A Hierarchical Scheme for Adapting Learned Quadruped Locomotion

Michel Aractingi, Pierre-Alexandre Léziart, Thomas Flayols, Julien Perez, Tomi Silander, Philippe Souères

A Hierarchical Scheme for Adapting Learned Quadruped Locomotion

Michel Aractingi[1,2], Pierre-Alexandre Léziart[1], Thomas Flayols[1],
Julien Perez[2], Tomi Silander[2] and Philippe Souères[1]

***Abstract*— The quadruped locomotion task is generally conditioned on a velocity command that is user-defined. However, certain features of the locomotion are not represented in this high-level definition of the task, e.g., swing feet height, step length and expended energy. Using reinforcement learning, many of these features can be determined by the reward function terms and scales that are often fixed. In this work, we propose a deep reinforcement learning (DRL) approach to learn control policies augmented with parameters that modify different aspects of the reward function and control setup which, in turn, result in variations of the locomotion. We can then define a hierarchical architecture where a high level policy infers the suitable parameters to complete a given task. We show that this setup makes it possible to learn more complex behaviours that can be adapted for different terrains and environments to ensure successful and efficient locomotion. We display our results by deploying the low-level parameterized policy on the MIT Mini-Cheetah quadruped.**

I. INTRODUCTION

Quadruped locomotion has emerged as an interesting field of research in both model-based control [1], [2], [3] and model-free learning [4], [5], [6]. In both approaches, the objective is similar, i.e., produce robust locomotion when following a desired velocity command while minimizing the used energy. However, this is a very abstract definition of the locomotion task as, in reality, many low-level features of the motion are required such as how much to lift the feet, the length and frequency of footsteps, or the best nominal joint pose to center the movement around.

These different decisions intrinsically make the locomotion problem a highly multi-task one at different levels regarding the motion of the base and the joints [2]. At the high-level, the velocity command determines the trajectory of the center of mass (COM) of the robot, but in more complex environments that require precise intended movements, e.g., going up the stairs or reaching a given location, the velocity on its own is not enough. At the same time, the low-level joint control pattern might also need to be modified for such specific tasks. For example, on rough terrains the robot might need to lift its feet higher, and in steep slopes the robot would need to take smaller steps or use more torque [7].

Therefore, there is a clear need to augment the task description and have the ability to convey the desired behavior to dictate the low-level motion. In this paper, we tackle the multi-task aspect of locomotion using hierarchical reinforcement learning (HRL). With HRL we can decompose a reinforcement learning (RL) problem into simpler sub-tasks that are easier to design and learn [8].

Fig. 1: Snapshots of the proposed low-level policy controlling the Mini-Cheetah through different challenging terrains. Full video can be found at: `https://youtu.be/B92HB964xq8`.

We propose learning a low-level policy with an elaborate reward function that governs many cost terms related to the desired joint-level control. Along with the state, this low-level policy is augmented with *command parameters* which are variables and weights that define the terms of the reward function and details of the state dynamics. We use the term *command* here because we view these parameters as an extension of the velocity commands that generally dictate the locomotion task.

The main contribution of this work is a hierarchical deep RL system that augments the baseline locomotion policy with a command parameters vector ω that manipulates different aspects of the locomotion (swing feet height, step length, initial pose and PD gains). This baseline policy can then be used in a hierarchical setup where a high-level policy determines suitable command parameters to adapt the locomotion to different situations. We show that this two-level design improves the performance of velocity tracking policies by reducing energy consumption and velocity tracking error in various situations while having better sample efficiency. We also show how these policies enable the real robot to cross different structured terrains that the baseline policy would regularly fail to be robust against in Figure 1. We illustrate our results on the MIT Mini-Cheetah quadruped [9].

Accepted at IEEE Humanoids 2023

[1]Authors at LAAS-CNRS, Université de Toulouse, Toulouse, 31400, France `first.last@laas.fr`

[2]Authors at NAVER LABS Europe, Meylan, 38240, France `first.last@naverlabs.com`

Fig. 2: Policy and control architecture of the low-level policy (right) and the high-level policy (left). The low-level policy's state and commands are fed to the policy network that outputs joint displacement offsets. A PD controller turns the network actions into torques. The high-level policy get task related information in order to actively control the low-level behaviour by actively adjusting the command parameters.

The paper is organized as follows. Section II discusses the related literature. Section III introduces some notions and definitions. Section IV presents in detail the approach and how the RL policies are designed and trained. Finally, the experimental results and conclusions are outlined in sections V and VII respectively.

II. Related Work

Recent development of model-based controllers for quadruped robots have contributed to the growth of locomotion research [1], [2], [3]. An efficient modular architecture was proposed in [2]. It includes a contact planner, a model predictive controller (MPC) that plans the base motion and the feet contact reaction forces over a receding horizon, and a whole body controller that executes the plan by controlling the joints at a higher frequency. A similar scheme with a simplified inverse kinematics approach and state estimation was later developed in [3]. While there is a hierarchy of optimization tasks in these model-based architectures, they are mainly built around base velocity tracking task and fixed values related to the gait pattern and feet trajectories. While they produce agile and efficient locomotion in nominal conditions, they do not allow for easy adaptation of that locomotion to new situations.

Alternative deep RL control paradigms were proposed to learn locomotion in an end-to-end manner. In [4], [5] authors detail different elements required to learn robust controllers that works on real robots, e.g., choice of action space, reward design and actuation modeling. In [10] state estimation is learned along with the policy which resulted in a rapid locomotion that broke the speed record of the Mini-Cheetah [9]. A robust walking gait was learned by biasing the action space with central pattern generators (CPGs) in [11]. Other authors suggested to learn an adaptation model along with the main policy to adapt its control to various environmental changes [6].

While these deep RL techniques learn robust locomotion to track a velocity command, they produce monotonic motion that can be hardly generalized to new situations. In [12] parameterized policies were learned for wheeled-robot navigation tasks. In order to adapt the behavior, as the dynamics of the underlying system changes, reward weightings were incorporated in the state. Recently, parameterized policies have also been introduced for versatile quadruped locomotion but without showing benefits of hierarchical RL [13].

Hierarchical reinforcement learning (HRL) methods have been proposed for learning locomotion strategies, for example by learning separate policies for recovery, standing and locomotion and then a high-level policy that learns to switch between them [14]. Hierarchical methods, in similar spirit, have also been developed in planning-based algorithms. The work in [7] proposes a contact planner for locomotion in tasks where there are multiple stages of contact and decomposing the planning into sub-problems is necessary. A two-level approach was proposed in [15] to control a biped in simulation, where a low-level policy learns to follow desired footstep placements and a high-level policy learns to place the footsteps based on the terrain information. In our work, we propose embedding a policy with several aspects related to the control and reward function such that the behaviour of the low-level policy is very flexible and can be adapted in different dimensions via a high-level policy. We also show the effectiveness of the low-level policy on the real system in challenging terrains and not only in simulation.

The proposed approach to learn parameterized policies for quadruped locomotion follows an idea similar to the one developed in [12]. However, we also use parameters like the gains of the low-level impedance controller that are not part of the reward, but can have a desired effect on the motion. In the literature of skill learning, some works propose to learn separate skills for different purposes [14], [16], [17], [18] often via imitation learning of trajectories generated by an optimal controller. Here, we are more interested in learning a general locomotion policy that we can control to fulfill specific task. Though it is also possible to follow a hierarchical approach to learn locomotion policies from visual input [19], the problem we consider in this work is to learn the best blind low-level locomotion policy so that we can control several aspects of it (foot height, stride length). Since our proposed policies are reactive, there is no planning involved.

III. Preliminaries

A. Low-level parametric policies

We model the low level learning environment as a set of Markov decision processes (MDPs), indexed by parameters ω, with common, continuous state and action spaces [20]: $\mathcal{M}_\Omega = \{(\mathcal{S}, \mathcal{A}, \mathcal{R}_\omega, \mathcal{P}, \rho_0)|\omega \in \Omega\}$, where $\mathcal{S}$ is an infinite set of states, and $\mathcal{A}$ is an infinite set of actions. In each $M_\omega \in \mathcal{M}_\Omega$, taking an action a in a state s yields a reward, defined by a function $\mathcal{R}_\omega : \mathcal{S}\times\mathcal{A}\times\mathcal{S} \to \mathbb{R}$. The environment dynamics is described by a conditional transition probability distribution $\mathcal{P} : \mathcal{S}\times\mathcal{A}\times\mathcal{S} \to \mathbb{R}_+$, with the interpretation that $\mathcal{P}(s, a, s') = p(s_{t+1} = s'|s_t = s, a_t = a)$ is a probability (density) that the next state is s' given that the current state is s, and the action taken is a. ρ_0 is the initial state probability distribution. We assume parameters $\omega \in \Omega \subset \mathbb{R}^d$ to be sampled from a static distribution ρ_Ω, and define the learning task to be finding the common parameters θ for all stochastic policies $\pi^\omega_\theta : \mathcal{S} \to Pr(\mathcal{A}|\mathcal{S})$ in order to attain a maximum expected discounted sum of rewards

$$J(\theta) := \mathbb{E}_{\rho_\Omega}\mathbb{E}_{\pi^\omega_\theta,\mathcal{P}_\omega,\rho_0}[\sum_{t=0}^{\infty}\gamma^t\mathcal{R}_\omega(s_t, \pi^\omega_\theta(s_t), s_{t+1})], \quad (1)$$

where $\gamma \in [0, 1]$ is a discount factor. In practice, π^ω_θ is implemented as a neural network thus θ corresponds to its weights.

B. High-level parameter controlling policy

Given a parameterized set of MDPs (as described above) and a parameterized set of low-level policies π^Ω_θ, we define the high-level control setting as an MDP $\mathcal{M}^H = (\mathcal{S}^H, \Omega, \mathcal{R}^H, \mathcal{P}^H, \rho^H_0)$, where the superscript H stands for "high-level". The state space $\mathcal{S}^H = \mathcal{S} \times \mathcal{S}^h$ may also contain additional information $\mathcal{S}^h$ (e.g., vision data) and the action space consists of the set of low-level parameters Ω. The transition dynamics is supposed to respect the low-level dynamics so that it obeys

$$\int_{s'^h} p^H((s', s'^h)|(s, s^h), \omega)ds'^h = \mathcal{P}(s, \pi^\omega_\theta(s), s') \quad (2)$$

for all $s, s' \in \mathcal{S}, s^h \in \mathcal{S}^h$, and $\omega \in \Omega$.

The high-level learning objective is to find the parameters ϕ for a high-level stochastic policy $\pi^h_\phi : \mathcal{S}^H \to Pr(\Omega|\mathcal{S}^H) : \pi^h_\phi(\omega|s) = p_\phi(\omega_t = \omega \mid s_t = s)$ that optimizes the expected discounted cumulative reward

$$J^H(\phi) := \mathbb{E}_{\pi^\Omega_\theta,\pi^h_\phi,\mathcal{P}^H,\rho^H_0}[\sum_{t=0}^{\infty}\gamma^t_H\mathcal{R}^H(s^H_t, \omega_t, s^H_{t+1})], \quad (3)$$

where $\gamma_H \in [0, 1]$ is a discount factor, $s^H_t = (s_t, s^h_t)$, and ω_t is computed by the high level policy $\omega_t = \pi^h_\phi(s^H_t)$. Notice also that $s^H_{t+1} = (\pi^{\omega_t}_\theta(s_t), s^h_{t+1})$.

IV. Method

In this section, we will first outline the main procedure for learning general end-to-end locomotion policies on the Mini-Cheetah quadruped. The low-level policy has to be robust while following a set of references determined by the command parameters. The learned policy also has to transfer to the real system. We will then describe the parameters that the low-level locomotion task is conditioned on. Finally, we introduce the high-level policy that adapts the parameterized locomotion.

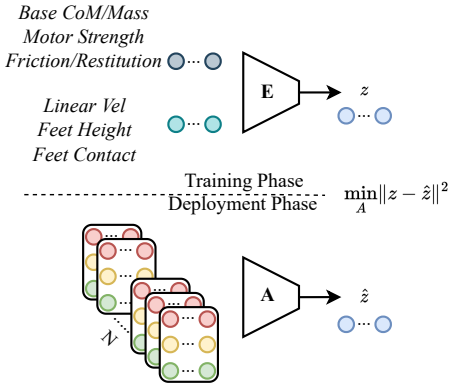

Fig. 3: The environment encoder E encodes a representation of the privileged information into z. The adaptation module A learns to regress the output of the encoder from the history of the observations and actions to predict $\hat{z}$.

A. Low-level policy

The goal of the low level policy is to produce joint angle control based on the state of the robot and the chosen command parameters. A general scheme of the low-level control is depicted in Figure 2.

Observation space. The observation of the robot mainly depends on the IMU readings and the proprioception of the joints. The observation at time t is:

$$o_t = (\theta_{\text{body}}, \omega_{\text{body}}, q_t, \dot{q}_t, a_{t-1}), \quad (4)$$

where θ_{body} is the orientation of the base, ω_{body} is the angular velocity of the base, q_t and $\dot{q}_t$ are the joint angles and velocities respectively. We also include the previous actions a_{t-1} to the observation.

State space. Observations in the real system are noisy and the dynamics of the robot and terrain could vary depending on the situation. In order to learn policies that are robust and transferable, one needs to inject noise and randomization in the observation space and environment's dynamics [4], [11]. Therefore, relying on a single timestep observation as input to the control policies would often hinder both learning and transfer.

To further robustify the robot behaviour, we use the rapid motor adaptation (RMA) strategy [6]. RMA is a distillation-based technique where an encoder network E is trained to encode useful privileged information (i.e., information only available in a simulator) to a latent vector z_t that represents environment's dynamics that are randomized to mimic the model mismatches and noisiness that are likely to appear in the real world. At a second stage, an adaptation model A is trained to learn an estimate $\hat{z}_t$ of z_t from the history of observations and actions with a supervised learning loss. We then use A when deploying the policy on the

real system. In the simulator, the state is $s_t = (o_t, z_t, \omega_t)$, and in the real robot $s_t = (o_t, \hat{z}_t, \omega_t)$. In this work, we propose to augment the encoder input with privileged state information related to the robot's feet positions and contacts and the base linear velocity. Ablation studies (see Section V-B.2) indicate that this helps learning low-level policies that reach the commanded references better and with less training iterations. A full scheme of the RMA approach is depicted in Figure 3.

Reward function The objective of the reward function is to achieve a specified base velocity while minimizing costs associated with acquiring resilient and stable locomotion skills. A detailed breakdown of the reward function components and weightings is presented in Table I. $V^{base}_{x,y,yaw}$ refers to the 3D vector consisting of the measured base linear velocity along x and y and angular yaw velocity. $p_{f,z,i}$ is the i^{th} foot height in world coordinates while $p^{max}_{f,z}$ refers to the desired foot height in world coordinates and $V_{f,xy,i}$ is the i^{th} linear foot velocity along x and y. The foot clearance reward depends on how close the current foot height is to the desired one, scaled by the foot velocity, e.g., the faster the feet are moving the higher the robot should lift its feet. The total reward function is the weighted sum of these terms. The main positive term in the reward function is related to tracking the desired velocity, while negative costs are used for refining the movement in terms of action magnitude and smoothness, for raising the feet, and maintaining an appropriate posture and base stability.

Reward Function	Formula
Velocity tracking	$c_{vel} exp(-\|\|V^{cmd} - V^{base}_{x,y,yaw}\|\|^2/\sigma_{vel})$
Base z velocity	$c_{vz}\|V_z\|^2$
Joint angle deviation	$c_{jpos}\|\|q_t - q_{nominal}\|\|^2$
Joint velocities	$c_{\dot{q}}\|\|\dot{q}_t\|\|^2$
Joint accelerations	$c_{\ddot{q}}\|\|\dot{q}_t - \dot{q}_{t-1}\|\|^2$
Joint torques	$c_\tau\|\|\tau\|\|^2$
Base orientation	$c_\theta\|\theta_{roll,pitch}\|^2$
foot clearance	$c_{fcl}\sum_i^4 (p_{f,z,i} - p^{max}_{f,z})^2 \|\|V_{f,xy,i}\|\|^{0.5}$
action smoothness	$c_{smooth}\|a_t - a_{t-1}\|^2$

TABLE I: Reward function terms and coefficients. The total reward is a weighted sum of all the terms above.

Locomotion command parameters. The reward design is similar to previous studies on end-to-end RL for locomotion [10], [5], [4], [11], [21]. However, in these previous works, the velocity command was usually considered the main command parameter in the reward function. While training in simulation it is randomly sampled, and on the real robot, it is controlled by the user with a gamepad device. In this work, we consider more elaborate command parameters that are hard to be prescribed by the user. The main parameter vector ω given to the policy is:

$$\omega = (V_{cmd}, p^{max}_{f,z}, c_{jpos}, Kp, Kd, q_{nominal}),$$

where V_{cmd} is 3D the velocity command, linear along x, y and angular about z. $p^{max}_{f,z}$ is the maximum foot height to reach in the swing phase, which is part of the foot clearance reward. c_{jpos} is the coefficient that weights the joint angle deviation penalty, and Kp and Kd are the proportional and derivative gains for the PD controller, and $q_{nominal}$ is the initial pose around which the action space is centered.

Action space. The network π_θ outputs the joint angle targets that are fed to a PD controller with zero joint velocity targets. We center the action space around the nominal joint angles at the initial position of the robot, so that the PD target at time t is $q^{target}_t = q_{nominal} + \lambda\pi_\theta(s_t, \omega_t)$. This centering is essential for the policy to learn and allows us to control the limits of the joint angles from the initial position. In our experiments we found $\lambda = 0.3$ to be the best choice. This action space was used in previous literature [4], [5].

B. High-level policy

The actions of the high-level policy are locomotion parameters that modulate the low-level policy. The main intuition is that the elements of ω can be varied to change the behaviour of the robot while still producing stable locomotion that is learned by the low-level policy.

The high-level policy π^h_ϕ can control V_{cmd} when it is not tuned by a user, like for example, in an autonomous point goal navigation task. The maximum desired foot height, $p^{max}_{f,z}$, is an important parameter that has consequences on transfer [10], [21]. On complex terrain the robot might need to lift its feet higher than on flat terrain where the robot can safely save some energy by not lifting its feet much. c_{jpos} determines the joint angle deviation penalty weight. In our experiments in Section V-B.1, we found that different values of this coefficient result in different stride length. Therefore, we use it as an indirect way of controlling the stride length. Controlling the PD gains adds variable impedance control aspects to learned policies [22]. It also adds robustness to variations in the environment [23] and makes it possible to control the expended torques to perform tasks that require more power.

The choice of parameters the high-level policy controls depends on the task. In this work, we experiment with a high-level task of traversing a variety of complex terrains while the high-level policy only optimizing for velocity tracking and energy consumption. The high-level policy takes a scan of the height measurements surrounding the robot as input Figure 5. The actions of this policy are deltas to the default command parameters vector ω excluding the velocity commands. We chose this task to show how we can easily learn to adapt the low-level policy behaviour that is trained on a flat terrain in order to be useful on more complex terrains. The high-level reward function r_h is based on the velocity tracking term (in Table I) and the energy consumption terms:

$$r^h_t = r^{vel}_t - 0.002|\tau^T_t \dot{q}_t|. \tag{5}$$

In the next section, an analysis of the resulting low-level policy with the elaborate locomotion parameters input is displayed. We then present some high-level tasks to showcase the benefits of the proposed hierarchical scheme. For each task, the high-level state, action and reward will be explained in detail along with the results.

V. Experimental Results

This section includes full implementation details about the entire approach. We conduct our experiments to answer the following questions: (1) Which aspects of the locomotion vary when modifying the command parameters of the low-level policy? (2) Does our additions to the encoder network improve the learned performance? (3) How would a hierarchical policy improve on the baseline end-to-end approaches on flat terrains and complex structured terrains in the presence of exteroception?

A. Implementation details

As mentioned before, we conduct our experiments with the Mini-Cheetah quadruped, which has 12 degrees-of-freedom with 3 actuators per leg [9]. The two levels are trained one after the other. First, the low-level policy is trained to follow different random values of the command parameters. After that, depending on the high-level task, one trains a high-level policy that controls the parameters of the already learned parametric low-level policy.

Both policies have the same architecture which is a multi-layer perceptron (MLP) with three hidden layers of sizes 512, 256, 128. The RL algorithm used to train both levels is PPO [24] with generalized advantage estimation [25]. We train our policy networks with actor-critic approach with critic having the same architecture as the actor but with a scalar value for the value estimation [26]. The policies run at a frequency of 50Hz while the simulation frequency runs at 200 Hz. On the robot the same control frequency is maintained while the low-level PD control runs in a high frequency feedback loop of 40 KHz.

Low-level policy training. The observation space is 45-dimensional while the actions are 12-dimensional, equal to the number of actuated joints. We use the exponential linear unit (ELU) activation function in the neural networks.The critic also receives directly the privileged state of the robot.

We use the IsaacGym simulator from Nvidia [27]. The environment code is based on the legged_gym repository [28]. In the current experiments, 4096 agents are run in parallel in an infinite horizon objective where the environment does not reset with each new training episode, but continues with the latest reached state. However, we found that introducing random resets also helps. We were able to learn the low-level policy with command parameters in 5000 iterations.

A linear curriculum was implemented on the penalty terms of the reward. A curriculum factor $k_c \in [0, 1]$ was introduced to scale the reward. This factor is increased as training progresses to give more weight to the penalties. At the start of training the agent is mostly concerned with learning a movement that follows the reference velocity in any possible way. As training progresses the movement is refined to optimize the penalty terms [4], [21]. The reward function term weights in Table I are chosen to be: $c_{vel} = 1.0$, $c_{v_z} = -1.2$, $c_{\dot{q}} = c_{\tau} = -0.0003$, $c_{\ddot{q}} = -0.00001$, $c_{\theta} = -3.0$, $c_{fcl} = -5$. and $c_{smooth} = -0.01$.

Command parameters ω are sampled at the start of a new training episode and given as input to the low-level policy. The parameters are resampled when the policy fails (e.g., robot falls) or at random times if the policy does not fail. The sampling range for each element in ω can be found in Table III[1]. Each parameter is sampled from a uniform distribution that samples deviations around the parameter's default value within the specified range. The sampling is also scaled by the curriculum factor k_c, i.e., we use the curriculum on the command parameters as well as on the reward terms.

Random Observations:		Random Dynamics:	
θ_{body}	$U^3(-0.05, 0.05)$	Motor Strength	$U^{12}(-0.9, 1.1)$
ω_{body}	$U^3(-0.20, 0.20)$	Ground Friction	$U(0.5, 1.0)$
q	$U^{12}(-0.05, 0.05)$	Ground restitution	$U(0.0, 1.0)$
$\dot{q}$	$U^{12}(-1.00, 1.00)$	Body Mass	$U(-1.0, 1.0)$
		COM displacement	$U^3(-0.2, 0.2)$

TABLE II: Ranges and dimensions of uniform noise for randomizing the dynamics and observations.

Element	Range	Default
Foot height target [cm]	[3.0 , 15.0]	7.0
Joint angle deviation	[-1.0, -0.2]	-0.5
Position gain	[17.0, 30.0]	20.0
Velocity reference [m/s]	X: [-2.0, 2.0] Y : [-1.0, 1.0] z_yaw: [-1.0, 1.0]	0.0 0.0 0.0
$q_{nominal}$ [rad]	Shoulder: [0.0, 0.1] Hip : [-2.0, -0.1] Knee: [1.3, 1.9]	0.05 -0.8 1.6

TABLE III: Default values of the command parameters and ranges in which the commands are sampled.

Domain randomization and RMA. For transferring the learned policy to the robot, it is essential to randomize various aspects of the observation and dynamics in order to mimic the perturbations and imperfections of the real system. The randomized values in the observations, robot dynamics, and terrain dynamics are shown in Table II. However, we found that, for the Mini-Cheetah, relying solely on domain randomization does not yield good transfer. To enhance transfer, we used rapid motor adaptation (RMA) [6], [29], in which one first learns a latent embedding z_t that encodes privileged information around the randomized dynamics, and then uses supervised learning to build an adaptation model that estimates this z_t based on the sequence of state variables that are also available in the real robot.

In our setup, the embedding size for z_t is 18 while the input size of the privileged information is 29. The privileged input is constructed from the vector of random dynamics values (ranges presented in Table II) and the privileged data regarding the feet height in world frame, terrain contact indicators and base linear velocities. The encoder E and adaptation model A are both MLPs with two hidden layers of sizes 256 and 128. The adaptation model takes as input the last $N = 15$ observations and outputs an estimate $\hat{z}$ of the latent vector z. Further information on this method can be found in [6], [29].

[1]Note the value of the derivative gain is coupled with the position gain, i.e., K_d is chosen to be a scaled value of $\sqrt{K_p}$ as in [22].

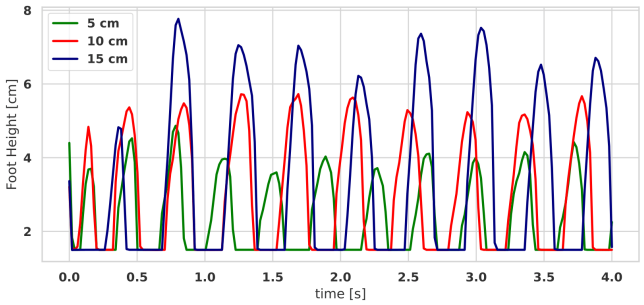

Fig. 4: Front left foot height for different $p_{f,z}^{max}$ targets (5, 10 and 15cm), when running the robot controlled by the low-level policy at a 1.0m/s forward velocity command.

B. Low-level policy study

1) Understanding the parameterized low-level control: It is important to first quantitatively and qualitatively observe the effect that the parameters ω have on the final locomotion in order to be able to define useful tasks that could benefit from a policy that controls ω. In this section we outline the difference in aspects of the resulting locomotion when changing some command parameters individually.

Joint angle penalty weight $= -0.1$			
Vx [m/s]	0.5	1.0	1.5
footstep [m]	0.27 $\pm$ 0.03	0.33 $\pm$ 0.03	0.42 $\pm$ 0.04
Joint angle penalty weight $= -0.5$			
Vx [m/s]	0.5	1.0	1.5
footstep [m]	0.22 $\pm$ 0.001	0.32 $\pm$ 0.005	0.39 $\pm$ 0.02
Joint angle penalty weight $= -1.0$			
Vx [m/s]	0.5	1.0	1.5
footstep [m]	0.17 $\pm$ 0.001	0.25 $\pm$ 0.001	0.33 $\pm$ 0.003

TABLE IV: Average foot step length as a function of the forward velocity command (angular velocity is set to zero) and joint angle deviation penalty weight.

The joint angle deviation term in the reward penalizes the policy based on how far the joint positions are from the nominal pose. Intuitively, modifying this penalty coefficient in the reward should allow us to control the range of joint movement which affects the step length. We verify that the step length in the parameterized low-level policy is a function of the coefficient c_{jpos} by examining the average foot step length of the forward left foot as a function of the forward velocity in Table IV. This table shows a clear relationship between the foot step length and coefficient value. For all values of c_{jpos} the average step length increases with the increase in the commanded V_x. However, the average step length for the same velocity command decreases when the absolute magnitude of c_{jpos} is increased. In other words, the more weight given to the joint angle deviation term the smaller the average foot steps are and vice versa. We also notice the high variance of the average step length when $c_{jpos} = -0.1$ which suggests the limit to which c_{jpos} could be varied while the penalty still retains an effect on the overall behaviour.

The $p_{f,z}^{max}$ parameter defines the target foot height when the foot is in swing mode. Figure 4 shows a plot of the achieved height of the front left foot as a function of time when running the low-level policy with a forward velocity command of 1.0m/s for four seconds. The figure shows patterns when the foot is swinging and when it is resting on the ground. We can clearly see the increase in foot height for higher values of $p_{f,z}^{max}$. Another interesting behaviour that emerges from changing the foot height target is a change in the stepping period. The 5cm height target results in shorter stepping period than the 10cm target, which in turn results in a shorter period than the 15cm target. Note that the achieved foot height is lower than the target because the foot clearance reward does not only depend on $p_{f,z}^{max}$ but is also scaled by the velocity of the feet, see Table I.

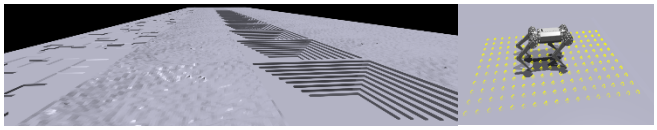

Fig. 5: Left: Terrain types in the HRL experiments. Right: Height-map measurements around the robot.

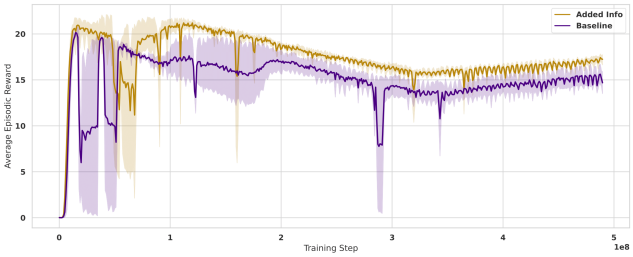

Fig. 6: The average episodic reward per training iteration. The curves show two experimental setups: when additional information is added to the encoder network (yellow), and when it is not (purple).

2) Ablation studies: In Section IV-A, we introduced additional information regarding the feet height, feet contacts and the base linear velocity as additional input to the encoder network E. We show in Figure 6 the average episodic reward achieved during training as a function of the training step with and without the additional data to the encoder input. Each curve in the plot is the average over five seeds of the same experiments. We notice the higher overall attained reward with the proposed additions. The additional input allows the learning to optimize rewards like foot clearance whose target value is varied from one episode to another. At the beginning of training, the average reward for both terms reaches higher values than at the end. This is the effect of the reward curriculum that scales down the negative reward penalties and the velocity curriculum that samples a very small range of the velocity command vector at the start of training.

C. High-level policy study

The strength of the proposed approach is the ability to reuse the low-level policy for different high-level tasks. In this section, we verify that we are able to learn a high-level policy

that adapts the low-level policy, learned on a flat terrain, in order to successfully traverse complex terrain that includes flat and rough plains, stairs, slopes and obstacles as shown in Figure 5.

Baseline comparison. We compare the results of the proposed hierarchic approach with a baseline end-to-end approach that learns to traverse the terrain from scratch with the height map state as input along with the low-level policy observation and reward function. However, we add the energy consumption cost term to the reward function for fairness of the comparison between the baseline and the proposed hierarchical approach. The training approach using the elevation map is inspired by the work done in [28]. We also use a similar curriculum on the terrain in order to gradually introduce the robot to terrains with increasing difficulty. It is important to note that the baseline training also uses the same RMA technique to enable the transfer of the learned policies on the real robot.

Figure 7 shows a scatter plot featuring the power consumption as a function of measured linear velocity. We collect the data points of the plot by running the two policies (hierarchic approach and baseline) on the velocity commands $V_x = \{0.0, 0.5, 1.0, 1.5, 2.0\}[m/s]$ for ten seconds at each velocity over a flat terrain. The figure shows that the proposed approach leads to lower overall power consumption with less variance than the baseline, especially at higher velocities. The hierarchic policy also tracks the commanded velocity better than the baseline policy both in terms of accuracy and variance.

Figure 8 shows the adaptation of the command parameters when the robot has to climb stairs. We see the change from their default value (dashed line). The high-level policy raises the foot height target (Fh) and the position gains which helps the robot use more power to climb the stairs. The angles of the hip and knee joints are lowered from their default values which could explain the overall improved energy efficiency of the robot (seen in Figure 7), since standing on straighter legs requires less power.

Sample Efficiency. Our hierarchical learning method also needs fewer samples for training since it splits complex problems into simpler sub-problems. In our experiments, the baseline method that learns to cross terrains from scratch required 20000 training iterations which is around two billion samples. However, with the hierarchical approach we are able to learn a low-level policy with 5000 iterations and a high-level policy with 3000 iterations. This means that our approach requires less than half the number of samples that the baseline needs because the complex problem of learning locomotion is done in the low-level policy on a simple terrain.

D. Real robot transfer

On its own, the low-level policy can be successfully transferred to the Mini-Cheetah robot without exteroception. We attach a video showing the policy in action with different command parameters. The video first shows the low-level policy run with different desired feet height and online

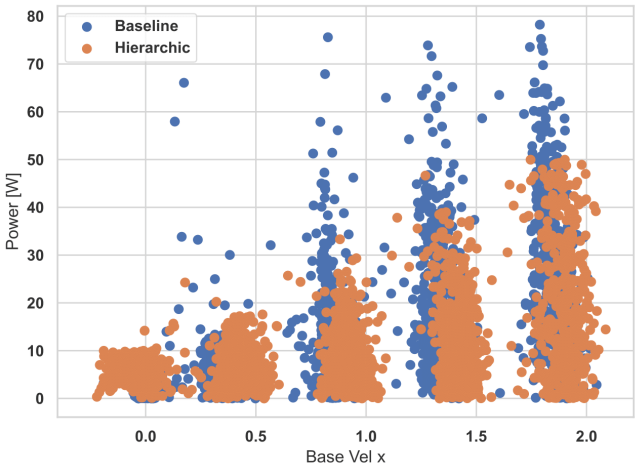

Fig. 7: Average power vs. measured velocity for the hierarchic and baseline policies.

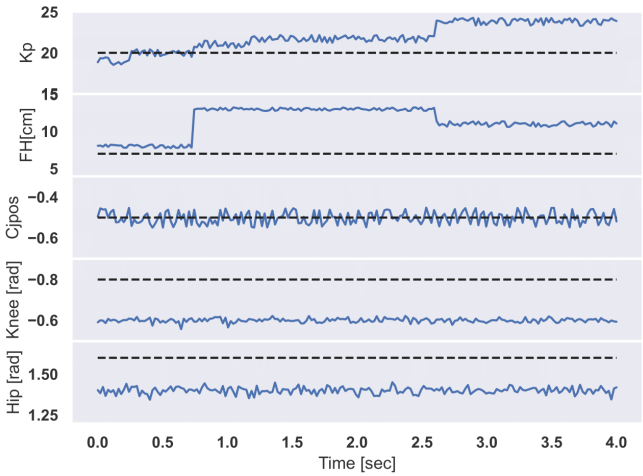

Fig. 8: Some command parameters' values adapted by the high-level policy while climbing stairs.

changing of the nominal pose. We can see the feet lifted differently and the robot changing its pose as it is moving. We also show multiple videos of the low-level policy traversing difficult surfaces (going up and down stairs, going over pavements and crossing steep slopes on muddy grass terrain) in a very robust manner when using parameters similar to the one learned by the high-level policy in the structured terrains. The video attachment also shows that a policy trained with fixed sets of parameters completely fails to be robust over structured terrains. The transfer of the high-level policy to the robot requires more instrumentation and handling exteroception (vision) that is beyond the scope of the current paper and will be the aim of a future work.

VI. Discussion and Limitations

We presented a hierarchical scheme for adapting learned quadruped locomotion. The main strength of the work lies in the versatile low-level policy training that can be reused for different tasks and in different environments and terrains. The low-level policy embeds several behaviour through the

command parameters. However, having a single low-level policy could also make training harder since we have to find one policy that is optimal for all combinations of the command vector. It is possible that training low-level policy on specialized subspaces of the command vectors would produce more specialized diverse skills. For example, we could train one policy that specializes in lifting its feet high and another one that specializes in taking long steps, and then combine these two policies in an ensemble manner [30]. This is an interesting question to explore in our future research.

We chose a specific set of commands that can directly and indirectly control aspects of the learned behaviour so we can simply show how they affect the policy when varied. For future work, an interesting objective could be to expand the command parameters to include all parameters related to the reward coefficients and learned pattern. For example, while we have assumed that PD gains are fixed for all joints, depending on the type of pattern we could propose learning a different gain for each leg or joint.

VII. Conclusion

In this work, we argued for expanding the notion of the command in learned locomotion policies by actively setting targets for locomotion features such as the swing feet height, stepping length, initial pose and gains. We show that we are able to learn a locomotion policy that reacts to the targets set by the command parameters. This makes it possible to adapt the low-level policy for different tasks. For this effect, we propose a hierarchical RL setup where a high-level policy learns to adapt the parameters of the low-level policy for versatile locomotion. We also showed that this hierarchical setup speeds up the learning of new tasks and that adapted policies can be better in quality than policies learnt from scratch. With the low-level parameterized policies we made our real robot successfully traverse complex terrains on which the baseline policy fails. In the future we will add exteroception to the Mini-Cheetah in order to be able to deploy and test high-level policies on the real robot in more complex tasks than those now demonstrated with a parameterized low-level policy.

References

[1] J. Di Carlo, P. M. Wensing, B. Katz, G. Bledt, and S. Kim, "Dynamic Locomotion in the MIT Cheetah 3 Through Convex Model-Predictive Control," in *IEEE International Conference on Intelligent Robots and Systems*, 2018.

[2] D. Kim, J. D. Carlo, B. Katz, G. Bledt, and S. Kim, "Highly dynamic quadruped locomotion via whole-body impulse control and model predictive control," 2019.

[3] P.-A. Léziart, T. Flayols, F. Grimminger, N. Mansard, and P. Souères, "Implementation of a Reactive Walking Controller for the New Open-Hardware Quadruped Solo-12," in *2021 IEEE International Conference on Robotics and Automation - ICRA*, Xi'an, China, May 2021.

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

[5] J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter, "Learning quadrupedal locomotion over challenging terrain," *Science Robotics*, vol. 5, no. 47, 2020.

[6] A. Kumar, Z. Fu, D. Pathak, and J. Malik, "RMA: Rapid Motor Adaptation for Legged Robots," jul 2021.

[7] S. Tonneau, A. Del Prete, J. Pettre, C. Park, D. Manocha, and N. Mansard, "An efficient acyclic contact planner for multiped robots," *IEEE Transactions on Robotics*, vol. 34, no. 3, pp. 586–601, jun 2018.

[8] S. Pateria, B. Subagdja, A.-h. Tan, and C. Quek, "Hierarchical reinforcement learning: A comprehensive survey," *ACM Comput. Surv.*, vol. 54, no. 5, jun 2021.

[9] B. Katz, J. D. Carlo, and S. Kim, "Mini cheetah: A platform for pushing the limits of dynamic quadruped control," in *Proceedings - IEEE International Conference on Robotics and Automation*, vol. 2019-May. Institute of Electrical and Electronics Engineers Inc., may 2019, pp. 6295–6301.

[10] G. Ji, J. Mun, H. Kim, and J. Hwangbo, "Concurrent Training of a Control Policy and a State Estimator for Dynamic and Robust Legged Locomotion," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 4630–4637, feb 2022.

[11] T. Miki, J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter, "Learning robust perceptive locomotion for quadrupedal robots in the wild," *Science Robotics*, vol. 7, no. 62, p. 2822, 2022.

[12] J. Choi, C. Dance, J.-e. Kim, K.-s. Park, J. Han, J. Seo, and M. Kim, "Fast adaptation of deep reinforcement learning-based navigation skills to human preference," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 2020, pp. 3363–3370.

[13] G. B. Margolis and P. Agrawal, "Walk these ways: Tuning robot control for generalization with multiplicity of behavior," *Conference on Robot Learning*, 2022.

[14] J. Lee, J. Hwangbo, and M. Hutter, "Robust Recovery Controller for a Quadrupedal Robot using Deep Reinforcement Learning," jan 2019.

[15] X. B. Peng, G. Berseth, K. Yin, and M. van de Panne, "Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning," *ACM Transactions on Graphics (Proc. SIGGRAPH 2017)*, vol. 36, no. 4, 2017.

[16] C. Li, M. Vlastelica, S. Blaes, J. Frey, F. Grimminger, and G. Martius, "Learning agile skills via adversarial imitation of rough partial demonstrations," in *6th Annual Conference on Robot Learning*, 2022.

[17] Y. Fuchioka, Z. Xie, and M. van de Panne, "Opt-mimic: Imitation of optimized trajectories for dynamic quadruped behaviors," 2022.

[18] 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," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, May 2023.

[19] D. Jain, A. Iscen, and K. Caluwaerts, "From Pixels to Legs: Hierarchical Learning of Quadruped Locomotion," nov 2020.

[20] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. The MIT Press, 2018.

[21] M. Aractingi, P.-A. Léziart, T. Flayols, J. Perez, T. Silander, and P. Souères, "Controlling the Solo12 Quadruped Robot with Deep Reinforcement Learning," Jul. 2023, scientific Reports.

[22] M. Bogdanovic, M. Khadiv, and L. Righetti, "Learning variable impedance control for contact sensitive tasks," *IEEE Robotics and Automation Letters*, vol. 5, no. 4, 2020.

[23] F. J. Abu-Dakka and M. Saveriano, "Variable impedance control and learning—a review," *Frontiers in Robotics and AI*, vol. 7, 2020.

[24] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *CoRR*, vol. abs/1707.06347, 2017.

[25] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," 2015.

[26] V. Konda and J. Tsitsiklis, "Actor-critic algorithms," in *Advances in Neural Information Processing Systems*, S. Solla, T. Leen, and K. Müller, Eds., vol. 12. MIT Press, 1999.

[27] V. Makoviychuk, L. Wawrzyniak, Y. Guo, M. Lu, K. Storey, M. Macklin, D. Hoeller, N. Rudin, A. Allshire, A. Handa, and G. State, "Isaac gym: High performance gpu-based physics simulation for robot learning," *CoRR*, vol. abs/2108.10470, 2021.

[28] N. Rudin, D. Hoeller, P. Reist, and M. Hutter, "Learning to walk in minutes using massively parallel deep reinforcement learning," *CoRR*, vol. abs/2109.11978, 2021.

[29] G. B. Margolis, G. Yang, K. Paigwar, T. Chen, and P. Agrawal, "Rapid locomotion via reinforcement learning," 2022.

[30] X. B. Peng, M. Chang, G. Zhang, P. Abbeel, and S. Levine, "Mcp: Learning composable hierarchical control with multiplicative compositional policies," in *Advances in Neural Information Processing Systems 32*, 2019, pp. 3681–3692.