kaba et al. (2017), Cao et al. (2017), Brunner et al. (2018) that focused on computer vision. Despite these successes, where reinforcement learning agents are shown to compete and outperform humans, researchers have struggled to achieve a similar level of success in robotics applications. We identify the following key bottlenecks, which we believe hinder the application of reinforcement learning to robotic systems. This also motivates our work, which proposes a new reinforcement learning scheme that addresses some of these shortcomings.

The first factor is that the lack of prior knowledge causes reinforcement learning algorithms to sometimes apply relatively aggressive feedback policies[1] during training. This has the potential to cause irreversible damage to robotic systems, which are often expensive and require careful maintenance (Moldovan & Abbeel, 2012; Schneider, 1996). Moreover, these aggressive policies are typically not effec-

✉ Hao Ma
hao.ma@tuebingen.mpg.de

Dieter Büchler
dieter.buechler@tuebingen.mpg.de

Bernhard Schölkopf
bernhard.schoelkopf@tuebingen.mpg.de

Michael Muehlebach
michael.muehlebach@tuebingen.mpg.de

[1] Learning and Dynamical Systems, Max Planck Institute for Intelligent Systems, Max-Planck-Ring 4, Tübingen 72076, Germany

[2] Empirical Inference, Max Planck Institute for Intelligent Systems, Max-Planck-Ring 4, Tübingen 72076, Germany

[1] The term “aggressive feedback policy" refers to a policy that generates actions with the potential to induce harm or cause damage to real-world systems.

tive, neither for revealing relevant system dynamics (exploration) nor for maximizing reward.

Second, reinforcement learning is often data hungry (Laskin et al., 2020), although the required amount of training data depends on the task at hand. In some cases, these data requirements can lead to weeks or months of training: For example, the work by Heess et al. (2017) reported that approximately 100 hours of simulation time (possibly more if conducted in real time) were required to train a 9-DoF mannequin to achieve walking behavior within the simulation environment. In a similar vein, Kalashnikov et al. (2018) collected over 800 hours of robot training data intermittently across seven robots over a span of four months to train the robots for challenging grasping tasks. Combined with the first factor, this also increases the possibility of destroying the robotic system during training.

Third, the behavior of the robot is characterized by the reward function. While in video games and board games, a binary reward function (success or failure) can be used to evaluate the performance of policies, it is much more difficult to characterize the desired behavior of a robotic system with a single reward function. For example, when two behaviors of a robot return the same reward, there is no way of judging which one is better (Kober et al., 2013). As a result, it often takes more effort to design the reward function of reinforcement learning in the field of robotics. More importantly, results from optimal control suggest that optimizing for a single criterion such as execution time, tracking error or energy often results in policies that are brittle and lack robustness with respect to modelling errors (Doyle, 1978).

To address the aforementioned first factor, our approach includes a model-based part, where prior knowledge can be incorporated. Model-based reinforcement learning methods have gained prominence in robotics research in recent years (Levine & Koltun, 2014; Deisenroth et al., 2014; Van Rooijen et al., 2014; Wilson et al., 2014; Kupcsik et al., 2017; Boedecker et al., 2014). In contrast to model-free reinforcement learning methods, model-based approaches leverage the (approximate) system dynamics, sometimes enabling faster convergence towards the optimal policy while reducing the number of interactions required between the robot and its environment (Polydoros & Nalpantidis, 2017; Luo et al., 2022; Wang et al., 2019b).

Numerous methodologies have emerged to address the second factor, including the influential paradigm of meta-learning (Vanschoren, 2018; Alet et al., 2018; Lake et al., 2016). Meta-learning entails leveraging previously acquired skills from related tasks, reusing successful approaches, and prioritizing potential strategies based on accumulated experience. This paradigm is also considered to be a form of transfer learning, often referred to as learning to learn (Zhuang et al., 2021; Thrun & Pratt, 1998). However, research by Kaushik et al. (2020) demonstrated that when faced with diverse and complex dynamics, a substantial number of observations from the real-world system might still be required to effectively learn a reliable dynamics model. It is worth noting that our work primarily focuses on the paradigm of learning from scratch and is therefore not directly related to meta-learning.

Additionally, sim-to-real learning has gained significant traction as a widely adopted technique in the realm of reinforcement learning (Zhao et al., 2020; Matas et al., 2018). This approach relies on collecting training data from simulated environments and subsequently applying the simulation-based policies to real-world scenarios. Nonetheless, the mismatches between simulated and real-world settings pose substantial challenges. Ongoing research endeavors concentrate on refining the fidelity of physics engines within simulations to better approximate real-world dynamics, aiming to facilitate the direct deployment of policies trained in simulated environments (Shah et al., 2017; Dosovitskiy et al., 2017; Furrer et al., 2016; McCord et al., 2019; Todorov et al., 2012). Another research direction involves augmenting the safety measures associated with real-world robot training, facilitating online training of robots in actual environments, even in the presence of mismatches between simulated and real-world systems (Garcia & Fernandez, 2015; Cheng et al., 2019; Ramya Ramakrishnan et al., 2020). In the context of this article, the use of pneumatic artificial muscles presents obstacles in accurately capturing dynamic characteristics in simulations. Our work is not concerned with optimizing the performance of physics engines and operates directly with the real-world system, thereby avoiding sim-to-real learning.

For the third factor, Laud (2004) and Grzes (2017) introduced a process known as reward shaping, which includes intermediate rewards in the reward function to guide the learning process to a reasonable behavior. In addition, multiple criteria are introduced to balance the essential factors of the interaction with the environment during learning (Bagnell et al., 2006). Nonetheless, we believe that this still represents an important open issue. In our setting, by contrast, we first decompose the original problem into a subproblem (trajectory tracking) that is easier to solve. As a result, we are able to design the reward function (tracking error) in a principled way and without any auxiliary terms.

The focus of our work is on alleviating and resolving the three issues mentioned above. This is achieved by first decomposing the original high-level problem of playing table tennis into three subproblems: (i) prediction of incoming ball trajectories, (ii) planning reference trajectories that lead to successful returns, (iii) tracking the reference trajectories with the robot arm. The latter subproblem (trajectory tracking) can by itself be formulated as a reinforcement learning problem, as we highlight in Sect. 2 below. The decomposition has the important benefits that each component can be designed, tuned, and debugged individually. Moreover,

the decomposition enables us to incorporate task-specific knowledge, which, as we will demonstrate in the following, makes our learning sample efficient. For the subproblem of trajectory tracking, we propose a reinforcement learning framework that optimizes over feedforward inputs (actions) instead of feedback policies and contains both a model-based and model-free part. In a first step, we use iterative learning control (ILC) to compute input commands (actions) that minimize tracking error. Due to the fact that we can incorporate prior knowledge, the learning is very efficient and requires typically to execute only about 20–30 iterations on the robot. We balance the exploration and exploitation trade-off by adjusting the distribution of reference trajectories and the number of ILC iterations. For example, if we increase the number of reference trajectories, and decrease the number of ILC iterations, we shift towards more exploration and less exploitation. This first step provides us with pairs of reference and input trajectories, which constitutes the data and the labels for the second step. Here, we use a model-free supervised learning approach for learning a parameterized policy network that returns a sequence of nearly optimal input commands for any given reference trajectory. By doing so, we split the learning into two parts, both of which are well-known and well-understood: An ILC part that produces data and labels in a sample-efficient manner and a supervised learning task, which can be tuned offline using a validation data set. Exploration and exploitation is traded off in a direct way by balancing the diversity and number of reference trajectories with the number of ILC iterations. We will apply our framework to control a table tennis robot that is actuated by pneumatic artificial muscles (PAMs) as shown in Fig. 1. The same robot arm has been used in prior works, see for example Büchler et al. (2023). While playing table tennis is a relatively standard task for humans, it is full of challenges for robots. We show that, with the help of our framework, the robot is able to successfully learn how to intercept and return ping-pong balls in a safe and sample-efficient manner.

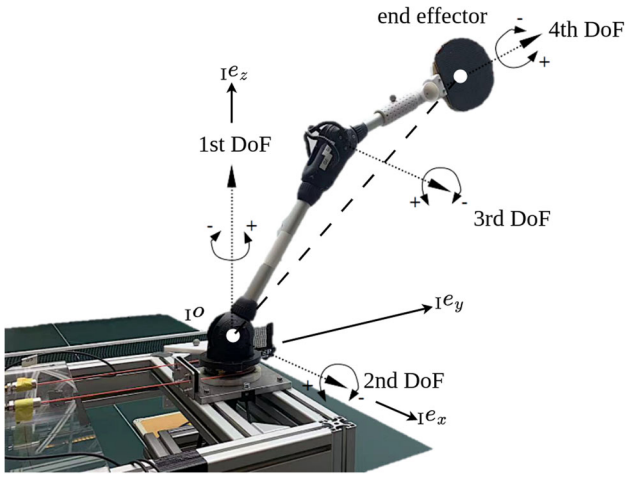


Fig. 1 The figure shows the structure of the robot arm. It has four rotational joints, and each joint is actuated by a pair of PAMs. The unit vectors ${}_I e_x$, ${}_I e_y$ and ${}_I e_z$ together form the global coordinate system {I} with ${}_I o$ as the origin. When intercepting the balls, the third degree of freedom is always above the line connecting the origin and end effector. For simplicity we consider only DoF 1–3, whereas DoF 4 is controlled with a proportional-integral-derivative (PID) controller. We note that DoF stands for degree of freedom

1.1 Related work

In this work, a dynamic model of the robot arm is introduced as prior knowledge, which speeds up the convergence of the learning process and enhances interpretability. However, the robot arm is actuated by artificial muscles, which make the derivation of accurate models challenging. Since 1960s, researchers have tried various approaches for modelling PAMs, which include first-principle models (Tondu & Zagal, 2006; Nickel et al., 1963; Ganguly et al., 2012), grey-box models (Hofer & D'Andrea, 2018; Kogiso et al., 2012), and black-box models (Ba et al., 2016). Our framework only requires a coarse model of the robot arm and, as we will see, even a low-complexity linear model will be enough to effectively guide the learning process and reduce its sample complexity.

In the first part of our two-step procedure we apply a variant of ILC, which has been widely and successfully used to tackle trajectory tracking problems in robotics. For example, Mueller et al. (2012) and Schoellig et al. (2012) achieved high-performance tracking of quadcopters using ILC; a similar performance was also achieved with other complex systems (Luo & Hauser, 2017; Zhao et al., 2015; Jian et al., 2019) including a thrust-vectored flying vehicle (Sferrazza et al., 2020). ILC stands for the repeatability of operating a given system, which can be exploited to update the control input based on previous data, thereby improving the transient performance over a fixed time interval (Arimoto et al., 1986; Bristow et al., 2006; Ahn et al., 2007). Due to the high nonlinearity of PAMs, Hofer et al. (2019) and Zughaibi et al. (2021) proposed the use of ILC to improve the tracking performance for an articulated soft robot arm during aggressive maneuvers. However, the most fatal flaw of ILC is that it only works for fixed reference trajectories. If the reference trajectory changes, ILC needs to be trained from scratch. While Chen et al. (2021) used deep learning to reduce the number of iterations required for ILC training, the approach still requires a couple of ILC executions when presented with a new trajectory.

In the context of robotics, researchers have also tried different methods to transform reinforcement learning into (semi-)supervised learning tasks. Finn et al. (2017) and Konyushkova et al. (2020) managed to use reinforcement learning to learn policies in labeled scenarios, and then gen-

eralized the policies to unlabeled scenarios with a deep neural network. However, their experiments are only carried out in simulation and it is unclear how effective their approach is for real-world robotic systems. Wang et al. (2019a) focused on a few specific learning problems and applied tools from supervised learning to address the problem of overfitting. While the underlying ideas share some common ground with our work, we apply ILC to transform a reinforcement learning task to a supervised learning problem in a very direct and intuitive way. Of course, statistical results that quantify uncertainty and/or sample complexity of supervised learning are also applicable in our setting (this is, however, not the focus of our work). Moreover, Fathinezhad et al. (2016) combined supervised learning and fuzzy control to generate initial solutions for reinforcement learning, thereby reducing the failure probability during training. Piche et al. (2022) made robots learn skills from a data set collected by policies of different expertise levels. However, this idea is more similar to imitation learning (Ravichandar et al., 2020), and the experiments are carried out only in simulation. In our system, for example, an imitation learning approach is very difficult to apply, since, due to the nonlinearity of PAMs and the requirements on the bandwidth and execution speed of motion, it is very difficult to generate expert policies.

Preliminary results from Sect. 3.2 have been presented in the conference publication Ma et al. (2022). While the focus in Ma et al. (2022) was on trajectory tracking, we apply our framework to intercept and return ping-pong balls that are played to the robot. We also compare the sample efficiency of our learning to the reinforcement learning approach used in Büchler et al. (2020). Finally, the article has a tutorial character and highlights the different steps that are required for solving a real-world robotic task with learning.

1.2 Contribution

The main contribution of this work is to demonstrate a potential way to decompose a high-level learning problem of playing table tennis into subproblems, which are easier to solve. To solve the trajectory tracking problem in the decomposed subproblems, we propose a sample-efficient reinforcement learning framework that transforms the given task into a supervised learning problem. We successfully apply our framework to a four-degrees-of-freedom robot arm driven by PAMs, where we demonstrate accurate tracking of a large number of reference trajectories relevant to our task. Our approach includes a model of the system as prior knowledge, which guides the ILC algorithm. The model can be obtained from first-principles, however, it can also be a black-box or grey-box model. For our experiments we rely on the black-box model that was identified in Ma et al. (2022). We use ILC to learn optimal feedforward inputs for a large number of reference trajectories. The reference trajectories are randomly sampled and are representative for the task at hand, which is, intercepting and returning ping-pong balls. The ILC learns and compensates repeatable disturbances when tracking the trajectories, which includes unmodeled nonlinearities, actuation biases, delays, and unmodeled dynamics. It achieves a remarkable tracking performance for a given reference trajectory, while requiring only 30 iterations. These reference trajectories and the feedforward inputs learned by the ILC will be used as data and labels for training a parameterized policy network in a supervised manner. This results in a nonlinear feedforward controller that can handle different reference trajectories and generalizes the excellent tracking performance of ILC to non-fixed reference trajectories.

Key advantages of our framework are a low number of hyperparameters, which have a physical interpretation and can be tuned in a principled way. Moreover, compared to so called model-free reinforcement learning algorithms, our framework incorporates prior knowledge about the system dynamics, which guides the learning by providing closed-form gradient information. This not only avoids gradient computations via finite differences or sampling-based approaches, but also improves the sample complexity of the learning. Due to the fact that our parameterized policy network is only used for computing feedforward controls, the method mitigates the risk of destabilizing the system. Thus, our framework can be directly applied to the robot without any pre-training in simulation.

In addition, we describe how our learning framework can be integrated with an existing vision system to intercept and return balls, achieving a 100% interception success rate for the balls that are played to the robot arm. This interception also requires the design of an extended Kalman filter (EKF) to estimate the state of the ball, which includes the position and velocity, as well as an impact model between the ball and the table. We use a data-driven approach to build the impact model. Finally, our interception also computes an interception point within the reach of the robot arm and plans a reference trajectory with minimum jerk.

1.3 Structure

This paper is structured as follows: in Sect. 2, we will introduce the main concepts of our framework. The overall control structure of our robot arm is based on the classical two-degrees-of-freedom control loop and includes a feedforward block and a feedback controller. The feedback controller will be fixed and our learning framework will only optimize over the feedforward block. Thus, unlike a classical reinforcement learning approach, we only learn feedforward inputs, which has numerous advantages. First, it allows us to incorporate a model, which provides important gradient information and reduces the sample complexity of the learning. Second, learning feedforward greatly reduces the risk of destabilizing the

underlying robotic system during training. This will also be further discussed in Sect. 2. In Sect. 3, we apply our framework to the task of playing table tennis with our robotic arm. The section contains an implementation tutorial and describes the ILC, the design of a convolutional neural network (CNN) for parameterizing the policy, the EKF, as well as the strategies for intercepting the balls. In Sect. 4, we compare our method with a traditional reinforcement learning algorithm that is also applied to the same robot arm. We also highlight the modularity and stability of our framework in this section. The aim of the comparison is not to argue that our approach is generally superior to black-box reinforcement learning, but to demonstrate, with a specific example, how much in terms of sample efficiency can be gained by incorporating task-specific knowledge in a principled way. The article concludes with a summary in Sect. 5.

2 Reinforcement learning as a supervised learning task

Reinforcement learning describes stochastic dynamic programming problems whereby the transition function is unknown (Bertsekas, 2019). The reinforcement learning task that is considered herein is formulated as follows:

$$\min_{\pi} J_\pi\left(x_0\right) = \min_{\pi} \mathbb{E}\left\{\sum_{k=0}^{q-1} g_k\left(x_k, u_k, \omega_k, y_{\mathrm{des},k}\right)\right\},$$

where $x_k \in \mathbb{R}^n$ denotes the state and x_0 is fixed, $u_k \in \mathbb{R}^m$ denotes the control input, $\omega_k \in \mathbb{R}^w$ a stochastic disturbance, $\pi = (\mu_0, \ldots, \mu_{q-1})$ the policy, and $y_{\mathrm{des}} = \left(y_{\mathrm{des},0}, y_{\mathrm{des},1}, \ldots, y_{\mathrm{des},q-1}\right) \in \mathbb{R}^{l\times q}$ a reference trajectory that we would like to track. The reference trajectory is unknown and uncertain, which will be modeled by assuming that y_{des} is random and distributed according to the distribution $p_{y_{\mathrm{des}}}$. The distribution $p_{y_{\mathrm{des}}}$ characterizes reference trajectories that are likely for the given task at hand. In our table tennis application, the reference trajectories arise from typical interception and return motions of the robot arm. The state x_k evolves through the system equations

$$\begin{aligned} x_{k+1} &= f_k\left(x_k, u_k, \omega_k\right), \\ y_k &= h_k\left(x_k, \omega_k\right), \quad k = 0, 1, \ldots, q-1, \end{aligned}$$

where $y_k \in \mathbb{R}^l$ denotes the output of the system, which is measured, and $u_k = \mu_k(y_0, \ldots, y_k, y_{\mathrm{des}}) \in \mathbb{R}^m$ denotes the control inputs (actions). In contrast to the problem formulation in Bertsekas (2019), for example, we do not assume to have access to the state x_k, which would allow for state feedback, and thus treat the more general case of output feedback. In addition, our problem formulation allows for reference tracking tasks, by allowing the running cost g_k and the functions μ_k to depend on y_{des}. The disturbance ω_k is stochastic and may explicitly depend on the state x_k and input u_k, but not on the prior disturbances $\omega_{k-1}, \omega_{k-2}, \ldots, \omega_0$. The system equations are unknown.

Computing a policy π that minimizes J_π is very difficult in general, even when the system equations are known (Bertsekas, 2012). We therefore deliberately simplify the problem at hand in two steps. First, we restrict our feedback functions μ_k to only depend on y_{des}, which amounts to feedforward control. In order to highlight this design choice we will denote the corresponding policy functions by $\mu_{\mathrm{ff},k}$ and the policy $\pi_{\mathrm{ff}} = (\mu_{\mathrm{ff},0}, \mu_{\mathrm{ff},1}, \ldots, \mu_{\mathrm{ff},q-1})$, where we added the subscript ff. Second, we define the running cost g_k to be the tracking error, that is,

$$g_k\left(x_k, \mu_k, \omega_k, y_{\mathrm{des},k}\right) = \frac{1}{2}\left|y_k - y_{\mathrm{des},k}\right|^2,$$

where $|\cdot|$ denotes the ℓ_2-norm.

This allows us to reformulate the minimization of J_π over π as follows:

$$\begin{aligned} &\min_{\pi_{\mathrm{ff}}} \mathbb{E}\left\{\frac{1}{2}\left|y - y_{\mathrm{des}}\right|^2\right\} \\ &\text{s.t. } y = F\left(x_0, u, \omega\right), \quad u = \pi_{\mathrm{ff}}(y_{\mathrm{des}}), \end{aligned} \tag{1}$$

where $u = (u_0, u_1, \ldots, u_{q-1}) \in \mathbb{R}^{m\times q}$ concatenates the entire sequence of inputs, $y = (y_0, y_1, \ldots, y_{q-1}) \in \mathbb{R}^{l\times q}$ the entire sequence of outputs, and $\omega = (\omega_0, \omega_1, \ldots, \omega_{q-1}) \in \mathbb{R}^{w\times q}$ the entire sequence of disturbances. The transition dynamics of our robotic system are represented with the function $F : \mathbb{R}^n \times \mathbb{R}^{m\times q} \times \mathbb{R}^{w\times q} \to \mathbb{R}^{l\times q}$. We note that F is unknown, possibly nonlinear, and can even model non-Markovian transition dynamics. The state of the system is hidden in the function F, since we focus on the input–output relationship and consider only trajectories of fixed length (we will describe how to handle trajectories with different length later on).

We note that the restriction of feedback policies π to feedforward policies π_{ff} is suboptimal and therefore $\min_\pi J(x_0) \leq \min_{\pi_{\mathrm{ff}}} J(x_0)$. If ω is deterministic, however, we have $\min_\pi J(x_0) = \min_{\pi_{\mathrm{ff}}} J(x_0)$.

Remark 1 In order to simplify the notation in the following sections, we will flatten all multi-dimensional vectors into one-dimensional vectors accordingly, that is $u \in \mathbb{R}^{mq}$, $y \in \mathbb{R}^{lq}$, $\omega \in \mathbb{R}^{wq}$, and $y_{\mathrm{des}} \in \mathbb{R}^{lq}$. Therefore, F can be redefined as $F : \mathbb{R}^n \times \mathbb{R}^{mq} \times \mathbb{R}^{wq} \to \mathbb{R}^{lq}$.

Due to the fact that π_{ff} is an arbitrary function of y_{des}, (1) is equivalent to

$$\mathbb{E}_{y_{\mathrm{des}}}\left\{\min_{u\in\mathbb{R}^{mq}} \mathbb{E}_{\omega|y_{\mathrm{des}}}\left\{\frac{1}{2}\left|y - y_{\mathrm{des}}\right|^2\right\}\right\}, \tag{2}$$

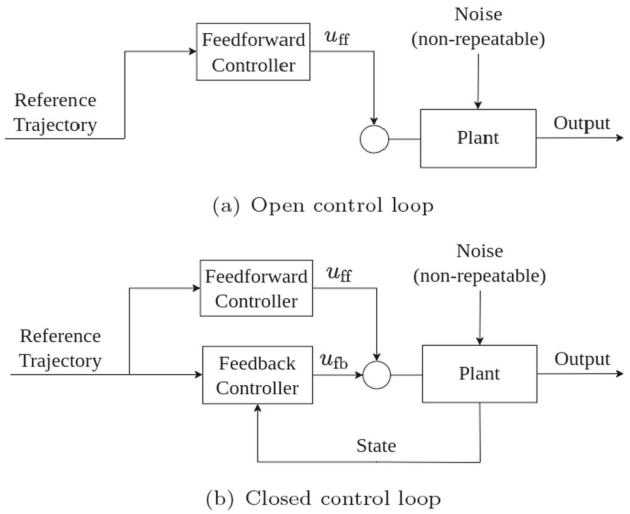


Fig. 2 The figure shows the open control loop (top) and the closed control loop (bottom)

where the minimization is subject to $y = F(x_0, u, \omega)$. This motivates our reinforcement learning framework, since we can solve the minimization over u in (2) with ILC in a very sample-efficient manner. We therefore sample different reference trajectories, apply ILC to compute the minimization in (2), which yields the minimizer $u^\star(y_{\text{des}})$, and finally fit a parametrized policy network π_{ff} for predicting $u^\star(y_{\text{des}})$. This addresses the two key challenges in our learning problem, which is that i) the dynamics F are unknown, ii) the reference trajectories are unknown/uncertain.

The detailed procedure is summarized with the following four steps:

1. We define a distribution $p_{y_{\text{des}}}$ that characterizes the uncertainty about the reference trajectories.
2. We sample a data set y^i_{des}, $(u^\star)^i$, with $i = 1, \ldots, N$, where $y^i_{\text{des}} \sim p_{y_{\text{des}}}$ and $(u^\star)^i = u^\star\left(y^i_{\text{des}}\right)$ is a minimizer of (2). The minimization in (2) is done with ILC.
3. We split the data set y^i_{des}, $(u^\star)^i$, with $i = 1, \ldots, N$ into a training and validation data set, and train a parameterized policy network $\pi_{\text{ff}} : \mathbb{R}^{l\times(2h+1)} \to \mathbb{R}^m$, which predicts the approximate optimal input (action) $u^\star_k \in \mathbb{R}^m$ at time point k for a given reference trajectory over the horizon $\left(y_{\text{des},k-h}, \ldots, y_{\text{des},k}, \ldots, y_{\text{des},k+h}\right)$, where $2h+1$ refers to the horizon length.
4. We integrate the parameterized policy network as the feedforward block in the two-degrees-of-freedom control structure shown in Fig. 2b. The feedback controller is fixed and not affected by the learning.

The following remarks are important. The distribution $p_{y_{\text{des}}}$ characterizes reference trajectories that are likely for the given task at hand. In our table tennis example, the reference trajectories arise from typical interception and return motions of the robot arm. More precisely, $p_{y_{\text{des}}}$ arises from sampling different interception points and planning minimum jerk trajectories that lead to these interception points. The procedure will be explained in further details in Sect. 3.1.

During the minimization of (2) with ILC, the system is operated in open loop, as shown in Fig. 2a. There are numerous advantages and disadvantages for performing ILC either on the open-loop or the closed-loop system. While in the closed-loop setting the feedback controller can attenuate measurement and process noise, and potentially pre-stabilize an unstable system, the ILC results, as well as the parameterized policy network π_{ff} are tailored to the specific feedback controller in use. In contrast, we run the ILC in open loop, which means that we can later change and adapt the feedback controller (see Fig. 2b and Step 4) without the need of retraining and rerunning the above steps. Moreover, as we will discuss later on, the ILC exploits a coarse model of the system, which provides gradient information and guides the learning. The model that we have at our disposal is obtained from open-loop experiments, which further motivates running ILC in open loop. The details about the ILC implementation and the corresponding results are summarized in Sect. 3.2.

The parameterized policy network π_{ff} computes a single input $u_k \in \mathbb{R}^m$ from a sliding window over the reference trajectories, which includes h past values, $y_{\text{des},k-h}, \ldots, y_{\text{des},k-1}$, the current values $y_{\text{des},k}$, and h future values, $y_{\text{des},k+1}, \ldots, y_{\text{des},k+h}$. The parameterized policy network π_{ff} can therefore directly be used as a nonlinear feedforward block in Fig. 2b. Further details are described in Sect. 3.3.

As is also highlighted with Step 4 (which is further described in Sect. 3.3) our reinforcement learning approach learns the feedforward block in Fig. 2b and does not affect the feedback controller. This is in sharp contrast to traditional approaches, which only optimize over feedback policies and where, in many cases, feedforward is completely ignored.

A more extended discussion of the stability guarantees of our framework can be found in Ma et al. (2022). In the following, we assume the plant to be stable and use ILC to optimize over feedforward inputs in open loop. However, if the open-loop system was unstable, it could be pre-stabilized with a feedback controller and our framework can be applied nonetheless (see also the previous discussion on learning in open loop versus learning in closed loop).

3 Implementation tutorial

In this section, we show how to decompose the problem of intercepting and returning a table tennis ball into the different subproblems (predicting ball trajectories, generating reference trajectories, and tracking), which are then solved

individually. The section follows Step 1–4 as described in Sect. 2.

3.1 Sampling reference trajectories

In this section, we will introduce the way we generate reference trajectories, which arise from the task of intercepting table tennis balls. We use a ball launcher, similar to Dittrich et al. (2022), and shoot ping-pong balls towards the robot. The balls are tracked with a vision system, Gomez-Gonzalez et al. (2019), and the resulting ball trajectories are stored in a data set that contains 43 trajectories. We now generate reference trajectories according to the following process.

1. For each ball trajectory we compute the highest point after impact with the table, and define this to be our interception point ${}^{\mathrm{I}}p_{\mathrm{int}} \in \mathbb{R}^3$ in the global coordinate frame {I}. If ${}^{\mathrm{I}}p_{\mathrm{int}}$ is not in the reachable range of the robot arm the trajectory is discarded.
2. Our robot arm has three main degrees of freedom, which correspond to DoF 1–3 in Fig. 1. For each of these degrees of freedom we plan a trajectory $y_1^i(t)$ from the rest position ${}^{\mathrm{I}}p_{\mathrm{ini}} \in \mathbb{R}^3$ of the end effector to ${}^{\mathrm{I}}p_{\mathrm{int}}$ and a trajectory $y_2^i(t)$ back to ${}^{\mathrm{I}}p_{\mathrm{ini}}$, with $i = 1, 2, 3$. Here, $y_1^i(t)$ and $y_2^i(t)$ denote the desired angles for degree of freedom d.
3. We merge the two trajectories $y_1^i(t)$ and $y_2^i(t)$ into one complete trajectory $y^i(t)$, with $i = 1, 2, 3$.

Remark 2 We constrain the angle θ_3 of the third degree of freedom to be negative (see Fig. 1), which means that we restrict ourselves to configurations where the joint of the third degree of freedom is above the line connecting the origin ${}_{\mathrm{I}}o$ with the end effector. As a result, the mapping between the position of the end effector in the global coordinate frame and the angles of DoF 1–3 is bijective.

We note that the robot arm will intercept the table tennis ball at the highest position after the ball collides with the table for the first time. At this position, the ball has the lowest velocity, which simplifies the interception task. The time T_1 from when the ball leaves the launcher to when the ball reaches the interception position ${}^{\mathrm{I}}p_{\mathrm{int}}$ is used to plan the first segment of the trajectory, while the second segment of the reference trajectory is set to a fixed duration T_2 of $T_2 = 1.5\,\mathrm{s}$. Immediately afterwards, the robot arm is required to remain stable at ${}^{\mathrm{I}}p_{\mathrm{ini}}$ for $T_3 = 0.2\,\mathrm{s}$. To sum up, in our design, the total time T_{total} required for the robot to complete a hit is $T_{\mathrm{total}} = T_1 + T_2 + T_3$.

It will be convenient to model the motion of the end effector in a polar coordinate frame, where the z-axis is aligned with DoF 1 (and the z-axis of the global frame {I}). The polar coordinate frame is defined as $\{\theta_1, \eta, \xi\}$, where θ_1 denotes the angle coordinate (which coincides with the angle of DoF 1), η denotes the radius coordinate, that is the distance between the origin to the projection of the end effector along the z-axis onto the $x - y$ plane of {I}, and ξ denotes the height, which coincides with the z-coordinate of {I}. The use of a polar coordinate system for describing and planning the motion of the end effector is motivated by the fact that the return velocity in tangential direction is given by $\dot{\theta}_1 \eta$, independent of where the ball is hit. We thus plan trajectories of the end effector in a polar coordinate system by minimizing jerk.

Minimum jerk trajectories are desirable for their amenability to path tracking and to limit robot vibration. They are also used in order to ensure continuity of velocities and accelerations (Piazzi & Visioli, 1997). This results in the following optimization problem, which is solved separately for each coordinate $\tau \in \{\theta_1, \eta, \xi\}$:

$$\min_{j:[0,T]\to\mathbb{R}} \int_0^T j(t)^2 + \alpha_{\tau_T}^6 \tau(t)^2 \mathrm{d}t, \tag{3}$$

$$\begin{aligned} \text{s.t.} \quad & j(t) = \dddot{\tau}(t), \\ & x_{\tau,T}^0 = (\tau(0), \dot{\tau}(0), \ddot{\tau}(0))^{\mathrm{T}}, \\ & x_{\tau,T}^{\mathrm{f}} = (\tau(T), \dot{\tau}(T), \ddot{\tau}(T))^{\mathrm{T}}, \end{aligned} \tag{4}$$

where $T \in \{T_1, T_2\}$.

We note that the term $\alpha_{\tau_T}^6 \tau(t)^2$ is added to penalize large motion ranges, which ensures that the physical constraints of our robot arm are not violated. The boundary conditions are set in such a way that the robot starts from ${}^{\mathrm{I}}p_{\mathrm{ini}}$ at rest, reaches ${}^{\mathrm{I}}p_{\mathrm{int}}$ at time T_1 with velocity $\dot{\eta}(T_1) = 0$, $\eta(T_1)\dot{\theta}_1(T_1) = 5\,\mathrm{m\,s^{-1}}$, $\dot{\xi}(T_1) = 0$ and returns to ${}^{\mathrm{I}}p_{\mathrm{ini}}$, where it arrives with zero velocity. In order to reduce impact, the initial and final accelerations of all coordinates are set to zero. We noticed that only θ_1 needs a penalty term to constrain the range of motion, while the rest of the coordinates are guaranteed to move within the physical constraints due to the boundary conditions. Therefore, in the experiment, $\alpha_{\theta_{1,T_1}}$ and $\alpha_{\theta_{1,T_2}}$ are set to one while the remaining α_{τ_T} are set to zero.

The minimum jerk problem (3) can be solved in closed form by applying Pontryagin's minimum principle (Geering, 2007). This results in the following co-state equation:

$$\dddot{\lambda}(t) = -2\alpha_{\tau_T}^6 \tau(t),$$

and the stationary condition of the associated Hamiltonian

$$2j(t) + \lambda(t) = 0, \quad \forall t \in [0, T],$$

with $T \in \{T_1, T_2\}$ and where λ denotes the co-state trajectory. Combining these equations results in the following boundary value problem

$$\tau^{(6)}(t) = \alpha_{\tau_T}^6 \tau(t),$$

subject to the boundary conditions listed in (4). This boundary value problem can be solved in closed form; for coordinates η and ξ, where the penalty term is set to zero, it is particularly straightforward and optimal η and ξ are given by fifth order polynomials; for θ_1, where $\alpha_{\theta_{1,T}} \neq 0$, the solution is more involved and includes exponential terms.

The important advantage of having closed-from solutions is that reference trajectories can be computed and generated extremely quickly. We will take advantage of this fact when performing interceptions and returns, where we will re-plan the reference trajectories in real time.

3.2 Label generation with ILC

We generate a data set of y^i_{des}, $i = 1, \ldots, N$ reference trajectories sampled from $p_{y_{\text{des}}}$ as described in the previous section. For each of these reference trajectories we will compute and learn an optimal input trajectory that minimizes tracking error by applying ILC.

3.2.1 Formulation

The ILC formulation is inspired by Hofer et al. (2019) and Schoellig et al. (2012). It assumes knowledge of a coarse model of the underlying dynamical system, which describes the output (the angles of DoF 1–3) in the following way:

$$y = \widetilde{F}\left(x_0, u, d + n_{\text{w}}\right),$$

where $u \in \mathbb{R}^{mq}$ is a sequence of inputs (actions) over a horizon of length q, $y \in \mathbb{R}^{lq}$ the corresponding outputs, $x_0 \in \mathbb{R}^n$ the initial condition, and $d + n_{\text{w}} \in \mathbb{R}^{lq}$ denotes the disturbances. The disturbance $d + n_{\text{w}}$ is separated into a repeatable part d (e.g. delays, friction and nonlinearity) and a non-repeatable part n_{w} (e.g. process noise). The disturbance d is in many cases implicitly dependent on the state and input of the robot. This dependence could, in principle, be arbitrarily complex (even non-smooth, Quintanilla and Wen (2007)). The disturbance d also contains interactions between the different degrees of freedom, which may not be fully captured in the nominal model $\widetilde{F}\left(x_0, u, 0\right)$. In particular, our model arises from a description via transfer functions that neglects the coupling between the degrees of freedom. It is given by:

$$\begin{aligned} y^i(z) = & \frac{\alpha^i_0 + \alpha^i_1 z^{-1} + \cdots + \alpha^i_{n^i_{\text{n}}} z^{-n^i_{\text{n}}}}{1 + \beta^i_1 z^{-1} + \cdots + \beta^i_{n^i_{\text{m}}} z^{-n^i_{\text{m}}}} z^{-n^i_{\text{d}}} u^i(z) \\ & + d^i(z) + n^i_{\text{w}}(z), \end{aligned} \tag{5}$$

where $y^i(z)$ denotes the z-transform of the angle of the i-th degree of freedom, $u^i(z)$ denotes the z-transform of the input of the i-th degree of freedom, and $d^i(z) + n^i_{\text{w}}(z)$ denotes the disturbance acting on the i-th degree of freedom, again separated in a repeatable and a non-repeatable part. The variable n^i_{n}, n^i_{m}, and n^i_{d} denote the order of the numerator, the order of the denominator and the delay of the i-th degree of freedom, with $i = 1, 2, 3$. All transfer functions are listed in "Appendix A" for completeness.

We note that the input is given by pressure differences that are sent to the low-level controller driving the PAMs. Both inputs and outputs are normalized such that they take the values zero when the robot is in its rest position x_0 (x_0 is an equilibrium). Thus as a result, the function $\widetilde{F}$ is linear and takes the form

$$y = A_0 x_0 + B_{\text{u}} u + B_{\text{d}}\left(d + n_{\text{w}}\right) + n_{\text{y}}, \tag{6}$$

where the matrices $A_0 \in \mathbb{R}^{lq \times n}$, $B_{\text{u}} \in \mathbb{R}^{lq \times mq}$, $B_{\text{d}} \in \mathbb{R}^{lq \times mq}$ are derived from (5) and are listed in "Appendix B". Here, $n_{\text{y}} \in \mathbb{R}^{lq}$ denotes the measurement noise.

The ILC aims at learning the repeatable disturbances d by applying the following principle:

1. We apply the input signal u to the system and record the angle trajectories of the degrees of freedom 1–3.
2. We update the estimate for the repeatable disturbances d in (6).
3. We update the input signal u and proceed with Step 1.

The repeatable disturbances d are learned with a Kalman filter, which is based on the following process equation

$$d^{k+1} = d^k + n^k_{\text{d}},\ n^k_{\text{d}} \sim \mathcal{N}\left(0, \Pi_{\text{d}}\right),\ d^0 \sim \mathcal{N}\left(0, \Pi\right),$$

and measurement equation

$$\underbrace{y^k - A_0 x_0 - B_{\text{u}} u^k}_{\text{measurement data}} = B_{\text{d}} d^k + n^k,\ n^k \sim \mathcal{N}\left(0, \Pi_{\text{mix}}\right),$$

where $(\cdot)^k$ denotes the number of ILC iterations. We use the following forms for the variance of n^k_{d}, the variance of d_0, and the variance of n^k:

$$\begin{aligned} \Pi_{\text{d}} &= \text{diag}\left\{\left(\sigma_{\text{d},i}\right)^2 I\right\}_{i=1}^{3}, \quad \Pi = \text{diag}\left\{\sigma_i^2 I\right\}_{i=1}^{3}, \\ \Pi_{\text{mix}} &= \text{diag}\left\{\left(\sigma_{\text{w},i}\right)^2 I\right\}_{i=1}^{3} B_{\text{d}} B_{\text{d}}^{\text{T}} + \text{diag}\left\{\left(\sigma_{\text{y},i}\right)^2 I\right\}_{i=1}^{3}, \end{aligned}$$

where $I \in \mathbb{R}^{q \times q}$ denotes the identity matrix and $\text{diag}\{\cdot\}$ refers to diagonal stacking. The concrete values of σ_i, $\sigma_{\text{d},i}$, $\sigma_{\text{w},i}$ and $\sigma_{\text{y},i}$ with $i = 1, 2, 3$ can be found in Table 1, and can be tuned in a principled manner. We note that if the model $\widetilde{F}$ was nonlinear we could apply the same approach with the difference that the measurement equation would be linearized in d about the mean estimate $\hat{d}^{k-1}$ from the previous iteration.

Table 1 This table lists the variance parameters for the ILC

	Vairance			
	$\sigma^2_{\mathrm{d},i}$	$\sigma^2_{\mathrm{y},i}$	$\sigma^2_{\mathrm{w},i}$	σ^2_i
DoF 1	10^{-9}	10^{-3}	10^{-9}	10^{-7}
DoF 2	10^{-11}	10^{-3}	10^{-11}	10^{-7}
DoF 3	10^{-8}	10^{-3}	10^{-8}	10^{-7}

We then use the mean value of the Kalman filter estimate, denoted by $\hat{d}^k$, at the k-th iteration to update the feedforward input u^{k+1} for the next iteration. More precisely, we update u in the following way:

$$u^{k+1} = \arg\min_u \frac{1}{2} \left| y_{\mathrm{des}} - A_0 x_0 - B_{\mathrm{u}} u - B_{\mathrm{d}} \hat{d}^k \right|^2 ,$$

where $y_{\mathrm{des}} \in \mathbb{R}^{lq}$ denotes the reference trajectory. Although there are pressure constraints on the input of the robot arm, we do not consider these when solving the above optimization problem. This leads to the following closed-form solution

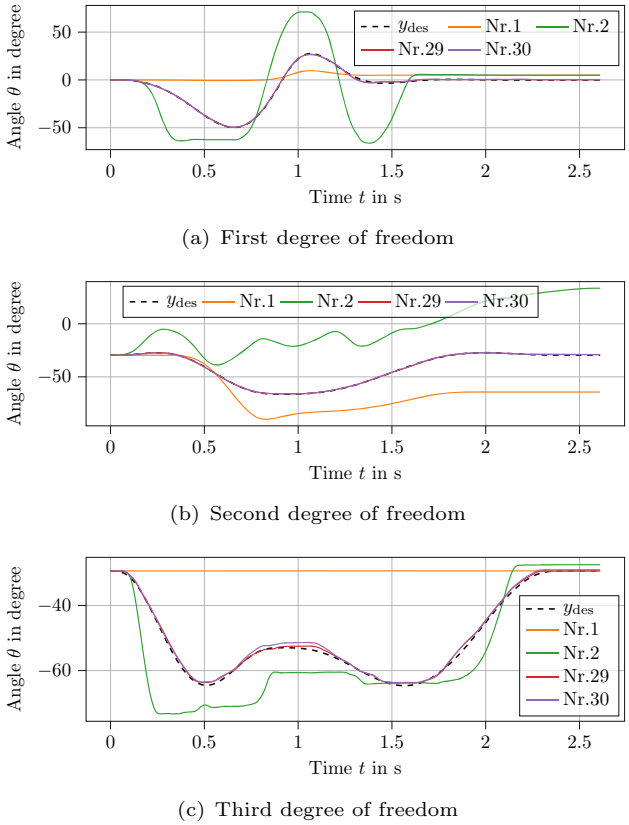
$$u^{k+1} = B_{\mathrm{u}}^{\dagger} \left(y_{\mathrm{des}} - A_0 x_0 - B_{\mathrm{d}} \hat{d}^k \right), \tag{7}$$

where $(\cdot)^{\dagger}$ denotes the Moore-Penrose pseudo inverse.

3.2.2 Learning results

In this section, we will show the learning results of ILC when tracking trajectories that are sampled from $p_{y_{\mathrm{des}}}$. We conduct 30 learning iterations for each sample, and the results for DoF 1–3 are shown in Fig. 3. It is worth mentioning that the input for the first iteration is directly generated by solving (7) with $\hat{d}^0 = 0$. The inputs for the last iteration are assumed to be a good approximation of $u^\star(y_{\mathrm{des}})$ as defined in (2) (we note the almost perfect tracking of the reference trajectory after 30 iterations shown in Fig. 3). As can be seen from the figure, our model $\widetilde{F}(x_0, y, 0)$ is a very coarse approximation of the underlying dynamics, resulting in a relatively poor tracking performance during the first couple of iterations. Nonetheless the model is very effective at providing gradient information, which the ILC can leverage, resulting in rapid improvement of the tracking error in the first several iterations. We also note that the learning converges, as the last two iterations remain almost the same. Considering the high nonlinearity of PAMs, and the fact that ILC only learns feedforward inputs, we conclude that ILC is very effective at solving (2).

We sample 43 different reference trajectories from $p_{y_{\mathrm{des}}}$, as described in Sect. 3.1. For each of these reference trajectories we apply 30 ILC iterations and store the resulting feedforward inputs $u^\star\left(y^i_{\mathrm{des}}\right), i = 1 \ldots, 43$. This provides us with the data and the labels to train a machine learning model as described in the next section.



(a) First degree of freedom

(b) Second degree of freedom

(c) Third degree of freedom

Fig. 3 The figure shows the learning results of ILC. The dashed line is the fixed reference trajectory and the solid lines are the results of the first and last two iterations

3.3 Generalization with parameterized policy network

In this section, we demonstrate how to transform the original interception task into a supervised learning problem. We will use the data and labels obtained in the previous section to train a CNN to approximate the optimal policy $\pi^\star_{\mathrm{ff}}$, thereby generalizing the ILC results to all $y_{\mathrm{des}} \sim p_{y_{\mathrm{des}}}$. The 43 trajectories obtained from Sect. 3.2.2 are divided into 30 trajectories for training and 13 trajectories for validation. To speed up the convergence of the neural network and improve accuracy, all inputs are normalized.

When intercepting the balls that are played to the robot arm, inference is done in real time. This motivates us to use a CNN instead of fully connected layers or recurrent neural networks. Moreover, the CNN is also found to be beneficial for handling the coupling between the various degrees of freedom and the temporal correlations. Our CNN has the same architecture for each degree of freedom. We will denote by π_i the CNN for degree of freedom i, which has three channels and maps $\mathbb{R}^{3\times 3\times(2h+1)} \to \mathbb{R}$, where the output approximates $u^\star(y_{\mathrm{des}})$ at time point k for degree of freedom i. The first channel takes a window of y_{des} of length $2h+1$

as input, that is, $\left(y_{\text{des},k-h}, \ldots, y_{\text{des},k}, \ldots, y_{\text{des},k+h}\right)$, whereas the second and third channels take the velocity and acceleration of y_{des} (again over the same window of size $2h+1$) as input. Both velocity and acceleration are computed with finite differences. In principle, the addition of the second and third channels is unnecessary, but we find in practice that this speeds up training and improves training and validation losses. The addition of velocity and acceleration components can be viewed as prior knowledge that is incorporated in the structure of the CNN. We believe that this is advantageous in situations where the size of the training data set is limited (30 trajectories). A slightly different type of prior knowledge is included and discussed in Ma et al. (2022). Each π_i is designed with a simple structure, characterized by a low number of parameters and a shallow architecture with few layers. The simplicity of the architecture reduces the risk of overfitting by limiting the model's capacity to memorize the training data. The architecture consists of six convolutional layers and four fully connected layers. The convolutional layers do not contain any pooling layers, and the fully connected layers do not have a dropout. We use `ReLU` as an activation function for all the layers, except the output layer. Empirical studies show that `ReLU` mitigates the risk of gradient disappearance or explosion. We note that each π_i incorporates the behavior and coupling of all three degrees of freedom.

We train each $\pi_i, i = 1, 2, 3$ on the training data set, and substitute the resulting machine learning models as the feedforward block in Fig. 2b. We apply the resulting control loop to the robot arm and evaluate the tracking performance on both training and validation data sets. The tracking error of the end effector, which we will use as our performance metric, is defined as follows:

$$\delta_i = \frac{1}{q_i} \sum_{k=0}^{q_i-1} \left| {}^{\mathrm{I}}p_i^{\star}(k) - {}^{\mathrm{I}}p_i(k) \right|, \tag{8}$$

where ${}^{\mathrm{I}}p_i^{\star}(k)$ denotes the i-th reference trajectory and ${}^{\mathrm{I}}p_i(k)$ the actual trajectory at time point k, where $i = 1, \ldots, 43$. We note that reference trajectories have sightly different lengths, which is described by the variable q_i.

The values of δ_i for all the trajectories are shown in Fig. 4. As shown in the figure, when relying only on feedforward control, the tracking accuracy of the neural network is much worse than ILC even though the neural network generalizes well (there is almost no performance difference between training and validation trajectories). The results from our two-degrees-of-freedom control loop (as shown in Fig. 2b) are also included and labeled LBIC (learning-based iterative control). The final tracking accuracy is as good as ILC, reaching an average error of under 0.02 m.

In Fig. 5 we compare the tracking results of the LBIC framework and ILC on the same reference trajectory as shown in Fig. 3 from the validation set. As can be seen from the figure, the generalization results of the LBIC framework are very close to the tracking results of ILC. It is also worth noting that the average root-mean squared tracking error over all ball trajectories obtained by the LBIC framework is almost as good as the learning results of ILC in both the training and validation data sets, see Fig. 4. This indicates that our LBIC framework successfully generalizes the results from ILC to reference trajectories sampled from $p_{y_{\text{des}}}$.

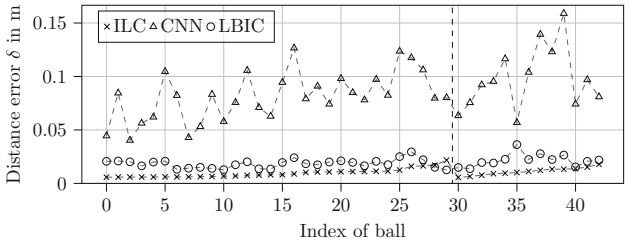
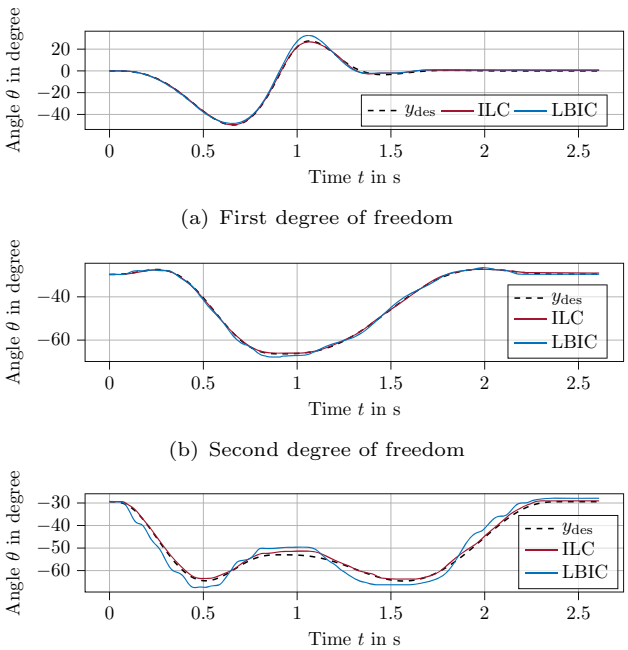


Fig. 4 The figure shows the tracking error in the global coordinate system {I} of all reference trajectories. The left side of the dashed line represents trajectories in the training set, whereas the right side corresponds to trajectories in the validation set. The index (x-axis) is sorted in ascending order according to the results of ILC on the training data set and validation data set, respectively



(a) First degree of freedom

(b) Second degree of freedom

(c) Third degree of freedom

Fig. 5 The figure shows the results of the LBIC framework and the last iteration of the ILC. The fixed reference trajectories y_{des} are shown with dashed curves. The results of the last iteration y obtained by the different methods are shown in solid curves. "ILC" denotes the results of the ILC method and "LBIC" our reinforcement learning framework. This reference trajectory is from the validation data set. We note that the movement range is large ($-50°$ in the first degree of freedom, $-65°$ in the second degree of freedom, and $-65°$ in the third degree of freedom) and that the motion is dynamic reaching $5\,\mathrm{m\,s^{-1}}$ at the interception point

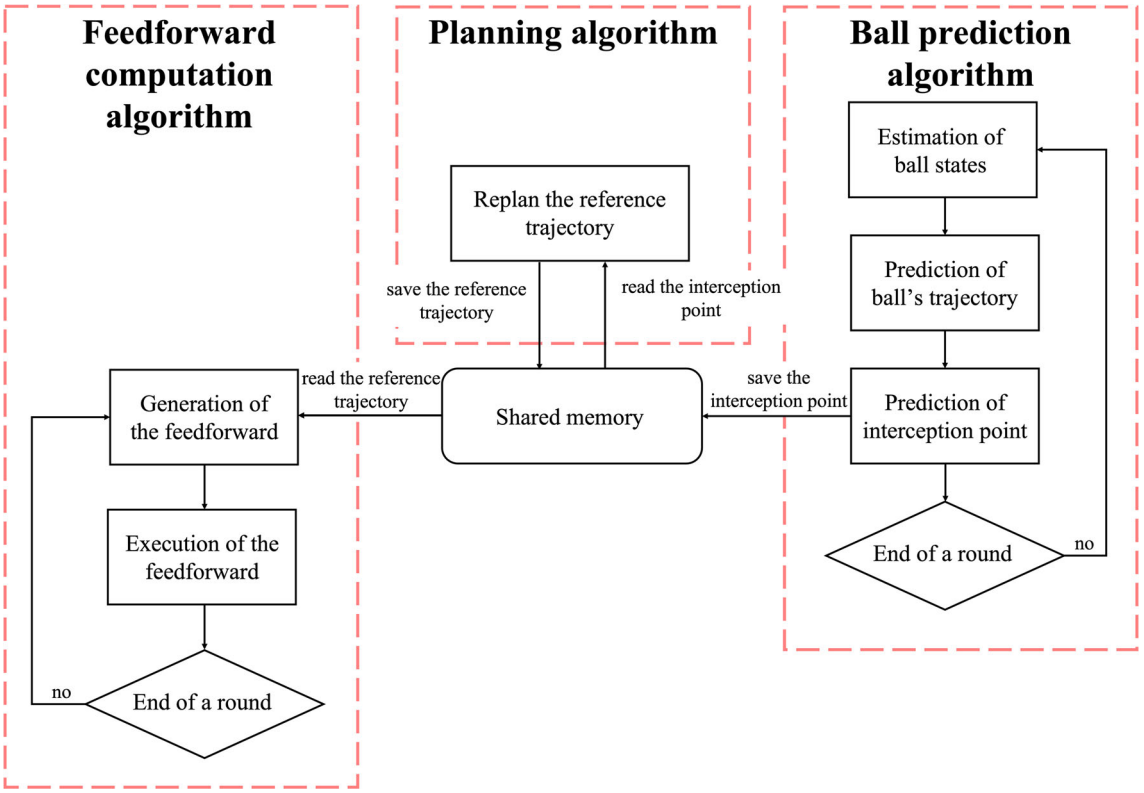


Fig. 6 The figure shows the interception control loop used to intercept the table tennis ball in real time. The interception control loop consists of three parts that run independently at different frequencies, and exchange data through shared memory

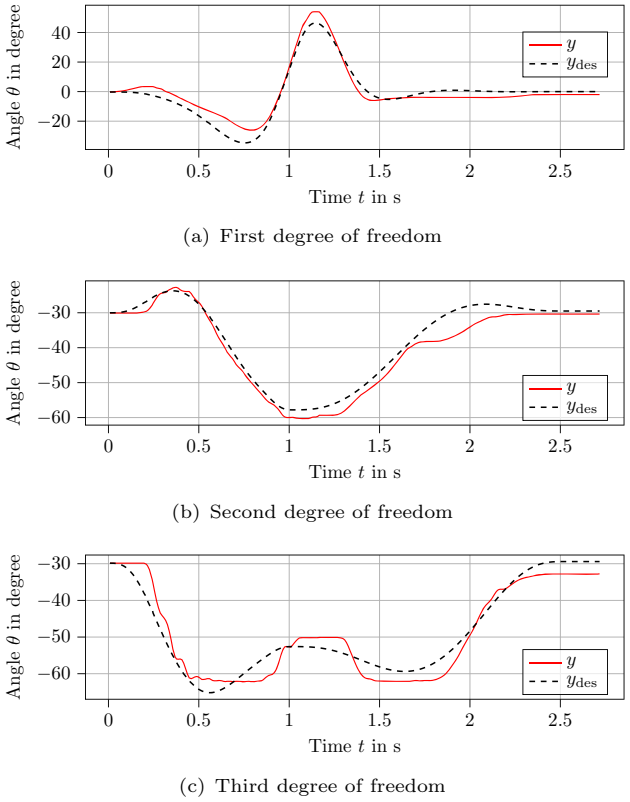
3.4 Online planning for intercepting ping-pong balls

So far, we have successfully used our reinforcement learning framework to achieve high-precision tracking of trajectories sampled from $p_{y_{\text{des}}}$. However, there is still a step missing to reach our ultimate goal of intercepting and returning table tennis balls. Here, we propose the interception control loop as shown in Fig. 6. We used the Python interface developed by Berenz et al. (2021) for controlling the robot arm, and we used the Pyhton package `SharedArray` to implement the shared memory.

As can be seen from the figure, our interception control loop consists of three parts, the feedforward computation algorithm, the planning algorithm and the ball prediction algorithm. These three parts operate independently at different frequencies, and exchange data through shared memory. A ball leaving the launcher is considered to be the start of the round, while a successful interception or a miss is considered to be the end of the round.

The ball prediction algorithm runs at 60 Hz and works as follows: While the table tennis ball is flying towards the robot arm, the vision system (Gomez-Gonzalez et al., 2019) tracks the table tennis ball through four `RGB` cameras hanging from the ceiling. The vision system returns measurements of the table tennis ball, which are processed with our EKF that estimates the state $\zeta^{\mathrm{T}} = \left({}^{\mathrm{I}}p^{\mathrm{T}}, {}^{\mathrm{I}}v^{\mathrm{T}}\right)$ of the table tennis ball, where ${}^{\mathrm{I}}p \in \mathbb{R}^3$ denotes the position and ${}^{\mathrm{I}}v \in \mathbb{R}^3$ the velocity. It is worth mentioning that in our experiments the influence of the ball's angular velocity is negligible, and therefore, not included in our model. A more detailed study about the influence of spin can be found in Achterhold et al. (2023).

After getting an estimate of the state ζ of the table tennis ball at time point k, we can predict the remaining part of the ball's trajectory using the ball's motion model (Zhao et al., 2017; Glover & Kaelbling, 2014). However, according to the rules of table tennis, we must intercept the ball after the ball collides with the table. In order to predict the ball's trajectory after the impact, an accurate impact model is required. We use a data-driven approach to obtain this model, and collect 120 table tennis ball trajectories, which include an impact with the table. We assume that the state ζ right before and after the impact are ζ^- and ζ^+, respectively, where the position remains unchanged, $p^+ = p^-$. We use the collected data to derive a linear impact model $\Gamma \in \mathbb{R}^{3\times3}$, $v^+ = \Gamma v^-$, by solving a least-squares problem. By combining the estimate of the state ζ, the ball's motion model and ball's impact model Γ, we can therefore predict its trajectory and calculate the interception point ${}^{\mathrm{I}}p_{\text{int}}$ as defined in Sect. 3.1. The interception point will be saved in the shared memory, where we overwrite ${}^{\mathrm{I}}p_{\text{int}}$ from the previous iteration.



(a) First degree of freedom

(b) Second degree of freedom

(c) Third degree of freedom

Fig. 7 The figure shows the tracking performance of our reinforcement learning framework in joint space for an interception round in real time. The figure shows the tracking performance of the first, second and third degree of freedom of the robot arm from top to bottom. The reference trajectory as computed and updated by the planning algorithm is shown with the dashed line, while the actual trajectory is shown with a solid line

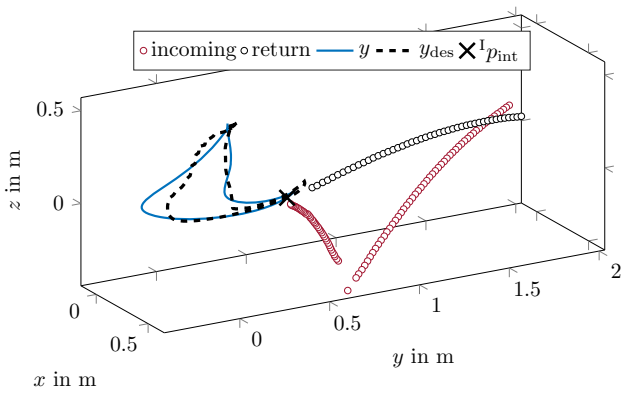


Fig. 8 The figure shows the an interception round in the global coordinate system {I}. The positions of the ping-pong ball before and after the interception are indicated by red and black circles, respectively. Sometimes the vision system is not able to correctly identify the ping-pong ball (due to occlusion), therefore some parts of the ball trajectory are missing, which also increases the difficulty of ball's trajectory prediction. The reference trajectory y_{des} and the actual trajectory y of the end effector are given by the black dashed line and the blue solid line, respectively. The final interception point ${}^{\text{I}}p_{\text{int}}$ is marked as a cross on the planned trajectory

The planning algorithm runs at 30 Hz. The algorithm will first read the latest interception point from the shared memory. If the interception point has been updated, the planning algorithm will re-plan the trajectory as described in Sect. 3.1, with the sole difference that the trajectory's starting point is changed from ${}^{\text{I}}p_{\text{ini}}$ to the planned position (according to the reference trajectory from the previous iteration) at the current time point. The updated trajectory will be saved in shared memory and overwrite the data from the previous iteration.

The feedforward computation algorithm runs at 10 Hz. The algorithm will read the current reference trajectory from the shared memory, and then evaluates the functions $\pi_i, i = 1, 2, 3$ and computes feedforward inputs $u^\star(y_{\text{des}})$ for the given reference trajectory (see Sect. 3.3). The resulting $u^\star(y_{\text{des}})$ are written to the shared memory, where they are read by the underlying low-level controller. This low-level control loop runs at 100 Hz and combines the nonlinear feedforward with the feedback from a PID controller (see Fig. 2b).

Figure 7 shows the tracking performance of an interception round in the joint space. We note that the tracking error increases compared to the static experiments presented in Sect. 3.2.2, which is due to the re-planning algorithm described in the previous paragraphs. The third degree of freedom has the largest tracking error, which can be attributed to its high nonlinearity (it has the largest friction due to the rope-pulley mechanism that connects the joints with the PAMs).

Figure 8 shows an interception round in the global coordinate system {I}. We notice that the tracking error in the initial phase of the trajectory is relatively high. We believe that this is due to the poor prediction of the interception point ${}^{\text{I}}p_{\text{int}}$ at the beginning, which leads to large differences in y_{des} when the trajectory is re-planned during the first several iterations. As a result, the prediction accuracy of the model π_i also decreases compared with the static experiments. We find that as the ball flies towards the arm, the prediction error of ${}^{\text{I}}p_{\text{int}}$ converges to zero, and the tracking error also decreases, in particular after the ball has impacted the table.

In Fig. 9, we show the distance error δ of 60 interceptions, and the results are sorted in ascending order. The average accuracy is about 0.07 m. Compared with the static experiments, there is a 0.05 m increase in average accuracy, which is, however, still sufficient for intercepting table tennis balls. In all 60 experiments, the robot managed to successfully intercept the ball.

4 Discussion and context

The following section provides additional context to our reinforcement learning framework and discusses data efficiency,

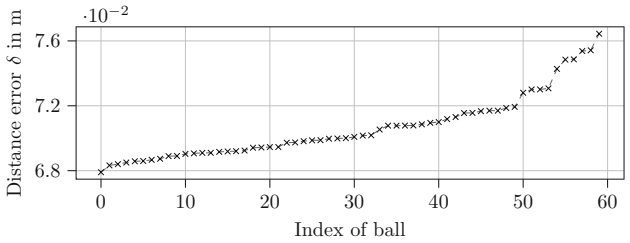


Fig. 9 The figure shows the tracking error of the end effector of the robot arm in the Euclidean space for 60 interceptions. The results are sorted in ascending order

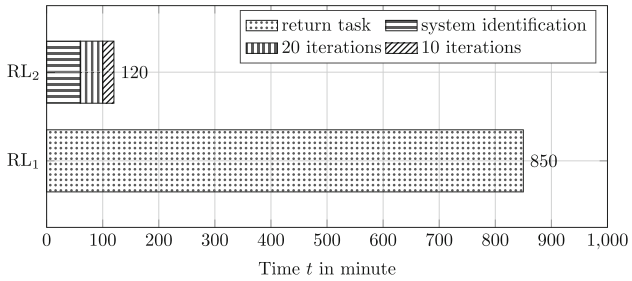


Fig. 10 The figure shows the comparison of two reinforcement learning algorithms in terms of sampling efficiency. Our method is denoted by RL_2. After learning for 20 iterations, we already obtain a reasonable estimate of $u^\star(y_{\text{des}})$. However, we perform ten more iterations for each trajectory to improve accuracy further

modularity and stability of the resulting control loop shown in Fig. 2b.

Data efficiency: As we discussed in Sect. 1, one of the drawbacks of many reinforcement learning algorithms is their high sample complexity resulting in training procedures that sometimes take weeks or even months. In our approach, the main part of the learning is done by the ILC. Due to the fact that an approximate model of the underlying system is incorporated, the ILC converges relatively quickly. We run the ILC for 30 iterations, however, the tracking performance improves only slightly over the last ten iterations. The observation that ILC is very data efficient is in line with numerous prior works (see Sect. 1). As a reference, we compare the sample complexity of our approach to Büchler et al. (2020), which optimizes over feedback policies. In Fig. 10 we visually show the comparison of the data efficiency of the two methods. The method in Büchler et al. (2020) was trained for about 14 hours for an interception task. In our experiments, we require about an hour for conducting an extended system identification that derives a non-parametric frequency response function and quantifies the nonlinearities, see Ma et al. (2022). The model $\widetilde{F}(x_0, u, 0)$, which is used as a starting point for the ILC, is obtained via a parametric fit through the non-parametric frequency response function. However, if the model structure is fixed, only about 15 min of data would be enough for identifying the parameters in (5). As discussed in Sect. 3.1 the trajectories y_{des} have about 2.5 s length. We require about 1.5 s extra to return to the initial configuration, which results in about 4 s for executing an ILC iteration. Our data set for training contains 30 trajectories sampled from $p_{y_{\text{des}}}$, which mean that the ILC takes 40 min to execute 20 iterations and 60 min to execute 30 iterations. We find that running ILC for 20 iterations is in principle enough for obtaining a reasonable approximation of $u^\star(y_{\text{des}})$. This means that in total our approach requires only about $1.5 - 2$ hours, which includes the conservative estimate of the time spent on the system identification. Alternatively $\widetilde{F}(x_0, u, 0)$ could also be modeled from first principles. It is important to note, however, that Büchler et al. (2020) did not focus on data efficiency at all. The training could be made more efficient and stopped at an earlier stage. We would like to highlight that while this approach proves to be feasible and beneficial for the trajectory tracking task addressed in this paper, its applicability and advantages may vary for different learning tasks.

Modularity: Compared to black-box approaches, our framework is much more modular. While this enables separate tuning of the different components in a principle manner, it also requires engineering skills and insights into the robotic platform (such as tuning the Kalman filter for the ILC or the estimation of the ball's state, defining the interception point ${}^{\text{I}}p_{\text{int}}$, tuning the architecture of the CNN). An important advantage, compared to a more black-box end-to-end learning approach is, however, that the individual components can be debugged separately.

Stability of the closed-loop: We note that our learning framework only optimizes over feedforward inputs. This departs from the Hamilton–Jacobi–Bellman perspective, which is prevalent in reinforcement learning. While feedforward cannot attenuate noise and disturbances, there is also very little risk that the system is destabilized during training. These observations even apply when running the interception control loop described in Sect. 3.4. Although we continuously re-plan feedforward commands, the reference trajectories do not depend on the current state of the robot.

5 Conclusion

In summary, we propose a new reinforcement learning framework for complex dynamic tasks in robotics. The framework transforms reinforcement learning into a supervised learning task. Important advantages include data efficiency, modularity and the fact that prior knowledge can be included to speed up learning.

We apply our framework to perform a trajectory tracking task with a robot arm driven by pneumatic artificial muscles. We use our framework to intercept and return ping-pong balls that are played to the robot arm and achieve an interception rate of 100% on more than 107 consecutive tries.

While this article is focused on an offline method to train the policy network, we believe that a fruitful and interesting avenue for future work would be to design online learning methods. Moreover, the results from this article form the basis for developing interception policies that return incoming balls to any predefined target on the table. This could then also enable two robots to play table tennis with each other.

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1007/s10514-023-10140-6.

Acknowledgements Hao Ma and Michael Muehlebach thank the German Research Foundation, the Branco Weiss Fellowship, administered by ETH Zurich, and the Center for Learning Systems for the support.

Author Contributions Hao Ma did the experimental work and prepared the first version of the manuscript. Hao Ma and Michael Muehlebach then improved the manuscript with feedback from Dieter Büchler. Bernhard Schölkopf funded the experimental setup.

Funding Open Access funding enabled and organized by Projekt DEAL. This work received funding from the German Research Foundation, the Branco Weiss Fellowship, administered by ETH Zurich, and the Center for Learning Systems.

Availability of data and materials The robotics platform is made from off-the-shelf components, as described in earlier works. The article contains all the necessary material to reproduce the results.

Declarations

Competing Interests The authors have no competing interests to declare that are relevant to the content of this article.

Code availability Important parts of our source code are available here: https://github.com/intelligent-soft-robots. The remaining parts can be reproduced from the article.

Appendix A: Transfer functions

The deterministic part of transfer functions for all degrees of freedom are shown as follows:

$$y^1(z) = \frac{\left(-0.27 + 0.39z^{-1} - 0.13z^{-2}\right) \cdot 10^{-5}}{1 - 2.82z^{-1} + 2.67z^{-2} - 0.84z^{-3}} u^1(z),$$

$$y^2(z) = \frac{\left(-0.26 + 0.25z^{-1}\right) \cdot 10^{-6}}{1 - 2.85z^{-1} + 2.71z^{-2} - 0.86z^{-3}} u^2(z),$$

$$y^3(z) = \frac{\left(-0.52 + 0.97z^{-1} - 0.46z^{-2}\right) \cdot 10^{-5}}{1 - 2.91z^{-1} + 2.84z^{-2} - 0.93z^{-3}} u^3(z).$$

Appendix B: State space matrices

The parametric model (5) for each degree of freedom i at time point k can be rewritten in the state-space as follows:

$$y^i(k) = \left(c^i\right)^{\mathrm{T}} A^i x^i(k) + \left(c^i\right)^{\mathrm{T}} b_{\mathrm{u}}^i u^i(k) + \left(c^i\right)^{\mathrm{T}} b_{\mathrm{d}}^i \left(d^i(k) + n_{\mathrm{w}}^i(k)\right),$$

where $c^i, x^i(k), b_{\mathrm{d}}^i, b_{\mathrm{u}}^i \in \mathbb{R}^{n_{\mathrm{n}}^i + n_{\mathrm{m}}^i}$, and $A^i \in \mathbb{R}^{\left(n_{\mathrm{n}}^i + n_{\mathrm{m}}^i\right) \times \left(n_{\mathrm{n}}^i + n_{\mathrm{m}}^i\right)}$. Thus, the dimension of the state is defined as $n = \sum_{i=1}^{3} \left(n_{\mathrm{n}}^i + n_{\mathrm{m}}^i\right)$. We note that both $y^i(k)$ and $u^i(k)$ are scalar, since it is a single-input single-output system for each degree of freedom. Next, the whole trajectory can be represented as follows:

$$y = A_0 x_0 + B_{\mathrm{u}} u + B_{\mathrm{d}} \left(d + n_{\mathrm{w}}\right),$$

where

$$(A_0)^{\mathrm{T}} = \left(A^{\mathrm{T}} c, \ \cdots, \ (A^q)^{\mathrm{T}} c\right),$$

$$B_{\mathrm{u}} = \begin{pmatrix} c^{\mathrm{T}} b_{\mathrm{u}} & & & \\ c^{\mathrm{T}} A b_{\mathrm{u}} & c^{\mathrm{T}} b_{\mathrm{u}} & & \\ \vdots & \vdots & \ddots & \\ c^{\mathrm{T}} A^{q-2} b_{\mathrm{u}} & c^{\mathrm{T}} A^{q-3} b_{\mathrm{u}} & \cdots & c^{\mathrm{T}} b_{\mathrm{u}} \\ c^{\mathrm{T}} A^{q-1} b_{\mathrm{u}} & c^{\mathrm{T}} A^{q-2} b_{\mathrm{u}} & \cdots & \cdots \quad c^{\mathrm{T}} b_{\mathrm{u}} \end{pmatrix},$$

$$B_{\mathrm{d}} = \begin{pmatrix} c^{\mathrm{T}} b_{\mathrm{d}} & & & \\ c^{\mathrm{T}} A b_{\mathrm{d}} & c^{\mathrm{T}} b_{\mathrm{d}} & & \\ \vdots & \vdots & \ddots & \\ c^{\mathrm{T}} A^{q-2} b_{\mathrm{d}} & c^{\mathrm{T}} A^{q-3} b_{\mathrm{d}} & \cdots & c^{\mathrm{T}} b_{\mathrm{d}} \\ c^{\mathrm{T}} A^{q-1} b_{\mathrm{d}} & c^{\mathrm{T}} A^{q-2} b_{\mathrm{d}} & \cdots & \cdots \quad c^{\mathrm{T}} b_{\mathrm{d}} \end{pmatrix},$$

and for notational convenience, we omitted all superscripts $(\cdot)^i$, which represent the degree of freedom. Finally, the matrices in (6) can be represented as follows:

$$A_0 = \operatorname{diag}\left\{A_0^i\right\}_{i=1}^3, \quad B_{\mathrm{u}} = \operatorname{diag}\left\{B_{\mathrm{u}}^i\right\}_{i=1}^3,$$

$$B_{\mathrm{d}} = \operatorname{diag}\left\{B_{\mathrm{d}}^i\right\}_{i=1}^3, \quad x_0^{\mathrm{T}} = \left(\left(x_0^1\right)^{\mathrm{T}}, \left(x_0^2\right)^{\mathrm{T}}, \left(x_0^3\right)^{\mathrm{T}}\right).$$

References

Achterhold, J., Tobuschat, P., Ma, H., et al. (2023). Black-box vs. gray-box: A case study on learning table tennis ball trajectory prediction with spin and impacts. In: *Proceedings of the learning for dynamics and control conference*, pp. 878–890.

Ahn, H. S., Chen, Y., & Moore, K. L. (2007). Iterative learning control: Brief survey and categorization. *IEEE Transactions on Systems, Man, and Cybernetics, Part C, 37*(6), 1099–1121.

Alet, F., Lozano-Perez, T., & Kaelbling, L. P. (2018). Modular meta-learning. In: *Proceedings of the conference on robot learning*, pp. 856–868

Arimoto, S., Kawamura, S., Miyazaki, F. (1986). Convergence, stability and robustness of learning control schemes for robot manipulators. In: *Proceedings of the international symposium on robot manipulators on recent trends in robotics: Modeling, control and education*, pp. 307–316

Ba, D. X., Dinh, T. Q., & Ahn, K. K. (2016). An integrated intelligent nonlinear control method for a pneumatic artificial muscle. *IEEE/ASME Transactions on Mechatronics, 21*(4), 1835–1845.

Bagnell, J., Chestnutt, J., Bradley, D., et al. (2006). Boosting structured prediction for imitation learning. In: *Proceedings of the advances in neural information processing systems*, pp. 1–8

Berenz, V., Naveau, M., Widmaier, F., et al. (2021). The o80 c++ templated toolbox: Designing customized python APIs for synchronizing realtime processes. *Journal of Open Source Software, 6*(66), 2752–2755.

Bertsekas, D. (2012). Dynamic programming and optimal control: Volume I. Athena Scientific

Bertsekas, D. (2019). *Reinforcement learning and optimal control*. Athena Scientific.

Boedecker, J., Springenberg, J. T., Wülfing, J., et al. (2014). Approximate real-time optimal control based on sparse gaussian process models. In *IEEE symposium on adaptive dynamic programming and reinforcement learning* pp. 1–8.

Bristow, D., Tharayil, M., & Alleyne, A. (2006). A survey of iterative learning control. *IEEE Control Systems Magazine, 26*(3), 96–114.

Brunner, G., Richter, O., Wang, Y., et al. (2018). Teaching a machine to read maps with deep reinforcement learning. In: *Proceedings of the AAAI conference on artificial intelligence*, pp. 2763–2770

Büchler, D., Guist, S., Calandra, R., et al. (2020). Learning to play table tennis from scratch using muscular robots. pp. 1–11. arXiv:2006.05935v1

Büchler, D., Calandra, R., & Peters, J. (2023). Learning to control highly accelerated ballistic movements on muscular robots. *Robotics and Autonomous Systems, 159*, 104230–104241.

Cao, Q., Lin, L., Shi, Y., et al. (2017). Attention-aware face hallucination via deep reinforcement learning. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 690–698

Chen, Z., Liang, X., & Zheng, M. (2021). Deep iterative learning control for quadrotor's trajectory tracking. In: *Proceedings of the American control conference*, pp. 1408–1413

Cheng, R., Orosz, G., Murray, R. M., et al. (2019). End-to-end safe reinforcement learning through barrier functions for safety-critical continuous control tasks. pp 1–11. arxiv:1903.08792v1

Deisenroth, M. P., Englert, P., Peters, J., et al. (2014). Multi-task policy search for robotics. In: *Proceedings of the IEEE international conference on robotics and automation*, pp. 3876–3881

Devrim Kaba, M., Gokhan Uzunbas, M., Nam Lim, S. (2017). A reinforcement learning approach to the view planning problem. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 6933–6941

Dittrich, A., Schneider, J., Guist, S., et al. (2022). AIMY: An open-source table tennis ball launcher for versatile and high-fidelity trajectory generation. pp 1–14. arXiv:2210.06048v2

Dosovitskiy, A., Ros, G., Codevilla, F., et al. (2017). CARLA: An open urban driving simulator. pp 1–16. arXiv:1711.03938v1

Doyle, J. (1978). Guaranteed margins for LQG regulators. *IEEE Transactions on Automatic Control, 23*(4), 756–757.

Fathinezhad, F., Derhami, V., & Rezaeian, M. (2016). Supervised fuzzy reinforcement learning for robot navigation. *Applied Soft Computing, 40*, 33–41.

Finn, C., Yu, T., Fu, J., et al. (2017). Generalizing skills with semi-supervised reinforcement learning, pp. 1–11. arXiv:1612.00429v2

Firoiu, V., Whitney, W. F., & Tenenbaum, J. B. (2017). Beating the world's best at super smash bros with deep reinforcement learning, pp. 1–7. arXiv:1702.06230v3

Furrer, F., Burri, M., Achtelik, M., et al. (2016). RotorS—A modular Gazebo MAV simulator framework. In: *Proceedings of the robot operating system: The complete reference*, pp. 595–625

Ganguly, S., Garg, A., Pasricha, A., et al. (2012). Control of pneumatic artificial muscle system through experimental modelling. *Mechatronics, 22*(8), 1135–1147.

Garcia, J., & Fernandez, F. (2015). A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research, 16*, 1437–1480.

Geering, H. P. (2007). *Optimal control with engineering applications*. Springer.

Glover, J., & Kaelbling, L. P. (2014). Tracking the spin on a ping pong ball with the quaternion bingham filter. In: *Proceedings of the IEEE international conference on robotics and automation*, pp. 4133–4140.

Gomez-Gonzalez, S., Nemmour, Y., Schölkopf, B., et al. (2019). Reliable real-time ball tracking for robot table tennis. *Robotics, 8*(4), 90–102.

Grzes, M. (2017). Reward shaping in episodic reinforcement learning. In: *Proceedings of the international conference on autonomous agents and multiagent systems*, pp. 565–573.

Heess, N., Tb, D., Sriram, S., et al. (2017). Emergence of locomotion behaviours in rich environments, pp. 1–14. arXiv:1707.02286v2

Hofer, M., & D'Andrea, R. (2018). Design, modeling and control of a soft robotic arm. In: *Proceedings of the IEEE/RSJ international conference on intelligent robots and systems*, pp. 1456–1463

Hofer, M., Spannagl, L., & D'Andrea, R. (2019). Iterative learning control for fast and accurate position tracking with an articulated soft robotic arm. In: *Proceedings of the IEEE/RSJ international conference on intelligent robots and systems*, pp. 6602–6607

Jian, Y., Huang, D., Liu, J., et al. (2019). High-precision tracking of piezoelectric actuator using iterative learning control and direct inverse compensation of hysteresis. *IEEE Transactions on Industrial Electronics, 66*(1), 368–377.

Kalashnikov, D., Irpan, A., Pastor, P., et al. (2018). QT-Opt: Scalable deep reinforcement learning for vision-based robotic manipulation, pp. 1–23. arXiv:1806.10293v3

Kansky, K., Silver, T., Mély, D. A., et al. (2017). Schema networks: Zero-shot transfer with a generative causal model of intuitive physics. In: *Proceedings of the international conference on machine learning*, pp. 1809–1818

Kaushik, R., Anne, T., Mouret, J. B. (2020). Fast online adaptation in robotics through meta-learning embeddings of simulated priors. In: *Proceedings of IEEE/RSJ international conference on intelligent robots and systems*, pp. 5269–5276.

Kober, J. J., Andrew, B., & Peters, J. (2013). Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research, 32*(11), 1238–1274.

Kogiso, K., Sawano, K., Itto, T., et al. (2012). Identification procedure for mckibben pneumatic artificial muscle systems. In *Proceedings*

of the IEEE/RSJ international conference on intelligent robots and systems, pp. 3714–3721

Konyushkova, K., Zolna, K., Aytar, Y., et al. (2020). Semi-supervised reward learning for offline reinforcement learning. pp. 1–12. arXiv:2012.06899v1

Kupcsik, A., Deisenroth, M. P., Peters, J., et al. (2017). Model-based contextual policy search for data-efficient generalization of robot skills. *Artificial Intelligence, 247*, 415–439.

Lake, B. M., Ullman, T. D., Tenenbaum, J. B., et al. (2016). Building machines that learn and think like people. pp. 1–58. arXiv:1604.00289v3

Laskin, M., Lee, K., Stooke, A., et al. (2020). Reinforcement Learning with Augmented Data. In: *Proceedings of the advances in neural information processing systems*, pp. 19884–19895

Laud, A. D. (2004). Theory and application of reward shaping in reinforcement learning. PhD thesis, University of Illinois at Urbana-Champaign

Levine, S., & Koltun, V. (2014). Learning complex neural network policies with trajectory optimization. In: *Proceedings of the international conference on machine learning*, pp. 829–837

Liu, F., Li, S., Zhang, L., et al. (2017). 3DCNN-DQN-RNN: A deep reinforcement learning framework for semantic parsing of large-scale 3D point clouds. In: *Proceedings of the IEEE international conference on computer vision*, pp. 5678–5687

Luo, F. M., Xu, T., Lai, H., et al. (2022). A survey on model-based reinforcement learning, pp. 1–28. arXiv:2206.09328v1

Luo, J., & Hauser, K. (2017). Robust trajectory optimization under frictional contact with iterative learning. *Autonomous Robots, 41*(6), 1447–1461.

Ma, H., Büchler, D., Schölkopf, B., et al. (2022). A learning-based iterative control framework for controlling a robot arm with pneumatic artificial muscles. In: *Proceedings of robotics: Science and systems XVIII*, pp. 1–10

Matas, J., James, S., & Davison, A. J. (2018). Sim-to-real reinforcement learning for deformable object manipulation, pp. 1–10. arXiv:1806.07851v2

McCord, C., Queralta, J. P., Gia, T. N., et al. (2019). Distributed progressive formation control for multi-agent systems: 2D and 3D deployment of UAVs in ROS/Gazebo with RotorS. In *European conference on mobile robots*, pp. 1–6.

Moldovan, T. M., Abbeel, P. (2012). *Safe exploration in markov decision processes*, pp. 1–10. arXiv:1205.4810v3

Mueller, F. L., Schoellig, A. P., D'Andrea, R. (2012). Iterative Learning of feed-forward corrections for high-performance tracking. In: *Proceedings of the IEEE/RSJ international conference on intelligent robots and systems*, pp. 3276–3281

Nickel, V. L., Perry, J., & Garrett, A. L. (1963). Development of useful function in the severely paralyzed hand. *The Bone and Joint Journal, 45*(5), 933–952.

Oh, J., Chockalingam, V., Satinder, et al. (2016). Control of memory, active perception, and action in minecraft. In: *Proceedings of the international conference on machine learning*, pp. 2790–2799

Paulus, R., Xiong, C., & Socher, R. (2017). A deep reinforced model for abstractive summarization, pp. 1–12. arXiv:1705.04304v3

Piazzi, A., & Visioli, A. (1997). An interval algorithm for minimum-jerk trajectory planning of robot manipulators. In: *Proceedings of the IEEE conference on decision and control*, pp. 1924–1927.

Piche, A., Pardinas, R., Vazquez, D., et al. (2022). Implicit offline reinforcement learning via supervised learning, pp. 1–14. arXiv:2210.12272v1

Polydoros, A. S., & Nalpantidis, L. (2017). Survey of model-based reinforcement learning: applications on robotics. *Journal of Intelligent & Robotic Systems, 86*(2), 153–173.

Quintanilla, R., Wen, J. T. (2007). Iterative learning control for nonsmooth dynamical systems. In: *Proceedings of the IEEE conference on decision and control*, pp. 245–251

Ramya Ramakrishnan, E. K., Debadeepta, D., Eric, H., et al. (2020). Blind spot detection for safe sim-to-real transfer. *Journal of Artificial Intelligence Research,67*(1), 1–24.

Ravichandar, H., Polydoros, A. S., Chernova, S., et al. (2020). Recent advances in robot learning from demonstration. *Review of Control, Robotics, and Autonomous Systems, 3*(1), 297–330.

Schneider, J. (1996). Exploiting model uncertainty estimates for safe dynamic control learning. In: *Proceedings of the advances in neural information processing systems*, pp. 1047–1053

Schoellig, A. P., Mueller, F. L., & D'Andrea, R. (2012). Optimization-based iterative learning for precise quadrocopter trajectory tracking. *Autonomous Robots, 33*(1), 103–127.

Sferrazza, C., Muehlebach, M., & D'Andrea, R. (2020). Learning-based parametrized model predictive control for trajectory tracking. *Optimal Control Applications and Methods, 41*(6), 2225–2249.

Shah, S., Dey, D., Lovett, C., et al. (2017). AirSim: High-fidelity visual and physical simulation for autonomous vehicles, pp 1–14. arXiv:1705.05065v2

Silver, D., Huang, A., Maddison, C. J., et al. (2016). Mastering the game of go with deep neural networks and tree search. *Nature, 529*(7587), 484–489.

Silver, D., Schrittwieser, J., Simonyan, K., et al. (2017). Mastering the game of go without human knowledge. *Nature, 550*(7676), 354–359.

Tessler, C., Givony, S., Zahavy, T., et al. (2017). A deep hierarchical approach to lifelong learning in minecraft. In: *Proceedings of the AAAI conference on artificial intelligence*, pp. 1553–1561

Thrun, S., & Pratt, L. (1998). *Learning to learn*. US: Springer.

Todorov, E., Erez, T., & Tassa, Y. (2012). MuJoCo: A physics engine for model-based control. In: *Proceedings of the IEEE/RSJ international conference on intelligent robots and systems*, pp. 5026–5033.

Tondu, B., & Zagal, S. (2006). McKibben artificial muscle can be in accordance with the Hill skeletal muscle model. In: *Proceedings of the RAS-EMBS international conference on biomedical robotics and biomechatronics*, pp. 714–720

Van Rooijen, J., Grondman, I., & Babuška, R. (2014). Learning rate free reinforcement learning for real-time motion control using a value-gradient based policy. *Mechatronics, 24*(8), 966–974.

Vanschoren, J. (2018). Meta-learning: A survey, pp. 1–29. arXiv:1810.03548v1

Wang, T., Bao, X., Clavera, I., et al. (2019b). Benchmarking model-based reinforcement learning, pp. 1–25. arxiv:1907.02057v1

Wang, H., Zheng, S., Xiong, C., et al. (2019a). On the generalization gap in reparameterizable reinforcement Learning. In: *Proceedings of the international conference on machine learning*, pp. 6648–6658

Wilson, A., Fern, A., & Tadepalli, P. (2014). Using trajectory data to improve Bayesian optimization for reinforcement learning. *Journal of Machine Learning Research, 15*, 253–282.

Yogatama, D., Blunsom, P., Dyer, C., et al. (2016). Learning to compose words into sentences with reinforcement learning, pp. 1–10. arXiv:1611.09100v1

Zhang, X., & Lapata, M. (2017). Sentence simplification with deep reinforcement learning, pp. 1–11. arxiv:1703.10931v2

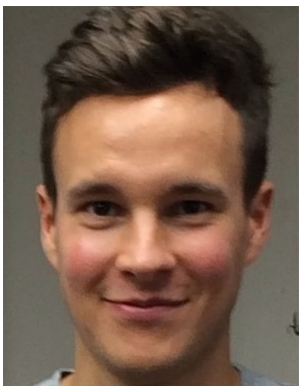
Zhao, W., Queralta, J. P., Westerlund, T. (2020). Sim-to-real transfer in deep reinforcement learning for robotics: A survey. In: *Proceedings of the IEEE symposium series on computational intelligence*, pp. 737–744.

Zhao, Y., Xiong, R., & Zhang, Y. (2017). Model based motion state estimation and trajectory prediction of spinning ball for ping-pong robots using expectation-maximization algorithm. *Journal of Intelligent & Robotic Systems,87*(3), 407–423.

Zhao, Y. M., Lin, Y., Xi, F., et al. (2015). Calibration-based iterative learning control for path tracking of industrial robots. *IEEE Transactions on Industrial Electronics, 62*(5), 2921–2929.

Zhuang, F., Qi, Z., Duan, K., et al. (2021). A comprehensive survey on transfer learning. *Proceedings of the IEEE, 109*(1), 43–76.
Zughaibi, J., Hofer, M., D'Andrea, R. (2021). A fast and reliable pick-and-place application with a spherical soft robotic arm. In: *Proceedings of the IEEE international conference on soft robotics*, pp. 599–606.

Hao Ma received the B.Sc. from Jilin University in 2017 and M.Sc. from TU Munich in 2021. He graduated with high distinction from TU Munich. His master's thesis focused on modeling and control of a robotic arm driven by pneumatic artificial muscles. Currently, he is a PhD student of the ETH-Max Planck Center for Learning Systems and is affiliated with both the Learning and Dynamical Systems group at the Max Planck Institute for Intelligent Systems and the Institute for Dynamic Systems and Control at ETH Zurich. His research interests include the control of complex systems and machine learning in the context of robotics.



Dieter Büchler received the B.Eng. degree in information and electrical engineering from the HAW Hamburg, Hamburg, Germany, in 2012, the M.Sc. degree in biomedical engineering from the Imperial College London, London, U.K., in 2013, and the Ph.D. degree from the Technische Universität Darmstadt, Darmstadt, Germany, in 2019 under the supervision of Jan Peters. He is currently a Research Group Leader with the Department of Empirical Inference, Max Planck Institute for Intelligent Systems, Tübingen, Germany, led by Bernhard Schölkopf. His research interests include control, machine learning, and robotics.

Bernhard Schölkopf scientific interests are in machine learning and causal inference. He has applied his methods to a number of different fields, ranging from biomedical problems to computational photography and astronomy. Bernhard studied physics and mathematics and earned his Ph.D. in computer science in 1997, becoming a Max Planck director in 2001. He has (co-)received the Berlin-Brandenburg Academy Prize, the Royal Society Milner Award, the Leibniz Award, the BBVA Foundation Frontiers of Knowledge Award, and the ACM AAAI Allen Newell Award. He is Fellow of the ACM and of the CIFAR Program "Learning in Machines and Brains", a member of the German Academy of Sciences, and a Professor at ETH Zurich. He helped start the MLSS series of Machine Learning Summer Schools, the Cyber Valley Initiative, the ELLIS society, and the Journal of Machine Learning Research, an early development in open access and today the field's flagship journal.

Michael Muehlebach studied mechanical engineering at ETH Zurich and specialized in robotics, systems, and control during his Master's degree. He received the B.Sc. and the M.Sc. in 2010 and 2013, respectively, before joining the Institute for Dynamic Systems and Control for his Ph.D. He graduated under the supervision of Prof. R. D'Andrea in 2018 and joined the group of Prof. Michael I. Jordan at the University of California, Berkeley as a postdoctoral researcher. In 2021 he started as an independent group leader at the Max Planck Institute for Intelligent Systems. Michael recieved the Outstanding D-MAVT Bachelor Award for his Bachelor's degree and the Willi-Studer prize for the best Master's degree. His Ph.D. thesis was awarded with the ETH Medal and the HILTI prize for innovative research. He was also awarded a Branco Weiss Fellowship and an Emmy Noether Fellowship, which funds his research group.