Online Optimization of Closed-Loop Control Systems

Hao Ma [1] **Melanie Zeilinger** [2] **Michael Muehlebach** [1]

Abstract

We propose a novel gradient-based online optimization framework for solving stochastic programming problems that frequently arise in the context of cyber-physical and robotic systems. We establish the connection between our algorithms and the cyber-physical systems through the classic two-degree-of-freedom control loop. We also incorporate an approximate model of the dynamics as prior knowledge into the learning process, and characterize the impact of modeling errors in the system dynamics on the convergence rate of the algorithms. We show that even rough estimates of the dynamics can significantly improve the convergence of our algorithms. Finally, we evaluate our algorithms in simulations of a flexible beam and a four-legged walking robot.[1][2]

1. Introduction

The increasing availability of sensors across various domains has led to the generation of vast volumes of data, ideal for analysis and training. However, a significant challenge arises from the traditional "Sampling-Training-Deployment" mode of machine learning algorithms. Once trained, most models remain static during deployment, unable to benefit from the continuous influx of new data. This limitation means that models risk becoming outdated as the environment evolves and new information emerges, leaving a substantial amount of potentially valuable data unused. This not only hinders improvements in performance but also falls short of enabling systems to continuously learn, adapt, and improve. Moreover, retraining models from scratch with new data is neither an economical nor a long-term solution. This issue is particularly prevalent in robotics, where deployed models struggle to adapt to ever-changing environments and continuous streams of new information and data.

In the field of machine learning, a strategy that incorporates streaming data is known as online leaning, which aims to minimize the expected regret as follows (Bubeck, 2012; Neu, 2015):

$$\begin{aligned}\text{Regret}\left(\mathcal{A}\right) = \mathbb{E}\left[\sum_{t=1}^{T} f\left(\omega_t;\zeta_t\right)\right] \\ - \min_{\omega\in\Omega} \mathbb{E}\left[\sum_{t=1}^{T} f\left(\omega;\zeta_t\right)\right],\ \zeta_t \overset{\text{i.i.d.}}{\sim} \mathcal{D}, \quad (1)\end{aligned}$$

where $\mathcal{A}$ denotes the specific online optimization algorithm that is used to minimize the regret and f are stochastic loss functions, where the random variable ζ_t is independently and identically sampled from the unknown distribution $\mathcal{D}$. The expectation is taken with respect to the loss functions and the decision variables $\omega_t \in \Omega$. The decision variables ω_t are generated by the algorithm $\mathcal{A}$ from a closed and convex set $\Omega \subseteq \mathbb{R}^{n_\omega}$, where $n_\omega \in \mathbb{N}_+$ denotes the number of decision variables ω. In addition, T denotes the total number of iterations. Intuitively, we claim that an online optimization algorithm $\mathcal{A}$ performs well if the regret induced by this algorithm is sub-linear as a function of T (i.e, $\text{Regret}\left(\mathcal{A}\right) = o\left(T\right)$), since this implies that on average, the algorithm performs as well as the best fixed decision variables $\omega^\star = \arg\min_{\omega\in\Omega} \mathbb{E}\left[\sum_{t=1}^{T} f\left(\omega;\zeta_t\right)\right]$ in hindsight. Therefore, when T is large enough or tends to infinity, online learning can cope with continually growing streams of data at the algorithm level, so that the decision variables ω_t continuously improve performance.

The loss functions f are typically assumed to be convex and bounded (Hazan, 2022; Shalev-Shwartz, 2012; Hall & Willett, 2015). However, one of the primary objectives of this article is to bridge the gap between theory and practice, enabling online learning algorithms to be deployed on cyber-physical systems. Therefore, in this article, we abandon the convexity assumption and instead rely on a more general smoothness assumption. Furthermore, we connect

[1]Learning and Dynamical Systems Group, Max Planck Institute for Intelligent Systems, Max-Planck-Ring 4, 72076, Tübingen, German [2]Department of Mechanical and Process Engineering, Institute for Dynamic Systems and Control, Leonhardstrasse 21, 8092 Zürich, Switzerland. Correspondence to: Hao Ma <hao.ma@tuebingen.mpg.de>.

Workshop on Foundations of Reinforcement Learning and Control at the 41st International Conference on Machine Learning, Vienna, Austria.

[1]This article summarizes Ma et al. (2024). Please see arXiv for the full version.

[2]See the supplementary video for a demonstration of the experiments: https://youtu.be/OLVvKGba7PA

our proposed algorithms with cyber-physical systems and provide a quantitaive analysis of their performance.

1.1. Related Work

The most typical algorithm for solving stochastic optimization problems is Stochastic Gradient Descent (SGD) (Robbins & Monro, 1951). In the context of large-scale machine learning, SGD is favored over batch algorithms for several reasons: SGD has a lower computational cost per iteration (Bottou et al., 2018), and SGD achieves a faster initial decline (Bertsekas, 2015). Current research is focused on how to reduce the noise in the gradient to enhance the convergence rate of SGD. This concept is evident in algorithms like the Stochastic Variance Reduced Gradient (SVRG) (Johnson & Zhang, 2013), the Stochastic Average Gradient (SAG) (Schmidt et al., 2017), and the Stochastic Average Gradient Accelerated-Descent (SAGA) (Defazio et al., 2014), which reduce the noise in the gradient obtained at each iteration through gradient aggregation without significantly increasing computational effort. The first two methods can achieve a linear convergence rate provided that the loss functions are strongly convex. The latter has been improved upon this basis, enabling it to reach a better convergence rate, still under the condition of strongly convex loss functions. However, these methods typically require traversing all or part of the data set, which is not feasible in the online learning setting.

There is little literature on deploying online learning algorithms on cyber-physical systems. Crespi & Ijspeert (2008) used a locomotion controller together with Powell's method, a gradient-free online optimization method, to optimize the gaits of an amphibious snake robot. The gradient-free online optimization enables fast optimization of the gaits in different media and takes only a few iterations to converge. Cheng & Chen (2014) developed an online parameter optimization method utilizing Gaussian process regression to approximate the relationship between the process parameters and system performance. The proposed online parameter optimization adapts to variations and satisfies the performance requirements. It also demonstrates higher efficiency and accuracy compared to existing methods. Wang et al. (2018) developed an approach based on online optimization for a robot to plan its actions in human-robot collaborations, enabling interactions with complex environments.

Different online optimization algorithms, such as online gradient descent, the online Newton algorithm and online mirror descent (Hazan, 2022; Bubeck, 2011) have been proposed and convergence results have been established. However, these theoretical proofs rely on the assumption of convexity. This is why, in the control community, there has been a nascent trend under the rubric of feedback optimization (Colombino et al., 2020; Hauswirth et al., 2017; Bernstein et al., 2019; He et al., 2024), where the steady-state of a cyber-physical system is optimized in an online manner. While the formulation does not measure and characterize regret in the sense of online learning and departs from convexity, it constrains the rate at which the underlying dynamics can change (Colombino et al., 2020) or restricts the interactions of the system with the environment (Hauswirth et al., 2017).

1.2. Structure

This article follows the structure outlined below: In Section 2, we will establish the connection between our algorithms and real-world cyber-physical systems through the classic two-degrees-of-freedom control loop, and provide a detailed formulation of the stochastic programming problem addressed in this article. Subsequently, in Section 3 we will propose an algorithmic framework to solve this problem, and derive the update schemes for our gradient descent algorithm in both open-loop and closed-loop systems. In Section 4, we will discuss the assumptions required for proving the convergence results, and characterize convergence rates in the presence of modeling errors. In Section 5, our algorithms are applied to various cyber-physical and robotic systems, including a flexible beam and a four-legged walking robot. The article concludes with a summary in Section 6.

2. Problem Formulation

We start with the classic two-degree-of-freedom control loop as shown in Figure 1, which has extensive applications in machine learning within the context of robotics. We can contrast our approach of learning feedforward and feedback controllers to reinforcement learning (RL), where the objective is to learn a feedback controller (policy) that minimizes a designated reward function (Sutton & Barto, 2018; Li, 2018). In contrast to RL, which is often based on approximately solving the Bellman equation, we do not use

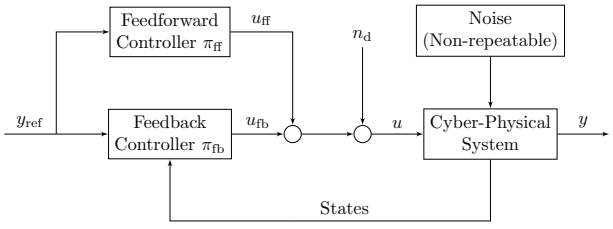


Figure 1. The figure shows the classic two-degree-of-freedom control loop, which includes a feedforward controller and a feedback controller. The variable n_{d} denotes a disturbance, which will subsequently be used to obtain an approximate gradient $\mathcal{G}(u_t)$.

any dynamic programming strategy in our approach.

We consider a specific form of (1), which is tailored to cyber-physical and robotic systems. We incorporate the system dynamics through the mapping G and parameterize actions (or control input) u_t via the function π, which include feedforward and feedback actions in the classic control loop. Our aim is to optimize performance by choosing the variable ω that parameterizes the function π and determines the actions. The dynamics G and the action parameterization π are both incorporated as constraints to establish the connection between our algorithms and the cyber-physical systems. Consequently, (1) is reformulated as follows:

$$\begin{aligned} &\mathbb{E}_{\zeta_t}\left[\sum_{t=1}^{T} l\left(y_t;\zeta_t\right)\right] - \min_{\omega\in\Omega} \mathbb{E}_{\zeta_t}\left[\sum_{t=1}^{T} l\left(y_t^\star;\zeta_t\right)\right] \\ \text{s.t. } & y_t = G\left(s_0,u_t;\zeta_t\right),\ y_t^\star = G\left(s_0,u_t^\star;\zeta_t\right) \\ & u_t = \pi\left(\omega_t,y_t;\zeta_t\right),\ u_t^\star = \pi\left(\omega,y_t^\star;\zeta_t\right), \\ & \zeta_t \overset{\text{i.i.d.}}{\sim} p_\zeta, \end{aligned} \tag{2}$$

where the constraints implicitly define y_t as a function of ω_t, s_0 and ζ_t, the superscript $(\cdot)^\star$ denotes the optimal value. The implicit equation arises due to the fact that feedback loops may potentially be present, and we assume that y_t exists and is well defined. The initial state of the cyber-physical system is denoted by $s_0 \in \mathcal{S} \subset \mathbb{R}^{n_s}$. The vectors $u_t = (u_{t,1},\ldots,u_{t,q})$ and $y_t = (y_{t,1},\ldots,y_{t,q})$ denote the input and output sequences of the system, where $u_{t,i} \in \mathcal{U} \subset \mathbb{R}^m$ and $y_{t,i} \in \mathcal{Y} \subset \mathbb{R}^n$, $i = 1,\ldots,q$ represent the input and output at a certain time point i and at iteration t of the learning process. The mapping $G(\cdot,\cdot;\zeta) : \mathcal{S}\times\mathcal{U}^q \to \mathcal{Y}^q$ transforms a sequence of inputs into a sequence of outputs and represents the input-output behavior of the cyber-physical system. The input-output behavior is not necessarily deterministic, due to process, measurement, and actuation uncertainty, which is modeled with the random variable ζ_t. In practice, the mapping G is typically unknown and may exhibit a high degree of nonlinearity, for example due to friction in the joints of a robot. The mapping $\pi(\cdot,y_t;\zeta) : \Omega \to \mathcal{U}^q$ describes how the decision variables ω_t affect the controls u_t. The feasible set $\Omega \subset \mathbb{R}^{n_\omega}$ is assumed to be closed and convex.

In (2), the function $l(\cdot;\zeta) : \mathcal{Y}^q \to \mathbb{R}$ describes the stochastic loss function, for example, tracking error, execution time, energy consumption, etc. To simplify the subsequent derivations and without loss of generality, we consider a specific form of l that models a trajectory tracking task:

$$l\left(y_t;y_{\text{ref},t}\right) := \frac{1}{2}\left|G\left(s_0,u_t;y_{\text{ref},t}\right) - y_{\text{ref},t}\right|^2, \quad y_{\text{ref},t} \overset{\text{i.i.d.}}{\sim} p_{y_{\text{ref}}},\ t = 1,\ldots,T. \tag{3}$$

where the random variable ζ denotes the reference trajectory $y_{\text{ref}} \in \mathcal{Y}^q$ that the system is required to track. This notation inherently suggests that the reference trajectory y_{ref} evolves in correspondence with the progression of iterations. Concurrently, at each iteration, the reference trajectory is randomly sampled from a fixed yet unknown distribution $p_{y_{\text{ref}}}$. For example, in the context of training a robot for table tennis, $p_{y_{\text{ref}}}$ is determined by the trajectories experienced by the end-effector during ball interception (Ma et al., 2022; 2023; Tobuschat et al., 2023). This implies that when solving (2) we have identified a nonlinear feedforward and feedback controller that yields accurate trajectory tracking for any $y_{\text{ref}} \sim p_{y_{\text{ref}}}$. For simplification and without compromising generality (we could extend the function π to also account for s_0), we assume that the system consistently initializes from an identical state prior to each iteration t of the learning process. As such, the initial state s_0 can be fixed and omitted. For example, in the experiments with the ping-pong robot we drive the robot back to a rest position after each iteration of the online learning with a simple proportional-integral-derivative (PID) controller.

Algorithm 1 Online-Quasi Newton Method

Input: initial parameters ω_1, constant ϵ and α, iterations T, step length $\{\eta_t\}_{t=1}^T$
for $t = 1$ **to** T **do**
  Sampling: $\zeta \sim p_\zeta \to \zeta_t$
  Implementation: $G\left(s_0,\pi\left(\omega_t,y_t;\zeta_t\right);\zeta_t\right) \to y_t$
  Evaluation: $l\left(y_t;\zeta_t\right)$
  Approximation: $\mathcal{L}_t \approx {}^{\partial y_t}/_{\partial \omega_t}$
  Hessian Calculation: $\Lambda_t = \frac{1}{\epsilon}\mathcal{L}_t^{\mathrm{T}}\nabla_y^2 l\left(y_t;\zeta_t\right)\mathcal{L}_t + \frac{\alpha}{\epsilon}\nabla_\omega\pi\left(\omega_t,y_t;\zeta_t\right)\nabla_\omega\pi\left(\omega_t,y_t;\zeta_t\right)^{\mathrm{T}} + \mathrm{I}$;
  $A_t = \frac{1}{t}\sum_{k=1}^{t}\Lambda_k$
  Update: $\omega_{t+1} = \omega_t - \eta_t A_t^\dagger \mathcal{L}_t^{\mathrm{T}}\nabla_y l\left(y_t;\zeta_t\right)$
end for

3. Stochastic Online Optimization

To address the online optimization problem (2), we propose Algorithm 1, which depending on the choice of ϵ represents either an online gradient descent or an online quasi-Newton method. The Moore–Penrose inverse is denoted by $(\cdot)^\dagger$.

We note that in Algorithm 1, the majority of variables can be obtained through measurement or simple calculations, except for the gradient of the outputs y_t with respect to the decision variables ω_t. The difficulty mainly arises from two aspects:

1. The dynamic behavior of the system G is unknown.
2. Due to the presence of the feedback loop, y_t is defined as an implicit function. This means that we must consider the effect of the feedback loop during the learning

process.

The former aspect can be addressed using various techniques such as system identification and finite difference estimation (Ljung, 2010; Pintelon & Schoukens, 2012; Carè et al., 2018; Tsiamis & Pappas, 2019; Campi & Weyer, 2002). In this article, we adopt a black-box representation for G, avoiding any explicit characterization of its internal dynamics. Although the dynamic characteristics of G are unknown, we assume that G is differentiable with respect to u. Furthermore, we adopt the following notational convention: $\mathcal{G}(u_t)$ represents an approximation of the gradient of the mapping G with respect to u, more precisely, $\mathcal{G}(u_t)$ denotes an approximation of $\partial G(s_0,u;\zeta)/\partial u\big|_{u=u_t}$. In Section 5, we will demonstrate that even a rough approximation of $\partial G(s_0,u;\zeta)/\partial u\big|_{u=u_t}$ can serve as valuable prior knowledge, significantly improving the convergence rate of our algorithms. In the following subsection we will focus on analyzing the impact of the feedback loop in the learning process, and derive the update scheme.

3.1. Impact of the Feedback Loop

In our approach, we use π_{ff} and π_{fb} to represent the parameterized feedforward and feedback networks, respectively, and ω_{ff} and ω_{ff} to denote their corresponding parameters. Then, the relation between y_t and u_t in (2) can be reformulated as follows:

$$\begin{aligned} u_t &= \pi_{\text{ff}}(\omega_{\text{ff},t};\zeta_t) + \pi_{\text{fb}}(\omega_{\text{fb},t};y_t-\zeta_t), \\ y_t &= G(s_0,u_t;\zeta_t),\ \zeta_t \overset{\text{i.i.d.}}{\sim} p_\zeta, \end{aligned} \tag{4}$$

that is, the input u_t is the combination of a feedforward part π_{ff} that does not depend on y_t and a feedback part π_{fb} that depends on the deviation of y_t from the reference trajectory ζ_t. Due to the inclusion of feedback π_{fb}, the calculation of the gradient in Algorithm 1 becomes more complex and less intuitive compared to the open-loop situation where $\pi_{\text{fb}}=0$. Hence, we will demonstrate the computation of the gradients $\partial y/\partial\omega_{\text{ff}}$ and $\partial y/\partial\omega_{\text{fb}}$ in the closed-loop system and discuss their implications. The critical aspect to note at this point is that (4) defines an implicit equation for y_t and also u_t. We should therefore think of y_t and u_t as functions of ω_{ff}, ω_{fb} and ζ_t, that is, $u=u(\omega_{\text{ff}},\omega_{\text{fb}};\zeta)$, $y=y(\omega_{\text{ff}},\omega_{\text{fb}};\zeta)$. The gradient of the loss function l with respect to the parameters can be calculated as follows:

$$\nabla_\omega f(\omega;\zeta) = \begin{bmatrix} \frac{\partial y^{\text{T}}}{\partial\omega_{\text{ff}}} & \frac{\partial y^{\text{T}}}{\partial\omega_{\text{fb}}} \end{bmatrix} \nabla_y l(y;\zeta).$$

By combining the two equations in (4) we get a single implicit equation for y. The differential $\partial y/\partial\omega_{\text{ff}}$ can now be obtained by differentiating the implicit equation with respect to ω_{ff} (implicit function theorem):

$$\begin{aligned} \frac{\partial y}{\partial\omega_{\text{ff}}} &= \frac{\partial G(u;\zeta)}{\partial u}\bigg|_{u=\pi_{\text{ff}}+\pi_{\text{fb}}} \frac{\partial\pi_{\text{ff}}(\omega_{\text{ff}};\zeta)}{\partial\omega_{\text{ff}}} \\ &+ \frac{\partial G(u;\zeta)}{\partial u}\bigg|_{u=\pi_{\text{ff}}+\pi_{\text{fb}}} \frac{\partial\pi_{\text{fb}}(\omega_{\text{fb}};\zeta^\circ)}{\partial\zeta^\circ}\bigg|_{\zeta^\circ=G(u;\zeta)-\zeta} \frac{\partial y}{\partial\omega_{\text{ff}}}. \end{aligned}$$

This can be rearranged to

$$\begin{aligned} \frac{\partial y}{\partial\omega_{\text{ff}}} = \left(\text{I} - \frac{\partial G(u;\zeta)}{\partial u}\frac{\partial\pi_{\text{fb}}(\omega_{\text{fb}};\zeta^\circ)}{\partial\zeta^\circ}\bigg|_{\zeta^\circ=G(u;\zeta)-\zeta}\right)^\dagger \\ \frac{\partial G(u;\zeta)}{\partial u}\frac{\partial\pi_{\text{ff}}(\omega_{\text{ff}};\zeta)}{\partial\omega_{\text{ff}}}. \end{aligned} \tag{5}$$

The expression $\partial y/\partial\omega_{\text{fb}}$ can be derived with a similar argument and results in

$$\begin{aligned} \frac{\partial y}{\partial\omega_{\text{fb}}} = \left(\text{I} - \frac{\partial G(u;\zeta)}{\partial u}\frac{\partial\pi_{\text{fb}}(\omega_{\text{fb}};\zeta^\circ)}{\partial\zeta^\circ}\bigg|_{\zeta^\circ=G(u;\zeta)-\zeta}\right)^\dagger \\ \frac{\partial G(u;\zeta)}{\partial u}\frac{\partial\pi_{\text{fb}}(\omega_{\text{fb}};\zeta^\circ)}{\partial\omega_{\text{fb}}}\bigg|_{\zeta^\circ=G(u;\zeta)-\zeta}. \end{aligned} \tag{6}$$

We observe that the term $\partial G(u;\zeta)/\partial u$ consistently represents the gradient of the open-loop system and, as previously mentioned, can be approximated using the estimate $\mathcal{G}(u)$. This approximation renders the terms $\partial y/\partial\omega_{\text{ff}}$ and $\partial y/\partial\omega_{\text{fb}}$ computable. We also note that the feedback controller may reduce the effect of estimation errors in $\mathcal{G}$ on the resulting gradient estimates. Indeed, if $\partial\pi_{\text{fb}}(\omega_{\text{fb}};\zeta^\circ)/\partial\zeta^\circ$ is large, both expressions reduce to

$$\frac{\partial\pi_{\text{fb}}(\omega_{\text{fb}};\zeta^\circ)^\dagger}{\partial\zeta^\circ}\frac{\partial\pi_{\text{fb}}(\omega_{\text{fb}};\zeta^\circ)}{\partial\omega_{\text{fb}}},\ \frac{\partial\pi_{\text{fb}}(\omega_{\text{fb}};\zeta^\circ)^\dagger}{\partial\zeta^\circ}\frac{\partial\pi_{\text{ff}}(\omega_{\text{ff}};\zeta)}{\partial\omega_{\text{ff}}},$$

respectively, which means that ∇f is approximately independent of G for large $\partial\pi_{\text{fb}}(\omega_{\text{fb}};\zeta^\circ)/\partial\zeta^\circ$. If the feedback gain is small, however, $\partial y/\partial\omega_{\text{ff}}$ and $\partial y/\partial\omega_{\text{fb}}$ reduce to

$$\frac{\partial G(s_0,u;\zeta)}{\partial u}\frac{\pi_{\text{ff}}(\omega_{\text{ff}};\zeta)}{\partial\omega_{\text{ff}}},\ \frac{\partial G(s_0,u;\zeta)}{\partial u}\frac{\pi_{\text{fb}}(\omega_{\text{fb}};\zeta^\circ)}{\partial\omega_{\text{fb}}}.$$

Moving forward, we will briefly show that the term

$$\left(\text{I} - \frac{\partial G(u;\zeta)}{\partial u}\frac{\partial\pi_{\text{fb}}(\omega_{\text{fb}};\zeta^\circ)}{\partial\zeta^\circ}\right)^\dagger \frac{\partial G(u;\zeta)}{\partial u} \tag{7}$$

is the gradient of the closed-loop system with respect to the external input n_{d} (see Figure 1). This finding enables us to directly derive gradient estimation approaches for the closed-loop system by performing stochastic finite difference, which will be denoted as $\mathcal{G}^\circ(\omega_{\text{ff}},\omega_{\text{fb}},\zeta)$. We perform the following calculations (implicit function theorem):

$$\begin{aligned} \frac{\partial y}{\partial n_{\text{d}}} &= \frac{\partial G(u;\zeta)}{\partial u}\bigg|_{u=\pi_{\text{ff}}+\pi_{\text{fb}}} \\ &+ \frac{\partial G(u;\zeta)}{\partial u}\bigg|_{u=\pi_{\text{ff}}+\pi_{\text{fb}}} \frac{\partial\pi_{\text{fb}}(\omega_{\text{fb}};\zeta^\circ)}{\partial\zeta^\circ}\bigg|_{\zeta^\circ=G(u;\zeta)-\zeta} \frac{\partial y}{\partial n_{\text{d}}}, \end{aligned}$$

which results in

$$\frac{\partial y}{\partial n_{\mathrm{d}}} = \left(\mathrm{I} - \frac{\partial G\left(u;\zeta\right)}{\partial u}\frac{\partial \pi_{\mathrm{fb}}\left(\omega_{\mathrm{fb}};\zeta^{\circ}\right)}{\partial \zeta^{\circ}}\right)^{\dagger}\frac{\partial G\left(u;\zeta\right)}{\partial u}.$$

Intuitively, $\partial y/\partial n_{\mathrm{d}}$ describes the sensitivity of y in closed-loop to changes in u.

Consequently, the terms $\partial y/\partial \omega_{\mathrm{ff}}$ and $\partial y/\partial \omega_{\mathrm{fb}}$, apart from being derived from (5) and (6), can also be obtained through the following more direct approach

$$\frac{\partial y}{\partial \omega_{\mathrm{ff}}} = \mathcal{G}^{\circ}\left(\omega_{\mathrm{ff}},\omega_{\mathrm{fb}},\zeta\right)\frac{\partial \pi_{\mathrm{ff}}\left(\omega_{\mathrm{ff}};\zeta\right)}{\partial \omega_{\mathrm{ff}}},$$
$$\frac{\partial y}{\partial \omega_{\mathrm{fb}}} = \mathcal{G}^{\circ}\left(\omega_{\mathrm{ff}},\omega_{\mathrm{fb}},\zeta\right)\frac{\partial \pi_{\mathrm{fb}}\left(\omega_{\mathrm{fb}};\zeta^{\circ}\right)}{\partial \omega_{\mathrm{fb}}},$$

thereby allowing for direct computations if $\mathcal{G}^{\circ}$ is known. As we will highlight with experiments in Section 5 $\mathcal{G}^{\circ}\left(\omega_{\mathrm{ff}},\omega_{\mathrm{fb}},\zeta\right)$ can be estimated by performing stochastic rollouts with different random perturbations n_{d} (stochastic finite difference).

4. Convergence Guarantees

In this section, we summarize the convergence guarantees of Algorithm 1 under the following assumptions:

Assumption 4.1 (L-Smoothness)**.** Let the loss functions $f\left(\cdot;\zeta\right):\Omega\to\mathbb{R}$ be L-smooth, that is,

$$\left|\nabla f\left(v;\zeta\right)-\nabla f\left(\omega;\zeta\right)\right|\leq L\left|v-\omega\right|,$$

for all $\omega,v\in\Omega$.

Assumption 4.2 (Bounded Variance)**.** There exists a constant $H\geq 0$ such that for all $\omega\in\Omega$ the following inequalities hold:

$$\mathbb{E}_{\zeta}\left[\left|\nabla f\left(\omega;\zeta\right)\right|^{2}\right]\leq H^{2},\ \mathbb{E}_{\zeta}\left[\left|\mathcal{F}\left(\omega;\zeta\right)\right|^{2}\right]\leq H^{2},$$

where $\mathcal{F}\left(\omega;\zeta\right)$ denotes the estimated gradient of $f\left(\omega;\zeta\right)$ induced by $\mathcal{G}\left(u\right)$, while $\nabla f\left(\omega;\zeta\right)$ denotes the true gradient, that is,

$$\mathcal{F}\left(\omega;\zeta\right)=\frac{\partial l\left(y;\zeta\right)}{\partial y}\left(\mathrm{I}-\mathcal{G}\left(u\right)\frac{\partial \pi\left(\omega,y;\zeta\right)}{\partial y}\right)^{\dagger}\mathcal{G}\left(u\right)\frac{\partial \pi\left(\omega,y;\zeta\right)}{\partial \omega},$$
$$\nabla f\left(\omega;\zeta\right)=\frac{\partial l\left(y;\zeta\right)}{\partial y}\left(\mathrm{I}-\frac{\partial G\left(s_0,u;\zeta\right)}{\partial u}\frac{\partial \pi\left(\omega,y;\zeta\right)}{\partial y}\right)^{\dagger}\frac{\partial G\left(s_0,u;\zeta\right)}{\partial u}\frac{\partial \pi\left(\omega,y;\zeta\right)}{\partial \omega}.$$

Assumption 4.3 (Bounded Hessian)**.** Given a sequence of single pseudo-Hessians Λ_t obtained according to Algorithm 1, there exists a constant $\lambda\geq 1$ such that for all $t=1,\ldots,T$ the following inequalities hold:

$$1\leq\lambda_{\min}\left(\Lambda_t\right)\leq\lambda_{\max}\left(\Lambda_t\right)\leq\lambda,$$

where $\lambda_{\min}$ and $\lambda_{\max}$ denote the minimum and maximum eigenvalues of a matrix, respectively.

In this work, we abandon the convexity assumption of the objective function f with respect to ω and employ a more general smoothness assumption instead (see Assumption 4.1). Assumption 4.1 and Assumption 4.2 are standard in non-convex optimization (Bottou et al., 2018). We note that the non-convexity of the objective function f arises from the nonlinear dynamics of the cyber-physical systems, while the function $l\left(y;\zeta\right)$ can still be chosen to be convex. Thereby, all additive terms in Λ_t in Assumption 4.3 are guaranteed to be positive semi-definite. Furthermore, the matrix Λ_t depends on the parameter ϵ, which can always be chosen large enough, such that Assumption 4.3 is satisfied. Beyond this, we also make the following assumption on the modeling errors of our gradient estimate:

Assumption 4.4 (Modeling Error)**.** Let the parameters ω_t evolve according to Algorithm 1. There exists a constant $\kappa\in[0,1)$ such that for all $t=1,\ldots,T$ the following inequality holds:

$$\left|\mathbb{E}_{\zeta}\left[\mathcal{F}\left(\omega_t;\zeta\right)\middle|\,\omega_t\right]-\mathbb{E}_{\zeta}\left[\nabla f\left(\omega_t;\zeta\right)\middle|\,\omega_t\right]\right|^{2}\leq\frac{\kappa^{2}}{\lambda}\left|\mathbb{E}_{\zeta}\left[\nabla f\left(\omega_t;\zeta\right)\middle|\,\omega_t\right]\right|^{2}. \quad (8)$$

In fact, the parameter λ arises from choosing the ℓ_2-norm in (8). If the inequality (8) is expressed in the metric $|\cdot|_{A_t^{-1}}$, the factor $1/\lambda$ can be avoided, where $|\cdot|_{A_t^{-1}}$ denotes the metric induced by the positive definite matrix A_t^{-1}, that is,

$$|x|_{A_t^{-1}}^{2}:=\sup_{|x|\leq 1}x^{\mathrm{T}}A_t^{-1}x.$$

Then, we have the subsequent conclusions for Algorithm 1:

Theorem 4.5. *Let the loss functions $f\left(\cdot;\zeta\right):\Omega\to\mathbb{R}$ satisfy Assumption 4.1 and Assumption 4.2, and let the pseudo-Hessian A_t satisfy Assumption 4.3. Let the estimate $\mathcal{G}\left(u_t\right)$ satisfy Assumption 4.4, and let the step size be chosen as*

$$\eta=\sqrt{\frac{2F\left(\omega_1\right)}{LH^{2}T}}.$$

Then the following inequality holds:

$$\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}_{\zeta_{1:T}}\left[\left|\nabla F(\omega_t)\right|_{A_t^{-1}}^{2}\right]\leq\sqrt{\frac{2LH^{2}F\left(\omega_1\right)}{\left(1-\kappa\right)^{2}T}}+\frac{\lambda H^{2}\left(\ln T+2\right)}{\left(1-\kappa\right)T}, \quad (9)$$

where $\omega^{\star}:=\arg\min_{\omega\in\Omega}F\left(\omega\right)$ denotes the global optimum and $F\left(\omega\right):=\mathbb{E}_{\zeta}\left[f\left(\omega;\zeta\right)\right]$.

From the above conclusion, it is evident that even when using approximate gradients and avoiding convexity assumptions, the expected value of the average of the squared gradients still converges at a rate comparable to many popular stochastic optimization algorithms (Bottou et al., 2018). We note that, due to the unavailability of $\partial G(s_0,u;\zeta)/\partial u$ in practical scenarios, the convergence rate of Algorithm 1 needs to be characterized using the modeling error modulus κ, and the convergence rate is governed by $1/1-\kappa$. If the modeling error modulus κ reaches one, the results become trivial since the right-hand side in (9) becomes arbitrarily large. When the modeling error modulus κ is zero, it implies that the estimate $\mathcal{G}(u)$ has no bias. The intuitive representation of Assumption 4.4 in two-dimensional space is illustrated in Figure 2. The expectation of the gradient estimate $\mathbb{E}_\zeta[\mathcal{F}(\omega_t;\zeta)|\,\omega_t]$ lies within the open ball with center $\mathbb{E}_\zeta[\nabla f(\omega_t;\zeta)|\,\omega_t]$ and radius $|\mathbb{E}_\zeta[\nabla f(\omega_t;\zeta)|\omega_t]|/\sqrt{\lambda}$. This implies that Assumption 4.4 constrains the estimate $\mathbb{E}_\zeta[\mathcal{F}(\omega_t;\zeta)|\,\omega_t]$ both in magnitude and direction. Therefore, the parameter κ provides a reference for evaluating the quality of the obtained estimates.

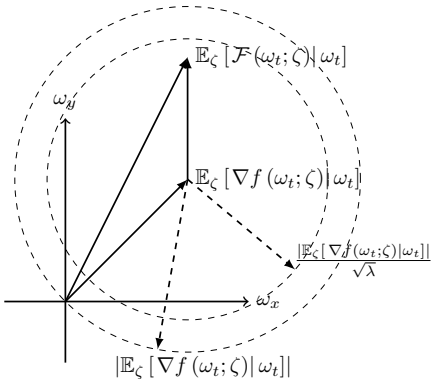


Figure 2. This figure illustrates the geometric meaning of the modeling error modulus κ in two-dimensional space. The expectation of the gradient estimate $\mathbb{E}_\zeta[\mathcal{F}(\omega_t;\zeta)|\,\omega_t]$ lies within the open ball with center $\mathbb{E}_\zeta[\nabla f(\omega_t;\zeta)|\,\omega_t]$ and radius $|\mathbb{E}_\zeta[\nabla f(\omega_t;\zeta)|\omega_t]|/\sqrt{\lambda}$.

We observe that by selecting a sufficiently large ϵ, the upper bound λ approaches one, thereby transforming Algorithm 1 from a Newton method to a gradient descent method. This suggests that by adjusting the value of ϵ, we can enable the algorithm to switch between Newton and gradient descent methodologies, leading to the following corollary:

Corollary 4.6. *Let the assumptions of Theorem 4.5 be satisfied and let $\epsilon \to +\infty$. Then, the following inequality holds:*

$$\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}_{\zeta_{1:T}}\left[|\nabla F(\omega_t)|^2\right] \leq \sqrt{\frac{2LH^2F(\omega_1)}{(1-\kappa)^2T}} + \frac{H^2(\ln T+2)}{(1-\kappa)T}.$$

Next, we will reveal the connection between online learning (1) and stochastic optimization (2), and provide the corresponding convergence guarantee. Prior to this, we make the following additional assumption:

Assumption 4.7 (Polyak-Łojasiewicz Inequality)**.** There exists a constant $\mu > 0$ such that for all $\omega \in \Omega$, the Polyak-Łojasiewicz (PL) inequality holds:

$$|\nabla F(\omega)|^2 \geq 2\mu(F(\omega) - F(\omega^\star)), \tag{10}$$

where $\omega^\star = \arg\min_{\omega\in\Omega} F(\omega)$ denotes the global optimum.

Following this assumption, we have the conclusion:

Corollary 4.8. *Let the assumptions in Theorem 4.5 be satisfied, and let Assumption 4.7 hold. Then, for any $\epsilon > 0$, the expected regret satisfies the following inequality:*

$$\mathbb{E}_{\zeta_{1:T}}\left[\sum_{t=1}^{T} f(\omega_t;\zeta_t)\right] - \min_{\omega\in\Omega}\mathbb{E}_{\zeta_{1:T}}\left[\sum_{t=1}^{T} f(\omega;\zeta_t)\right] \leq \frac{\hbar_1\sqrt{T} + \hbar_2\ln T + \hbar_3}{2\mu},$$

where

$$\hbar_1 = \frac{\lambda\sqrt{2LH^2F(\omega_1)}}{1-\kappa},\ \hbar_2 = \frac{\lambda H^2}{1-\kappa},\ \hbar_3 = \frac{2\lambda H^2}{1-\kappa}.$$

For a detailed proof of Theorem 4.5 and Corollary 4.8, please refer to Ma et al. (2024).

5. Experiments

In this section, we will demonstrate the effectiveness of our algorithms through experiments conducted on various cyber-physical systems. We highlight that even shallow networks work well with our algorithms. Depending on different scenarios, we will employ an appropriate method to obtain gradient estimates $\mathcal{G}(u)$ or $\mathcal{G}^\circ$.

5.1. Cantilever Beam

We consider a flexible cantilever beam illustrated in Figure 3, where the left end of the beam is hinged to a joint, and the active torque τ is applied only at the left end. The total

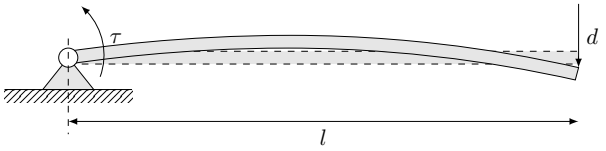


Figure 3. Deformation of the cantilever beam under the active torque and an external disturbance d, where the dashed line represents the position of the cantilever beam when at rest.

length of the entire cantilever beam in a rest configuration is denoted by l. The aim of this experiment is to utilize Algorithm 1 to learn the parameters of networks π_{ff} and π_{fb} in an online manner, in order to minimize the tracking error of the end-effector $|y - y_{\text{ref}}|$ for reference trajectories sampled from the unknown distribution $p_{y_{\text{ref}}}$. The outputs of the parameterized networks yield the active torque τ. The output y describes the distance in y-direction between the tip of the beam and the horizontal plane. We note that current reinforcement learning algorithms have difficulties in dealing with continuous state and action spaces exceeding two dozen states, while the example illustrates how our approach can easily handle dynamical systems with a large number of hidden states (here 100). The cantilever model and the following experiments are implemented in `Matlab` and `Simulink`. We intentionally increase the nonlinearity of the cantilever beam deformation, complemented by 100 hidden states in the discrete model in `Simulink`, making this task highly challenging.

All reference trajectories used in the experiment arise from sampling an unknown but fixed distribution. At each iteration, we randomly generate the reference trajectory according to the method described in Appendix A.1. We observe that the range of the trajectory is extensive and is not limited to small deformations. In the subsequent experiments, we will see that the parameterized networks trained by our algorithms effectively generalize well across the entire support of $p_{y_{\text{ref}}}$.

In this experiment, we employ system identification in the frequency domain to obtain a rough linear estimate of $\partial G(s_0,u;\zeta)/\partial u$ (Pintelon & Schoukens, 2012). It is important to emphasize that in this case the gradient estimate $\mathcal{G}$ is static, meaning that it does not change as a function of u. Additionally, we estimate the closed-loop system gradient $\mathcal{G}^{\circ}(\omega_{\text{ff}}, \omega_{\text{fb}}, \zeta)$ using (7). More information about the gradient estimation can be found in Appendix A.2.

We employ the strategies described in Appendix A.3 to parameterize the feedforward network π_{ff}. One is a linear network, denoted as π_{ff}^1. The other is a nonlinear network, represented by π_{ff}^2, which is a fully-connected network with a single hidden layer. The `ReLU` function is used as the activation function for the hidden layer, and no activation function is applied to the output layer. We consistently use a linear feedback network π_{fb}.

The experimental setups and results can be found in Appendix A.4. In a noise-free environment, trajectory tracking can be viewed as a purely feedforward control task. Therefore, in Experiments 1-4, we employ only the feedforward network π_{ff} and adjust the parameter ϵ, allowing Algorithm 1 to transition between gradient descent ($\epsilon \to \infty$) and the quasi-Newton method ($\epsilon < \infty$). Through these experiments, we explore the convergence rates of different networks and investigate the influence of different algorithms on convergence as well as their robustness to the selection of hyper-parameters. Subsequently, we intentionally introduce noise n_{d} to the inputs of the system (see Figure 1), rendering the pure feedforward network ineffective for the task at hand (see Experiment 5). Experiment 6 demonstrates the ability of the combined feedforward and feedback control (π_{ff} and π_{fb}) to resist noise in online learning. We evaluate the performance of all the obtained parameterized networks trained in different experiments on a newly generated test data set previously unseen by our algorithms (see the average loss in Table 2), in order to investigate the generalization capability of the networks. Although we only utilize shallow networks and a linear static gradient estimate $\mathcal{G}$ (which, unsurprisingly, is a very poor estimate), the algorithms still perform well using either gradient descent method or quasi-Newton method, reflecting its strong robustness to modeling errors.

5.2. Four-Legged Robot

In this experiment, we adopt the ant model (Schulman et al., 2018) frequently used to demonstrate reinforcement learning algorithms to evaluate the effectiveness of our algorithms. We choose `Isaac Gym` (Makoviychuk et al., 2021) as our simulation environment for its ability to support large-scale parallel simulations, which enables us to rapidly estimate system gradients using a stochastic finite difference method. It is important to emphasize that the traditional reinforcement learning task on this model focuses on enabling the ant to move forward as quickly as possible. However, in our experiment, our aim is to enable the ant to track any reference trajectory of the center of the mass of the torso. It should be noted that in this experiment, we do not artificially introduce system noise, thus the trajectory tracking task can be considered a purely feedforward control task. Therefore, we only employ a feedforward model π_{ff}, which implies that $\pi_{\text{fb}} = 0$.

In this experiment, we can only measure and observe the information about the torso, which includes the position of the torso, its orientation represented by a quaternion, and the translational and angular velocities of the torso. The

ant moves on a rough and infinitely flat plane, therefore the reference trajectories contain only three components: the planar position of the torso, i.e., the x and y coordinates, and the yaw, which is the rotation of the torso around the z-axis. The method for generating the reference trajectories can be found in Appendix B.1.

In the context of the ant model, which is a system characterized by contacts and non- smooth motion, employing system identification methods as described in Section 5.1 is not applicable. Fortunately, the powerful parallel simulation capability of the `Isaac Gym` environment allows us to easily estimate the system gradient $G(u_t)$ using a stochastic finite difference method (see Appendix B.2).

We recognize that the learning of the motion of the ant, without any prior knowledge, is a challenging task. Compared to the experiments in Section 5.1, the learning of the motion present the following differences and difficulties: First, due to the contacts and interactions between the ant and the environment, the motion of the ant is non-smooth, and accordingly, its gradients are discontinuous (though still assumed to be bounded). Second, the states of the ant are not fully observable. In fact, in this experiment only the information about the torso is assumed to be measurable and observable, including its positions, orientations, and corresponding velocities and angular velocities. This means that changes in control inputs do not necessarily cause changes in the outputs. For instance, when one of the legs is not in contact with the ground, the positional change of this leg caused by input variation will not affect the posture of the torso. Third, walking, as a periodic behavior, should follow specific gaits and frequencies. Training a network model from scratch may lead to the ant exhibiting anomalous behaviors. Based on the aforementioned perspectives, we make adjustments to the network structure and use a pre-trained linear model to provide prior knowledge of ant motion patterns. For more detailed information, please refer to Appendix B.3.

The experimental setups and results can be found in Appendix B.4. We note that the loss of both algorithms eventually converges to the same level with the same rate. However, it is important to emphasize that to ensure the convergence of the gradient descent method, its step size η_t must be carefully designed. In contrast, the quasi-Newton method demonstrates much stronger robustness to the step size selection. Finally, through this experiment, we demonstrate that in such complex cyber-physical system, even with a poor gradient estimate, our algorithms still ensure convergence and exhibit high robustness to modeling errors. It is important to emphasize that, unlike RL, which optimizes a feedback policy to enable the ant to move forward as fast as possible, our algorithms learn a feedforward model π_{ff} that allows the ant to track any reference trajectories sampled from the distribution $p_{y_{\text{ref}}}$, thereby truly enabling it to learn the skill of walking. Additionally, our algorithms are capable of continuously improving the tracking performance of the feedforward network through online learning during deployment.

6. Conclusion

In this article, we propose a novel gradient-based online learning framework operating under the assumptions that the loss functions are smooth but not necessarily convex. Thanks to gradient information that is incorporated within the algorithm, we obtain a sample efficient online learning approach that is applicable to cyber-physical and robotic systems. The framework presented in this article includes a stochastic optimization algorithm, various designs for neural networks and input structures for feedforward and feedback control scenarios. We have not only theoretically proven the convergence of the algorithm without relying on convexity, but also evaluated the effectiveness of our proposed framework through simulation experiments. These experiments highlight fast convergence of our algorithms and robustness against modeling errors. Furthermore, they provide empirical evidence that this algorithm can be deployed in real-world applications in the future.

References

Bernstein, A., Dall'Anese, E., and Simonetto, A. Online Primal-Dual Methods With Measurement Feedback for Time-Varying Convex Optimization. *IEEE Transactions on Signal Processing*, 67(8):1978–1991, 2019.

Bertsekas, D. *Convex Optimization Algorithms*. Athena Scientific, 2015.

Bottou, L., Curtis, F. E., and Nocedal, J. Optimization Methods for Large-Scale Machine Learning. *SIAM Review*, 60 (2):223–311, 2018.

Bubeck, S. Introduction to Online Optimization. *Lecture Notes*, 2(1):1–86, 2011.

Bubeck, S. Regret Analysis of Stochastic and Nonstochastic Multi-armed Bandit Problems. *Foundations and Trends in Machine Learning*, 5(1):1–122, 2012.

Campi, M. C. and Weyer, E. Finite Sample Properties of System Identification Methods. *IEEE Transactions on Automatic Control*, 47(8):1329–1334, 2002.

Carè, A., Csáji, B. C., Campi, M. C., and Weyer, E. Finite-Sample System Identification: An Overview and a New Correlation Method. *IEEE Control Systems Letters*, 2(1): 61–66, 2018.

Cheng, H. and Chen, H. Online Parameter Optimization in Robotic Force Controlled Assembly Processes. In *Proceedings of IEEE International Conference on Robotics and Automation*, pp. 3465–3470, 2014.

Colombino, M., Dall'Anese, E., and Bernstein, A. Online Optimization as a Feedback Controller: Stability and Tracking. *IEEE Transactions on Control of Network Systems*, 7(1):422–432, 2020.

Crespi, A. and Ijspeert, A. J. Online Optimization of Swimming and Crawling in an Amphibious Snake Robot. *IEEE Transactions on Robotics*, 24(1):75–87, 2008.

Defazio, A., Bach, F., and Lacoste-Julien, S. SAGA: A Fast Incremental Gradient Method With Support for Non-Strongly Convex Composite Objectives. In *Proceedings of Advances in Neural Information Processing Systems*, pp. 1–9, 2014.

Geering, H. P. *Optimal Control with Engineering Applications*. Springer, 2007.

Hall, E. C. and Willett, R. M. Online Convex Optimization in Dynamic Environments. *IEEE Journal of Selected Topics in Signal Processing*, 9(4):647–662, 2015.

Hauswirth, A., Zanardi, A., Bolognani, S., Dorfler, F., and Hug, G. Online Optimization in Closed Loop on the Power Flow Manifold. In *Proceedings of IEEE Manchester PowerTech*, pp. 1–6, 2017.

Hazan, E. *Introduction to Oline Convex Optimization*. MIT Press, 2022.

He, Z., Bolognani, S., Muehlebach, M., and Dörfler, F. Gray-Box Nonlinear Feedback Optimization. *arXiv*, 2404.04355:1–16, 2024.

Hofer, M., Spannagl, L., and D'Andrea, R. Iterative Learning Control for Fast and Accurate Position Tracking with an Articulated Soft Robotic Arm. In *Proceedings of International Conference on Intelligent Robots and Systems*, pp. 6602–6607, 2019.

Johnson, R. and Zhang, T. Accelerating Stochastic Gradient Descent using Predictive Variance Reduction. In *Proceedings of Advances in Neural Information Processing Systems*, pp. 1–9, 2013.

Li, Y. Deep Reinforcement Learning: An Overview. *arXiv*, 1701.07274:1–85, 2018.

Ljung, L. Perspectives on System Identification. *Annual Reviews in Control*, 34(1):1–12, 2010.

Ma, H., Büchler, D., Schölkopf, B., and Muehlebach, M. A Learning-based Iterative Control Framework for Controlling a Robot Arm with Pneumatic Artificial Muscles. In *Proceedings of Robotics: Science and Systems*, pp. 1–10, 2022.

Ma, H., Büchler, D., Schölkopf, B., and Muehlebach, M. Reinforcement learning with model-based feedforward inputs for robotic table tennis. *Autonomous Robots*, 47 (8):1387–1403, 2023.

Ma, H., Zeilinger, M., and Muehlebach, M. Stochastic Online Optimization for Cyber-Physical and Robotic Systems. *arXiv*, 2404.05318:1–46, 2024.

Makoviychuk, V., Wawrzyniak, L., Guo, Y., Lu, M., Storey, K., Macklin, M., Hoeller, D., Rudin, N., Allshire, A., Handa, A., and State, G. Isaac Gym: High Performance GPU-Based Physics Simulation For Robot Learning. *arXiv*, 2108.10470:1–32, 2021.

Mueller, F. L., Schoellig, A. P., and D'Andrea, R. Iterative Learning of Feed-Forward Corrections for High-Performance Tracking. In *Proceedings of International Conference on Intelligent Robots and Systems*, pp. 3276–3281, 2012.

Neu, G. Explore no more: Improved high-probability regret bounds for non-stochastic bandits. In *Proceedings of Advances in Neural Information Processing Systems*, pp. 1–9, 2015.

Piazzi, A. and Visioli, A. An Interval Algorithm for Minimum-Jerk Trajectory Planning of Robot Manipulators. In *Proceedings of IEEE Conference on Decision and Control*, pp. 1924–1927, 1997.

Pintelon, R. and Schoukens, J. *System Identification: A Frequency Domain Approach*. John Wiley & Sons, 2012.

Robbins, H. and Monro, S. A Stochastic Approximation Method. *The Annals of Mathematical Statistics*, 22(3): 400–407, 1951.

Schmidt, M., Le Roux, N., and Bach, F. Minimizing Finite Sums with the Stochastic Average Gradient. *Mathematical Programming*, 162(1):83–112, 2017.

Schoellig, A. P., Mueller, F. L., and D'Andrea, R. Optimization-Based Iterative Learning for Precise Quadrocopter Trajectory Tracking. *Autonomous Robots*, 33(1):103–127, 2012.

Schulman, J., Moritz, P., Levine, S., Jordan, M., and Abbeel, P. High-Dimensional Continuous Control Using Generalized Advantage Estimation. *arXiv*, 1506.02438:1–14, 2018.

Shalev-Shwartz, S. Online Learning and Online Convex Optimization. *Foundations and Trends in Machine Learning*, 4(2):107–194, 2012.

Sutton, R. S. and Barto, A. G. *Reinforcement Learning: An Introduction*. MIT Press, 2018.

Tobuschat, P., Ma, H., Büchler, D., Schölkopf, B., and Muehlebach, M. Data-Efficient Online Learning of Ball Placement in Robot Table Tennis. In *Proceeding of IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 567–573, 2023.

Tsiamis, A. and Pappas, G. J. Finite Sample Analysis of Stochastic System Identification. In *Proceedings of IEEE Conference on Decision and Control*, pp. 3648–3654, 2019.

Wang, W., Li, R., Diekel, Z. M., and Jia, Y. Robot Action Planning by Online Optimization in Human–Robot Collaborative Tasks. *International Journal of Intelligent Robotics and Applications*, 2(2):161–179, 2018.

Zughaibi, J., Hofer, M., and D'Andrea, R. A Fast and Reliable Pick-and-Place Application with a Spherical Soft Robotic Arm. In *Proceedings of International Conference on Soft Robotics*, pp. 599–606, 2021.

Zughaibi, J., Nelson, B. J., and Muehlebach, M. Balancing a 3D Inverted Pendulum Using Remote Magnetic Manipulation. *arXiv*, 2402.06012:1–8, 2024.

Table 1. Summary of the parameters used for generating reference trajectories.

PARAMETER	DISTRIBUTION	UNIT
t_a	UNIFORM$(1.2, 1.8)$	s
t_b	UNIFORM$(2.9, 3.5)$	s
y_a, y_b	UNIFORM$(-0.2, 0.2)$	m
v_a, v_b	UNIFORM$(-2.0, 2.0)$	m/s

A. Cantilever Beam

A.1. Reference Trajectory

We randomly generate the reference trajectory based on the following principles: 1. Over a time span of T_{sim} seconds, the trajectory starts from rest and eventually returns to its initial position, and remains still for an additional $0.5\,\text{s}$. 2. Apart from the starting and ending points (y_0 and y_T), two other time points, t_a and t_b, will be randomly selected within the time duration T_{sim}. The displacements (y_a and y_b) and velocities (v_a and v_b) at these moments will also be randomly generated, with the accelerations being set to zero. 3. The four points are connected using trajectories that minimize jerk[3] (Geering, 2007; Piazzi & Visioli, 1997). The values of the various parameters are summarized in Table 1.

Figure 4 illustrates the sampling procedure for the reference trajectories along with 400 samples. The total duration is set to $T_{\text{sim}} = 5.5\,\text{s}$. The red dashed boxes indicate the spatial and temporal distribution range of the points y_a and y_b, respectively.

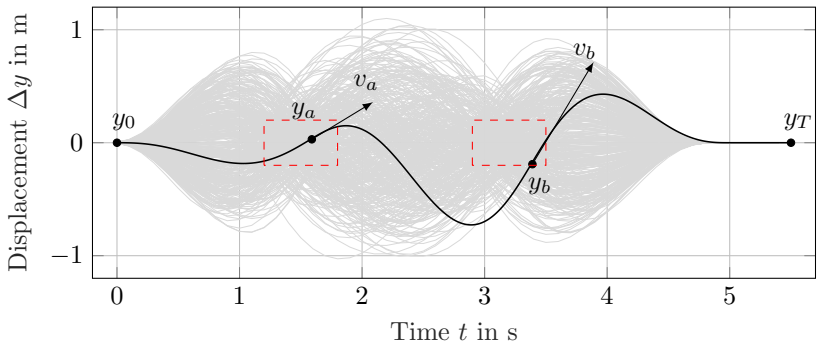


Figure 4. The figure illustrates the range of reference trajectories used for training, where the gray lines are composed of 400 randomly sampled reference trajectories. The red dashed boxes indicate the spatial and temporal distribution range of the points y_a and y_b, respectively.

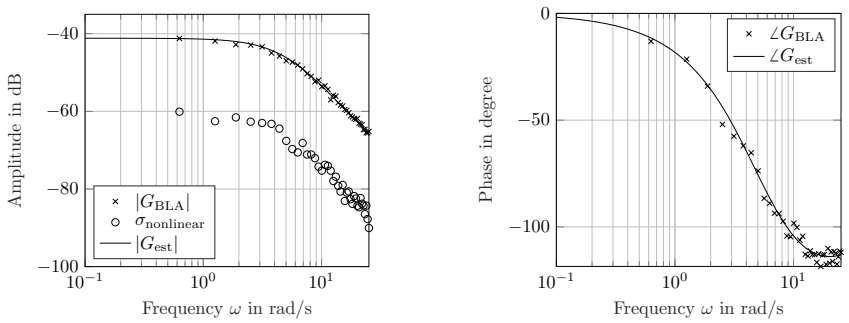
A.2. Gradient Estimate

To estimate the gradient of the system G with respect to the inputs u, we first excite the model in `Simulink` with an excitation signal ranging from $0\,\text{Hz}$ to $4\,\text{Hz}$, with an interval of $0.1\,\text{Hz}$ to get a linear transfer function. The resulting system response in the frequency domain and the estimated linear transfer function are shown in Figure 5. Then, we use the obtained transfer function to construct a linear approximation of $\partial G(s_0,u;\zeta)/\partial u$, which is denoted by $\mathcal{G}$. For the specific construction method, please refer to Ma et al. (2022; 2023).

A.3. Model Structure

The structure of the networks is illustrated in Figure 6. The policy network π_{ff} takes in a horizon of h_1 steps in the past and h_2 steps in the future to produce the input $u_{k,\text{ff}}$ at time k (see Figure 6a), while π_{fb} takes only h_1 steps in the past to produce $u_{k,\text{fb}}$(see Figure 6b). In instances where the horizon surpasses the range of the reference trajectory, we employ a

[3]Jerk is defined as the derivative of acceleration of the third derivative of displacement.

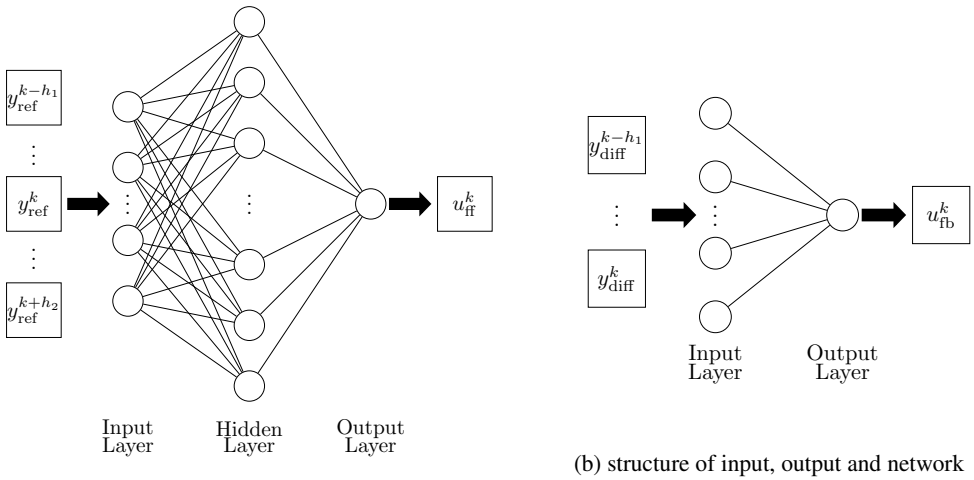


(a) amplitude diagram

(b) phase diagram

Figure 5. The figure displays the amplitude diagram (left) and phase diagram (right) of the system response in the frequency domain. Crosses represent the measured data obtained through system identification in frequency domain, while the solid line represents the fitted transfer function. The nonlinearity system is denoted by circles in the amplitude diagram.

zero-padding strategy to compensate for the absent elements.



(a) structure of input, output and network for π_{ff}

(b) structure of input, output and network for π_{fb}

Figure 6. The figure illustrates the input, output, and network structures of both π_{ff} and π_{fb}. The feedforward network π_{ff}, which is a fully connected network, utilizes the reference trajectory at time k, as well as the reference trajectories for the horizons of h_1 and h_2 before and after this time, as its input. The output is the corresponding feedforward input $u_{k,\text{ff}}$ at time k. On the other hand, the feedback network π_{fb}, a linear network, employs the trajectory difference for a horizon of h_1 units leading up to time k as its input. The output is the respective feedback input $u_{k,\text{fb}}$ at time k. The trajectory difference is defined as the difference between the output trajectory and the reference trajectory.

A.4. Results

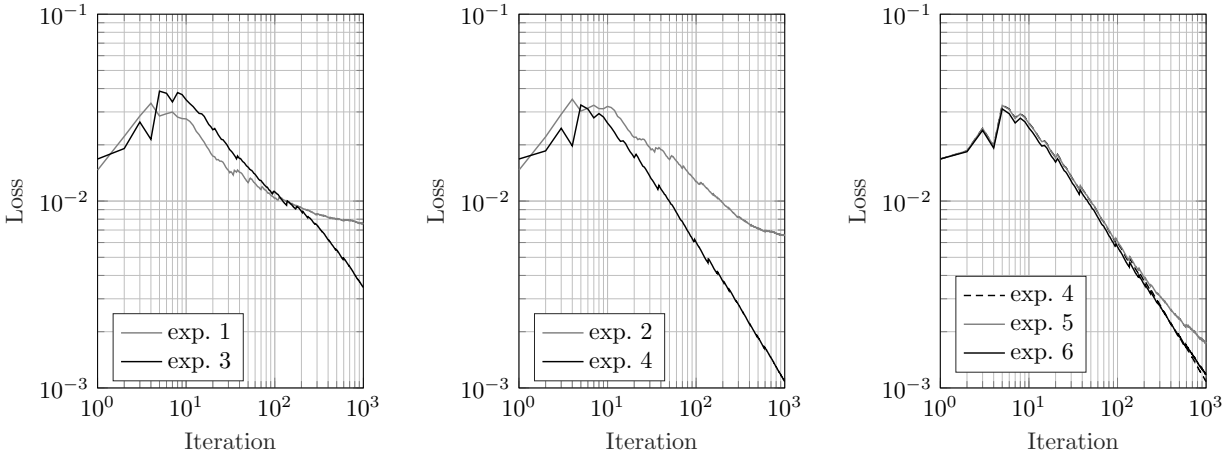
The overview of different experiments is presented in Table 2. In each experiment, we train the parameterized networks for 1000 iterations. The average loss δ_t is given by

$$\delta_t = \frac{1}{t} \sum_{k=1}^{t} l\left(y_k; y_{\text{ref},k}\right), \; t = 1, \ldots, T.$$

The convergence results of Experiments 1 and 3 are illustrated in Figure 7a, while the results of Experiments 2 and 4 are shown in Figure 7b. In Experiment 5, we artificially introduce noise n_{d} to render the purely feedforward control ineffective

Table 2. Overview of parameters, network configurations, and experimental results.

NO.	MODEL(S)	NOISE	h_1	h_2	HIDDEN NEURONS	ϵ	α	η	AVERAGE LOSS
1	π_{ff}^1	$\times$	100	100	-	$+\infty$	-	0.1	6.90×10^{-3}
2	π_{ff}^1	$\times$	100	100	-	1.0	0.1	15.0	6.30×10^{-3}
3	π_{ff}^2	$\times$	100	100	40	$+\infty$	-	0.1	8.27×10^{-4}
4	π_{ff}^2	$\times$	100	100	40	1.0	0.1	15.0	3.19×10^{-4}
5	π_{ff}^2	$\surd$	100	100	40	1.0	0.1	15.0	1.10×10^{-3}
6	π_{ff}^2 π_{fb}	$\surd$	100 25	100 -	40 -	1.0	0.1	15	4.65×10^{-4}



(a) results of exp. 1 and 3

(b) results of exp. 2 and 4

(c) results of exp. 4, 5 and 6

Figure 7. This figure depicts the convergence results $\delta_t,\ t = 1, \ldots, 1000$ of different experiments.

in trajectory tracking and introduce a feedback controller in Experiment 6 to reject noise, with the convergence results shown in Figure 7c.

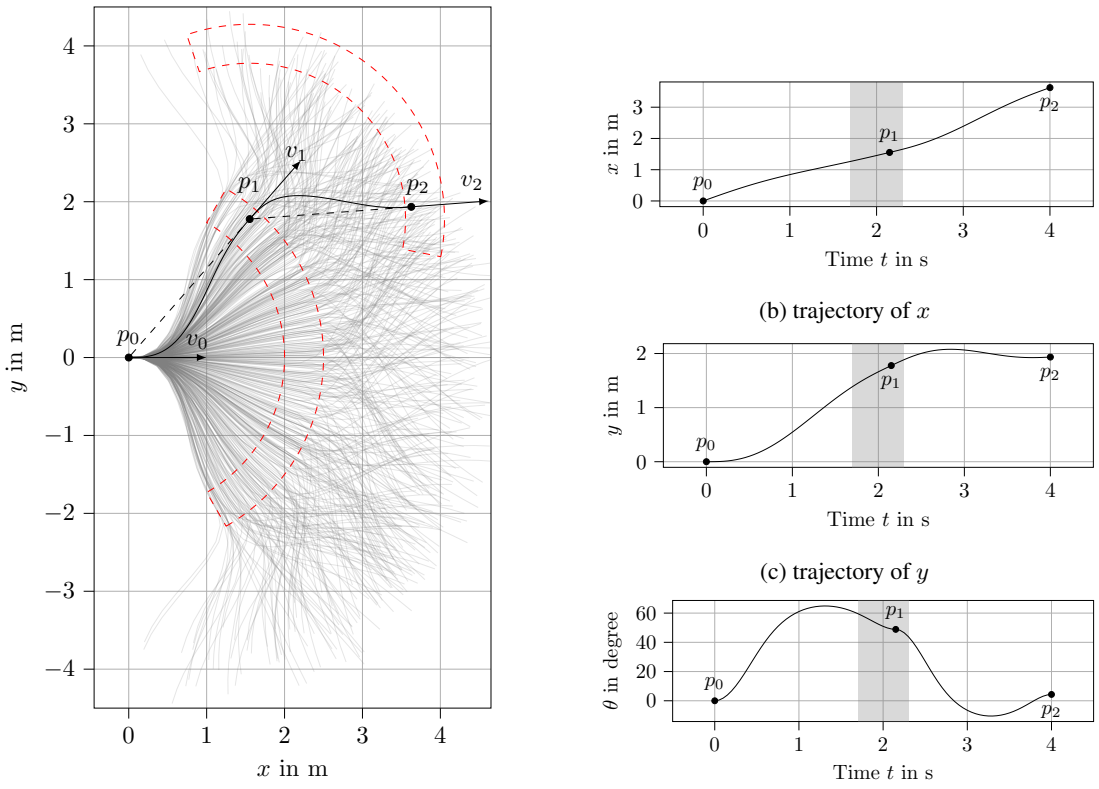
B. Four-Legged Robot

B.1. Reference Trajectory

Figure 8 displays the distribution of the reference trajectories used for tracking. We take one of the sampled trajectories as an example to illustrate the general rules for generating reference trajectories. The trajectories are generated over a time duration of $T_{\text{sim}} = 4\,\text{s}$. The starting point p_0 is fixed at the point $[0, 0]^{\text{T}}$ in the x-y plane at time $t_0 = 0\,\text{s}$, and the initial velocity v_0 is also fixed at $1\,\text{m}\,\text{s}^{-1}$ directed along the positive x-axis. Next, we uniformly generate the point p_1 within a disk centered around p_0 with radii of $2\,\text{m}$ and $2.5\,\text{m}$, and an angular span of $\pm 60^\circ$ centered around v_0 (see the red dashed disk). The velocity v_1 at p_1 is also set to $1\,\text{m}\,\text{s}^{-1}$, in the direction of the line from p_0 to p_1. The time t_1 for generating p_1 is uniformly within a range of $\pm 0.3\,\text{s}$ centered around $t = 2\,\text{s}$. Based on the point p_1, the point p_2 and its corresponding velocity v_2 are generated in the same manner, with the time point $t_2 = 4\,\text{s}$ being fixed for p_2. The acceleration at each point is set to zero. Finally, we connect these three points using a trajectory that minimizes jerk.

B.2. Gradient Estimate

At each iteration, we run n_{env} (here $n_{\text{env}} = 2000$) identical environments in parallel in addition to the nominal environment. The nominal input $u_t,\ t = 1, \ldots, T$, is fed into the nominal environment, yielding the corresponding nominal output y_t. For the remaining parallel environments, normally distributed noise n_{d} with a mean of zero and a variance of one is added to



(a) distribution of reference trajectories

(b) trajectory of x

(c) trajectory of y

(d) trajectory of yaw

Figure 8. The left subfigure shows the area generated by 500 randomly sampled reference trajectories used for tracking (depicted as grey lines), along with an example reference trajectory to illustrate the rules for generating trajectories (shown as a black line). The red dashed disks represent the distribution range for the position of the next point in the x-y plane, assuming the previous point is determined. The radial gap of the disk is $0.5\,\mathrm{m}$, with an angular span $\pm\,60^\circ$ centered around the tangent direction at the previous point. The right subfigure illustrates the temporal evolution of the example reference trajectory, showing its x, y, and yaw components. The time points for generating points p_0 and p_2 are fixed. The grey areas in the right subfigures represent the time range for point p_1, which is centered around $t = 2\,\mathrm{s}$ with a permissible deviation of $\pm 0.3\,\mathrm{s}$.

the nominal input u_t (see Figure 1), denoted as $\widetilde{u}_{t,i},\ i = 1, \ldots, n_{\text{env}}$, resulting in the respective outputs $\widetilde{y}_{t,i}$. Finally, we estimate the system gradient $\mathcal{G}(u_t)$ using least squares[4]:

$$\mathcal{G}(u_t)^{\mathrm{T}} = \begin{bmatrix} (\widetilde{u}_{t,1} - u_t)^{\mathrm{T}} \\ (\widetilde{u}_{t,2} - u_t)^{\mathrm{T}} \\ \vdots \\ (\widetilde{u}_{t,n_{\text{env}}} - u_t)^{\mathrm{T}} \end{bmatrix}^{\dagger} \begin{bmatrix} (\widetilde{y}_{t,1} - y_t)^{\mathrm{T}} \\ (\widetilde{y}_{t,2} - y_t)^{\mathrm{T}} \\ \vdots \\ (\widetilde{y}_{t,n_{\text{env}}} - y_t)^{\mathrm{T}} \end{bmatrix},$$

where we stack all inputs and outputs by columns respectively.

B.3. Neural Network with Pre-Trained Motion Patterns

In order to enable online learning with the ant model, we parameterize our networks as follows:

$$\pi_{\text{ff}}(\omega_{\text{ff}}; \zeta) = U\phi\left(\omega_{\text{ff}}; V^{\mathrm{T}}\zeta\right),\ \pi_{\text{fb}} = 0,$$

where the matrices $U \in \mathbb{R}^{mq \times n_\sigma}$ and $V \in \mathbb{R}^{nq \times n_\sigma}$ represent linear transformation to a lower dimensional latent space and $\phi : \mathbb{R}^{n_{\omega_{\text{ff}}}} \times \mathbb{R}^{n_\sigma} \to \mathbb{R}^{mq}$ is a neural network comprising one hidden layer.

[4]In the experiment, only the ants that remain upright until the end are considered for estimating the gradient.

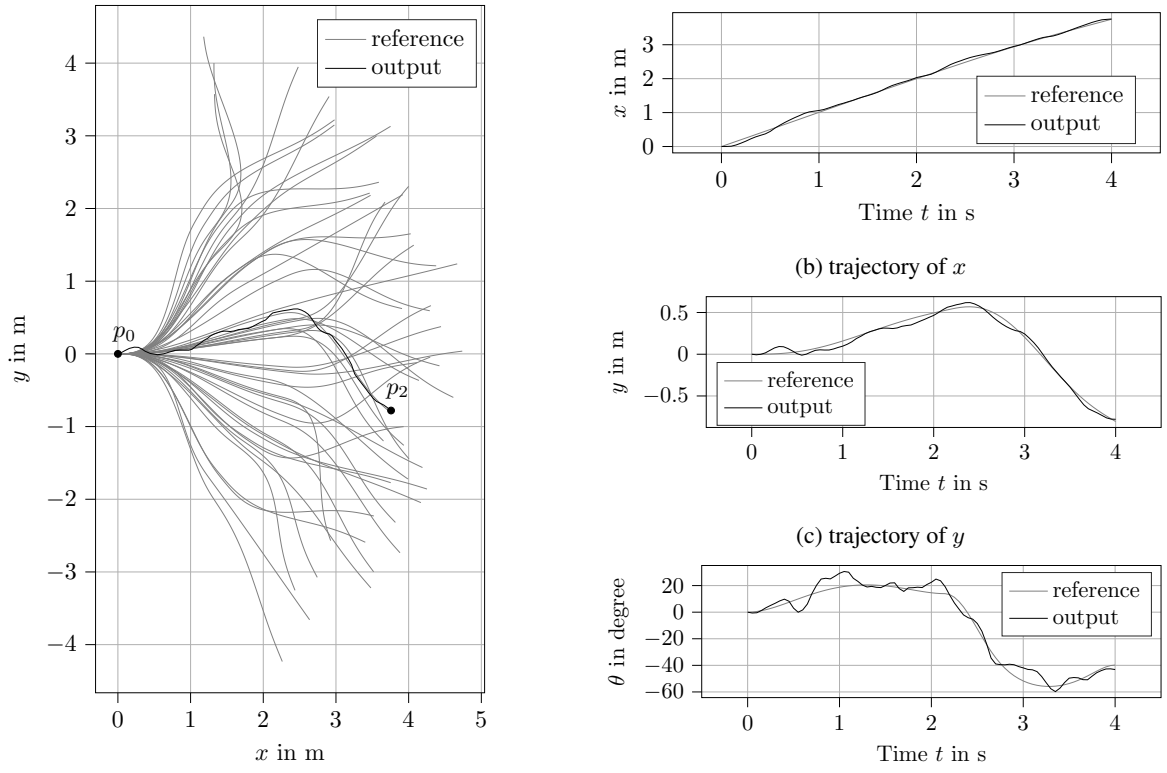
The matrices U and V are obtained through the singular value decomposition of the matrix R:

$$R = U \operatorname{diag}(\sigma) V^{\mathrm{T}},$$

where the matrix R is derived by solving the following ridge regression:

$$\min_{R \in \mathbb{R}^{mq \times nq}} \frac{1}{2} \sum_{i=1}^{n_{\mathrm{ILC}}} |u_{\mathrm{ref},i} - R y_{\mathrm{ref},i}|^2 + \frac{\rho}{2} \|R\|_{\mathrm{F}}^2,$$

where ρ is a positive constant, and $\|\cdot\|_{\mathrm{F}}$ denotes the Frobenius norm. The ideal input u_{ref} represents the input required for accurately tracking a given reference trajectory y_{ref}. The ideal input is unknown, and we therefore employ iterative learning control (ILC) to approximate it (Ma et al., 2022; Hofer et al., 2019; Zughaibi et al., 2021; Mueller et al., 2012; Schoellig et al., 2012; Zughaibi et al., 2024). The variable n_{ILC} denotes the number of pre-trained trajectories using ILC. In this experiment, we sample 50 reference trajectories and get their corresponding ideal inputs using ILC, and each reference trajectory takes 200 to 300 iterations to obtain the ideal inputs. We then use 45 trajectories (90%) along with their ideal inputs to perform ridge regression. Figure 9 displays the distribution composed of all 50 reference trajectories used for pre-training in the left subfigure, whereas the right subfigure showcases the final training result of ILC for one reference trajectory as an example. The right subfigures illustrate the tracking performance of the corresponding x, y, and yaw components of this trajectory. We note that the tracking of the x and y components by the ILC is very effective; however, due to the presence of collisions, the tracking of the yaw component is slightly less accurate. Nevertheless, we consider this as a sufficiently good ideal input for tracking the given reference trajectory, which is able to capture the motion patterns.



(a) distribution of trajectories for ILC

(b) trajectory of x

(c) trajectory of y

(d) trajectory of yaw

Figure 9. The left subfigure shows the distribution of 50 randomly sampled reference trajectories used for pre-training (represented by gray lines), and illustrates the tracking performance of ILC with one of these trajectories (depicted as a black line). The right subfigures demonstrate the tracking performance of the ILC for the particular trajectory in the x, y, and yaw components.

Table 3. Parameters for training the ant model.

NO.	MODEL	n_σ	HIDDEN NEURONS	ϵ	α	η
1	π_{ff}	90	45	$+\infty$	-	DIMINISHING
2	π_{ff}	45	20	0.1	0.5	0.1

B.4. Results

In this experiment, we use a fully connected network with only one hidden layer containing 20 neurons. The hidden layer employs the `ReLU` activation function, while the output layer does not have an activation function. We employ different methods to train the network, and the parameters are shown in Table 3.

The two experiments were conducted with over 1500 and 3500 iterations, respectively, and their convergence results are shown in Figure 10.

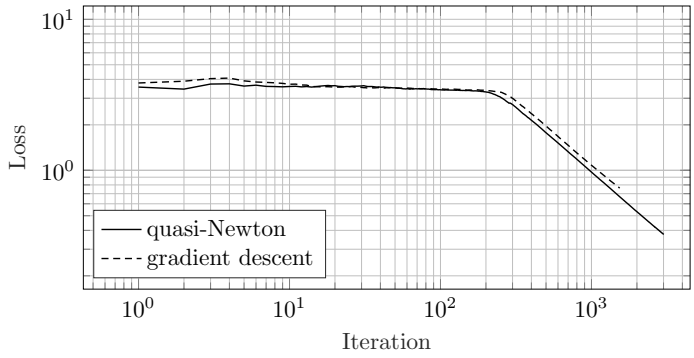


Figure 10. The figure shows the convergence results of the gradient descent and quasi-Newton method. The gray line represents the average loss of the gradient descent method, and the black line indicates the average loss of the quasi-Newton method.