Robust Jumping with an Articulated Soft Quadruped via Trajectory Optimization and Iterative Learning

Ding, Jiatao; Sels, Mees A.van Loben; Angelini, Franco; Kober, Jens; Santina, Cosimo Della

DOI
10.1109/LRA.2023.3331288

Publication date
2023

Document Version
Final published version

Published in
IEEE Robotics and Automation Letters

Robust Jumping With an Articulated Soft Quadruped Via Trajectory Optimization and Iterative Learning

Jiatao Ding, Mees A. van Löben Sels, Franco Angelini, *Member, IEEE*, Jens Kober, *Senior Member, IEEE*, and Cosimo Della Santina, *Senior Member, IEEE*

***Abstract*—Quadrupeds deployed in real-world scenarios need to be robust to unmodelled dynamic effects. In this work, we aim to increase the robustness of quadrupedal periodic forward jumping (i.e., pronking) by unifying cutting-edge model-based trajectory optimization and iterative learning control. Using a reduced-order soft anchor model, the optimization-based motion planner generates the periodic reference trajectory. The controller then iteratively learns the feedforward control signal in a repetition process, without requiring an accurate full-body model. When enhanced by a continuous learning mechanism, the proposed controller can learn the control inputs without resetting the system at the end of each iteration. Simulations and experiments on a quadruped with parallel springs demonstrate that continuous jumping can be learned in a matter of minutes, with high robustness against various types of terrain.**

***Index Terms*—Legged robots, optimization and optimal control, motion control.**

I. Introduction

LEGGED robots are expected to take on various tasks such as industrial inspection, surveillance, and outdoor monitoring [1], [2], [3]. To achieve high traversability of quadrupedal robots, a robust dynamic motion such as jumping, needs further investigation. Joint compliance of articulated soft robots [4] such as Spacebok [5], Birdbot [6] and ANYmal [7], provides a promising solution towards this goal. For this reason, in this work, we focus on a quadrupedal robot with parallel elastic actuation (PEA). Although there are many implementations of PEA in quadrupedal locomotion, realizing highly robust periodic forward jumping, i.e., pronking, with a PEA-driven quadruped is still an open challenge.

Based on the full-body or centroidal dynamic models, various planners such as [8], [9], [10], [11], [12], [13] have been proposed to generate jumping motion, using trajectory optimization (Topt) techniques that can deal with a large number of constraints [14], [15]. However, they are applied to rigid robots without considering parallel elasticity. Instead, the spring-loaded inverted pendulum (SLIP) model [16], [17] is able to capture the system compliance. The reduced-order template, together with its variants such as 3D SLIP [18], flywheel SLIP [19] and p-SLIP [20] makes the problem of generating jumping motions more tractable, which however usually ignores the actuation limits. To tackle this issue, a compliant anchor model with a similar morphology to a real quadruped could be used, without causing a heavy computing burden.

Once the trajectory is planned, the main obstacle is designing a robust controller accounting for the highly-nonlinear underactuated dynamics and the intense contact with uncertain environments. The standard approach to execute jumping motions is feedback control [9], [10], [11], even with the exploitation of the natural dynamics of elastic legged systems [21], [22], [23], [24]. However, feedback alters the stiffness of elastic systems with a factor proportional to the feedback gain [25], defeating the purpose of introducing physical compliance. In this regard, feedforward control with minimal reliance on feedback is preferable [4]. For quadrupedal jumping, feedforward torque compensation could be realized by quadratic programming [26] or model predictive control (MPC) [8], [27], yet they are only applied to rigid quadrupedal robots. Recent work in [28] uses normal mode theory to exploit and stabilize modal oscillations for efficient locomotion of an elastic quadrupedal robot, but no jumping motion is executed. Differing from the above approaches, iterative learning control (ILC) [29] methods learn the control inputs through repetitive iterations, without requiring any accurate models. In addition, the feedforward nature of ILC in the time domain preserves the physical compliance of the system, and the feedback in the iteration domain enhances the tracking performance. ILC has already been applied to articulated soft robots such as [30], [31], [32], however, the application to quadrupedal jumping has never been found before.

In this work, we make a step towards unifying trajectory optimization and iterative learning for robust jumping of a PEA-driven quadruped, see Fig. 1. Inspired by the SLIP model, the trajectory optimizer generates periodic jumping motions, based on a complaint single-body dynamics (CSBD) model with a similar morphology to that of the quadrupedal system. Then the ILC learns the control inputs through feedback in the iteration domain, omitting the need of identifying the compliant full-body

Manuscript received 30 April 2023; accepted 20 October 2023. Date of publication 8 November 2023; date of current version 22 November 2023. This letter was recommended for publication by Associate Editor H. S. Ahn and Editor A. Faust upon evaluation of the reviewers' comments. This work was supported in part by the European Union's Horizon 2020 Research and Innovation Programme under Grant Agreement 101016970 (Natural Intelligence) and in part by Ministry of University and Research (MUR) as part of the PON 2014-2021 "Research and Innovation" resources - Green/Innovation Action - DM MUR 1062/2021. *(Jiatao Ding and Mees A. van Löben Sels contributed equally to this work.) (Corresponding author: Jiatao Ding.)*

Jiatao Ding, Mees A. van Löben Sels, and Jens Kober are with the Department of Cognitive Robotics, Delft University of Technology, 2628 CD Delft, The Netherlands (e-mail: j.ding–2@tudelft.nl; M.A.vanLobenSels@student.tudelft.nl; j.kober@tudelft.nl).

Franco Angelini is with the Centro di Ricerca "Enrico Piaggio" and the Dipartimento di Ingegneria dell'Informazione, Università di Pisa, 56126 Pisa, Italy (e-mail: frncangelini@gmail.com).

Cosimo Della Santina is with the Department of Cognitive Robotics, Delft University of Technology, 2628 CD Delft, The Netherlands, and also with the Institute of Robotics and Mechatronics, German Aerospace Center (DLR), 82234 Wessling, Germany (e-mail: cosimodellasantina@gmail.com).

Digital Object Identifier 10.1109/LRA.2023.3331288

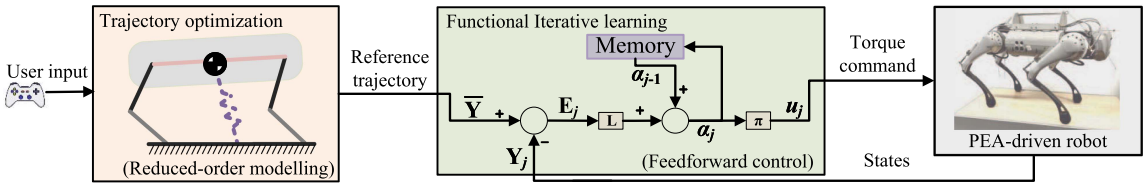

Fig. 1. Overview of the proposed jumping control framework for the PEA-driven quadruped. From left to right: The user provides a desired forward velocity, and the trajectory optimizer uses a simplified model to generate a periodic jumping motion that minimizes the cost of transport. The functional iterative learning stage generates a continuous evolution of torque control action to implement the desired trajectory. The direction of learning is guided by the knowledge of the approximate model.

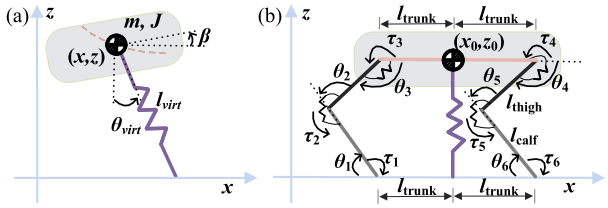

Fig. 2. TSLIP at touchdown (a) and the CSBD anchor model in homing configuration (b).

dynamics. Specifically, we extend functional ILC (fILC) [33] to nonlinear systems to track the optimal trajectory at selected time instances of interest. To avoid resetting the system state after each iteration, we introduce a continuous learning mechanism. The proposed strategy is tested in simulations and validated on the Delft E-Go robot with parallel elastic actuators.

The contributions are three-fold:

1) We formulate a SLIP-inspired trajectory optimizer that generates periodic jumping motion explicitly considering the parallel elasticity, by employing a CSBD template with a similar morphology to a real quadrupedal.
2) We introduce fILC for nonlinear robust jumping control, alleviating the need to build an accurate full-body dynamic model. Enhanced by a continuous learning mechanism, the fILC can find optimal actions for pronking through unseen scenarios within a few minutes, without resetting the system state after each iteration.
3) We preliminarily validate our approach on a newly-designed PEA-driven robot. The experiments demonstrate the stability and robustness of the proposed framework.

II. SLIP-Inspired Jumping Motion Optimization

In this section, we first introduce an anchor CSBD model to describe the sagittal quadrupedal jumping motion, by bridging the trunk SLIP model with the full-body model. Then, we derive the reduced-order jumping dynamics with explicitly considering parallel compliance. Finally, we present the Topt for jumping motion generation

A. CSBD: Bridging the SLIP and Quadruped

The trunk SLIP (TSLIP), extending the naive SLIP [17] by adding a rotating trunk on the CoM (see Fig. 2(a)), is employed to capture the quadrupedal jumping dynamics. During the stance phase, the TSLIP motion is governed by the ground reaction force and the torque at the trunk while during the flight phase, governed by gravity. In the sagittal plane, the system states include the CoM position (consists of forward position (x) and vertical position (z)) and the pitch angle of the rotating trunk (β). Besides, we assume the robot touches down with a virtual touchdown angle (θ_{virt}) and a virtual length (l_{virt}), as illustrated in Fig. 2(a).

The TSLIP template behaviour is then embedded into a more realistic anchor model whose morphology is closer to that of the actual system, through adding links, actuators, and elastic joints, as depicted in Fig. 2(b). To bridge the TSLIP and quadrupedal, we propose the following mapping rules:

1) *legs are massless, formulating a single-mass model,*
2) *each leg joint is enhanced with a parallel spring,*
3) *the leg position in the TSLIP model is in the middle of the hind and front foot positions in the anchor model.*

In addition, we assume that the two feet touch down and lift off simultaneously, i.e., pronking gait. As a result, two feet positions can be parameterized at touchdown (TD) using touchdown angle θ_{virt}, leg length l_{virt} (see Fig. 2(a)) and the trunk length l_{trunk} (see Fig. 2(b)).

B. Hybrid Reduced-Order Jumping Dynamics

The configuration of the 2D sagittal motion can be represented by the special Euclidean group $SE(2)$, parameterized by the generalized coordinates $\boldsymbol{q} = [x, z, \beta]^\mathsf{T} \in \mathbb{R}^3$. The system inputs are the motor torques $\boldsymbol{\tau} = [\tau_2, \tau_3, \tau_4, \tau_5]^\mathsf{T} \in \mathbb{R}^4$ at the thigh and calf joints, as can be seen in Fig. 2(b).

1) Flight Dynamics: During the flight phase, the system follows a ballistic trajectory, which is modelled as

$$\mathbf{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \mathbf{G}(\boldsymbol{q}) = \mathbf{0}, \tag{1}$$

where $\mathbf{M} \in \mathbb{R}^{3\times3}$ is the mass matrix and $\mathbf{G} \in \mathbb{R}^3$ is the gravitational term.

2) Stance Dynamics: During the stance phase, the generalized coordinates $\boldsymbol{q}$ are mapped to the joint angles $\boldsymbol{\theta} \in \mathbb{R}^6$ through the mapping function $\boldsymbol{h} : \boldsymbol{q} \mapsto \boldsymbol{\theta}$, where $\boldsymbol{h}$ is obtained using inverse kinematics. In joint space, the contribution of the parallel spring can be easily captured [28]. The equation of motion (EoM) of the anchor model is obtained using Lagrangian mechanics, resulting in the stance dynamics

$$\mathbf{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \mathbf{G}(\boldsymbol{q}) + \mathbf{J_h}^\mathsf{T}(\boldsymbol{q})\mathbf{K}\left(\boldsymbol{\theta} - \boldsymbol{\theta}_0\right) = \boldsymbol{\mathcal{F}}, \tag{2}$$

where $\mathbf{K} \in \mathbb{R}^{3\times3}$ is the spring stiffness matrix, $\boldsymbol{\theta}_0$ is the spring rest positions, $\mathbf{J_h} = \frac{\mathrm{d}\boldsymbol{\theta}}{\mathrm{d}\boldsymbol{q}} \in \mathbb{R}^{6\times3}$ is the Jacobian that maps from joint space to generalized coordinates, and $\boldsymbol{\mathcal{F}} \in \mathbb{R}^3$ is the spatial wrench which is a function of the motor torques $\boldsymbol{\tau}$. The derivation of EoM (2) is explained in the Appendix.

Solving (1) and (2) for the accelerations $\ddot{\boldsymbol{q}}$ leads to

$$\begin{aligned} \ddot{\boldsymbol{q}} &= -\mathbf{M}^{-1}\mathbf{G} && \text{(flight)} \\ \ddot{\boldsymbol{q}} &= \mathbf{M}^{-1}\left(\boldsymbol{\mathcal{F}} - \mathbf{G} - \mathbf{J_h}^\mathsf{T}\mathbf{K}\left(\boldsymbol{\theta} - \boldsymbol{\theta}_0\right)\right) && \text{(stance)} \end{aligned} \tag{3}$$

Choosing the system state $\boldsymbol{x} = \left[\boldsymbol{q}, \dot{\boldsymbol{q}}\right]^{\mathsf{T}} \in \mathbb{R}^6$ and control input $\boldsymbol{u} = \boldsymbol{\tau} \in \mathbb{R}^4$, (3) is then rewritten as

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}). \tag{4}$$

C. Topt Formulation

In this work, one jumping stride is divided into three phases, i.e., an initial flight phase, a stance phase and a final flight phase, which are separated by a TD event and a takeoff (TO) event. The optimization problem is then transcribed using multiple shooting by dividing it into $N-1$ intervals with N grid points. The TD and TO events separately take place at the n_{TD}-th and n_{TO}-th grid points. The time step $\mathrm{d}t$ of each interval is also automatically chosen by the Topt problem, which is formulated as

$$\begin{aligned} &\min_{\substack{\mathrm{d}t, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_N \\ \boldsymbol{u}_1, \ldots, \boldsymbol{u}_{N-1}}} \text{Objective}, \\ &\text{s.t. } \textit{Feasibility constraints}. \end{aligned} \tag{5}$$

1) Objective Function: The objective of the optimization problem is to find a periodic jumping trajectory. To this end, we minimize the cost of transport (CoT)[1] while obeying the periodicity constraints. Since CoT is defined as the energy input E required to move a system of weight mg over a distance d [34], the objective function is then formulated as

$$\text{CoT} = \frac{E}{mgd} = \frac{\sum_{n=1}^{N} \sum_{i=2}^{5} \left| \tau_i^n \dot{\theta}_i^n \right|}{mg x_N}, \tag{6}$$

where x_N is the longitudinal position on the final node.

2) Feasibility Constraints: Various constraints are taken into consideration in the Topt problem, including

Initial condition: The stride starts at an apex, i.e. at zero vertical velocity, and at zero initial horizontal position, i.e.,

$$\left[x_1, \dot{z}_1\right]^{\mathsf{T}} = \boldsymbol{0}. \tag{7}$$

Periodicity constraints: The periodicity is enforced by constraining the final state $\boldsymbol{x}_N$ equate to the initial state $\boldsymbol{x}_1$ except for the longitudinal position, formulated as

$$\boldsymbol{x}_1 \backslash \{x_1\} = \boldsymbol{x}_N \backslash \{x_N\}. \tag{8}$$

Dynamics constraints: The system dynamics of the CSBD model are integrated within each step using the EoM in (4). The following constraint is set at each grid point to guarantee the continuity between the adjacent nodes:

$$\boldsymbol{x}_{n+1} = \boldsymbol{F}(\boldsymbol{x}_n, \boldsymbol{u}_n, \boldsymbol{f}(\boldsymbol{x}_n, \boldsymbol{u}_n), \mathrm{d}t), \tag{9}$$

where $\boldsymbol{F}(\cdot)$ are determined by state $\boldsymbol{x}_n$ and control input $\boldsymbol{u}_n$. Various numerical processes can be used to this end, and we use the fourth-order Runge-Kutta algorithm.

In (9), the dynamic transfer is chosen in the following way such that the system is in flight before n_{TD} and after n_{TO},

$$\boldsymbol{f}(\boldsymbol{x}_n, \boldsymbol{u}_n) = \begin{cases} \boldsymbol{f}_{\text{flight}}(\boldsymbol{x}_n, \boldsymbol{u}_n) & \forall n \in [1, n_{\text{TD}}) \cup [n_{\text{TO}}, N] \\ \boldsymbol{f}_{\text{stance}}(\boldsymbol{x}_n, \boldsymbol{u}_n) & \forall n \in [n_{\text{TD}}, n_{\text{TO}}) \end{cases}, \tag{10}$$

where $\boldsymbol{f}_{\text{flight}}$ and $\boldsymbol{f}_{\text{stance}}$ are determined in (4) by using the first and second row in (3), respectively.

Switch conditions: Constraints are imposed at the switching nodes to guarantee kinematical feasibility. A TD event is formulated as the moment when the system state is in the touchdown manifold, given by

$$\boldsymbol{x}_{n_{\text{TD}}} \in \boldsymbol{\mathcal{X}}_{\text{TD}} = \{\boldsymbol{x} \mid z - l_{\text{virt}} \cos(\theta_{\text{virt}}) = 0,\ \dot{z} < 0\}. \tag{11}$$

Similarly, a TO event occurs when the system state is in the take-off manifold, given by

$$\boldsymbol{x}_{n_{\text{TO}}} \in \boldsymbol{\mathcal{X}}_{\text{TO}} = \{\boldsymbol{x} \mid \sqrt{x^2 + z^2} - l_{\text{virt}} = 0,\ \dot{z} > 0\}. \tag{12}$$

State limits: Robot states are constrained by

$$\boldsymbol{x}_{\min} \leq \boldsymbol{x}_n \leq \boldsymbol{x}_{\max}, \tag{13}$$

where $\boldsymbol{x}_{\min}$ and $\boldsymbol{x}_{\min}$ are the lower and upper boundaries.

Actuator constraints: Input torques are limited by

$$\boldsymbol{u}_{\min} \leq \boldsymbol{u}_n \leq \boldsymbol{u}_{\max}, \tag{14}$$

where $\boldsymbol{u}_{\min}$ and $\boldsymbol{u}_{\max}$ are determined by actuation capability.

Time step constraints: Time step $\mathrm{d}t$ is limited by

$$\mathrm{d}t_{\min} \leq \mathrm{d}t \leq \mathrm{d}t_{\max}, \tag{15}$$

where $\mathrm{d}t_{\min}$ and $\mathrm{d}t_{\max}$ are determined by the shortest and longest jumping stride.

III. Iterative Learning-Based Jumping Control

To control the quadruped with parallel compliance, we propose the use of fILC to overcome the dynamic uncertainties by iteratively learning the feedforward control signal. A block diagram of the controller is depicted in Fig. 1.

A. Functional Iterative Learning Control

1) Background: Originally, fILC was developed for linear systems [33]. Given a linear continuous system

$$\dot{\boldsymbol{x}}_j(t) = \mathbf{A}\boldsymbol{x}_j(t) + \mathbf{B}\boldsymbol{u}_j(t), \quad \boldsymbol{y}_j(t) = \mathbf{C}\boldsymbol{x}_j(t), \tag{16}$$

with iteration index j, $\mathbf{A} \in \mathbb{R}^{n \times n}$, $\mathbf{B} \in \mathbb{R}^{n \times l}$, $\mathbf{C} \in \mathbb{R}^{m \times n}$, $\boldsymbol{x}_j \in \mathbb{R}^n$, $\boldsymbol{u}_j \in \mathbb{R}^l$, $\boldsymbol{y}_j \in \mathbb{R}^m$, and $l < m$, fILC learns the feedforward control signals obeying the following principles:

1) Differing from the classical ILC approaches, fILC does not track the reference $\bar{\boldsymbol{y}}$ completely, but rather tracks it at certain time instances of interest, given by $\{T^1, \ldots, T^o\}$, where o is the number of time instances and the superscript denotes the index. Denoting the desired outputs (obtained by the above motion planner) at the time instances as $\{\bar{\boldsymbol{y}}^1 \ldots \bar{\boldsymbol{y}}^o\}$, the goal of the controller then is to iteratively learn the control input $\boldsymbol{u}_j(t)$, such that
$$\lim_{j \to \infty} \boldsymbol{y}_j(T^k) = \bar{\boldsymbol{y}}^k, \quad \forall k \in 1, \ldots, o. \tag{17}$$
2) The control input $\boldsymbol{u}_j(t)$ is learned in a functional subspace of continuous basis functions $\boldsymbol{\pi}$. In particular, parameterized as a linear combination of functions, the control input is given by
$$\boldsymbol{u}_j(t) = \boldsymbol{\pi}(t)\boldsymbol{\alpha}_j, \quad \boldsymbol{\pi} = \left[\boldsymbol{\pi}^1 \ldots \boldsymbol{\pi}^o\right] \in \mathbb{R}^{l \times mo}, \tag{18}$$
where $\boldsymbol{\pi}$ is a matrix of basis functions, with $\boldsymbol{\pi}^i \in \mathbb{R}^{l \times m}$ and weight vector $\boldsymbol{\alpha}_j \in \mathbb{R}^{mo}$.

The closed form solution of (16) is

$$\boldsymbol{y}_j(t) = \mathbf{C}e^{\mathbf{A}t}\boldsymbol{x}(0) + \mathbf{C}\int_0^t e^{\mathbf{A}(t-\tau)}\mathbf{B}\boldsymbol{u}_j(\tau)\mathrm{d}\tau. \tag{19}$$

Substituting (18) and then sampling (19) at the i-th time instance, we have

$$\boldsymbol{y}_j\left(T^i\right) = \mathbf{C}e^{\mathbf{A}T^i}\boldsymbol{x}(0) + \left(\int_0^{T^i} \mathbf{C}e^{\mathbf{A}\left(T^i-\tau\right)}\mathbf{B}\boldsymbol{\pi}(\tau)\mathrm{d}\tau\right)\boldsymbol{\alpha}_j. \tag{20}$$

[1]CoT is widely used in optimization formulation for locomotion generation. Considering we are focusing on robust jumping, we make no claim of energy efficiency in this work.

Using the super-vector notation for all instances, we have $\mathbf{Y}_j = \boldsymbol{d}_j + \mathbf{H}\boldsymbol{\alpha}_j$, where $\boldsymbol{d}_j \in \mathbb{R}^{mo}$ is the free response and $\mathbf{H} \in \mathbb{R}^{mo\times mo}$ is the forced response to all $\boldsymbol{\pi}$.

The weights $\boldsymbol{\alpha}_j$ are learned iteratively through proportional error feedback, given a typical ILC learning rule

$$\begin{aligned}\boldsymbol{\alpha}_{j+1} &= \boldsymbol{\alpha}_j + \mathbf{L}\mathbf{E}_j = \boldsymbol{\alpha}_j + \mathbf{L}\left(\bar{\mathbf{Y}} - \mathbf{Y}_j\right)\\ &= \boldsymbol{\alpha}_j + \mathbf{L}\begin{bmatrix}\bar{\boldsymbol{y}}^1 - \boldsymbol{y}_j\left(T^1\right)\\ \vdots\\ \bar{\boldsymbol{y}}^o - \boldsymbol{y}_j\left(T^o\right)\end{bmatrix},\end{aligned} \tag{21}$$

where $\mathbf{L} \in \mathbb{R}^{mo\times mo}$ is the learning gain, $\mathbf{E}_j \in \mathbb{R}^{mo}$ is the iteration error, $\bar{\mathbf{Y}} \in \mathbb{R}^{mo}$ is the reference, and $\mathbf{Y}_j \in \mathbb{R}^{mo}$ is the iteration output.

2) Nonlinear System Control: For a nonlinear quadruped system, we have the following dynamic at j-th iteration, $\dot{\boldsymbol{x}}_j(t) = \boldsymbol{f}(t, \boldsymbol{x}_j(t), \boldsymbol{u}_j(t))$, $\boldsymbol{y}_j(t) = \boldsymbol{g}(t, \boldsymbol{x}_j(t), \boldsymbol{u}_j(t))$, with $\boldsymbol{x}_j \in \mathbb{R}^n$, $\boldsymbol{u}_j \in \mathbb{R}^l$, $\boldsymbol{y}_j \in \mathbb{R}^m$, and $l < m$. In the nonlinear case, $\mathbf{H}$ usually cannot be obtained as in (20). As $\mathbf{H}$ is the input-output map of the system response to the basis functions $\boldsymbol{\pi}$, which are sampled at the time instances $\{T^1, \ldots, T^o\}$, it can be formulated as

$$\mathbf{H} = \begin{bmatrix}\boldsymbol{g}(T^1, \boldsymbol{x}, \boldsymbol{\pi}^1) & \boldsymbol{g}(T^1, \boldsymbol{x}, \boldsymbol{\pi}^2) & \cdots & \boldsymbol{g}(T^1, \boldsymbol{x}, \boldsymbol{\pi}^{mo})\\ \boldsymbol{g}(T^2, \boldsymbol{x}, \boldsymbol{\pi}^1) & \boldsymbol{g}(T^2, \boldsymbol{x}, \boldsymbol{\pi}^2) & \cdots & \boldsymbol{g}(T^2, \boldsymbol{x}, \boldsymbol{\pi}^{mo})\\ \vdots & \vdots & \ddots & \vdots\\ \boldsymbol{g}(T^o, \boldsymbol{x}, \boldsymbol{\pi}^1) & \boldsymbol{g}(T^o, \boldsymbol{x}, \boldsymbol{\pi}^2) & \cdots & \boldsymbol{g}(T^o, \boldsymbol{x}, \boldsymbol{\pi}^{mo})\end{bmatrix}. \tag{22}$$

The matrix $\mathbf{H}$ can then be obtained from experiments, by exciting the system from equilibrium and recording the responses, omitting the need to identify an explicit model.

B. Controller Design

The learning gain $\mathbf{L}$ and basis function matrix $\boldsymbol{\pi}$ are determined as follows,

1) Learning Rule: We use a linear quadratic learning rule to compute the optimal learning gain $\mathbf{L}$ by minimizing

$$\mathbf{J}(\boldsymbol{\alpha}_j) = \|\mathbf{E}_j\|^2_{\mathbf{Q}_{\mathrm{LQ}}} + \|\boldsymbol{\alpha}_j - \boldsymbol{\alpha}_{j-1}\|^2_{\mathbf{S}_{\mathrm{LQ}}}, \tag{23}$$

such that

$$\mathbf{L} = \left(\mathbf{H}^{\mathsf{T}}\mathbf{Q}_{\mathrm{LQ}}\mathbf{H} + \mathbf{S}_{\mathrm{LQ}}\right)^{-1}\mathbf{H}^{\mathsf{T}}\mathbf{Q}_{\mathrm{LQ}}, \tag{24}$$

where $\mathbf{Q}_{\mathrm{LQ}}$ and $\mathbf{S}_{\mathrm{LQ}}$ are diagonal gain matrices [35].

2) Basis Functions: The selection of an appropriate family of basis functions determines the behaviour of the controller. [33] argued that any choice of π suffices that $\mathbf{H}$ is full rank is accepted. To be simple, we here select a set of mo Gaussians as our basis functions, with the mean μ and variance σ^2 per Gaussian to select (A detailed discussion can be found in Section IV-C1). As the robot can only be controlled during the stance phase, the means are distributed evenly between the TD and TO timings, i.e., t_{TD} and t_{TO} respectively. Furthermore, we use the same variance for each basis function. Outside this interval, the basis functions are set to zero, formulated as

$$\pi^i(t) = \begin{cases}\frac{1}{\sigma\sqrt{2\pi}}e^{-\frac{1}{2}\left(\frac{t-\mu^i}{\sigma}\right)^2} & \text{if } t_{\mathrm{TD}} \le t \le t_{\mathrm{TO}},\\ 0 & \text{otherwise.}\end{cases} \tag{25}$$

The configuration of the basis functions π^i in the matrix $\boldsymbol{\pi}$ should also be designed to prevent coupling of the control inputs, as each column of $\boldsymbol{\pi}$ is multiplied with a single scalar weight α^i. To this end, each column of $\boldsymbol{\pi}$ only contains one basis function. Furthermore, each row of $\boldsymbol{\pi}$ preferably has an equal amount of basis functions, such that the control authority is distributed equally over available control inputs. We use the following configuration for $\boldsymbol{\pi}$,

$$\begin{aligned}\boldsymbol{\pi}(t) &= \begin{bmatrix}\boldsymbol{\pi}^1 & \ldots & \boldsymbol{\pi}^o\end{bmatrix}\\ &= \begin{bmatrix}\pi^1(t) & 0 & 0 & 0 & \pi^5(t) & \cdots & 0\\ 0 & \pi^2(t) & 0 & 0 & 0 & \cdots & 0\\ 0 & 0 & \pi^3(t) & 0 & 0 & \cdots & 0\\ 0 & 0 & 0 & \pi^4(t) & 0 & \cdots & \pi^{mo}(t)\end{bmatrix}.\end{aligned} \tag{26}$$

TABLE I
OPTIMIZATION MODEL PARAMETERS

Parameter		Value	Parameter		Value
J	($\mathrm{kg\,m^2}$)	0.103	N	(-)	150
l_{calf}	(m)	0.213	n_{TD}, n_{TO}	(-)	50, 100
l_{thigh}	(m)	0.213	$\mathrm{d}t_{\min}$, $\mathrm{d}t_{\max}$	(s)	0.001, 0.01
l_{trunk}	(m)	0.188	$x_{\min}$, $x_{\max}$	(m)	0, 2
l_{virt}	(m)	0.301	$z_{\min}$, $z_{\max}$	(m)	0, 2
θ_{virt}	(°)	15	$\beta_{\min}$, $\beta_{\max}$	(m)	0, 5
θ_{0_2}	(°)	45	$\dot{x}_{\min}$, $\dot{x}_{\max}$	(m/s)	0, 5
θ_{0_3}	(°)	-135	$\dot{z}_{\min}$, $\dot{z}_{\max}$	(m/s)	-2, 2
θ_{0_4}	(°)	-135	$\dot{\beta}_{\min}$, $\dot{\beta}_{\max}$	(m)	0, 2
θ_{0_5}	(°)	45	$\tau_{2_{\min}}$, $\tau_{2_{\max}}$	(N m)	-71.1, 71.1
k_2,k_5	($\mathrm{N\,m\,rad^{-1}}$)	100	$\tau_{3_{\min}}$, $\tau_{3_{\max}}$	(N m)	-47.4, 47.4
k_3,k_4	($\mathrm{N\,m\,rad^{-1}}$)	100	$\tau_{4_{\min}}$, $\tau_{4_{\max}}$	(N m)	-47.4, 47.4
m	(kg)	13.013	$\tau_{5_{\min}}$, $\tau_{5_{\max}}$	(N m)	-71.1, 71.1

C. Continuous Learning Mechanism

The discontinuous learning process inherent to ILC requires resetting the system state after each iteration, which is tedious for the hardware test. To tackle this issue, we introduce the continuous learning mechanism, whereby the final state of the current iteration is used as the initial state for the next iteration. To this end, we modify the learning rule in (21) by adding a diagonal scaling matrix $\mathbf{W}$, enabling us to increase the learning rate in some states and to switch it off in others. The new learning rule becomes

$$\boldsymbol{\alpha}_{j+1} = \boldsymbol{\alpha}_j + \mathbf{L}(\mathbf{W}\mathbf{E}_j), \tag{27}$$

where $\mathbf{W} \in \mathbb{R}^{mo\times mo}$ is the scaling matrix.

With this continuous learning mechanism, we do not need to reset the system after each iteration. Note that, aside from the change in (27), the controller design is unchanged.

IV. Simulation Evaluations

The section validates the proposed method via full-body dynamic simulations.[2]

A. Setup

In simulation, the offline trajectory planner and the online learning controller are both implemented in Python. We use the CasADi [36] to formulate the Topt problem, using '`IPOPT`' as the solver. The PyBullet [37] is utilized to emulate the full-body quadrupedal system.

The model parameters are listed in Table I. Particularly, the springs are selected such that the simulated quadrupedal robot in PyBullet is able to sustain itself without using its actuators while in standstill, which was found to be $50\,\mathrm{Nm/rad}$ for all calf and thigh joints. Considering the 2D anchor model should produce twice the amount of torque per leg, the spring constants

[2]Videos of all the results can be accessed at: https://youtu.be/j-5WtDPZHCw

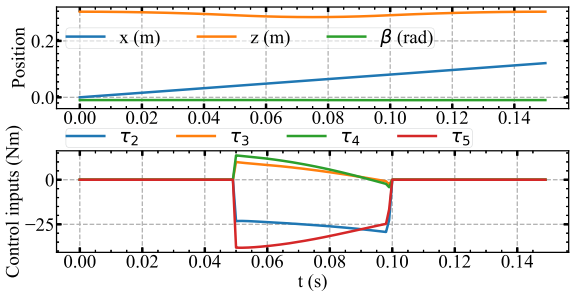

Fig. 3. Optimization results of the CSBD model with springs. We show both the evolution of the system and the control action for one single stride.

TABLE II
CoTs and MAEs in Jumping, 'Planner' and 'Controller' Separately Present the Results From the Trajectory Optimizer and Jumping Controller

	Planner			Controller		
	stiff (CoT)	elastic (CoT)	elastic (τ_{sum})	elastic (no learning)	elastic (fILC)	elastic (MPC)
CoT	0.596	0.535	0.937	3.83	0.765	1.565
MAE	-	-	-	0.350	0.00448	0.0986

are set at $100\,\mathrm{Nm/rad}$. Also, the torque boundaries are doubled in the anchor template model.

B. Jumping Motion Generation

Setting $\theta_{\text{virt}} = 15^\circ$, the NLP solver finds the optimal solution within approximately $7\,\text{s}$, resulting in a CoT of 0.535. The generated trajectories are shown in Fig. 3. We observe smooth trajectories in three phases, with a symmetric height trajectory ('z' curve in Fig. 3) around $0.075\,\text{s}$. During the first flight phase ($0\sim0.05\,\text{s}$) and the final flight phase ($0.1\sim0.15\,\text{s}$), the robot falls freely under gravity without torque inputs.

When setting all the elements in $\mathbf{K}$ (in (2) and (3)) to be zeros, jumping trajectories for a rigid quadruped without parallel actuators can be generated. In this rigid case, a larger CoT is obtained (see Table II). That is, compared to stiff locomotion, parallel elastic elements improve energy efficiency.

To further demonstrate the advantage of the current Topt formulation, we compare it with an alternative formulation, where the control inputs are minimized. That is, we replace the object (6) by $\tau_{\text{sum}} = \sum_{n=1}^{N}\sum_{i=2}^{5}(\tau_i^n)^2$. As a result, the jumping trajectory can still be generated. However, the resultant CoT is 0.937, which is much larger than that of the current formulation. Thus, in this work, we minimize the CoT when generating the periodic jumping trajectory.

C. FILC-Based Jumping Control

For the fILC, the selected time instances of interest are the lowest point of the trajectory, the take-off event and the final apex, i.e., at the node indices $\{75, 100, 150\}$. At the start of the learning process, the system is set to the initial state determined by the planner and the weights of the first iteration are set to zeros, namely, $\boldsymbol{\alpha}_0 = \mathbf{0}$. Once an iteration is finished, the system is reset to the initial state and the next control signal is executed.

1) fILC Using Different Basis Functions: Firstly, the torque profiles generated by the Topt are fed into the simulated quadrupedal robot directly, leading to large state errors with the maximal absolute error (MAE) being 0.350.

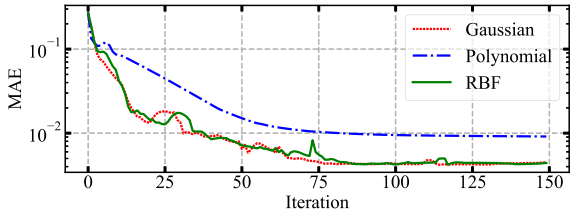

Fig. 4. Evolution of the maximal absolute error during the fILC learning process when using different kernel functions.

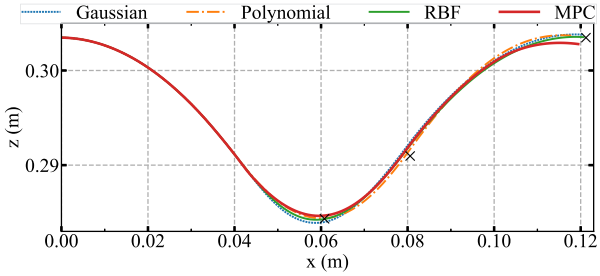

Fig. 5. Evolution of the CoM as resulting from the application of fILC with three different choices of basis functions. The performance of an MPC controller which relies on a perfect knowledge of the model is also shown as a comparison. '×' marks the reference way-point of interest.

Before applying fILC, we compare Gaussian kernel with other two basis functions, i.e., Polynomial kernel and Radial basis function (RBF), which are formulated as

$$\text{Polynomial:}\quad \pi^i(t) = A_1\frac{((t-\mu^i)^2 + B_1(t-\mu^i))}{\sigma_1^2},$$
$$\text{RBF:}\quad \pi^i(t) = A_2 e^{-\frac{1}{2}\left(\frac{t-\mu^i}{\sigma_2}\right)^2}, \tag{28}$$

with $\sigma_1^2{=}3\text{e}^{-5}, A_1{=}0.22, B_1{=}1.5$ and $\sigma_2^2{=}4\text{e}^{-5}, A_2{=}57$. For Gaussian kernel in (25), we have $\sigma^2 = 5\text{e}^{-5}$.

Fig. 4 plots the evolutionary MAE as iteration grows. As can be seen, after 150 iterations, the MAE drops to a very small value. Fig. 5 draws the CoM trajectories after 150 iterations. We can see that the RBF kernel contributes to the least MAE, with the best tracking performance. However, the Gaussian kernel results in a smoother MAE profile, with a decent tracking performance. Thus, in this work, we use the Gaussian kernel in the remaining parts. As listed in Table II, the resultant MAE with Gaussian kernel is 4.48×10^{-3}, with the measured CoT 0.765. The learned weights and control inputs using the Gaussian kernel are presented in Fig. 6.

2) fILC vs MPC: To further demonstrate the effectiveness, we compare fILC with an MPC scheme (similar to [27])[3], which is introduced here for the first time for periodic jumping control of a PEA-driven quadruped. The measured CoM is plotted by a red solid curve in Fig. 5. Results in Table II demonstrate that the fILC obtains a smaller MAE and a lower CoT than MPC.

[3]To our best knowledge, no MPC strategy is reported on the continuous jumping control of a PEA-driven quadrupedal robot. However, the MPC framework in [27] is widely used in dynamic locomotion control.

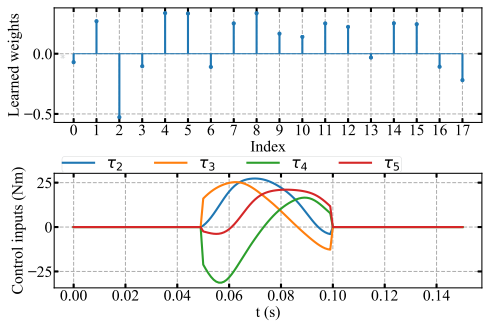

Fig. 6. Example of the learned weights (top) and resulting torque inputs (bottom) obtained when using the truncated Gaussian kernels.

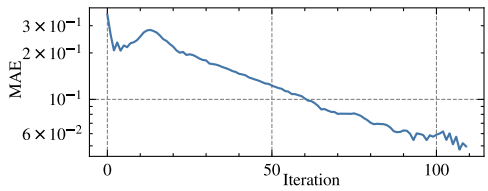

Fig. 7. Evolution of maximal absolute error when using the continuous learning variation of fILC.

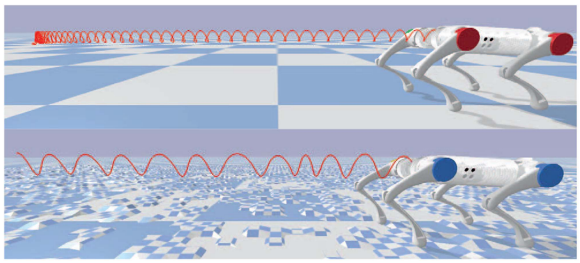

Fig. 8. Continuous fILC learning for periodic forward jumping on the flat ground (top) and across rough terrain (bottom).

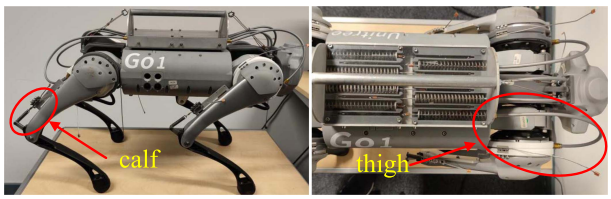

Fig. 9. Two views of the E-Go robot, a Unitree Go1 robot enhanced with springs acting in parallel to all the 12 joints. The springs acting in the sagittal plane for the E-Go robot are highlighted.

D. Continuous fILC in PyBullet Simulation

To realize continuous jumping without resetting the state after each stride, we apply the continuous learning approach with the scaling factors $\mathbf{W} = \text{diag}(s, s, s)$ $(s = (0, 10, 1, 2, 2, 1))$. The fILC gains are $\mathbf{Q}_{\text{LQ}} = 0.05\mathbf{I}_{18}$ and $\mathbf{S}_{\text{LQ}} = \mathbf{I}_{18}$, and $\sigma^2 = 0.001$. On flat ground, the quadruped is able to learn pronking gait after 110 iterations. The decrease of MAE is plotted in Fig. 7 and the CoM trajectory is visualized at the top of Fig. 8.

The robustness is then validated on the quadrupedal jumping on randomly generated uneven terrain, by initializing the weights with the previous ones learned for the flat ground. It demonstrates that the robot managed to traverse the uneven terrain without modelling it by virtue of continuous fILC, as can be seen in the bottom of Fig. 8.

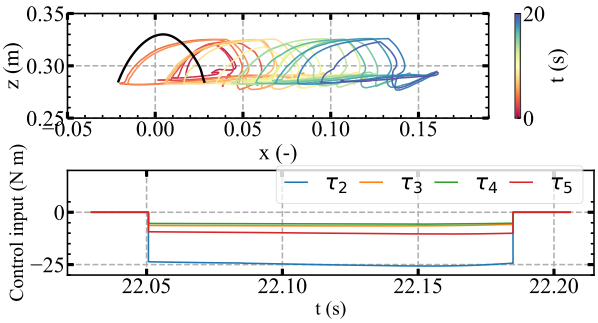

Fig. 10. Learned results for E-Go jumping on flat ground. The top shows the x-z trajectory after 52 s, with the black curve indicating the simulation result. The bottom panel shows the torque inputs of one jumping cycle. Note that the x-axis at the top has no unit.

It is worth mentioning that without updating the reference in real-time or adding an extra compensation mechanism for earlier/later contact, the robot can hardly pronk on flat ground when using the MPC scheme. A comparison motion can be found in the attached video.

V. Hardware Experiments

A. Experimental Setup

This section validates the proposed method on our PEA-driven robot (we call it E-Go here) that is presented in Fig. 9. The parallel springs are separately attached to the knee joints and hip joints of the Unitree Go1 robot [38]. With PEA add-ons, the E-Go weighs in total about 13 kg. To apply fILC on the quadrupedal robot, the input-output map in (22) needs to be determined in advance. However, the inherent learning property of fILC does not necessarily require an accurate $\mathbf{H}$ matrix. Thus, we directly adopt the $\mathbf{H}$ matrix achieved in the simulation when conducting hardware experiments.

The springs of the E-Go quadruped are much softer than the ones used in the PyBullet simulation, namely $6\,\text{Nm/rad}$ for the calf joints and $16\,\text{Nm/rad}$ for the thigh joints, compared to $50\,\text{Nm/rad}$ used for all joints in simulation. As a result, the physical springs are not stiff enough to support the weight of the robot without additional motor inputs. We compensate for this using a PD controller, $\boldsymbol{\tau}_{\text{PD}} = \mathbf{K}_{\text{P}}(\boldsymbol{q}_{\text{j}}^{\text{ref}} - \boldsymbol{q}_{\text{j}}) - \mathbf{K}_{\text{D}}\dot{\boldsymbol{q}}_{\text{j}}$, where $\boldsymbol{q}_{\text{j}} \in \mathbb{R}^8$ and $\boldsymbol{q}_{\text{j}}^{\text{ref}} \in \mathbb{R}^8$ are the real and reference joint angles in the sagittal plane, $\dot{\boldsymbol{q}}_{\text{j}} \in \mathbb{R}^8$ the real joint velocities, $\mathbf{K}_{\text{P}} \in \mathbb{R}^{8\times8}$ and $\mathbf{K}_{\text{D}} \in \mathbb{R}^{8\times8}$ separately are proportional and derivative gain. The gains are $K_{\text{P}} = 60$ and $K_{\text{D}} = 1$ for all joints such that the robot can stand up. For continuous jumping, the command torque sent to the motor is then the sum of the fILC output (run at 500 Hz) and the feedback torque (run at 10 KHz).

B. Jumping on Flat Ground

To start, the fILC is tuned without engaging the parallel springs, following the trajectory generated for a rigid robot. The scaling factors $\mathbf{W} = \text{diag}(s, s, s)$ $(s = (0, 10, 1, 2, 2, 1))$, together with controller parameters including $\mathbf{Q}_{\text{LQ}} = 0.1\mathbf{I}_{18}$, $\mathbf{S}_{\text{LQ}} = \mathbf{I}_{18}$ and $\sigma^2 = 0.1$ are used for jumping on flat ground. Thereafter, the learned weights are used to initialize the controller for jumping with springs engaged.

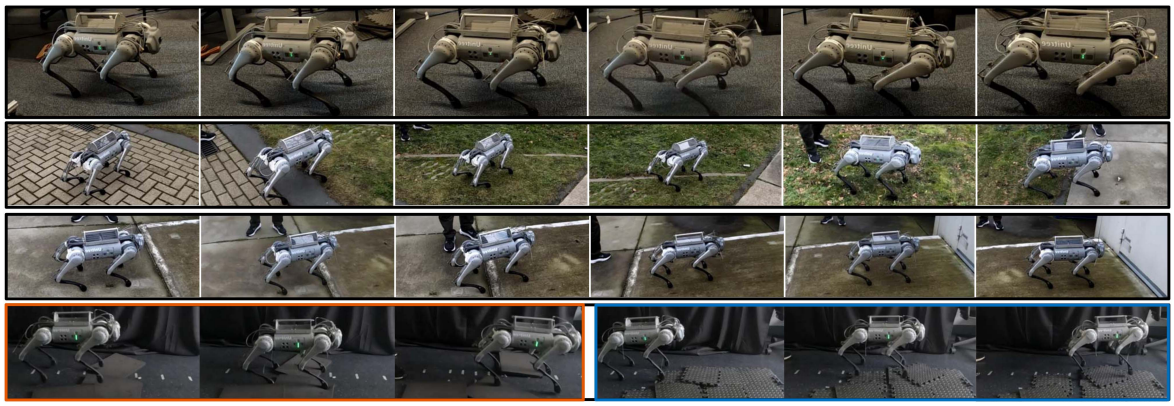

Fig. 11. Continuous jumping on flat terrain (first row), rough grassy terrain (second row), a slope (third row), and uneven movable pads (bottom). At the bottom, the rigid quadruped successfully jumps across the soft pads (on the left) and also the rigid pads (on the right).

With springs engaged, we start from a standstill and let the PEA-driven robot learn weights (following the reference trajectory for a compliant robot) until it runs up to $3\,\mathrm{m}$, which occurred after $52\,\mathrm{s}$. Then, we transfer the learned weights to a new learning cycle. This time, it took only $20\,\mathrm{s}$ to reach the target distance. Fig. 10 shows learned results for flat-ground jumping with parallel springs engaged.

The evolutionary x-z trajectory at the top panel of Fig. 10 demonstrates that, without identifying the $\mathbf{H}$ matrix for the hardware system from scratch, the fILC can converge and accomplish the jumping task. In addition, the shapes of the measured x-z curve are quite close to the one obtained in the simulation (black curve at the top panel)[4]. Aside from this, the feedforward torque inputs computed by fILC are below the physical limits, as can be seen from the bottom panel.

C. Robust Pronking

We then test the method's robustness in three unfavourable conditions. In all cases, the ground is modeled as flat and the robot is assumed to be PEA-driven.

1) Outdoor Uneven Terrain: The method is initialized with the learned weights from the previous indoor learning cycle. The other controller parameters are left unchanged. We start the new learning cycle when the robot is placed on concrete bricks while facing the grassy ground. We observe that the controller is able to successfully transit between different types of terrain, e.g., from uneven rigid concrete to uneven grassy ground and from grassy ground to flat concrete ground. The second row of Fig. 11 shows several snapshots.

2) Slope Terrain: The controller is also tested on an inclined slope of $5\,^\circ$. Again, the cycle is initialized with the weights learned indoors. We let the robot face the slope, starting from level ground. The controller successfully transits from the level ground to the slope. Several snapshots are presented in the third row of Fig. 11.

3) Model and Environment Uncertainties: To further validate the robustness, we test the continuous jumping on uneven ground, without springs engaged. Note that the reference trajectory was generated for a PEA robot, not for the rigid case, causing modelling discrepancies. Even though, it turns out that the robot can learn to jump across uneven terrain using fILC. Snapshots at the bottom of Fig. 11 show that the robot can jump across randomly-distributed movable pads on the way, including soft pads (see the orange box on the left side) and rigid pads (see the blue box on the right side).

[4]Considering the large state estimation error due to the landing impact, we do not plot the global jumping distance in Fig. 10.

VI. Conclusion

This work controls quadrupedal jumping using trajectory optimization and iterative learning. By introducing a compliant single-mass anchor model, we achieve periodic jumping gait. The use of fILC enables learning the feedforward control in only a matter of minutes. To the best of the authors' knowledge, this is the first report to use ILC to control a PEA-driven quadruped jumping.

Several recommendations can be made to improve this work. First, instead of giving the virtual touch-down angle θ_{virt} and the virtual leg length l_{virt} in advance, we can incorporate them into the optimization formulation to achieve a more efficient gait accompanied with better jumping performance, such as longer jumping distance and larger jumping height. Second, regarding fILC design, other basis functions such as a high-order polynomial kernel could be investigated. Besides, alternative learning rules such as those with Q-filter [39] potentially increase the convergence rate and improve the tracking accuracy.

Alternatively, learning from expertise reference [40], [41], [42] helps to achieve natural and efficient locomotion stills, among which the reinforcement learning (RL)approach is very promising since it allows the robot to learn by interacting with environments. However, the RL-based methods usually require a large number of iterations before converging. In contrast to fILC, it is very hard to run the RL-based method online. Still, it would be interesting to combine RL with ILC in learning the versatile strategy in more challenging scenarios in the future.

Appendix

The EoM of the quadruped during the stance phase is computed using Lagrangian mechanics,

$$\begin{bmatrix} m & 0 & 0 \\ 0 & m & 0 \\ 0 & 0 & J \end{bmatrix} \begin{bmatrix} \ddot{x} \\ \ddot{z} \\ \ddot{\beta} \end{bmatrix} + \begin{bmatrix} \sum_{i=2}^{5} k_i(\theta_i - \theta_{0_i}) \frac{\mathrm{d}}{\mathrm{d}x}\theta_i \\ \sum_{i=2}^{5} k_i(\theta_i - \theta_{0_i}) \frac{\mathrm{d}}{\mathrm{d}z}\theta_i \\ \sum_{i=2}^{5} k_i(\theta_i - \theta_{0_i}) \frac{\mathrm{d}}{\mathrm{d}\beta}\theta_i \end{bmatrix} + \begin{bmatrix} 0 \\ mg \\ 0 \end{bmatrix} = \begin{bmatrix} F_x \\ F_z \\ \tau_\beta \end{bmatrix} \tag{29}$$

or, alternatively,

$$\mathbf{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \mathbf{J_h}^{\mathsf{T}}(\boldsymbol{q})\mathbf{K}\left(\boldsymbol{\theta} - \boldsymbol{\theta}_0\right) + \mathbf{G}(\boldsymbol{q}) = \boldsymbol{\mathcal{F}}\,, \tag{30}$$

where $\mathbf{K} = \mathrm{diag}(0, k_2, k_3, k_4, k_5, 0)$. The partial derivatives of $\boldsymbol{\theta}$ with respect to $\mathbf{q}$ are obtained using SymPy. The spatial wrench $\boldsymbol{\mathcal{F}}$ is given by

$$\boldsymbol{\mathcal{F}} = \begin{bmatrix} F_x \\ F_z \\ \tau_\beta \end{bmatrix} = \begin{bmatrix} F_{\mathrm{H}_x} + F_{\mathrm{F}_x} \\ F_{\mathrm{H}_z} + F_{\mathrm{F}_z} \\ \boldsymbol{F}_{\mathrm{H}} \times \boldsymbol{r}_{\mathrm{H}} + \boldsymbol{F}_{\mathrm{F}} \times \boldsymbol{r}_{\mathrm{F}} \end{bmatrix}, \tag{31}$$

where $\boldsymbol{F}_{\mathrm{H}}$ and $\boldsymbol{F}_{\mathrm{F}}$ are the ground reaction forces of each leg as a result of the motor torque $\boldsymbol{\tau}$, while $\boldsymbol{r}_{\mathrm{H}}$ and $\boldsymbol{r}_{\mathrm{F}}$ are the vectors from the CoM to the hind and front legs.

The motor torques $\boldsymbol{\tau}$ are then converted from joint coordinate space to generalized forces using the contact Jacobian.

REFERENCES

[1] C. D. Bellicoso et al., "Advances in real-world applications for legged robots," *J. Field Robot.*, vol. 35, no. 8, pp. 1311–1326, 2018.

[2] F. Angelini et al., "Robotic monitoring of habitats: The natural intelligence approach," *IEEE Access*, vol. 11, pp. 72575–72591, 2023.

[3] J. Ding, L. Han, L. Ge, Y. Liu, and J. Pang, "Robust locomotion exploiting multiple balance strategies: An observer-based cascaded model predictive control approach," *IEEE/ASME Trans. Mechatron.*, vol. 27, no. 4, pp. 2089–2097, Aug. 2022.

[4] C. D. Santina, M. G. Catalano, A. Bicchi, M. Ang, O. Khatib, and B. Siciliano, "Soft robots," *Encyclopedia Robot.*, vol. 489, pp. 1–15, 2021.

[5] H. Kolvenbach, E. Hampp, P. Barton, R. Zenkl, and M. Hutter, "Towards jumping locomotion for quadruped robots on the moon," in *Proc. IEEE/RSJ Int. Conf. Intell. Robot. Syst.*, 2019, pp. 5459–5466.

[6] A. Badri-Spröwitz, A. A. Sarvestani, M. Sitti, and M. A. Daley, "Birdbot achieves energy-efficient gait with minimal control using avian-inspired leg clutching," *Sci. Robot.*, vol. 7, no. 64, 2022, Art. no. eabg4055.

[7] F. Bjelonic et al., "Learning-based design and control for quadrupedal robots with parallel-elastic actuators," *IEEE Robot. Automat. Lett.*, vol. 8, no. 3, pp. 1611–1618, Mar. 2023.

[8] C. Nguyen, L. Bao, and Q. Nguyen, "Continuous jumping for legged robots on stepping stones via trajectory optimization and model predictive control," in *Proc. IEEE Conf. Decis. Control*, 2022, pp. 93–99.

[9] Q. Nguyen, M. J. Powell, B. Katz, J. D. Carlo, and S. Kim, "Optimized jumping on the MIT Cheetah 3 robot," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2019, pp. 7448–7454.

[10] Z. Song et al., "An optimal motion planning framework for quadruped jumping," in *Proc. IEEE/RSJ Int. Conf. Intell. Robot. Syst.*, 2022, pp. 11366–11373.

[11] M. Chignoli and S. Kim, "Online trajectory optimization for dynamic aerial motions of a quadruped robot," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2021, pp. 7693–7699.

[12] M. Chignoli, S. Morozov, and S. Kim, "Rapid and reliable quadruped motion planning with omnidirectional jumping," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2022, pp. 6621–6627.

[13] Y. Ding, C. Li, and H.-W. Park, "Kinodynamic motion planning for multi-legged robot jumping via mixed-integer convex program," in *Proc. IEEE/RSJ Int. Conf. Intell. Robot. Syst.*, 2020, pp. 3998–4005.

[14] M. Posa, C. Cantu, and R. Tedrake, "A direct method for trajectory optimization of rigid bodies through contact," *Int. J. Robot. Res.*, vol. 33, no. 1, pp. 69–81, 2014.

[15] A. W. Winkler, C. D. Bellicoso, M. Hutter, and J. Buchli, "Gait and trajectory optimization for legged systems through phase-based end-effectorparameterization," *IEEE Robot. Automat. Lett.*, vol. 3, no. 3, pp. 1560–1567, Jul. 2018.

[16] R. Blickhan, "The spring mass model for running and hopping," *J. Biomech.*, vol. 22, no. 11/12, pp. 1217–1227, 1989.

[17] H. Geyer, A. Seyfarth, and R. Blickhan, "Compliant leg behaviour explains basic dynamics of walking and running," *Proc. Roy. Soc. B: Biol. Sci.*, vol. 273, no. 1603, pp. 2861–2867, 2006.

[18] P. M. Wensing and D. E. Orin, "High-speed humanoid running through control with a 3D-SLIP model," in *Proc. IEEE/RSJ Int. Conf. Intell. Robot. Syst.*, 2013, pp. 5134–5140.

[19] X. Xiong and A. D. Ames, "Sequential motion planning for bipedal somersault via flywheel SLIP and momentum transmission with task space control," in *Proc. IEEE/RSJ Int. Conf. Intell. Robot. Syst.*, 2020, pp. 3510–3517.

[20] D. Calzolari, C. D. Santina, A. M. Giordano, and A. Albu-Schäffer, "Single-leg forward hopping via nonlinear modes," in *Proc. Amer. Control Conf.*, 2022, pp. 506–513.

[21] D. Lakatos, C. Rode, A. Seyfarth, and A. Albu-Schäffer, "Design and control of compliantly actuated bipedal running robots: Concepts to exploit natural system dynamics," in *Proc. IEEE-RAS Int. Conf. Humanoid Robots*, 2014, pp. 930–937.

[22] M. Hutter, C. Gehring, M. Bloesch, M. Hoepflinger, P. Fankhauser, and R. Siegwart, "Excitation and stabilization of passive dynamics in locomotion using hierarchical operational space control," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2014, pp. 2977–2982.

[23] G. M. Gasparri et al., "Efficient walking gait generation via principal component representation of optimal trajectories: Application to a planar biped robot with elastic joints," *IEEE Robot. Automat. Lett.*, vol. 3, no. 3, pp. 2299–2306, Jul. 2018.

[24] C. D. Santina and A. Albu-Schaeffer, "Exciting efficient oscillations in nonlinear mechanical systems through eigenmanifold stabilization," *IEEE Control Syst. Lett.*, vol. 5, no. 6, pp. 1916–1921, Dec. 2021.

[25] C. D. Santina et al., "Controlling soft robots: Balancing feedback and feedforward elements," *IEEE Robot. Automat. Mag.*, vol. 24, no. 3, pp. 75–83, Sep. 2017.

[26] M. Focchi, A. D. Prete, I. Havoutis, R. Featherstone, D. G. Caldwell, and C. Semini, "High-slope terrain locomotion for torque-controlled quadruped robots," *Auton. Robots*, vol. 41, no. 1, pp. 259–272, 2017.

[27] J. D. Carlo, P. M. Wensing, B. Katz, G. Bledt, and S. Kim, "Dynamic locomotion in the MIT Cheetah 3 through convex model-predictive control," in *Proc. IEEE/RSJ Int. Conf. Intell. Robot. Syst.*, 2018, pp. 1–9.

[28] M. J. Pollayil et al., "Planning natural locomotion for articulated soft quadrupeds," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2022, pp. 6593–6599.

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

[30] F. Angelini et al., "Decentralized trajectory tracking control for soft robots interacting with the environment," *IEEE Trans. Robot.*, vol. 34, no. 4, pp. 924–935, Aug. 2018.

[31] R. Mengacci, F. Angelini, M. G. Catalano, G. Grioli, A. Bicchi, and M. Garabini, "On the motion/stiffness decoupling property of articulated soft robots with application to model-free torque iterative learning control," *Int. J. Robot. Res.*, vol. 40, no. 1, pp. 348–374, 2021.

[32] M. Pierallini, F. Angelini, A. Bicchi, and M. Garabini, "Swing-up of underactuated compliant arms via iterative learning control," *IEEE Robot. Automat. Lett.*, vol. 7, no. 2, pp. 3186–3193, Apr. 2022.

[33] C. D. Santina and F. Angelini, "Iterative learning in functional space for non-square linear systems," in *Proc. IEEE Conf. Decis. Control*, 2021, pp. 5858–5863.

[34] H. C. Doets, D. Vergouw, H. E. Veeger, and H. Houdijk, "Metabolic cost and mechanical work for the step-to-step transition in walking after successful total ankle arthroplasty," *Hum. Movement. Sci.*, vol. 28, no. 6, pp. 786–797, 2009.

[35] R. J. Li and Z. Z. Han, "Survey of iterative learning control," *Kongzhi yu Juece/Control Decis.*, vol. 20, no. 9, pp. 961–966, 2005.

[36] J. A. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, "CasADi: A software framework for nonlinear optimization and optimal control," *Math. Program. Computation*, vol. 11, no. 1, pp. 1–36, 2019.

[37] E. Coumans and Y. Bai, "PyBullet, a Python module for physics simulation for games, robotics and machine learning," 2021. [Online]. Available: https://pybullet.org/

[38] U. Robotics, "Unitree Go1," 2023. [Online]. Available: https://www.unitree.com/en/go1/

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

[40] J. Ding, T. L. Lam, L. Ge, J. Pang, and Y. Huang, "Safe and adaptive 3-D locomotion via constrained task-space imitation learning," *IEEE/ASME Trans. Mechatron.*, early access, Feb. 20, 2023, doi: 10.1109/TMECH.2023.3239099.

[41] Y. Yang, X. Meng, W. Yu, T. Zhang, J. Tan, and B. Boots, "Learning semantics-aware locomotion skills from human demonstration," in *Proc. Conf. Robot Learn.*, 2023, pp. 2205–2214.

[42] G. Bellegarda and Q. Nguyen, "Robust quadruped jumping via deep reinforcement learning," 2020, *arXiv:2011.07089*.