2024 IEEE International Conference on Robotics and Automation (ICRA 2024)
May 13-17, 2024. Yokohama, Japan

Two-Stage Learning of Highly Dynamic Motions with Rigid and Articulated Soft Quadrupeds

Francecso Vezzi[1,⋆], Jiatao Ding[1,⋆,*], Antonin Raffin[2], Jens Kober[1], Cosimo Della Santina[1,2]

***Abstract*— Controlled execution of dynamic motions in quadrupedal robots, especially those with articulated soft bodies, presents a unique set of challenges that traditional methods struggle to address efficiently. In this study, we tackle these issues by relying on a simple yet effective two-stage learning framework to generate dynamic motions for quadrupedal robots. First, a gradient-free evolution strategy is employed to discover simply represented control policies, eliminating the need for a predefined reference motion. Then, we refine these policies using deep reinforcement learning. Our approach enables the acquisition of complex motions like pronking and back-flipping, effectively from scratch. Additionally, our method simplifies the traditionally labour-intensive task of reward shaping, boosting the efficiency of the learning process. Importantly, our framework proves particularly effective for articulated soft quadrupeds, whose inherent compliance and adaptability make them ideal for dynamic tasks but also introduce unique control challenges.**

I. Introduction

Quadrupedal robots [1]–[3] promise to address real-world challenges due to their potential advantages in terms of mobility and versatility. To improve the performance and extend mission time [4], elastic and soft elements [5] have been applied to quadrupedal robots in recent works, such as [6]–[9]. Although stable locomotion can be achieved by off-the-shelf solvers, accomplishing highly dynamic motion is still an open challenge due to the under-actuated, hybrid, nonlinear dynamics.

Acrobatic motions have been achieved with quadrupedal robots by virtue of trajectory optimization (TO) and model predictive control (MPC), including jumping [10]–[12] and back-flip [13], [14]. However, these approaches are computationally intensive and require accurate models. To guarantee the solution when solving the TO problems, simplifications in problem formulations are often needed, resulting in a conservative solution.

Reinforcement learning (RL) has emerged as a promising alternative to model-based controllers for achieving dynamic motions. Although normal gaits like trotting can be easily learned from scratch, learning acrobatic motions needs extra setup, such as tedious reward shaping [15]–[17] and careful curriculum design [18]–[21], due to the highly nonlinear dynamics and reward sparsity. To address this problem, the current RL frameworks assume the availability of prior knowledge on how to solve the task in the form of a reference motion/trajectory [22]–[26], a pre-existing controller [27], [28] or both [29], [30]. This reliance on expert knowledge limits the applicability of these methods as it may not be readily available and possibly biases the learning towards sub-optimal policies.

Apart from motion generation for rigid robots, many works have argued that introducing elasticity can yield better performance [31], which, however, brings extra control challenges. Up to now, very limited work has been done on generating highly dynamic quadrupedal motions exploiting parallel elasticity. RL-based works such as [7] and [9] employ passive elasticity for quadrupedal locomotion but do not report such acrobatic motions as done in this work. The only exception is SpaceBok [8], which is, however, specifically designed to work in a low-gravity environment.

To overcome these limitations, we propose here a two-stage strategy for learning highly dynamic motions from scratch by combining RL with an evolution strategy (ES), as illustrated in Fig. 1. To start with, an evolution stage is employed to directly learn joint commands meeting the task demands, without requiring an expert demonstration defined in advance. In this stage, we use the augmented random search (`ARS`) [32] algorithm to learn a linear policy. Then, we transfer the learned policy to a more expressive neural network via deep reinforcement learning (DRL) [33]. We use proximal policy optimization (`PPO`) [34] to refine the policy that is warm-started by mimicking the learned motions in the first stage. Note that a similar two-stage method containing ES and DRL has been proposed in [35] for quadrupedal locomotion. However, the ES, specifically the natural evolution strategies algorithm [36], was used there to search the parameters for a center pattern generator that generates reference trajectories for the second-stage RL, and no dynamic motions like ours were reported.

We validate[1] the approach by rich simulations when learning versatile acrobatic motions, including jumping, pronking and back-flip, see Fig. 2. Furthermore, we apply the approach to both a rigid and an articulated soft quadruped. We demonstrate that the proposed architecture enables exploiting parallel elasticity in achieving highly dynamic motions. Taking the jumping task as an example, the articulated soft quadrupedal robot could jump 15.4% higher in the in-placed jumping task, and jump 23.1% further in the forward jumping.

II. Methodology

We introduce basic notation in Section II-A and the two-stage procedure in a general fashion in Section II-B.

This work is supported by the EU project 101016970 NI. [1]Authors are with the Department of Cognitive Robotics, Delft University of Technology, Building 34, Mekelweg 2, 2628 CD Delft, Netherlands (f.vezzi.96@gmail.com, J.Ding-2@tudelft.nl, J.Kober@tudelft.nl, C.DellaSantina@tudelft.nl). [2]Authors are with the Institute of Robotics and Mechatronics, German Aerospace Center (DLR), 82234 Wessling, Germany (Antonin.Raffin@dlr.de). ⋆ These authors contributed equally to this work. * Jiatao Ding is the corresponding author.

[1]All the videos can be found at: `https://www.youtube.com/watch?v=liIeYU71a5w`

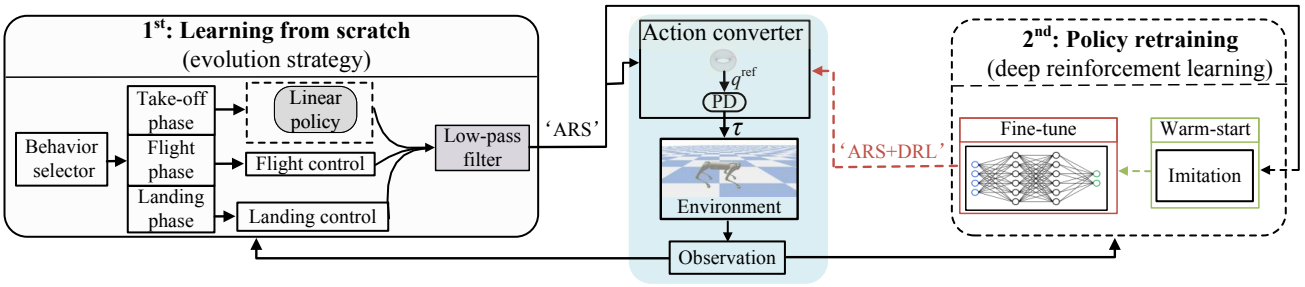
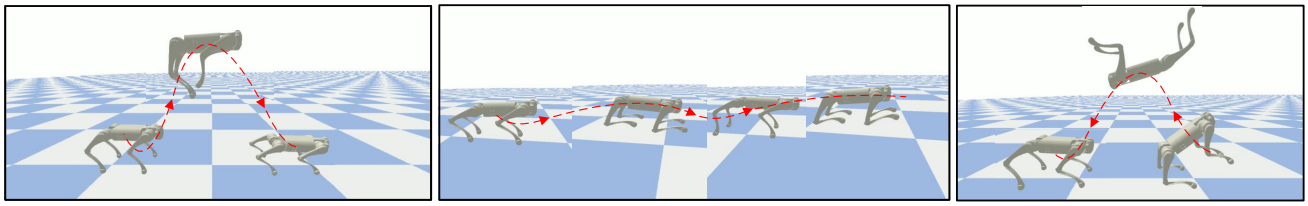


Fig. 1. Two-stage learning procedure for the acrobatic motion with a flight phase. Without defining reference motions, the (rigid and soft) quadrupedal robot realizes (a) jumping in place and jumping forward, (b) pronking and (c) back-flip. 'ARS' and 'ARS+DRL' separately represent the linear policies generated by first-stage ES and the refined policy after the second-stage retraining.



Jumping forward Pronking Back-flip

Fig. 2. Examples of highly dynamic motions that the robot could learn using the proposed strategy. Red curves plot the CoM movements where the arrows point to the movement directions.

Acrobatic-specific details follow in Section II-C - yielding the general architecture shown in Fig. 1. Task-specific details will be provided later in Section III.

A. RL notation

The RL problem is formulated as a Markov decision process (MDP), described by the tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathcal{P}$ is the transition probability, $\mathcal{R}$ is the reward distribution, and γ is the discount factor. The states ($\boldsymbol{s}_k \in \mathcal{S}$), actions ($\boldsymbol{a}_k \in \mathcal{A}$), and rewards ($r(\boldsymbol{s}_k, \boldsymbol{a}_k) \in \mathcal{R}$) in an episode of length T constitute a rollout of the policy. In each rollout, the cumulative return is $G_T = \sum_{k=1}^{T} \gamma^{k-1} r(\boldsymbol{s}_k, \boldsymbol{a}_k)$. In the following, we will assume that the whole state is observable, and thus we will refer to elements in $\mathcal{S}$ as observations. The goal of RL is then to find an approximation of the optimal policy ($\pi^\star$) such that the agent achieves the maximal expected return, i.e., $\pi^\star = \arg\max_{\pi} \mathbb{E}\left[G_T \mid \pi\right]$.

B. Two-stage learning

1) First step-learning a simple policy from scratch via evolution strategy: Despite DRL's success in sequential decision-making, it struggles with sparse-reward tasks [37]. To address this, we use a gradient-free ES scheme, specifically ARS, in the first stage to treat the RL tasks as black-box optimization [38], where the linear policy is adopted[2]

$$\pi_{\texttt{ars}}(\boldsymbol{s}_k) = \mathbf{W}\, \boldsymbol{s}_k + \boldsymbol{b}, \tag{1}$$

where $\mathbf{W} : \mathcal{S} \to \mathcal{A}$ is a linear application, and $\boldsymbol{b} \in \mathcal{A}$.

2) Second step-refining and retraining via DRL: We then resort to the more expressive deep neural network (DNN) to refine the policy. Using the DRL method, specifically PPO, we achieve the nonlinear expression

$$\pi_{\texttt{PPO}}(\boldsymbol{a}_k | \boldsymbol{s}_k; \boldsymbol{\theta}) = \mathrm{NN}(\boldsymbol{s}_k; \boldsymbol{\theta}), \tag{2}$$

[2]The work in [32] argued that the linear policy is simple but enough to obtain competitive results for RL tasks. A comparison with other representations on the jumping task is found in Section IV-B.

where $\mathrm{NN} : \mathcal{S} \times \mathbb{R}^n \to \Delta(\mathcal{A})$ is a policy function that maps states to a probability distribution on the action space, and $\boldsymbol{\theta} \in \mathbb{R}^n$ is the vector of the network parameters.

At this stage, we start training a DNN by replicating the observation-action pair provided by the first-stage policy, using the imitative reward function $r^{\text{ini}} = w_{\text{a}} \exp^{(-w_{\text{b}} ||\boldsymbol{a} - \boldsymbol{a}^{\texttt{ars}}||^2)}$, with $\boldsymbol{a}, \boldsymbol{a}^{\texttt{ars}} \in \mathcal{A}$ respectively being the output of the DNN and the first-stage ARS given an observation, and $w_{\text{a}}, w_{\text{b}} \in \mathbb{R}$ being the hyperparameters. Afterwards, we fine-tune the DNN using the task-specific reward ${}^2r$, which we detail in section III.

C. Learning acrobatic motions for quadrupeds

We define an acrobatic motion as a dynamic movement involving three (possibly repeated) phases: (i) take-off, (ii) flight, (iii) landing. On top of what we discussed in the previous section, we introduce a *behavior selector* mechanism. This is a state machine that transitions through the three phases discussed above, characterizing every acrobatic motion. The first phase is the take-off, which uses (1). The second is the flight phase, which generates a constant action equal to the homing pose in Fig. 3(b). The third phase passes the same pose through a proportional-derivative (PD) controller thus reducing the effective joint stiffness. This is sufficient to ensure a compliant landing. The guarding conditions of the flight phase and landing phase separately are $F_{\text{contact}} = 0$, and $\dot{h} < 0$ and $F_{\text{contact}} > 0$, where F_{contact} is the total contact force between the robot feet and the ground, and $\dot{h}$ is the vertical velocity. We also add a low-pass filter to the action a before feeding it into the environment as ARS can result in non-smooth control signals. We remove these additions in the second stage of learning, relying on the capability of the *NN* in (2) to learn the smooth actions.

III. Learning Task Formulation

This section defines the key elements of the RL tuple for learning acrobatic motions. We specifically focus on jumping, pronking, and back-flip (see Fig. 2). Note that if

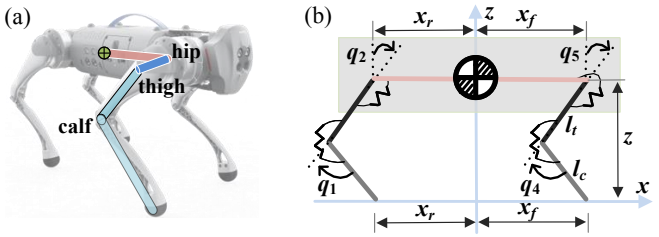


Fig. 3. Quadrupedal robot Go1 (a) and its PEA arrangement (b). In the homing pose, we have initial height $z_0 = 0.32$m. We call q_1, q_4 the calf angles, q_2, q_5 the thigh angles, and q_3, q_6 the hip angles.

not differently specified, the observation and action spaces are the same during the two learning stages and the three tasks.

We conduct simulation experiments using the Go1 robot from UniTree and its soft variant featuring parallel elastic actuation, as detailed in Appendix A. Both models are depicted in Fig. 3. Hence, when discussing specifics, we'll assume these are the robots in question, although our architecture is broadly applicable to any quadrupedal system. Moreover, it is worth pointing out here that we will focus on Sagittal motions - although the environment is fully 3D.

A. Observation/state

These include the base (i.e., trunk) height ($h \in \mathbb{R}$), vertical velocity ($\dot{h} \in \mathbb{R}$), and the pitch angle ($\theta \in \mathbb{R}$). Furthermore, the average[3] joint angles ($\boldsymbol{q} \in \mathbb{R}^6$) and velocities ($\dot{\boldsymbol{q}} \in \mathbb{R}^6$) are also added to the observations. Finally, we include a Boolean flag $b \in \{0, 1\}$ that indicates whether the robot is in a flight phase or not. This observation is, however, only available to π_{PPO}. The resulting observations vector is $\boldsymbol{s}_k = \{h, \dot{h}, \theta, \boldsymbol{q}, \dot{\boldsymbol{q}}, b\}$. In simulations, the white noise is added to the observations to emulate the sensory noise.

B. Action space

Assuming symmetry of the robot and of the task between the right and left sides, we only learn the 3 joints in the front leg and 3 joints in the rear leg.

We convert the policy outputs (i.e., $\boldsymbol{a}_k \in [-1, 1]^6$) to joint angle ($\boldsymbol{q}^{\text{ref}}$) commands by a linear scaling defined such that the maximum actions corresponds to the joint limits. which we feed into a PD controller that generates motor torques as follows $\tau_i = k_{\text{p}}(q_i^{\text{ref}} - q_i) - k_{\text{d}}\dot{q}_i$, where $k_{\text{p}} \in \mathbb{R}$ and $k_{\text{d}} \in \mathbb{R}$ are the proportional and derivative gains, respectively.

The pipeline of converting action to motor torque command is summarized by the 'Action converter' in Fig. 1.

C. Episode design

In our settings, one episode is terminated when the time runs out or early termination is triggered. In particular, the early termination is triggered if one of the following is true

- the robot falls, which we detect by checking the height, i.e., $h < 0.1$ m,
- the knees touch the ground, or links collide with each other,
- the trunk is parallel enough to the ground, that we quantify by evaluating the scalar product between the local z-axis and the global one and testing that it is less than 0.85.

The third condition is not present in the back-flip task.

[3]More precisely, $q_i = (q_i^{\text{left}} + q_i^{\text{right}})/2$, where $q_i^{\text{left}}, q_i^{\text{right}} \in \mathbb{R}$ are the i-th joint according to the numbering in Fig. 3 on the left and right sides of the robot when cut by the Sagittal plane.

D. Rewards for the two stages

1) Notation: We concisely refer to the exponential kernel as follows

$$g_l^t : \mathbb{R} \to \mathbb{R}, \quad g_l^t(x) \coloneqq a_l^t e^{-b_l^t |x|} \tag{3}$$

with a_l^t and b_l^t being hyperparameters. With this notation, the kernel function for different tasks is encapsulated inside the subscript and superscript, where the superscript (t) specifies the task, while the subscript (l) refers to the input variable. Hereafter, we use r to denote $r(\boldsymbol{s}_k, \boldsymbol{a}_k)$ for brevity

2) Reward shaping: To define the reward function for each task, we first normalize and clip the maximal height ($\overline{h} \in \mathbb{R}$). For the jumping task, we also normalize and clip the maximal jumping distance ($\overline{d} \in \mathbb{R}$). That is

$$h_n = \text{clip}\left\{0, \overline{h}/h_{\max}, 1\right\}, \quad d_n = \text{clip}\left\{0, \overline{d}/d_{\max}, 1\right\}, \tag{4}$$

where $h_{\max}$ and $d_{\max}$ denote the maximal height and maximal jumping distance that can be achieved by the robot, which we tuned heuristically.

We go now more in detail, taking the jumping task as an example. Pronking and back-flip are similarly defined by making minor changes to the reward functions. More details can be found in Appendix B.

a) Reward function for the first stage: Since ES search can deal with the learning tasks with sparse rewards, alleviating the efforts for rewarding shaping, we assume that the robot does not get any reward during one episode until it ends. The other reason is that dense reward might lead to unwanted behavior, whereas sparse reward makes it simple to obtain the correct behavior. Thus, the reward for the first-stage ARS training (${}^1r^{\text{jump}}$) is straightforwardly defined as

$${}^1r^{\text{jump}} = \begin{cases} 0 & \text{when episode is running,} \\ r_{\text{bonus}} + r_{\text{end}} & \text{when episode terminates,} \end{cases} \tag{5}$$

where the ending reward r_{end} is computed by evaluating the state when the episode terminates, and the bonus reward r_{bonus} is by evaluating the jumping performance.

To achieve a jumping motion, we expect the robot to increase its normalized height (h_n) as much as possible so as to take off from the ground. Besides, to reduce the control efforts for posture adjustment in the air while minimizing the landing recovery efforts after touchdown, we penalize the maximal pitch angle ($\overline{\theta} \in \mathbb{R}$) the robot experiences during one episode. Finally, one additional term is added to regulate the normalized jumping distance (d_n). For jumping in place, this term minimizes the forward distance while, for jumping forward, this term encourages a large forward distance.

Reward for jumping in place are defined as

$$\begin{aligned} r_{\text{end}}^{\text{jip}} &= \left[g_\theta^{\text{jip}}(\overline{\theta}) + g_d^{\text{jip}}(d_n) + c_h^{\text{jip}}\right] h_n, \\ r_{\text{bonus}}^{\text{jip}} &= \begin{cases} +\, b^{\text{jip}} h_n & \text{not early termination,} \\ -\left[q^{\text{jip}} + m^{\text{jip}} h_n\right] & \text{otherwise,} \end{cases} \end{aligned} \tag{6}$$

where g_θ^{jip} and g_d^{jip} are computed using (3). c_h^{jip}, b^{jip}, q^{jip} and m^{jip} are the positive hyperparameters.

Reward for jumping forward are then defined as

$$r_{\text{end}}^{\text{jf}} = \left[g_\theta^{\text{jf}}(\overline{\theta}) + c_d^{\text{jf}} d_n + c_h^{\text{jf}}\right] h_n,$$
$$r_{\text{bonus}}^{\text{jf}} = \begin{cases} +\, b^{\text{jf}}(h_n + d_n) & \text{no early termination,} \\ -\left[q^{\text{jf}} + m^{\text{jf}}(h_n + d_n)\right] & \text{otherwise,} \end{cases} \quad (7)$$

where g_θ^{jf} is computed using (3). c_d^{jf}, c_h^{jf}, b^{jf}, q^{jf} and m^{jf} are the positive hyperparameters.

Note that the reward functions for jumping forward are quite similar to those for jumping in place. The only difference is that the jumping distance appears proportionally when computing $r_{\text{end}}^{\text{jf}}$ rather than inside the kernel function such that a longer jumping distance is encouraged. Besides, when computing the bonus for jumping forward ($r_{\text{bonus}}^{\text{jf}}$), we consider both the normalized maximal jumping distance d_n and the normalized maximal height h_n.

b) Reward function for the second stage: We discuss here the reward function used for refinement. The reader can refer to Section II-B.2 for the warm-starting reward r^{ini}. Then, the dense refinement reward (${}^2r$) for retraining is

$${}^2r^{\text{jump}} = r_h + r_c + r_d + r_s + r_\theta + {}^2r_{\text{bonus}}. \quad (8)$$

In (8), the term r_h is defined as

$$r_h = \begin{cases} a_h h & \text{if } h_{\min} \le h \le h_{\max}, \\ 0 & \text{otherwise,} \end{cases} \quad (9)$$

with a_h being a positive hyperparameter, which encourages the agent to not exceed $h_{\max}$ when jumping and avoids rewarding it when not jumping at all ($h < h_{\min}$). Note that here we use time-varying height h instead of the normalized maximal height h_n.

The term r_c is used to penalize the average contact force ($f \in \mathbb{R}$) so as to encourage soft landing.

$$r_c = \begin{cases} -a_c f & \text{if } f \ge f_{\min}, \\ 0 & \text{otherwise,} \end{cases} \quad (10)$$

where f is computed by dividing total force (F_{contact}) by the number of contact legs, $f_{\min}$ is the lower threshold, and a_c is a hyperparameter.

The term r_d regulates jumping distance ($d \in \mathbb{R}$). Particularly, for jumping in place, we minimize the jumping distance whereas, for jumping forward, we maximize the jumping distance. To this end, we have

$$r_d^{\text{jip}} = g_d^{\text{jip}}(d), \quad r_d^{\text{jf}} = k_d^{\text{jf}} d. \quad (11)$$

with k_d^{jf} being a hyperparameter.

The term r_s rewards a smooth torque profile $r_s = g_s(\boldsymbol{\delta}_\tau)$ with $\boldsymbol{\delta}_\tau \in \mathbb{R}^6$ being the differences between the torques applied at two consecutive time steps.

The term r_θ minimizes the pitch angle, i.e., $r_\theta = g_\theta(\theta)$.

Furthermore, similar to the first stage, a bonus is added at the end of one episode. For both tasks, we define

$${}^2r_{\text{bonus}}^{\text{jip}} = \begin{cases} 0 & \text{not early termination,} \\ -m^{\text{jip}}\,\overline{h} & \text{otherwise.} \end{cases}$$
$${}^2r_{\text{bonus}}^{\text{jf}} = \begin{cases} b^{\text{jf}}\,(\overline{h} + \overline{d}) & \text{not early termination,} \\ 0 & \text{otherwise.} \end{cases} \quad (12)$$

with m^{jip} and b^{jf} being the hyperparameters.

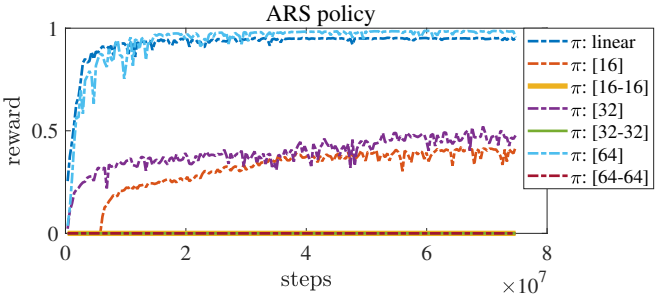


Fig. 4. Reward profiles for the in-place jumping achieved by the ARS algorithm when using different policy representations.

IV. Evaluation

The proposed architecture consistently succeeded in generating effective acrobatic motions, as the one presented in Fig. 2 and in the video attachment.

In this section, we report an in-depth analysis of these performance, including ablation studies and comparisons.

A. Technical details

We build the PyBullet environment [39] to emulate Go1's motion. To mimic the parallel spring in the training process, we add the resultant torque by spring extensions (computed by the formula in Appendix A) to each joint. To run the experiments, we use the RL-Zoo training framework [40]. The neural network implementing π_{PPO} is a two-layer DNN with 64 units each. We use $\tanh$ as the activation function and set the learning rate to be $2e^{-4}$. The batch size is 4096, and the discount factor γ is 0.999. The clip range is 0.1[4].

B. ES: linear policy vs nonlinear network

We first compare the linear policy with the NN-based nonlinear policies in ARS training. Taking the in-placed jumping, for example, the learned curves using different representations are plotted in Fig. 4. It demonstrates that the linear policy (see 'π : linear' curve) is among the best representations, obtaining the highest rewards. The comparable performance with higher rewards is achieved by the NN-based policy with one hidden layer of 64 units ('π:[64]' curve). Even though, the liner policy converges faster with a smoother reward profile. Thus, it demonstrates that the linear policy representation in the first-stage ARS training process is good enough to achieve competitive results.

C. Learning explosive jumping motion

Using the hyperparameters in Appendix C, the quadrupedal robot learns the jumping motion. The first three columns in Fig. 5 present the results under two stages, i.e., 'ARS' and 'ARS+DRL' (described in Fig. 1).

As can be seen from the first column of Fig. 5, both schemes can accomplish two jumping tasks, i.e., jumping in place and jumping forward, successfully. Among the two strategies, 'ARS+DRL' achieves the maximal mean reward (see the green bar in the second column), accompanied by the larger jumping height for both tasks (see the green bar in the third column). In addition, compared with 'ARS', a longer jumping distance is achieved by 'ARS+DRL' in the forward jumping task, as can be seen from the yellow bar

[4]Code is available: https://github.com/francescovezzi/quadruped-springs

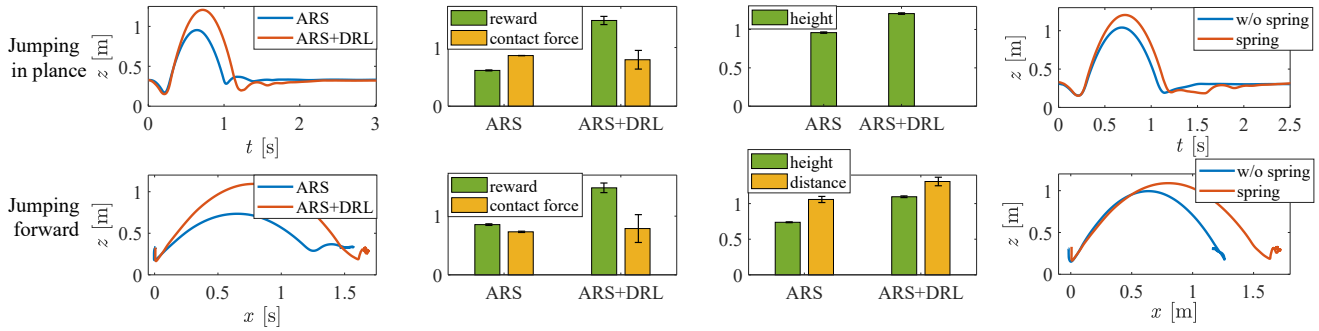


Fig. 5. Jumping performance comparison. In the second column, a high reward means a better performance, while a low contact force means a compliant landing motion. Note the contact forces are divided by 1000N. In the third column, the 'height' and 'length' separately denote the maximal jumping height and jumping distance. In the fourth column, the learned CoM trajectories (under 'ARS+DRL' policy) for a rigid robot and a soft robot are compared. 'w/o spring' denotes the rigid case without springs engaged.

TABLE I

JUMPING PERFORMANCE WITH/WITHOUT SPRING ENGAGED

	Jumping in place		Jumping forward	
	spring	w/o spring	spring	w/o spring
Height [m]	**1.206±0.010**	**1.045±0.003**	1.100±0.014	0.995±0.002
Distance [m]	0.069±0.015	0.207±0.024	**1.301±0.073**	**1.057±0.019**
Force [N]	799±189	871±228	796±211	800±369

in the third column. Further observations reveal that for in-placed jumping, 'ARS+DRL' achieves a smaller contact force, contributing to a compliant landing motion.

It should be mentioned that the first-stage ES itself ('ARS' in Fig. 5) could accomplish two jumping tasks successfully, using the sparse reward defined in Section III-D.2.a. Considering the 'ARS+DRL' policy works better, in the following section, we evaluate it by default.

D. Articulated soft quadruped vs rigid quadruped

Using the two-stage learning scheme, we train the control policies for both the articulated soft robot (with parallel springs engaged) and the rigid robot (without engaging parallel springs). To our knowledge, this is the first report on learning quadrupedal jumping with parallel elasticity in a normal gravity environment.

The fourth column of Fig. 5 compares center of mass (CoM) trajectories in both cases (using 'ARS+DRL' policy), and Table I lists the meaning values of jumping height, jumping distance and the largest landing force. It turns out that a larger height (15.4% higher) and a longer distance (23.1% further) are achieved when exploiting the parallel springs. Besides, parallel springs help to gain smaller contact forces with fewer oscillations. More details can be found in the attached video.

E. Beyond single jumping

Using the reward function defined in Appendix B.1 and Appendix B.2, the proposed two-stage learning scheme enables learning the pronking and back-flip motions, which are illustrated in Fig. 2. Note that no reference motion is used here either. Fig. 6 presents the limit cycle behaviour when executing the pronking motion. We find that the vertical CoM and pitch rotation would converge to the limit cycle using the learned policy. Besides, we find that without engaging parallel springs, the back-flip motion is hardly realized.

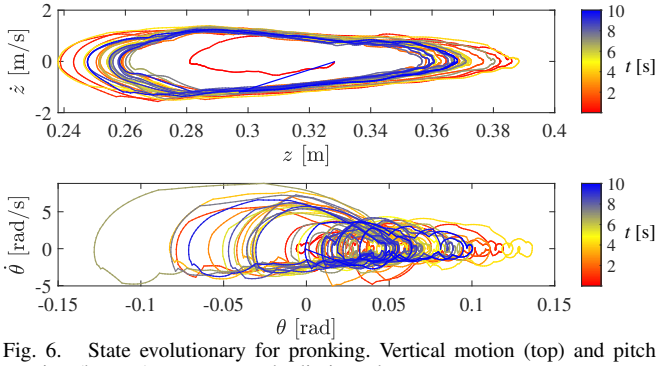


Fig. 6. State evolutionary for pronking. Vertical motion (top) and pitch rotation (bottom) converge to the limit cycle.

TABLE II

ENVIRONMENT RANDOMIZATION

leg mass	mass offset	CoM offset	trunk mass	spring stiffness	spring damping
±20%	[0 4] [kg]	±[0.2 0 0.2] [m]	accordingly	±30%	±30%

F. Environment randomization

To bridge the sim2real gap, we randomize the springs coefficient including damping and stiffness, the total mass, robot mass distribution, CoM offset, and the inertia tensor, as reported in Table II. In each episode, the environment changes according to the random sampling.

To save space, we here only present the results of jumping motion. Fig. 7 compares the jumping performance when using different strategies, where 'ARS-r' and 'ARS+DRL-r' separately denote the retrained policy using domain randomization techniques [41]. For the in-placed jumping task, domain randomization retraining helps to increase the success rate. In particular, the two-stage retraining with dynamic randomization, i.e., 'ARS+DRL-r', increases the jumping height. For forward jumping, the success rate is also improved by domain randomization. One interesting thing is that the forward jumping distance is decreased compared to the 'ARS+DRL' policy without domain randomization. We guess it is because the learning scheme is prone to guarantee the successful rate of forward jumping against environmental uncertainties.

V. CONCLUSIONS

In this work, we propose a two-stage approach for learning highly dynamic quadrupedal motions from scratch. In the first stage, an evolution strategy is used to generate the linear

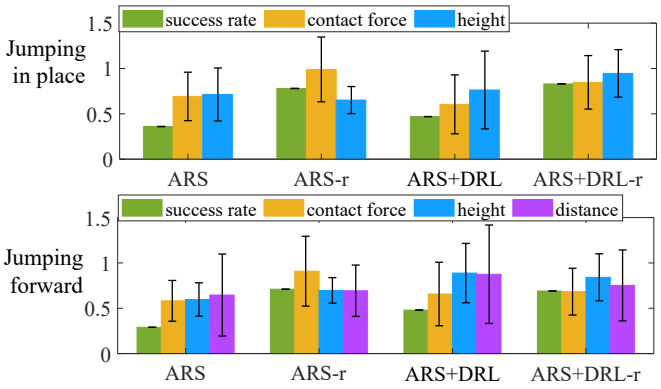


Fig. 7. Jumping performances in randomized environments. 'xx-r' denotes the retrained policy with environment randomization.

policy, without defining a reference trajectory in advance. Then, we refine and retrain the policy by deep reinforcement learning, achieving higher performance. Compared with current studies that focus on one specific task, it has been demonstrated that, by slightly modifying the reward function, versatile acrobatic motions including pronking and back-flip can be learned, with shared state space and observation space.

Aside from the rigid case, we also apply the method to the articulated soft robot with parallel elasticity. It turned out that the soft robot could still learn highly dynamic motions using the proposed method. Furthermore, exploitation of the parallel springs enhances the motion performance.

Future work will focus on hardware validation. In addition, we would like to apply the method to humanoid robots, generating general locomotion control polices [42], [43].

APPENDIX

A. Exploiting parallel elasiticity

We also explore parallel elasticity in achieving highly dynamic motions by attaching parallel springs to the actuated joints of the Unitree Go1 robot [3]. For the sake of simplicity, we here consider the springs attached to the joints in the sagittal plane, i.e., thigh and calf joints (see Fig. 3). The excessive torque on the i-th joint caused by the spring deformation (τ^{s}) is computed as $\tau_i^{\mathrm{s}} = k_i\left(q_i^{\mathrm{ref}} - q_i^0\right) + c_i\dot{q}_i^{\mathrm{ref}}$ with $k_i \in \mathbb{R}$ and $c_i \in \mathbb{R}$ separately being the spring stiffness and damping, $q_i^{\mathrm{ref}} \in \mathbb{R}$, $\dot{q}_i^{\mathrm{ref}} \in \mathbb{R}$, and $q_i^0 \in \mathbb{R}$ separately denoting the commanded angle, angular velocity) and rest angle of the i-th joint.

B. Rewards for pronking and back-flip

1) Reward for pronking: To begin with, we evaluate single jump by

$$p_{\mathrm{sj}} = w_h h_n + w_d d_n. \tag{13}$$

Then, we define the performance array for n jumps during one episode as $p_j = \left[p_{\mathrm{sj}}^1, p_{\mathrm{sj}}^2, \ldots, p_{\mathrm{sj}}^n\right]$. Given p_j, we can compute the average performance p_j^{avg} and the performance entropy $S(p_j)$ (considering $p_{\mathrm{sj}}^i \in [0,1]$ and $\sum_{i=1}^n p_{\mathrm{sj}}^i = 1$). The sparse reward used for the first-stage learning is then defined as

$$\begin{aligned} r_{\mathrm{avg}} &= p_j^{\mathrm{avg}}\left\{g_\theta(\bar{\theta}) + w_t t_n + w_s S(p_j) + k\right\}, \\ {}^1r^{\mathrm{pronking}} &= w_{\mathrm{avg}} r_{\mathrm{avg}} + w_{\mathrm{max}} p_{\mathrm{sj}}^{\mathrm{max}} + w_c c_j + b, \end{aligned} \tag{14}$$

TABLE III

REWARD PENALTIES FOR THE FIRST-STAGE ARS TRAINING.

Jumping in place	h_f^{jip}	a_θ^{jip}	b_θ^{jip}	a_d^{jip}	b_d^{jip}	c_h^{jip}	b^{jip}	q^{jip}	m^{jip}
	0.9	0.3	0.0225	0.05	0.05	0.7	0.1	0.08	0.064
Jumping forward	h_f^{jf}	a_θ^{jf}	b_θ^{jf}	d_f^{jf}	c_d^{jf}	c_h^{jf}	b^{jf}	q^{jf}	m^{jf}
	0.3	0.25	0.0225	1.3	0.5	25	0.1	0.08	0.096

TABLE IV

REWARD PENALTIES FOR THE SECOND-STAGE PPO TRAINING.

Jumping in place	a_h^{jip}	$h_{\mathrm{min}}^{\mathrm{jip}}$	$h_{\mathrm{max}}^{\mathrm{jip}}$	a_c^{jip}	$f_{\mathrm{min}}^{\mathrm{jip}}$	a_d^{jip}
	0.01	0.29	1.3	$1.5e^{-4}$	800	$6.5e^{-4}$
	b_d^{jip}	a_s^{jip}	b_s^{jip}	a_θ^{jip}	b_θ^{jip}	m^{jip}
	40	$3e^{-3}$	0.1	$4.2e^{-3}$	26	0.25
Jumping forward	a_h^{jf}	$h_{\mathrm{min}}^{\mathrm{jf}}$	$h_{\mathrm{max}}^{\mathrm{jf}}$	a_c^{jf}	$f_{\mathrm{min}}^{\mathrm{jf}}$	k_d^{jf}
	$6.5e^{-3}$	0.29	1.1	$1.2e^{-4}$	800	$1.52e^{-2}$
	$d_{\mathrm{max}}^{\mathrm{jf}}$	a_s^{jf}	b_s^{jf}	a_θ^{jf}	b_θ^{jf}	b^{jf}
	1.3	$3e^{-3}$	0.1	$4.2e^{-3}$	26	0.025

where c_j is the number of jumps whose performance is above a certain threshold, b is a bonus for a successful termination, t_n is the ratio between the time the episode ends and the maximal episode time, k is a constant and $p_{\mathrm{sj}}^{\mathrm{max}}$ is the maximal element of the array p_j. $w_\times$ denotes the hyperparameter.

For the second-stage training, the reward function is similar to the one already defined in (8), added by two additional terms. One term (r_p) is used to minimize the actuators' energy consumption, and the other term (r_j) provides a reward each time the robot successfully performs a jump, which are separately defined as

$$r_p = g_p(E), \qquad r_j = w_j p_j^{\mathrm{avg}}\left[g_s\left(S\left(p_j\right)\right) + l_j\right], \tag{15}$$

where E is the energy cost in one episode, l_j is a constant reward, and the weight w_j is 0 if the jump performance is below a certain threshold.

2) Reward for back-flipping: Denoting $\theta_n \in \mathbb{R}$ as the normalized maximum pitch angle in one episode (computed by (4)), the reward function for the first-stage ARS training is defined as

$${}^1r^{\mathrm{flip}} = w_h\bar{h} + w_\theta\theta_n + w_{\mathrm{h}\theta}(\bar{h}\theta_n) + b, \tag{16}$$

where b is a bonus if the episode ends successfully, $w_\times$ denotes the hyperparameter.

The reward for the second-stage DRL training is similar to the one defined in (8). One exception is the reward with regard to the jumping distance (r_d in (8)) is discarded and the one with regard to the pitch angle is proportional to it, i.e., $r_\theta = g_\theta(\theta)$. Another point is that we change r_{bonus} in (8) to be

$$r_{\mathrm{bonus}} = \begin{cases} w_{\mathrm{bonus}}(\bar{h}\theta_n) & \text{not early termination,} \\ 0 & \text{otherwise.} \end{cases} \tag{17}$$

C. Hyperparameters for learning jumping motion

Hyperparameters for the first-stage and second-stage training are separately listed in Table III and Table IV.

REFERENCES

[1] M. Hutter, C. Gehring, D. Jud, A. Lauber, C. D. Bellicoso, V. Tsounis, J. Hwangbo, K. Bodie, P. Fankhauser, M. Bloesch *et al.*, "Anymal-a highly mobile and dynamic quadrupedal robot," in *Proc. IEEE/RSJ Int. Conf. Intell. Robot. Syst.*, 2016, pp. 38–44.

[2] B. Katz, J. Di Carlo, and S. Kim, "Mini cheetah: A platform for pushing the limits of dynamic quadruped control," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2019, pp. 6295–6301.

[3] U. Robotics, "Unitree robot go1: https://m.unitree.com/products/go1," 2021.

[4] F. Angelini, P. Angelini, C. Angiolini, S. Bagella, F. Bonomo, M. Caccianiga, C. Della Santina, D. Gigante, M. Hutter, T. Nanayakkara *et al.*, "Robotic monitoring of habitats: the natural intelligence approach," *IEEE Access*, 2023.

[5] C. Della Santina, M. G. Catalano, A. Bicchi, M. Ang, O. Khatib, and B. Siciliano, "Soft robots," *Ency. Robot.*, vol. 489, 2021.

[6] A. Badri-Spröwitz, A. Aghamaleki 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, p. eabg4055, 2022.

[7] F. Bjelonic, J. Lee, P. Arm, D. Sako, D. Tateo, J. Peters, and M. Hutter, "Learning-based design and control for quadrupedal robots with parallel-elastic actuators," *IEEE Robot. Autom. Lett.*, vol. 8, no. 3, pp. 1611–1618, 2023.

[8] P. Arm *et al.*, "Spacebok: A dynamic legged robot for space exploration," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2019, pp. 6288–6294.

[9] A. Raffin, D. Seidel, J. Kober, A. Albu-Schäffer, J. Silvério, and F. Stulp, "Learning to exploit elastic actuators for quadruped locomotion," *arXiv preprint arXiv:2209.07171*, 2022.

[10] Q. Nguyen, M. J. Powell, B. Katz, J. Di Carlo, and S. Kim, "Optimized jumping on the mit cheetah 3 robot," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2019, pp. 7448–7454.

[11] 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.

[12] Z. Song, L. Yue, G. Sun, Y. Ling, H. Wei, L. Gui, and Y.-H. Liu, "An optimal motion planning framework for quadruped jumping," in *Proc. IEEE/RSJ Int. Conf. Intell. Robot. Syst.*, 2022, pp. 11 366–11 373.

[13] Y. Ding, A. Pandala, C. Li, Y.-H. Shin, and H.-W. Park, "Representation-free model predictive control for dynamic motions in quadrupeds," *IEEE Trans. Robot.*, vol. 37, no. 4, pp. 1154–1171, 2021.

[14] G. García, R. Griffin, and J. Pratt, "Time-varying model predictive control for highly dynamic motions of quadrupedal robots," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2021, pp. 7344–7349.

[15] J. Hwangbo, J. Lee, A. Dosovitskiy, D. Bellicoso, V. Tsounis, V. Koltun, and M. Hutter, "Learning agile and dynamic motor skills for legged robots," *Sci. Robot.*, vol. 4, no. 26, p. eaau5872, 2019.

[16] Z. Fu, A. Kumar, J. Malik, and D. Pathak, "Minimizing energy consumption leads to the emergence of gaits in legged robots," *arXiv preprint arXiv:2111.01674*, 2021.

[17] N. Rudin, D. Hoeller, P. Reist, and M. Hutter, "Learning to walk in minutes using massively parallel deep reinforcement learning," in *Conference on Robot Learning*. PMLR, 2022, pp. 91–100.

[18] Z. Tang, D. Kim, and S. Ha, "Learning agile motor skills on quadrupedal robots using curriculum learning," in *Proc. Int. Conf. Robot. Intell. Tech. Appl.*, vol. 3, 2021.

[19] G. B. Margolis, G. Yang, K. Paigwar, T. Chen, and P. Agrawal, "Rapid locomotion via reinforcement learning," *arXiv preprint arXiv:2205.02824*, 2022.

[20] A. Iscen, G. Yu, A. Escontrela, D. Jain, J. Tan, and K. Caluwaerts, "Learning agile locomotion skills with a mentor," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2021, pp. 2019–2025.

[21] S. Chen, B. Zhang, M. W. Mueller, A. Rai, and K. Sreenath, "Learning torque control for quadrupedal locomotion," *arXiv preprint arXiv:2203.05194*, 2022.

[22] G. B. Margolis, T. Chen, K. Paigwar, X. Fu, D. Kim, S. Kim, and P. Agrawal, "Learning to jump from pixels," *arXiv preprint arXiv:2110.15344*, 2021.

[23] Y. Fuchioka, Z. Xie, and M. Van de Panne, "Opt-mimic: Imitation of optimized trajectories for dynamic quadruped behaviors," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2023, pp. 5092–5098.

[24] M. Bogdanovic, M. Khadiv, and L. Righetti, "Model-free reinforcement learning for robust locomotion using demonstrations from trajectory optimization," *arXiv preprint arXiv:2107.06629*, 2021.

[25] C. Li, S. Blaes, P. Kolev, M. Vlastelica, J. Frey, and G. Martius, "Versatile skill control via self-supervised adversarial imitation of unlabeled mixed motions," *arXiv preprint arXiv:2209.07899*, 2022.

[26] C. Li, M. Vlastelica, S. Blaes, J. Frey, F. Grimminger, and G. Martius, "Learning agile skills via adversarial imitation of rough partial demonstrations," in *Conference on Robot Learning*. PMLR, 2023, pp. 342–352.

[27] L. Smith, J. C. Kew, T. Li, L. Luu, X. B. Peng, S. Ha, J. Tan, and S. Levine, "Learning and adapting agile locomotion skills by transferring experience," *arXiv preprint arXiv:2304.09834*, 2023.

[28] Y. Yang, X. Meng, W. Yu, T. Zhang, J. Tan, and B. Boots, "Continuous versatile jumping using learned action residuals," in *Learning for Dynamics and Control Conference*. PMLR, 2023, pp. 770–782.

[29] Y. Yang, G. Shi, X. Meng, W. Yu, T. Zhang, J. Tan, and B. Boots, "Cajun: Continuous adaptive jumping using a learned centroidal controller," *arXiv preprint arXiv:2306.09557*, 2023.

[30] Z. Li, X. B. Peng, P. Abbeel, S. Levine, G. Berseth, and K. Sreenath, "Robust and versatile bipedal jumping control through multi-task reinforcement learning," *arXiv preprint arXiv:2302.09450*, 2023.

[31] N. Kashiri, A. Abate, S. J. Abram, A. Albu-Schaffer, P. J. Clary, M. Daley, S. Faraji, R. Furnemont, M. Garabini, H. Geyer *et al.*, "An overview on principles for energy efficient robot locomotion," *Front. Robot. AI*, vol. 5, p. 129, 2018.

[32] H. Mania, A. Guy, and B. Recht, "Simple random search of static linear policies is competitive for reinforcement learning," *Adv. Neural Inf. Process.*, vol. 31, 2018.

[33] V. François-Lavet, P. Henderson, R. Islam, M. G. Bellemare, J. Pineau *et al.*, "An introduction to deep reinforcement learning," *Found. Trends Mach. Learn.*, vol. 11, no. 3-4, pp. 219–354, 2018.

[34] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[35] H. Shi, B. Zhou, H. Zeng, F. Wang, Y. Dong, J. Li, K. Wang, H. Tian, and M. Q.-H. Meng, "Reinforcement learning with evolutionary trajectory generator: A general approach for quadrupedal locomotion," *IEEE Robot. Autom. Lett.*, vol. 7, no. 2, pp. 3085–3092, 2022.

[36] T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever, "Evolution strategies as a scalable alternative to reinforcement learning," *arXiv preprint arXiv:1703.03864*, 2017.

[37] A. Y. Majid, S. Saaybi, V. Francois-Lavet, R. V. Prasad, and C. Verhoeven, "Deep reinforcement learning versus evolution strategies: a comparative survey," *IEEE Trans. Neural Netw. Learn.*, 2023.

[38] N. Hansen, D. V. Arnold, and A. Auger, "Evolution strategies," *Springer handbook of computational intelligence*, pp. 871–898, 2015.

[39] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning," http://pybullet.org, 2016–2019.

[40] A. Raffin, "Rl baselines3 zoo," https://github.com/DLR-RM/rl-baselines3-zoo, 2020.

[41] Z. Xie, X. Da, M. Van de Panne, B. Babich, and A. Garg, "Dynamics randomization revisited: A case study for quadrupedal locomotion," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2021, pp. 4955–4961.

[42] J. Ding, X. Xiao, N. G. Tsagarakis, and Y. Huang, "Robust gait synthesis combining constrained optimization and imitation learning," in *Proc. IEEE/RSJ Int. Conf. Intell. Robot. Syst.*, 2020, pp. 3473–3480.

[43] J. Ding, X. Xiao, and N. G. Tsagarakis, "Nonlinear optimization of step duration and step location," in *Proc. IEEE/RSJ Int. Conf. Intell. Robot. Syst.*, 2019, pp. 2849–2854.