Controlling the Solo12 Quadruped Robot with Deep Reinforcement Learning

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

Controlling the Solo12 Quadruped Robot with Deep Reinforcement Learning

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

***Abstract*—Quadruped robots require robust and general locomotion skills to exploit their mobility potential in complex and challenging environments. In this work, we present the first implementation of a robust end-to-end learning-based controller on the Solo12 quadruped. Our method is based on deep reinforcement learning of joint impedance references. The resulting control policies follow a commanded velocity reference while being efficient in its energy consumption, robust and easy to deploy. We detail the learning procedure and method for transfer on the real robot. In our experiments, we show that the Solo12 robot is a suitable open-source platform for research combining learning and control because of the easiness in transferring and deploying learned controllers.**

***Index Terms*—Quadruped Locomotion, Deep Learning, Reinforcement Learning.**

I. Introduction

LEGGED robots can traverse challenging, uneven terrains. The interest in the design and control of legged robots has resurged due to the development of many quadruped platforms such as the Mini-Cheetah [1], HyQ [2], ANYmal [3], Solo [4], Spot Mini [5] and Laikago [6]. These platforms serve as suitable test-benches for control and locomotion research. Finding the right way to control such systems is crucial to fully exploit quadruped mobility. In this paper we conduct our experiments using the Solo12 [7] robot which is a recent alternative platform that provides a reliable low-cost open-access quadruped within the Open Dynamic Robot Initiative[1].

Many control methods based on motion planning and trajectory optimization have been proposed for quadrupeds. Winkler et al. [8] suggest using a tree search to plan the body path and footsteps positions in the environment for the HyQ robot [2]. Bellicoso et al. [9] show a ZMP-based motion planner for executing dynamic transitions between gaits on the ANYmal robot [3]. The approaches proposed by Di Carlo et al. [10] and Kim et al. [11] use model-predictive control (MPC) on a centroidal model to plan the base trajectory and ground reaction forces of the feet in contact for the Mini-cheetah [1]. Kim et al. [11] propose a whole body control formulation that outputs the necessary low-level control in order to track the base trajectory on a shorter-time horizon. Leziart et al. [7] implement a similar MPC-based approach for Solo12 [4] while simplifying solutions for the computation of the whole-body control. While all these methods produce robust dynamic controllers, they often require some aspects of control, such as gait, feet trajectories, body height and orientation etc., to be determined by hand-tuned parameters that are hard to adapt for all the different environments a quadruped might encounter in the real-world. These controllers often rely on models that are hard to design and observe in many situations. Furthermore, these methods are computationally heavy at run time and sometimes laborious to set up.

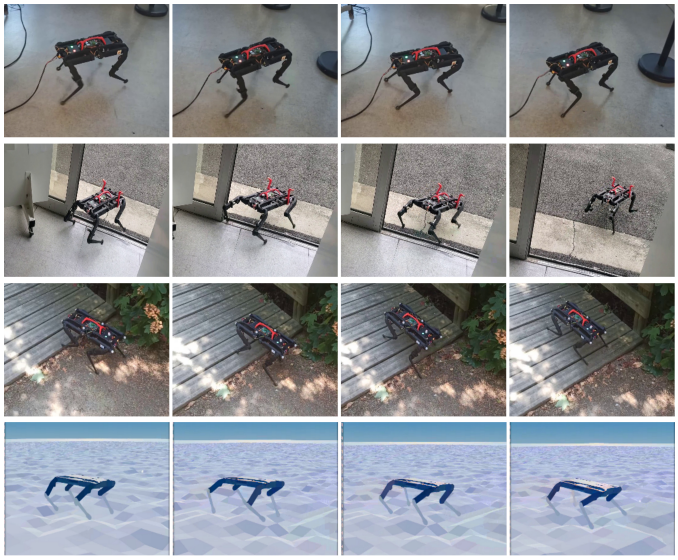

Fig. 1. Snapshots of the Solo12 quadruped in simulation and in real settings driven by a reactive controller learned through deep reinforcement learning. With learned controllers the robot can traverse various outdoor environments with slopes and rough ground. Full video: https://youtu.be/t-67qBxNyZI

In contrast to optimization methods, data-driven methods that are based on learning can be used for designing controllers. Specifically, reinforcement learning (RL) is an alternative approach for obtaining highly performant agents that act in their environment in which the dynamics and transitions are modeled as a Markov decision process (MDP) [12]. There are many early examples of applying RL to robotic tasks such as manipulation [13]–[16] and locomotion [17], [18]. However, RL used to be hard to scale and was often limited to solving small sub-problems in the control pipeline in which most of the components were hand-designed. With increased computing power and recent evolution of deep learning methods that use large scale neural networks we can now solve problems requiring high-dimensional data [19]–[21]. Deep RL combines neural networks with RL algorithms to learn value function approximations [22]–[24] and/or, directly, policies [25]–[27]. Using images from camera, deep RL has been successfully

[1] LAAS-CNRS, Université de Toulouse, CNRS, Toulouse, France.
[2] NAVER LABS Europe, Grenoble, France.
michel.aractingi@naverlabs.com
[1]https://open-dynamic-robot-initiative.github.io/

applied for manipulation tasks such as object insertion, peg in a hole [28], and reaching and grasping objects [29].

In recent works, deep RL has been applied to quadrupeds [30] and bipeds [31] for the purpose of learning end-to-end controllers. Hwangbo et al. [30] outline a general RL method for learning joint angle controllers from the base and joint states of the robot. The authors propose learning a model of the actuation dynamics of ANYmal [3] from real-data that can then be deployed in simulation thus enabling the learned policies to transfer to the real-world. In the work by Miki et al. [32], the authors deploy a similar learning scheme and augment the action space with a central pattern generator (CPG) layer that produces a baseline walking gait pattern for the feet [33]. Using proprioception and a LIDAR based reconstruction of the environment, the policy then learns to manipulate the CPG phase and joint angles to modify the gait. Similarly, Lee et al. [34] learn a policy that modifies the phase and shift of CPG functions that determine the foot trajectories which are fed to model-based controller to produce joint angle control. Ji et al. [35] propose learning a control policy through RL and a state estimation network with supervised learning that tries to predict state variables that are not measured on the real robot but are available in simulation and provide vital information for learning robust policies, e.g., feet contact states and linear velocity of the base. These works mostly rely on domain randomization techniques that add noise to the sensory input of the policy and to the dynamics of the simulation in order to learn policies that transfer to the real system. Rapid motor adaptation (RMA) presents an alternative method for transfer by adding an adaptation network to the training architecture [36], [37]. The original policy is initially trained in different simulated conditions by varying ground friction, payload, motor strength, etc. In this first learning phase the algorithm also constructs a compact latent descriptor of the relevant aspects of these different conditions. In the second phase of learning the system learns an adaptation network that estimates this latent condition descriptor using only the history of measurements available in the real robot. This ability to adapt to different conditions also compensates for discrepancy between simulated and real settings.

In this paper, we present an RL approach for learning robust controllers on the Solo12 robot [7]. We detail our procedure for setting up the MDP components, i.e., state space, action space and reward function, along with the additional techniques required to make the learning converge and transfer to the real robot. We use proximal policy optimization (PPO) [27] as the RL algorithm, and present results on the real Solo12 robot with videos and snapshots. Figure 1 depicts examples of Solo12 controlled by learned policies in simulation and real-world using two different joint angle configurations. Our main contributions are:

- Detailed description and analysis of a deep RL method for learning controllers for the Solo12 that transfer to the real-robot.
- Introduction and study of a realistic energy loss penalty for policy learning based on actuator friction and Joules losses identification.
- Open-source implementation to make the work reproducible that is in line with the open-source mission of Solo12. [2]

In Section II we present notations and preliminaries for RL and MDPs. Section III details our learning methods and notably the core components of the MDP, i.e., the state, actions, reward function and transfer methods. Section IV showcases our results in simulation and with the real robot. Finally, we conclude in Section V.

II. Reinforcement Learning Preliminaries

We model the reinforcement learning (RL) environment as a Markov decision process (MDP) with continuous state and action spaces [12]. An MDP is defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T}, P_0)$, where $\mathcal{S} \subset \mathbb{R}^{d_S}$ is a set of states, and $\mathcal{A} \subset \mathbb{R}^{d_A}$ is a set of actions. In RL setting, only spaces $\mathcal{S}$ and $\mathcal{A}$ of the MDP are known to the learning agent. The agent starts by observing the initial state $s_0 \in \mathcal{S}$ and it performs actions $a_t \in \mathcal{A}$ in the environment at discrete times indexed by $t \in \mathbb{N}$, after which it receives a stochastic reward $r_{t+1} \in \mathbb{R}$ and observes a new stochastic state s_{t+1}.

The environment dynamics is described by a transition probability distribution $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}_+$, such that $\mathcal{T}(s, a, s') = p(s'|s, a)$ is the probability (density) that the next state is s' given that the current state is s and that the action taken is a. P_0 is the initial state probability distribution. Similarly, the stochastic reward $r \in \mathbb{R}$ after taking an action a in a state s and observing a state s' next is governed by the function $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \times \mathbb{R} \to \mathbb{R}_+$ that defines the probability densities $p(r|s, a, s')$.

To formalize the goal of learning, we define a stochastic policy $\pi_\theta(h, a) = p_\theta(a_t = a \mid h_t = h)$, parameterized by θ, that gives the probability density of taking an action a given a state-action history $h = (s_0, a_0, s_1, a_1, ... a_t)$. The learning objective is to find the parameters θ of the policy for which the expected discounted sum of rewards $J(\theta) := \mathbb{E}[\sum_{t=1}^{H} \gamma^{t-1} r_t]$ is maximized. In this expression H is the horizon of the episode and $\gamma \in [0, 1]$ is a discount factor. The expectation is taken over the stochastic policy, the initial state distribution, and the stochasticity of rewards and state dynamics.

III. Method

Our goal is to define an RL method that can learn to control a Solo12 robot conditioned on a user-defined velocity command. The Solo12 quadruped is a 12 degrees of freedom version of Solo8 [4] that can be torque controlled. We will describe the design of our state space, action space and reward function in the following sections.

In general, our control policy is implemented as a neural network that takes the state as an input, and outputs the actions. The actions that define joint angle targets are then fed to a Proportional Derivative (PD) feedback controller in order to get the desired torques for commanding the robot joints. Figure 2 shows a summary of the control scheme in terms of the inputs/outputs of the control network and how it is deployed on the real robot. The estimation network in Figure 2

[2]Code available at: https://github.com/Gepetto/soloRL

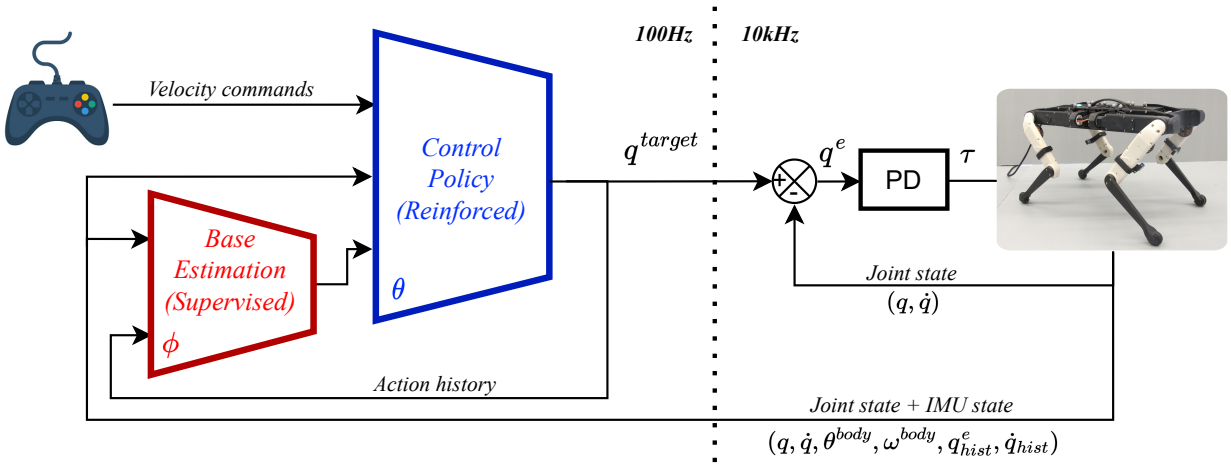

Fig. 2. Summary of the control scheme. The control policy receives a desired 3D velocity to follow. Using the linear velocity prediction from the base estimation network and other state values measured by the robot's sensors, the control policy outputs joint angle displacements to a nominal joint angle configuration. The target is fed to a proportional derivative (PD) joint impedance controller to calculate the torques. The estimation network predicts the linear velocity of the base from the state information. The control scheme on the real robot is split into two levels, the RL policy (θ) is queried at 100Hz to give a joint target action that will be executed at a high frequency low level PD loop at 10kHz.

is trained with supervised learning to predict the linear velocity of the base. The control policy parameters are optimized using the proximal policy gradients objective (PPO) [27].

A. State space

The state space of the MDP is constructed from the proprioception of the robot, i.e., the sensory readings from the joint encoders, and the inertial measurement unit (IMU). The state at time t includes the base state and the joint state. The base state consists of the orientation $\theta_t^{\text{body}} \in \mathbb{R}^3$, linear velocity $v_t^{\text{body}} \in \mathbb{R}^3$ and angular velocity $\omega_t^{\text{body}} \in \mathbb{R}^3$ of the body. The joint state consists of the joint angles $q_t \in \mathbb{R}^{12}$, joint velocities $\dot{q}_t \in \mathbb{R}^{12}$ along with history of the joint target errors $\{q^e_{t-j} \in \mathbb{R}^{12}\}^{j=1...N}$ (explained below) and joint velocities $\{\dot{q}_{t-j} \in \mathbb{R}^{12}\}^{j=1...N}$. In our work $N = 3$, i.e., the velocities and joint target errors from last three policy steps are stored and added to the state. We also include to the state s_t the last two actions $\{a_{t-j} \in \mathbb{R}^{12}\}^{j=1...(N-1)}$. Finally, the 3D velocity command is also given as an input to the policy neural network. The design of this state space is similar in spirit to those proposed in other works [32], [34], [35].

The orientation and angular velocity of the base can be provided by an IMU on-board the robot, which internally uses an Extended Kalman Filter (EKF) to estimate angular orientation from raw gyroscope and accelerometer sensor data. At each joint an optical encoder measures the joint angles from which one can then compute the joint velocities. The joint target errors are the differences between the target joint angles conveyed to the PD controller and the measured joint angles, i.e., $q^e_t = q^{target}_{t-1} - q_t$. The error q^e_t is related to a torque, and it implicitly provides rich information, such as the contact state of the feet with the ground, about the environment. The target errors also vary by terrain as the vertical foot position shifts if the terrain is not flat, which changes the resulting joint angles. Therefore, it is also crucial to add the last two actions of the policy to the state so that the learning can observe the change of the joint target errors for the similar actions which indicates a change in the terrain.

The on-board IMU does not directly measure linear velocity, and estimating the velocity from accelerations often diverges over time due to sensor bias. Like Ji et al. [35], we propose training a separate state estimation network for estimating the base linear velocity from the IMU and joint encoder measurements. The state estimation network is trained through supervised learning and it receives as input the base orientation and angular velocity along with the joint angles, joint velocity, history of the past joint angle errors, joint velocities and actions. The output is a three-dimensional vector that estimates the linear velocity in the x, y, z directions. Implementation details can be found in Section IV.

B. Action space

The design of the action space can make a difference on the learning speed and policy quality. Peng et al. [38] showed that direct torque control is harder to learn than joint position control in RL-based systems. Similar observations were made in the literature on learning quadruped robots' locomotion [30], [36]. We also argue that torque control policies are harder to transfer than joint angle control policies, due to the fact that joint angle control is inherently stable after choosing appropriate impedance gains K_p and K_d. While direct torque control can result in diverging motion especially during the flying phases of the legs where the apparent joint inertia is low, the position-based impedance control forces the joints to behave like a spring damper system.

In this work, we propose learning a policy π that outputs displacements of the reference joint angles with respect to the nominal pose of the robot, i.e., $\pi_\theta(s_t) = \Delta q^\theta_t$, where π is implemented by the policy neural network parameterized by θ, and s_t is the state input to the policy at time t. The target joint angles can then be computed as:

$$q^{target}_t = q_{init} + \lambda_q \Delta q^\theta_t,$$

where q_{init} are robot's nominal joint angles around which the policy actions are centered. We define λ_q as a scalar that scales the output of the network before adding to q_{init}. Given q_t^{target}, we use a PD controller to compute the torques:

$$\tau_t = K_p(q_t^{target} - q_t) - K_d\dot{q}_t$$

with the proportional and derivative gains K_p and K_d. It is important to note that using such a joint controller doesn't imply having a rigid position control. The reference angle q_t^{target} should not be interpreted as positions to be reached, but rather as intermediate control variables. The resulting system is analog to elastic strings that pull the joint angles toward q_t^{target}.

C. Reward function

The reward function defines the task. The main task in our work is to follow a given reference velocity. In order to get natural locomotion that can be deployed on the robot, one needs some constraints on the robot's pose, joint torques, joint velocities, etc. After each action a_t, the robot receives a reward r_{t+1}. We split our reward r into one main positive term that rewards the tracking of the commanded velocity and several weighted penalty terms that act as negative costs in the reward. The values of the weights are listed in Table III. The reward terms and state variables below are implicitly indexed by the time step index t but we only include this index when necessary for clarity.

Command velocity tracking. The reward r_{vel} for following the command velocity is based on the squared Euclidean distance between the 3D vector V_{x,y,w_z} consisting of the forward, lateral and yaw velocities of the body and the 3D velocity command V^{cmd}, i.e.,

$$r_{vel} = c_{vel}e^{-||V^{cmd}-V_{x,y,w_z}||^2}.$$

with coefficient c_{vel} that scales the reward.

Foot clearance penalty. To encourage the robot to lift its feet high even when training on a flat surface, we use the foot clearance objective proposed by Ji et al. [35]. Denoting the height of the i-th foot by $p_{z,i}$, we set a constant foot height target p_z^{max} and define the foot clearance penalty as

$$r_{clear} = c_{clear}\sum_{i=1}^{4}(p_{z,i} - p_z^{max})^2||\dot{p}_{xy,i}||^{0.5},$$

where $\dot{p}_{xy,i}$ stands for the velocity of the foot i in the x, y direction so that the target is not active during the ground contact and it is approximately maximal in the middle of the swing phase. Scalar c_{clear} is a weight for this penalty.

Foot slip penalty. When a foot comes in contact with the ground, its x, y velocity should be zero in order to avoid slipping. We define a foot slip penalty as

$$r_{slip} = c_{slip}\sum_{i=1}^{4}C_i||\dot{p}_{xy,i}||^2,$$

where C_i is a binary indicator of the ground contact of the i-th foot, and c_{slip} is penalty weight.

Base orientation and velocity penalties. The base pitch, roll and velocity in the z direction should all be near zero to produce stable motion. With scalars c_{orn} and c_{vz}, we define this penalty as

$$r_{base} = c_{orn}(roll^2 + pitch^2) + c_{vz}V_z^2.$$

Joint pose penalty. We add a penalty on the joint angles in order to learn to avoid large joint displacement. We define this penalty as the deviation from the nominal joint angles at the initial state, as

$$r_{joint} = c_q||q_t - q_{init}||^2$$

with weight c_q.

Power loss penalty. For safety reasons and for saving energy, we would usually prefer to minimize the overall power consumption of the robot. The power loss term encapsulates the relationship between the torque and velocity at the joint level - we use the model proposed and identified by Fadini et al. [39] which includes the heating by Joules loss in the motors P_J as well as the losses by friction P_f.

We denote with τ_f the torque necessary to overcome the joint friction :

$$\tau_f = \tau_u\text{sign}(\dot{q}) + b\dot{q},$$

where q, $\dot{q}$ are respectively the joint position and velocity. The identified model parameter are the Coulomb friction $\tau_u = 0.0477$[Nm] and the the viscous friction coefficient $b = 0.000135$[Nm·s].

The two sources of power losses can then be expressed as

$$\begin{aligned} P_f &= \tau_f\dot{q} \quad [W], \\ P_J &= K^{-1}(\tau + \tau_f)^2 \quad [W], \end{aligned}$$

where τ is the joint output torque and $K = 4.81$[Nm·s] is linked to the motor coil resistance and motor constant.

The total power over joints and the penalty term used in the reward is taken as the sum over all joints:

$$r_E = c_E\sum_{j=1}^{12}P_{f,j} + P_{J,j}$$

with the weight c_E.

Action smoothness penalties. To generate joint trajectories without vibrations and jitter, we define a penalty on the first and second order differences in the joint angle values:

$$\begin{aligned} r_{smooth} = \quad & c_{a1}||q_t^{target} - q_{t-1}^{target}||^2 \\ & + c_{a2}||q_t^{target} - 2q_{t-1}^{target} + q_{t-2}^{target}||^2 \end{aligned}$$

with weights c_{a1} and c_{a2}.

Total reward. The final reward is a weighted sum of the positive velocity tracking reward minus a sum r_{pen} of all the penalties explained above:

$$r_{total} = r_{vel} - r_{pen}.$$

D. Domain and dynamic randomization

In order to learn policies that transfer to the real robot we have to identify and bridge the sim-to-real gap. We decided to use domain randomization techniques by adding noise to the state and randomizing some aspects of the simulator dynamics. Table I shows the noise models used for each element in the state and dynamics. For the dynamics, we found that for Solo12 it was enough to randomize the gains of the PD controller in order to learn policies that adapt to some stochasticity in the low level control that can come from many factors. This is in contrast to previous work on ANYmal and Mini-cheetah where more randomization is needed for the center of mass, mass of the body and links, positions of the joints and motor friction [30], [32], [34], [35]. Randomizing the state is essential in order to overcome sensory noise. Our results show that one can learn a transferable policy on Solo12 using this simple randomization strategy.

TABLE I
UNIFORM NOISE FOR EACH OF THE STATE OBSERVATIONS AND PD CONTROLLER GAINS.

Observation Noise	
θ_{body}	$U^3(-0.05, 0.05)$
ω_{body}	$U^3(-0.10, 0.10)$
v_{body}	$U^3(-0.10, 0.10)$
q	$U^{12}(-0.05, 0.05)$
$\dot{q}$	$U^{12}(-0.50, 0.50)$
Dynamics Noise	
K_p	$U(-1.0, 3.0)$
K_d	$U(-0.1, 0.1)$

E. Curriculum learning

Reward curriculum. Due to the elaborate penalty terms of the reward function, we observe that the agent may learn to neglect the positive reinforcement signal from following the command velocity and learn to stand still since this optimizes several penalty terms in the reward. In order to bypass this problem, we introduce a linear curriculum on the reward. Curriculum learning is a popular method that introduces easier tasks to learn at the start of training and gradually increases the level of difficulty as training progresses [40]. Like Hwangbo et al. [30], we multiply the cost terms of the reward function by a curriculum factor $k_c \in [0, 1]$ that is equal to zero at the start of the training and slowly increases up to one through the training iterations. The reward function becomes $r_{total} = r_{vel} - k_c r_{pen}$. This way we first train the agent to follow the command velocity in any manner before emphasizing the cost terms in the reward in order to refine locomotion.

Noise curriculum. We also propose a curriculum on the injected noise for randomizing the state and dynamics. We found that decoupling the curriculum of the reward and randomization works better. Therefore the sampled noise in Table I is multiplied by another curriculum factor $k_{c,noise} \in [0.0, 1.0]$ that is increased at a slower pace than k_c.

Terrain curriculum. We introduce rough terrains at the end of training to learn from more complex interactions when the ground is not flat. This helps in refining the robot's locomotion in terms of lifting all feet equally in order to keep balance. At the last 1000th training iteration, we start sampling random heightmaps at the start of the episodes. We also lower some PPO parameters to perform more conservative updates to the policy in order to avoid catastrophic forgetting [41] of locomotion on flat terrain once the rough terrains are introduced and the training data distribution changes. The PPO parameter values before and after introducing the rough terrains are listed in Table II, we refer to Schulman et al. [27] for a description of these parameters.

TABLE II
PPO PARAMETERS WHEN TRAINING ON FLAT TERRAIN AND NON-FLAT TERRAIN.

PPO parameters	Flat terrain	Non-flat terrain
Clip ratio	0.200	0.050
Gradient norm clip	0.500	0.300
Entropy coefficient	0.010	0.000
Learning rate	0.005	0.001

IV. RESULTS

In this section, we analyze the locomotion produced by our learned control policies. We test both symmetric ($\overline{><}$) and non-symmetric ($\overline{<<}$) poses of the legs with the policy being able to learn both successfully. We display results about velocity tracking and energy consumption of the learned controller. Successful real robot transfer experiments are conducted and discussed in the following sections.

TABLE III
REWARD TERMS' WEIGHTS.

c_{vel}	c_{clear}	c_{slip}	c_{orn}	c_{vz}	c_q	c_E	c_{a1}	c_{a2}
6.0	20.0	0.07	3.0	1.2	0.5	2.0	2.5	1.5

A. Implementation details

The control policy is implemented as a multi-layer perceptron with three hidden layers of sizes 256, 128 and 32 with Leaky ReLU activations between each layer. The control policy runs at a frequency of 100Hz. We use the Raisim simulator [42] for training. The simulator frequency is set at 1kHz which means that the PD control between each RL step is executed ten times. On the real-robot we have a low-level loop at 10kHz for communicating with the actuators, but the policy network is still queried every 0.01 seconds (see Figure 2). In simulation, 300 different versions of the robot are run in parallel processes in order to collect diverse data faster. The value of the PD control gains are $K_p = 3$ and $K_d = 0.2$ respectively. On the robot, the computation of actions from states only takes 10 μs on a Raspberry Pi 4 which makes this approach particularly appealing due to its simple setup and high computational speed.

The state estimation network is also a multi-layer perceptron with two hidden layers of sizes 256 and 128 with Leaky ReLU activations and a three dimensional output corresponding to the linear velocity. To train the state estimation network, we collect a dataset by running learned policies on random velocity commands. We found that a dataset of 50,000 samples (policy

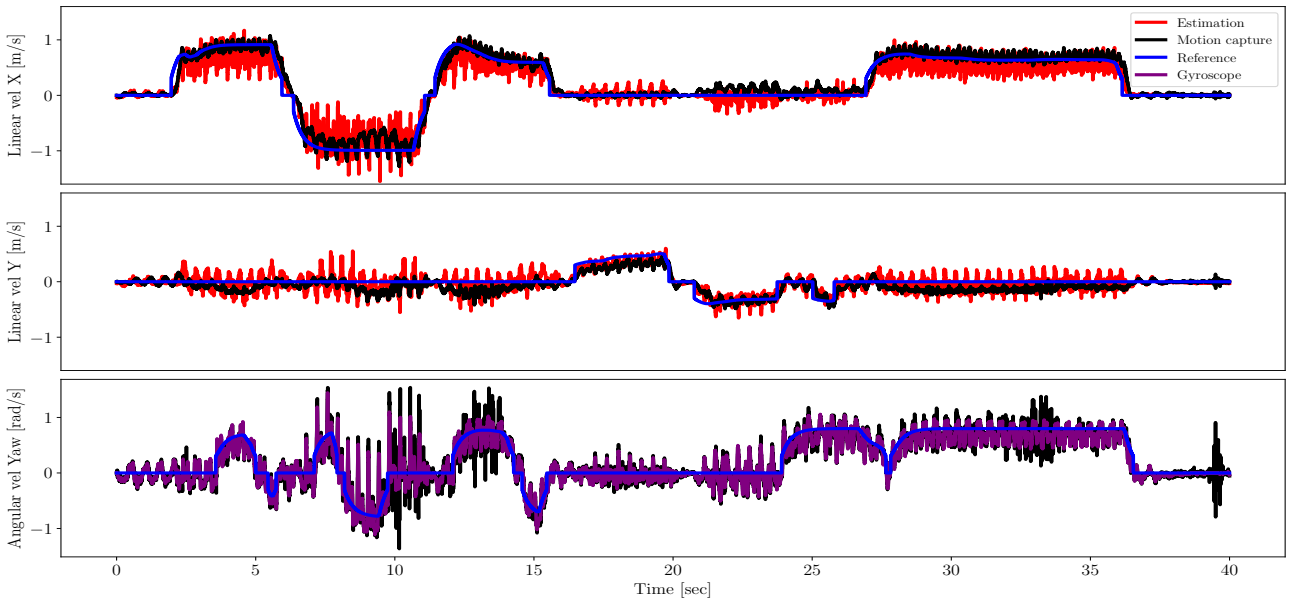

Fig. 3. Plot of the 3D velocity command controlled by a gamepad to command the real robot in blue. The red curve plots the output of the state estimation. The black plot is the motion capture of the real solo12 to convey the ground-truth base velocity. The purple plot is the yaw velocity estimate from the gyroscope in the IMU. The x-axis shows time in seconds.

steps) is enough to train the estimation network to a good accuracy. The data is collected with the random noise added to the observations and PD gains along with randomizing the terrains between rough and flat. We train on a supervised cost to minimize the mean squared error loss using the Adam optimization algorithm [43].

We use the objective from PPO [27] to train the policy network. This is done in an actor-critic setup where, in addition to the policy network (actor), we train another network (critic) that learns to map the state to a single scalar value that estimates the desirability of the state. This scalar value is commonly used for reducing the variance of the RL objective [27]. In each training episode, the policy is run for 100 steps (= 1 second of real-time) to collect data for optimizing the objective. The episode ends if the body of the robot comes in contact with the ground. Even though locomotion is not an episodic task with a natural endpoint and the episode is not reset between each training epoch, we choose to introduce random resets at the beginning of some episodes since this appears to stabilize training. At the start of each episode, a random velocity command is sampled and then scaled by the noise curriculum factor so that the network starts learning gradually from one low velocity towards higher ones. The initial state at the start of each episode is set at the nominal joint pose q_{init} with zero joint velocity. We use the `stable-baselines` [44] open source implementation of the PPO algorithm.

As mentioned before, at the beginning of training the ground is flat, but in order to learn more robust policies, we gradually introduce some non-flat terrains by sampling random height values for points in a regular grid. At the last 1000th training iteration, 80% of the parallel processes start sampling non-flat terrains. We found that we need around 10,000 training iterations which equates to 300 million collected samples with 300 parallel processes.

Table III shows the coefficient values that are used to scale each term in the reward function. Along with choosing the right values of the weights, we choose the desired maximum foot height in the foot clearance reward to be $p_z^{max} = 6cm$. We scale the output of the policy network, with scalar $\lambda_q = 0.3$ before integrating towards the target joint angles.

B. Velocity tracking

We first judge the quality of the learned controller by its ability to follow the reference velocity in the forward, lateral and yaw directions. During training we randomly sample the velocity vector based on the following uniform distributions: $V_x \sim U(-1.5, 1.5)$, $V_y \sim U(-1, 1)$ and $W_z \sim U(-1, 1)$. As mentioned before, these values are scaled by k_{noise} in order to start learning with low velocities before gradually increasing the range of sampled velocities.

Figure 3 shows the velocity plots of a random walk recorded while guiding the robot with the gamepad across the room. The blue lines plot the reference velocity command in three directions. The black lines represent the robot's body velocity estimation from motion capture data. The red lines in the first two plots are the state estimation network's velocity estimates in the x and y directions. From the plots we see that the real robot is able to follow the commanded velocity well, as indicated by the alignment between the motion capture plots – which provides ground-truth values – and the reference command plots. The velocity predictions from the state estimation network are similar to the ones from motion capture while being more noisy. The noise in the prediction, that is given as an input to the control network, does not appear to downgrade the performance of the controller. Indeed, this robustness to noisy estimation is expected as noise is added to the linear velocity input during training.

Figure 4 shows the plot of the hind right joint angle target vs. the measured joint angles for the same random run. We

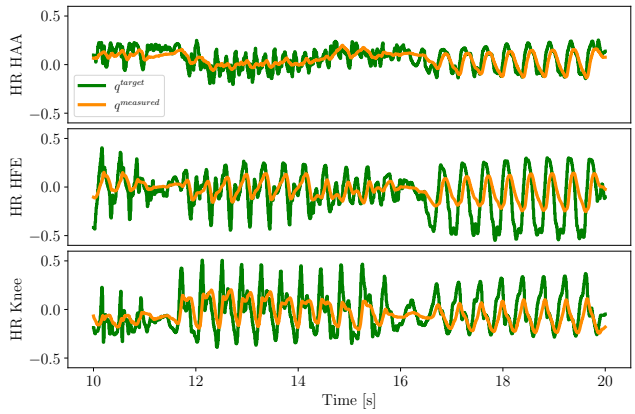

Fig. 4. Plot of the desired joint angle command vs. the measured joint angles over a random run for the hind right leg.

observe that the target joint angles are not reached. The difference between the command and the achieved angles showcases the nature of the soft impedance control which resembles elastic strings where the desired joint velocity is zero. Similar behaviour is observed for the other legs.

C. Energy consumption

In order to verify the usefulness of the proposed power loss penalty in the reward function, we run several experiments while varying the power loss weight c_E in the reward and observe its effect on the learned policy. We run the policies in simulation for five seconds for the maximum forward velocity command of $1.5[m/s]$. This test focuses on a rapid and dynamic task that would require most energy.

Table IV lists the effect of c_E on the average power consumption, velocity error and base height during the test task. We first observe that the increase of c_E decreases the power loss. This confirms that the power term on the learned policy makes intuitive sense and that it can be tuned to learn locomotion with different power profiles. For $c_E = 50$ the increase in power consumption seems to contradict this general conclusion. However, with such a high power loss penalty the policy merely learns to stand still in an inefficient pose.

Increasing the weight c_E makes the policy prioritize optimizing the power loss rather than other rewards such as velocity tracking. We observe this effect in the table as the velocity error increases when using policies that have learned to consume less power due to higher c_E. The velocity error column contains the l_1 norm of the difference between the desired velocity and the achieved velocity. Note that even though the error increases, we see a big decrease in the consumed power which would make the policies with $c_E \in [3, 4]$ an attractive option since the robot would have slightly less accurate velocity tracking but still save more than 30% on the consumed power.

The base height could be another indicator of energy efficiency since standing on straighter legs requires less power. In Table IV we list the body height as a function of c_E, and observe a gradual $2\,cm$ increase in the base height when c_E increases from zero to ten. Beyond $c_E = 10$ the RL ceases to produce good policies as mentioned before.

D. power vs. torque penalty

In previous work [30], [32], [35] penalty terms on the torque magnitude, joint velocity magnitude and joint accelerations are used in the reward. We trained several policies using these penalty terms to compare with the proposed power cost. The last row in Table IV shows the power loss vs. velocity error averaged over three policies trained with those penalties. The learned policies are less energy efficient than most of the policies that have the power term with high variance between the policies. In practice we found it easier to tune a single power weight during experimentation rather than tuning three separate weights for torque, velocity and acceleration terms with different units. The power loss formula expresses the relationship between the torque and the velocity by effectively combining the other three penalties into a one single physical and coherent term.

TABLE IV
AVERAGE POWER VS. VELOCITY ERROR AS A FUNCTION OF THE POWER WEIGHT c_E.

c_E	Power [W]	Velocity error [m/s]	Base height [m]
0.0	17.7	0.079 ± 0.054	0.23 ± 0.004
0.1	16.2	0.083 ± 0.067	0.23 ± 0.006
1.0	13.7	0.092 ± 0.065	0.24 ± 0.009
2.0	12.0	0.121 ± 0.064	0.24 ± 0.007
3.0	11.0	0.141 ± 0.086	0.24 ± 0.014
4.0	10.2	0.145 ± 0.091	0.25 ± 0.004
10.0	7.7	0.198 ± 0.164	0.25 ± 0.014
20.0	5.5	0.275 ± 0.113	0.23 ± 0.005
50.0	7.5	1.51 ± 0.039	0.17 ± 0.005
With joint torque, velocity and acceleration penalty			
-	15.5	0.122 ± 0.054	0.27 ± 0.008

E. Comment on the policy transfer to Solo12

As explained earlier, random uniform noise was added to the robot dynamics and state observations during training. This noise was progressively inserted through the curriculum factor $k_{c,noise}$, starting with noiseless simulations and increasing the noise magnitude as the training progressed. The goal was to prepare the policy network for sim-to-real transfer so that, once deployed on a real Solo12, it would still produce a robust behavior even if the model did not perfectly fit the system. Such discrepancy is inevitable since different motors have slightly different characteristics that vary as coils get warmer, and the model does not include joint friction, its inertia matrices are not perfectly accurate, etc.

Despite these inevitable model inaccuracies, the policy was successfully transferred on the very first try and the robot moved around without falling. The transfer did not require learning an actuator model, as done in other works [30], or modelling the actuation dynamics in the PD formula. This demonstrates how a simple randomization during training is enough for direct transfer to Solo12, probably by virtue of the fast dynamics of this robot (lightweight quadruped powered by low inertia actuators with high bandwidth) which leads to a limited sim-to-real gap. This all makes the Solo platform an attractive choice for deploying RL schemes.

V. Conclusion

We presented an end-to-end approach for learning controllers for the Solo12 quadruped robot. We described the training method in detail with the choice of state space, action space and reward function along with the curriculum strategy and domain/dynamic randomization method for learning transferable policies for following 3D velocity commands. We presented results for the velocity tracking and energy loss. Numerous experimental tests on the real robot have shown that robust locomotion policies with different energy profiles can be learned by randomizing the weights of the power loss variables. Based on this work and previous publications, we plan to conduct a large scale study in the near future to compare the potential of current model-based and RL-based controllers on Solo12.

References

[1] B. Katz, J. D. Carlo, and S. Kim, "Mini cheetah: A platform for pushing the limits of dynamic quadruped control," in *Proceedings - IEEE Int. Conf. Robot. Automat.*, vol. 2019-May. Institute of Electrical and Electronics Engineers Inc., 2019, pp. 6295–6301.

[2] C. Semini, N. G. Tsagarakis, E. Guglielmino, M. Focchi, F. Cannella, and D. G. Caldwell, "Design of hyq – a hydraulically and electrically actuated quadruped robot," *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering*, vol. 225, no. 6, pp. 831–849, 2011.

[3] M. Hutter *et al.*, "Anymal - a highly mobile and dynamic quadrupedal robot," in *2016 IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2016, pp. 38–44.

[4] F. Grimminger *et al.*, "An Open Torque-Controlled Modular Robot Architecture for Legged Locomotion Research," *IEEE Robot. Automat. Lett.*, vol. 5, no. 2, pp. 3650–3657, 2019.

[5] B. Dynamics, "Spot Mini Robot," www.bostondynamics.com/spot.

[6] X. Wang, "Unitree Robotics," www.unitree.com.

[7] 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 *IEEE Int. Conf. Robot. Automat.*, Xi'an, China, 2021.

[8] A. W. Winkler, C. Mastalli, I. Havoutis, M. Focchi, D. G. Caldwell, and C. Semini, "Planning and execution of dynamic whole-body locomotion for a hydraulic quadruped on challenging terrain," *Proceedings - IEEE Int. Conf. Robot. Automat.*, pp. 5148–5154, 2015.

[9] C. D. Bellicoso, F. Jenelten, C. Gehring, and M. Hutter, "Dynamic Locomotion Through Online Nonlinear Motion Optimization for Quadrupedal Robots," *IEEE Robot. Automat. Lett.*, vol. 3, no. 3, pp. 2261–2268, 2018.

[10] 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 Int. Conf. Intell. Robots Syst.*, 2018.

[11] 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.

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

[13] V. Gullapalli, "Skillful control under uncertainty via direct reinforcement learning," *Robot. Autono. Syst.*, vol. 15, no. 4, pp. 237–246, 1995.

[14] J. Peters and S. Schaal, "Reinforcement learning of motor skills with policy gradients," *Neural Networks*, vol. 21, no. 4, pp. 682–697, 2008.

[15] J. Kober, K. Mülling, O. Krömer, C. H. Lampert, B. Schölkopf, and J. Peters, "Movement templates for learning of hitting and batting," *Proceedings - IEEE Int. Conf. Robot. Automat.*, pp. 853–858, 2010.

[16] M. Kalakrishnan, L. Righetti, P. Pastor, and S. Schaal, "Learning force control policies for compliant robotic manipulation," *29th Int. Conf. Machine Learning*, vol. 1, pp. 4639–4644, 2012.

[17] H. Benbrahim and J. A. Franklin, "Biped dynamic walking using reinforcement learning," *Robot. Autono. Syst.*, vol. 22, no. 3-4, pp. 283–302, 1997.

[18] N. Kohl and P. Stone, "Policy gradient reinforcement learning for fast quadrupedal locomotion," *Proceedings - IEEE Int. Conf. Robot. Automat.*, vol. 2004, no. 3, pp. 2619–2624, 2004.

[19] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2323, 1998.

[20] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," pp. 1097–1105, 2012.

[21] Y. Lecun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[22] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with Deep Reinforcement Learning," 2013.

[23] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature 2015 518:7540*, vol. 518, no. 7540, pp. 529–533, 2015.

[24] J. Koutník, G. Cuccu, J. Schmidhuber, and F. Gomez, "Evolving large-scale neural networks for vision-based reinforcement learning," *GECCO 2013 - Proceedings of the 2013 Genetic and Evolutionary Computation Conference*, pp. 1061–1068, 2013.

[25] S. Levine and V. Koltun, "Guided policy search," in *30th Int. Conf. Machine Learning*, ser. ICML'13, no. PART 2. JMLR.org, 2013, pp. 1038–1046.

[26] J. Schulman, S. Levine, P. Moritz, M. Jordan, and P. Abbeel, "Trust region policy optimization," *32nd Int. Conf. Machine Learning*, vol. 3, pp. 1889–1897, 2015.

[27] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017.

[28] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-End Training of Deep Visuomotor Policies," *Journal of Machine Learning Research*, vol. 17, pp. 1–40, 2015.

[29] D. Kalashnikov *et al.*, "QT-Opt: Scalable Deep Reinforcement Learning for Vision-Based Robotic Manipulation," 2018.

[30] 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.

[31] Z. Li, X. Cheng, X. B. Peng, P. Abbeel, S. Levine, G. Berseth, and K. Sreenath, "Reinforcement Learning for Robust Parameterized Locomotion Control of Bipedal Robots," *Proceedings - IEEE Int. Conf. Robot. Automat.*, vol. 2021-May, pp. 2811–2817, 2021.

[32] 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. eabk2822, 2022.

[33] D. F. Hoyt and C. R. Taylor, "Gait and the energetics of locomotion in horses," *Nature*, vol. 292, no. 5820, pp. 239–240, 1981.

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

[35] 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 Robot. Automat. Lett.*, vol. 7, no. 2, pp. 4630–4637, 2022.

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

[37] Z. Fu, A. Kumar, J. Malik, and D. Pathak, "Minimizing Energy Consumption Leads to the Emergence of Gaits in Legged Robots," 2021.

[38] X. B. Peng and M. van de Panne, "Learning Locomotion Skills Using DeepRL: Does the Choice of Action Space Matter?" *Proceedings - SCA 2017: ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, 2016.

[39] G. Fadini, T. Flayols, A. del Prete, N. Mansard, and P. Souères, "Computational design of energy-efficient legged robots: Optimizing for size and actuators," in *IEEE Int. Conf. Robot. Automat.*, 2021.

[40] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," in *ACM International Conference Proceeding Series*, vol. 382. New York, New York, USA: ACM Press, 2009, pp. 1–8.

[41] R. M. French, "Catastrophic forgetting in connectionist networks," *Trends in Cognitive Sciences*, vol. 3, no. 4, pp. 128–135, 1999.

[42] J. Hwangbo, J. Lee, and M. Hutter, "Per-contact iteration method for solving contact dynamics," *IEEE Robot. Automat. Lett.*, vol. 3, no. 2, pp. 895–902, 2018. [Online]. Available: www.raisim.com

[43] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015.

[44] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021.