Reinforcement Learning of Single Legged Locomotion

Péter Fankhauser, Marco Hutter, Christian Gehring, Michael Bloesch, Mark A. Hoepflinger, Roland Siegwart

***Abstract*— This paper presents the application of reinforcement learning to improve the performance of highly dynamic single legged locomotion with compliant series elastic actuators. The goal is to optimally exploit the capabilities of the hardware in terms of maximum jump height, jump distance, and energy efficiency of periodic hopping. These challenges are tackled with the reinforcement learning method Policy Improvement with Path Integrals (PI^2) in a model-free approach to learn parameterized motor velocity trajectories as well as high-level control parameters. The combination of simulation and hardware-based optimization allows to efficiently obtain optimal control policies in an up to 10-dimensional parameter space. The robotic leg learns to temporarily store energy in the elastic elements of the joints in order to improve the jump height and distance. In addition, we present a method to learn time-independent control policies and apply it to improve the energetic efficiency of periodic hopping.**

I. INTRODUCTION

The nature of legged robots raises big challenges for controlling these systems. High degrees of freedom (DOF) and highly nonlinear non-smooth dynamics (due to interaction with the environment) count amongst the difficulties that researchers face in the development of control algorithms for robots with legs. Model-based control approaches are naturally limited by the fact that accurate models are difficult to obtain and are hence less suitable to achieve maximal performance, efficiency, and accuracy on the hardware. A common method to overcome modeling inaccuracies is to apply reinforcement learning directly to the physical robot. Some successful results have been achieved on four- [1] and two-legged robots [2]–[4]. They focused on robots with position controlled, stiff actuators, which fundamentally differentiates it from our system. Learning of a vertical jump with a robot with compliant actuators was presented in [5]. The control input was parameterized with two step functions with variable timing as basis functions for each actuator. The optimization was conducted by the combination of random exploration and hill-climbing search. This approach focuses on the pneumatic actuation of this robot, which restricts the applicability to our work. For purely simulation-based approaches, earlier work on learning control for single legged robots has been presented in [6], [7]. The topology of a neural network is learned in simulation with a simple model. However, these methods are not suitable for learning on the real robot as the training requires a big number of trials (up to $\sim 10^6$ in [7]) to cover the state space. In a closely related work [8], a genetic algorithm was used to generate a vertical jump for a simulated single legged robot with mechanical springs in the joints. The algorithm has evolved to a countermovement jump at which a torque in the springs is developed before the upward movement of the jump. We aim at achieving similar results for a real robot with an additional horizontal DOF.

This work was supported by the Swiss National Centre of Competence in Research (NCCR). The authors are with the Autonomous Systems Lab (ASL), ETH Zurich, Switzerland, pfankhauser@ethz.ch

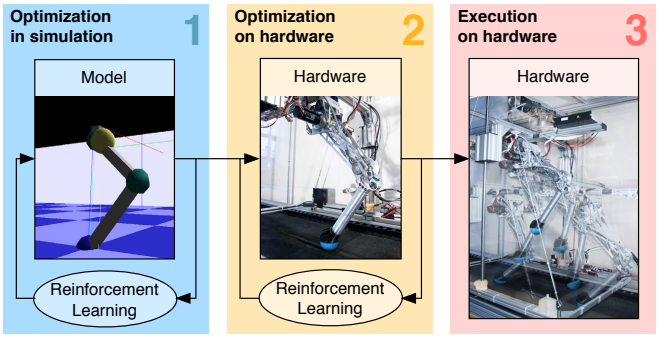
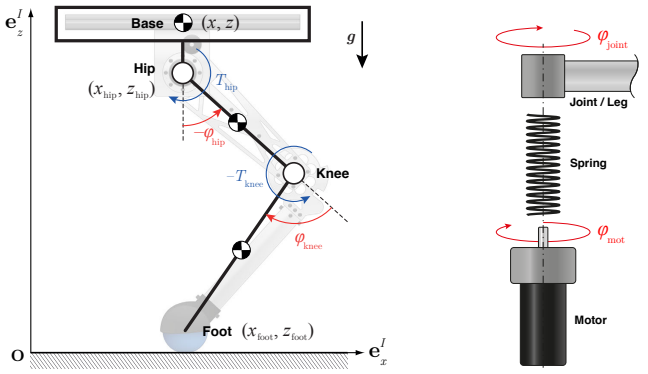


Fig. 1: A two-step learning process allows for efficient optimization in simulation (1) and circumvention of the model inaccuracies on the hardware (2).

Scalability, high convergence rate and numerical robustness are crucial for reinforcement learning on real robots. While classic reinforcement learning methods do not scale well to high-dimensional continuous state and action spaces, recent developments in robot learning have overcome these limitations (see [9], [10] for a comprehensive overview). In particular, the method *Policy Improvement with Path Integrals* (PI^2) [11], has lately shown promising results in learning complex behavior for interaction with the environment [12], [13]. It combines path integral methods for optimal control with direct reinforcement learning in a model-free approach. The algorithm allows to iteratively update the control policy in high dimensions and has shown fast convergence in comparison to other reinforcement learning methods (e.g. REINFORCE, PoWER). PI^2 does not rely on the computation of gradients, which is of great importance considering the application on a physical system with noise and non-smooth dynamics. Also, tuning of open parameters is reduced in PI^2 to the manual definition of the exploration noise. This way, the user can focus on the design of the cost function. However, the scalability to high dimensions comes at the cost that the PI^2 algorithm (as all trajectory-based learning algorithms) finds only locally optimal solutions. Therefore, it can be crucial to start with reasonably good initial policies depending on the complexity of the task.

In this work, we apply PI^2 to learn jumping and hopping maneuvers on the robotic single leg *ScarlETH* [14]. ScarlETH is a planar system with an articulated leg driven by highly compliant series elastic actuators (SEA) [15] in the hip and knee joint. To maximize the performance and locomotion efficiency, we apply a two-step optimization



(a) (b)

Fig. 2: The one-legged robot has a total number of 6 DOF (x, z, $\varphi_{\text{hip,mot}}$, $\varphi_{\text{hip,joint}}$, $\varphi_{\text{knee,mot}}$, and $\varphi_{\text{knee,joint}}$). The torque in each joint is given by the deflection of the spring that is attached between the motor and the joint.

framework by combining simulation-based and hardware-based reinforcement learning as illustrated in Fig. 1. The main contributions of this paper are presented in the context of reinforcement learning of jumping and hopping maneuvers for a single robotic leg with highly compliant joints: We extend the application range of PI^2 from typically slow reaching, grasping and manipulation tasks [12], [13] to highly dynamic jumping maneuvers. We directly parameterize the motor velocity trajectories (input to the SEA) and do not enforce joint position (output of the SEA) tracking with an additional controller. This way, the learning algorithm learns to excite the inherent dynamics of the system and to exploit them to achieve maximal performance. Additionally, we present a novel strategy to learning of time-independent periodic control policies with PI^2 to find an optimal excitation frequency. We have summarized the experiments and the learning progress in a video: youtu.be/xw6pSal-OgI

The remainder of this paper is structured as follows. The one legged platform ScarlETH is described in Section II and an introduction to PI^2 is given in Section III. The learning framework and the results for single jumps is given in Section IV and for energy efficient hopping in Section V.

II. SYSTEM DESCRIPTION

ScarlETH is a planar one-legged robotic system [14]. It is modeled consisting of a main body/base, a thigh and a shank connected at the hip and knee joint respectively as shown in Fig. 2(a). A guiding unit allows the robot to move freely in horizontal and vertical direction while the rotation about the pitch axis is blocked. The position of the base (x, z) is measured with wire sensors while the position of the foot $(x_{\text{foot}}, z_{\text{foot}})$ is given by the hip and knee joint angles φ_{hip} and φ_{knee}. A series elastic actuator [15] drives each joint and is described by the motor position φ_{mot} and joint position φ_{joint} (Fig. 2(b)). The joint torque is given by the deflection of the spring as $T = c_{\text{sprg}}(\varphi_{\text{mot}} - \varphi_{\text{joint}})$ with c_{sprg} denoting the spring constant. Contact of the foot with the ground is detected through a threshold value for the deflection of the knee joint spring.

The controller for each joint provides different actuation modes. In the *motor velocity control* mode, the desired motor velocities for the hip $\dot{\varphi}^{\text{des}}_{\text{hip,mot}}$ and knee $\dot{\varphi}^{\text{des}}_{\text{knee,mot}}$ are directly transferred to the motor electronics. This is the lowest control input method to the system. In the *joint position control* mode, the desired joint positions for hip $\varphi^{\text{des}}_{\text{hip}}$ and knee $\varphi^{\text{des}}_{\text{knee}}$ are tracked. In the *torque control* mode, precise torque regulation is achieved through control of the spring deflection for the desired torques of hip $T^{\text{des}}_{\text{hip}}$ and knee $T^{\text{des}}_{\text{knee}}$.

The system has a total moving mass of 6.5 kg of which ~50% are on account of the vertical gliders and can be interpreted as payload. For the long jump (Subsection IV-B) and the periodic hopping (Section V) experiments, we attach springs with a constant force to compensate for the payload. Our simulation and software setup is identical to the framework described in [16]. This setup allows to quickly change from the simulation to the real robot while running the same high-level control software.

III. OPTIMIZATION FRAMEWORK

In the first step of our optimization framework (1 in Fig. 1), we run the simulation based on the model of the rigid body dynamics (Section II) in order to find a control policy that is optimized for the dominating dynamics of the system. In the second optimization step (2 in Fig. 1), we continue the learning procedure by applying it directly on the real robot. By leveraging the results from the first step, we are now circumventing the modeling errors and achieve a convergence to a new cost minimum in few trials. Executing the policy from the second optimization process on the hardware (3 in Fig. 1) then shows a consistent repeatability.

The reinforcement learning algorithm is based on PI^2 [11], a model-free, direct policy learning method. The algorithm evaluates a path integral with Monte Carlo rollouts (repeated, random sampling). It optimizes a control policy $a(t)$, which is parameterized by a basis function representation

$$a(t) = \boldsymbol{g}(t)^{\mathsf{T}}\boldsymbol{\theta}\,, \tag{1}$$

with the policy parameter vector $\boldsymbol{\theta} \in \mathbb{R}^{M\times 1}$ and a set of basis functions $\boldsymbol{g}(t) \in C(\mathbb{R}, \mathbb{R}^{M\times 1})$ with M being the number of parameters/basis functions. The aim is to find a set of parameters $\boldsymbol{\theta}$ which minimizes a cost function J with

$$J(\boldsymbol{\theta}) = \phi_{t_N} + \int_{t_i}^{t_N} \left(q(t) + \frac{1}{2}\boldsymbol{\theta}^{\mathsf{T}}\boldsymbol{R}\,\boldsymbol{\theta} \right) \mathrm{d}t\,, \tag{2}$$

for the time interval from t_i to t_N. The cost function consists of the terminal cost term $\phi(t_N)$, the immediate state cost $q(t)$, and the control cost $\frac{1}{2}\boldsymbol{\theta}^{\mathsf{T}}\boldsymbol{R}\,\boldsymbol{\theta}$ with $\boldsymbol{R} \in \mathbb{R}^{M\times M}$ denoting the control cost matrix.

During the learning procedure the cost function is minimized in an iterative process of exploration and parameter updating. The state space is explored by injecting the Gaussian mean-zero noise $\boldsymbol{\varepsilon} \in \mathbb{R}^{M\times 1}$ in the policy parameters and executing a number of stochastic control policies for $k = 1...K$ with

$$a_k(t) = \boldsymbol{g}(t)^{\mathsf{T}}(\boldsymbol{\theta} + \boldsymbol{\varepsilon}_k)\,. \tag{3}$$

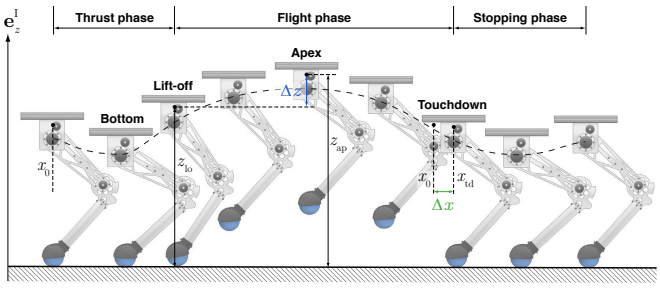


Fig. 3: The time evolution of a vertical jump consists of the three phases *thrust*, *flight*, and *stopping phase*. The goal is to maximize the jump height Δz while minimizing the touchdown offset Δx.

The basis functions are nonlinear and distributed in such a way that they provide an appropriate expressivity for the particular task. We use normalized Gaussian basis functions,

$$g_m(t) = \frac{\Psi_m(t)}{\sum_{j=1}^{M} \Psi_j(t)} \quad \text{with} \quad \Psi_m(t) = \exp\left(-\frac{1}{2\sigma_m^2}(t - c_m)^2\right) , \quad (4)$$

where σ_m and c_m are the widths and centers of the basis function with index $m = 1...M$.

The variation of the policy parameters in (3) leads to different costs for each k-th execution of the policy (rollout). The policy improvement step (PI^2 update) is computed based on the trajectory costs and performs the parameter update

$$\boldsymbol{\theta}_{r+1} \longleftarrow \boldsymbol{\theta}_r + \delta\boldsymbol{\theta} , \quad (5)$$

for r the current parameter update step. The updated policy is expected to generate trajectories that lead to decreasing costs in the future. We refer to [11] for the derivation of the PI^2 update algorithm.

IV. JUMPS FROM REST

A jump from a resting posture is a highly explosive maneuver that requires correct timing and precise coordination of the joints. First, we focus on learning the motion for a purely vertical jump (Subsection IV-A) and then extend to jumps with different jump distances (Subsection IV-B).

A. Vertical Jump

The robot starts the vertical jump with the *thrust phase* from rest with the base horizontal start position x_0 (Fig. 3). During this phase, the leg has to develop a maximal thrust to achieve a high vertical lift-off velocity. It reaches the apex point (highest point in the flight trajectory) in the *flight phase*. The jump height Δz is defined as the difference between the base vertical lift-off position z_{lo} and the base vertical apex position z_{ap} as

$$\Delta z = z_{\text{ap}} - z_{\text{lo}} . \quad (6)$$

The touchdown offset Δx is given by the difference of the base horizontal start position x_0 and the base horizontal touchdown position x_{td} as

$$\Delta x = x_0 - x_{\text{td}} . \quad (7)$$

The goal of the vertical jump optimization is to maximize the jump height Δz while minimizing the touchdown offset Δx.

1) Control Structure: For the *thrust phase*, we choose the desired motor velocity for each joint, $\dot{\varphi}^{\text{des}}_{\text{hip,mot}}(t)$ and $\dot{\varphi}^{\text{des}}_{\text{knee,mot}}(t)$, as a feedforward control input (motor velocity control mode) whereby the control loop over the SEA is not closed (as opposed to e.g. torque control mode). This way, the learning framework can directly excite the SEA at the lowest control level which allows to include the dynamics of the SEA as part of the optimization. As soon as a lift-off is detected, the control is switched to the joint position control mode with predefined configuration for the duration of the *flight phase*.

2) Policy: Optimization is applied to the control policy of the thrust phase. The two-dimensional policy $\boldsymbol{a}_t = \left(\dot{\varphi}^{\text{des}}_{\text{hip,mot}}(t),\ \dot{\varphi}^{\text{des}}_{\text{knee,mot}}(t)\right)^{\mathsf{T}}$ is parameterized with equally distributed Gaussian basis functions. The basis functions are parameterized directly by the time t. The number of basis functions per dimension M, the execution time of the policy t_N and the start base vertical position z_0 influence the degree of optimization that can be achieved. We experience that providing the policy enough time ($t_N = 0.45\,\text{s}$), the trajectories from different start base positions $z_0 = \{0.32, 0.4, 0.48\}\,\text{m}$ converge to a trajectory with almost identical jump height. The combination of $M = 5$ (10 parameters in total), $t_N = 0.3\,\text{s}$ and $z_0 = 0.4\,\text{m}$ reflects a sensible trade-off between expressivity and complexity of the policy. Increasing the execution time or number of parameters does not lead to significantly better performance.

The initial policy is chosen manually such that the learning procedure is biased towards a trajectory with lift-off at the end of the policy execution. This ensures that all parameters are used to shape the trajectory. The variance of the exploration noise Σ_ε is set to $1.2\,\frac{\text{rad}}{\text{s}}$ for learning in simulation and to $0.7\,\frac{\text{rad}}{\text{s}}$ for learning on the hardware. The number of rollouts per update is chosen to be $K = 10$ and the 5 best rollouts from the last iteration are additionally reused for the policy improvement step.

3) Cost Function: We evaluate the jump height Δz (in m) and touchdown offset Δx (in m) in the terminal cost term as

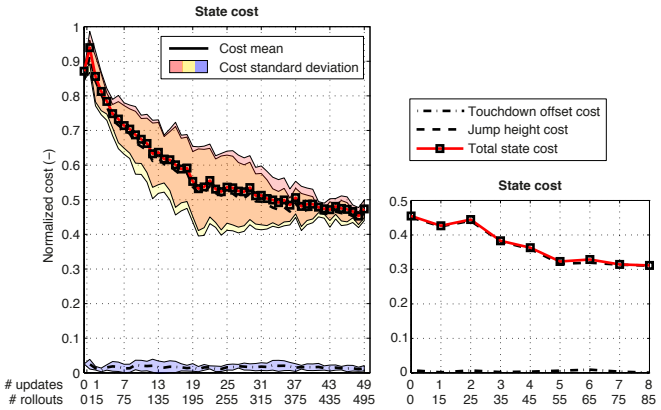
$$\phi_{t_N} = 0.3\ |\Delta x| - \Delta z\ .^{\text{a}} \quad (8)$$

The weighting factor 0.3 has been tuned manually in simulation to reach a reasonable trade-off between the jump height and touchdown offset. The immediate state cost q_t and the control cost are neglected, as they are not relevant for the optimality of this task.

4) Results: The mean and the standard deviation of the cost evolution for multiple learning sessions for the simulation-based learning is given in Fig. 4(a). For the learning process on the hardware, the initial policy is given by the optimized policy obtained from the simulation. The algorithm is able to converge to a new cost minimum within only a few update steps (Fig. 4(b)).[b]

[a] Negative costs $\phi_{t_N} < 0$ are valid values for the optimization framework.

[b] It is coincidental that the cost for the last update in the simulation Fig. 4(a) is close to the value for the initial rollout on the hardware Fig. 4(b). This depends on the motor power limitation in simulation is not relevant for the success of the learning procedure.



(a) Learning in simulation (b) Learning on hardware

Fig. 4: The normalized cost evolution of the simulation-based learning is shown in (a) (averaged over 5 learning sessions). The initial policy of the learning procedure on the hardware (b) is given by the learned parameters in simulation. By leveraging the simulation-based learning, convergence to a new cost minimum on the hardware can be reached in only a few update steps (<100 rollouts).[b]

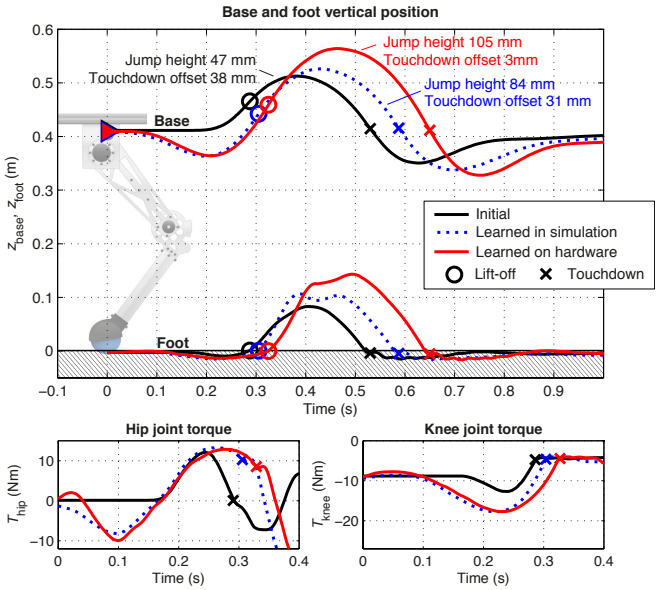


Fig. 5: The policy learning converges to a jump with characteristic countermovement behavior. The policy learned on the hardware exceeds the jump height of the policy learned in simulation.

Fig. 5 shows the initial and the learned jump with characteristic countermovement trajectory. In the learned policy, the actuators are pre-activated and the body is first lowered to temporarily store energy in the joint springs. This energy is then released during an explosive upward motion before lift-off.

The simulation-based policy (executed on the hardware) increases the initial jump height from 47 mm to 84 mm. The trajectory learned on the hardware pushes the jump height further to 105 mm by circumventing the modeling inaccuracies. While the policy learned in simulation reduces the touchdown offset on the hardware Δx from 38 mm to 31 mm, only the policy learned on the hardware is able to minimize the touchdown offset to a large extent to 3 mm.

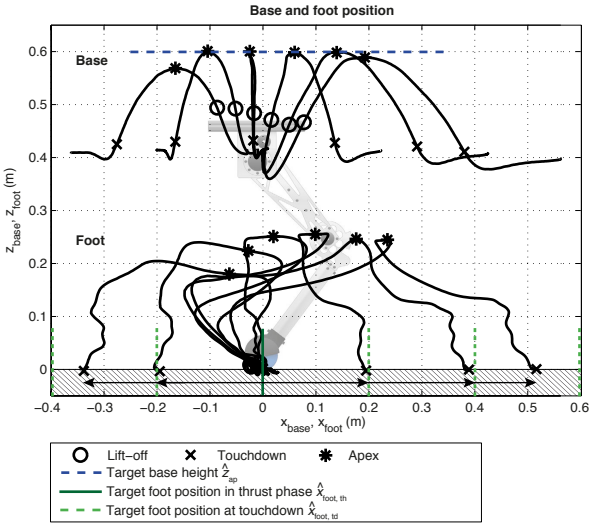


Fig. 6: The learning procedure for each long jump is initialized with the policy from the previous jump distance (starting from $\hat{x}_{\text{foot,td}} = 0\,\text{m}$) and converges within 35 rollouts. The longest target jump distances $\hat{x}_{\text{foot,td}} = 0.6\,\text{m}$ and $\hat{x}_{\text{foot,td}} = -0.4\,\text{m}$ are not fully reached because of limited foot traction.

B. Long Jump

As an extension of the vertical jump, we conduct experiments in which we let ScarlETH learn jumps with different jump distances while maintaining a defined jump height. This is incorporated as the base apex height z_{ap} (in m) and the foot touchdown position $x_{\text{foot,td}}$ (in m) in the terminal cost term

$$\phi_{t_N} = 2\,(z_{\text{ap}} - \hat{z}_{\text{ap}})^2 + (x_{\text{foot,td}} - \hat{x}_{\text{foot,td}})^2\,, \tag{9}$$

with the target base height at apex $\hat{z}_{\text{ap}}$ and the target foot touchdown position $\hat{x}_{\text{foot,td}}$. When targeting relatively large jump distances (up to 0.5 m), we have to deal with foot slippage as a result of a high horizontal thrust force. To avoid this, we penalize a horizontal displacement of the foot during the thrust phase by defining the immediate state cost as

$$q(t) = \frac{10}{N}(x_{\text{foot}}(t) - \hat{x}_{\text{foot,th}})^2\,, \tag{10}$$

with N the number of time steps of the policy execution, and $x_{\text{foot}}(t)$ (in m) and $\hat{x}_{\text{foot,th}}$ the actual and target horizontal foot position during thrust phase. We rely on hardware-based learning only as we can iteratively and efficiently reuse the policy of a previous jump with different distance (as opposed to learn each jump from scratch). The variance of the exploration noise Σ_ε is set to $1.35\,\frac{\text{rad}}{\text{s}}$. Fig. 6 shows the results for the learned policies for different target jump lengths which all converged within 35 rollouts (3 parameter updates). The longest target jump distances ($\hat{x}_{\text{foot,td}} = 0.6\,\text{m}$ and $\hat{x}_{\text{foot,td}} = -0.4\,\text{m}$) are not fully reached as a result of the trade-off between jump length and foot slippage.

We observe that ScarlETH is able to jump further in forward direction (positive x-direction) than backwards, which is due to its articulated leg design. For a forward jump, both hip and knee joints can contribute actively towards the necessary lift-off velocity, while a backward jump has to be actuated mainly by the knee joint.

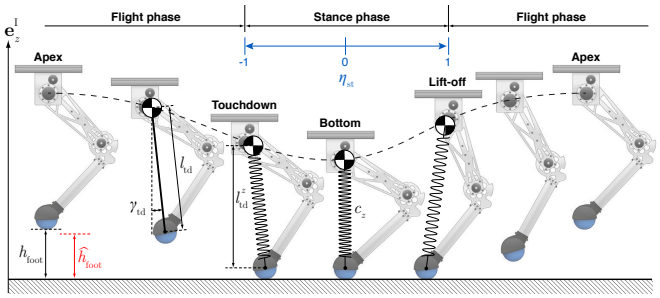


Fig. 7: High-level control parameters are optimized to maximize the energetic efficiency of periodic hopping.

V. PERIODIC HOPPING

Periodic hopping can be achieved with a control method presented in [14] that is based on virtual model control [17]. The controller enables robust hopping and a simple combination of control for the vertical and horizontal DOF. The goal is to learn a set of control parameters that minimizes the energy consumption for periodic hopping.

1) Control Structure: A hopping sequence from apex to apex is illustrated in Fig. 7. During the flight phase, the leg is positioned to a predefined configuration (joint position control) with touchdown leg angle γ_{td} and touchdown leg length l_{td}. For the stance phase, a virtual force element $\boldsymbol{F}_{\text{virtual}}$ is emulated between foot and hip joint (which roughly corresponds to the robot's center of mass). The vertical component of the virtual force is given by an emulated spring element as

$$F^z_{\text{virtual}} = c_z(\eta_{\text{st}})\left(l^z_0 - \left(z_{\text{hip}} - z_{\text{foot}}\right)\right) + g_z \text{ ,}^{\text{c}} \tag{11}$$

with the zero spring length l^z_0 and gravity compensation term g_z. The spring stiffness $c_z(\eta_{\text{st}})$ is parameterized by the time-independent locomotion variable η_{st} which is defined below. The control in horizontal direction is achieved by the x-component of the virtual force $F^x_{\text{virtual}} = k_x(\hat{x} - x)$ with the proportional gain k_x and the desired horizontal base position $\hat{x}$. The virtual force element $\boldsymbol{F}_{\text{virtual}}$ is mapped to the joint torques with Jacobi transposed mapping as laid out in [14]. The desired torques are sent to the joint controller running in the torque control mode.

2) Locomotion Time: The virtual spring stiffness $c_z(\eta_{\text{st}})$ determines the hopping frequency and thus the execution time of the policy. To comply with the time-invariant structure of the controller, we introduce a locomotion variable for the stance phase η_{st} as a time-independent, dimensionless parameter that describes the fraction of the stance time that has elapsed. It maps the progress between touchdown and lift-off to the interval $[-1, 1]$ with the bottom point at $\eta_{\text{st}} = 0$. We define the locomotion time similarly to [18] as

$$\begin{aligned}\eta^2_{\text{st}} = \frac{E_{\text{kin}}}{E_{\text{tot}}} &= \frac{\frac{1}{2}m\dot{z}^2}{\frac{1}{2}m\dot{z}^2 + mg(z - z_{\text{td}}) + \int_{t_{\text{td}}}^{t} F^z_{\text{virtual}}\,\mathrm{d}z} \text{ ,} \\ \eta_{\text{st}} &= \frac{\dot{z}}{\sqrt{\dot{z}^2 + 2g(z - z_{\text{td}}) + \frac{2}{m}\int_{t_{\text{td}}}^{t} F^z_{\text{virtual}}\,\mathrm{d}z}} \text{ ,}\end{aligned} \tag{12}$$

[c]For our choice of the coordinate system (see Fig. 2(a)), $z_{\text{foot}} \approx 0\,\text{m}$ holds during the stance phase.

based on the kinetic energy E_{kin} and the total energy of the system E_{tot} (as the sum of kinetic energy, potential energy in the gravitational field, and potential energy of the virtual spring). This approximation captures the main characteristics sufficiently as it covers big parts of the interval $[-1, 1]$ and is, at least experimentally, monotonic strictly increasing.

3) Policy: The learning algorithm is applied to a subset of the control parameters (in total 7 open parameters), namely the leg length at touchdown l_{td}, the virtual spring stiffness $c_z(\eta_{\text{st}})$, and the virtual zero spring length l^z_0 (Fig. 7). The touchdown leg length l_{td} determines the extension angle of the knee at touchdown. A higher leg length (and thus higher knee extension angle) leads to higher impact energy losses [19] and thus influences the energy efficiency. The parameterized virtual spring stiffness $c_z(\eta_{\text{st}})$ gives the learning algorithm the opportunity to optimize for energetic efficiency by influencing the instantaneous motor power. In accordance to the policies of the previous tasks (Section IV), $M = 5$ Gaussian functions are chosen as basis functions. Finally, the zero spring length l^z_0 determines how much energy is virtually introduced to the system at every time step to compensate for the energy losses. It is important to adapt the parameter accordingly to maintain the minimal foot clearance $\hat{h}_{\text{foot}}$.

The minimal foot clearance is given with $\hat{h}_{\text{foot}} = 16\,\text{cm}$ and the variance of the exploration noise for the virtual spring stiffness is set to $\Sigma_\varepsilon = 300\,\text{N/m}$. Again, the number of rollouts per update is $K = 10$ with an additional 5 rollouts reused from the last update.

4) Cost Function: The energy consumption is captured in the immediate state cost with the positive motor work (motor power defined as $P_{\text{mot}} = \dot{\varphi}_{\text{mot}} T$ in W) for the stance phase as

$$q_t = \max(P_{\text{hip,mot}}, 0) + \max(P_{\text{knee,mot}}, 0)\,. \tag{13}$$

Reaching the minimum foot clearance $\hat{h}_{\text{foot}}$ (in m) for each step allows us to compare the energy expenditure. We incorporate the minimal foot clearance in the terminal cost term as

$$\phi_{t_N} = 12 \cdot 10^3 \left(\min\left(h_{\text{foot}} - \hat{h}_{\text{foot}}, 0\right)\right)^2 \text{ ,} \tag{14}$$

where the weighting factor is tuned manually. One rollout is executed with 10 steps/jumps, of which the last 6 steps are evaluated with the cost function. This makes sure that the transient towards the limit cycle is omitted. The cost for each jump of the rollout are averaged and mapped on the locomotion variable η_{st} before performing the PI^2 policy update step.

5) Results: The cost evolution for multiple learning sessions in simulation is shown in Fig. 8. The learning algorithm has consistently converged from the leg length at touchdown $l_{\text{td}} = 35.0\,\text{cm}$ to a learned leg length with $l_{\text{td}} = 29.3\,\text{cm}$. The initial (linear) and the learned (nonlinear) virtual spring characteristic $c_z(\eta_{\text{st}})$ is plotted in Fig. 9. The learned control parameters reduce the positive motor work for the stance phase from 7.0 J to 6.1 J per step.

We have conducted a learning session on the real robot and initialized the policy with the learned parameters from

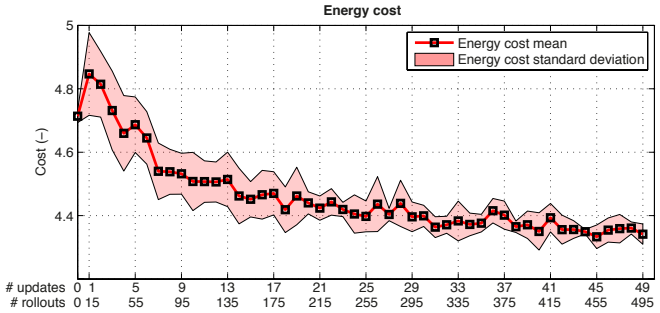


Fig. 8: Evolution of the energy cost for periodic hopping (learned in simulation, averaged over 5 learning sessions)

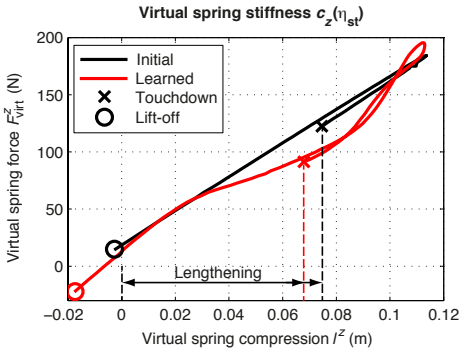


Fig. 9: Initial and learned virtual spring force F^z_{virtual} in dependency of the virtual spring compression $l^z = z_{\text{hip}} - z_{\text{foot}}$

simulation. A significant convergence to a new cost minimum could not be established. By closing the control loop with the introduced controller, big parts of the modeling errors are cancelled and an additional optimization on the hardware is unnecessary. However, we can ensure that parameters learned from simulation are at least locally optimal on the real robot as well.

VI. CONCLUSION

We have shown that reinforcement learning based on simulations and hardware experiments is highly effective to improve the locomotion performance and energetic efficiency. For jumps from rest, the motor velocity trajectories converged to a countermovement jump in which the elastic elements in the joints are exploited to improve the jump height. This is very similar to what can be observed in nature and has been examined i.a. by [20], [21]. By continuing the learning on the hardware, modeling errors are compensated for and the jumping performance is improved significantly within a few trials. Learning different jump lengths, we generate a policy library with which ScarlETH can accurately perform accurate jumps. For energy efficient hopping, we introduce a new method for the optimization of time-independent periodic control policies. The algorithm is able to minimize the energy consumption without sacrificing the robustness of the controller.

For the vertical and long jumps, we ascribe the learning success to the choice of the control policy. The policy directly acts on the motor velocity as input to the SEA without additional joint tracking controller and allows to efficiently exploit temporal energy storage and power amplification of the compliant actuators. This is in contrast to the closed-loop hopping controller where we restrict the optimization to the high-level control parameters in order to maintain active control of the horizontal DOF. This choice naturally restricts the degree of optimization of the energy efficiency that can be achieved. While the PI^2 algorithm has worked to our satisfaction in these experiments, we do not exclude that other reinforcement learning methods could have achieved similar results. As a next step, we are building on our findings to learn jumping, hopping, and running maneuvers on our quadruped robot *StarlETH* [16].

REFERENCES

[1] N. Kohl and P. Stone, "Policy Gradient Reinforcement Learning for Fast Quadrupedal Locomotion," in *International Conference on Robotics and Automation (ICRA)*, 2004.

[2] W. Miller, "Real-time neural network control of a biped walking robot," *IEEE Control Systems*, vol. 14, no. 1, pp. 41–48, Feb. 1994.

[3] H. Benbrahim, "Biped dynamic walking using reinforcement learning," *Robotics and Autonomous Systems*, vol. 22, 1997.

[4] R. Tedrake, T. W. Zhang, and H. S. Seung, "Learning to Walk in 20 Minutes," in *Yale Workshop on Adaptive and Learning Systems*, New Haven, CT, 2005.

[5] R. Niiyama, K. Kakitani, and Y. Kuniyoshi, "Learning to jump with a musculoskeletal robot using a sparse coding of activation," *International Conference on Robotics and Automation (ICRA)*, 2009.

[6] P. Doerschuk, W. Simon, V. Nguyen, and A. Li, "A modular approach to intelligent control of a simulated jointed leg," *IEEE Robotics and Automation Magazine*, vol. 5, no. 2, pp. 12–21, June 1998.

[7] R. Tedrake and H. S. Seung, "Improved Dynamic Stability Using Reinforcement Learning," in *International Conference on Climbing and Walking Robots (CLAWAR)*, 2002, pp. 341–348.

[8] S. Curran and D. E. Orin, "Evolution of a jump in an articulated leg with series-elastic actuation," *International Conference on Robotics and Automation (ICRA)*, pp. 352–358, May 2008.

[9] S. Schaal and C. G. Atkeson, "Learning control in robotics," *IEEE Robotics and Automation Magazine*, no. June, 2010.

[10] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement Learning in Robotics: A Survey," *International Journal of Robotics Research*, 2013.

[11] E. A. Theodorou, J. Buchli, and S. Schaal, "A generalized path integral control approach to reinforcement learning," *The Journal of Machine Learning Research*, vol. 11, pp. 3137–3181, 2010.

[12] J. Buchli, F. Stulp, E. A. Theodorou, and S. Schaal, "Learning variable impedance control," *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 820–833, Apr. 2011.

[13] P. Pastor, M. Kalakrishnan, S. Chitta, E. Theodorou, and S. Schaal, "Skill learning and task outcome prediction for manipulation," in *International Conference on Robotics and Automation (ICRA)*, 2011.

[14] M. Hutter, C. D. Remy, M. A. Hoepflinger, and R. Siegwart, "ScarlETH: Design and control of a planar running robot," in *International Conference on Intelligent Robots and Systems (IROS)*, Sept. 2011, pp. 562–567.

[15] G. A. Pratt and M. M. Williamson, "Series elastic actuators," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 1995, pp. 3137–3181.

[16] M. Hutter, C. Gehring, M. Bloesch, M. A. Hoepflinger, C. D. Remy, and R. Siegwart, "StarlETH: A compliant quadrupedal robot for fast, efficient, and versatile locomotion," in *International Conference on Climbing and Walking Robots (CLAWAR)*, 2012.

[17] J. Pratt, "Virtual Model Control: An Intuitive Approach for Bipedal Locomotion," *The International Journal of Robotics Research*, vol. 20, no. 2, pp. 129–143, Feb. 2001.

[18] J. Helferty and M. Kam, "Adaptive control of a legged robot using an artificial neural network," in *International Conference on Systems Engineering*. IEEE, 1989, pp. 165–168.

[19] M. Hutter, C. D. Remy, M. A. Hoepflinger, and R. Siegwart, "SLIP Running with an Articulated Robotic Leg," in *International Conference on Intelligent Robots and Systems (IROS)*, 2010.

[20] M. F. Bobbert, K. G. M. Gerritsen, M. C. A. Litjens, and A. J. Van Soest, "Why is countermovement jump height greater than squad jump height?" *Medicine and Science in Sports and Exercise*, 1996.

[21] D. A. Chu, *Jumping Into Plyometrics*. Human Kinetics, 1998.