RLOC: Terrain-Aware Legged Locomotion Using Reinforcement Learning and Optimal Control

Siddhant Gangapurwala, Mathieu Geisert, Romeo Orsolino, Maurice Fallon, Senior Member, IEEE, and Ioannis Havoutsis

Abstract—We present a unified model-based and data-driven approach for quadrupedal planning and control to achieve dynamic locomotion over uneven terrain. We utilize on-board proprioceptive and exteroceptive feedback to map sensory information and desired base velocity commands into footstep plans using a reinforcement learning (RL) policy. This RL policy is trained in simulation over a wide range of procedurally generated terrains. When run online, the system tracks the generated footstep plans using a model-based motion controller. We evaluate the robustness of our method over a wide variety of complex terrains. It exhibits behaviors that prioritize stability over aggressive locomotion. Additionally, we introduce two ancillary RL policies for correcting whole-body motion tracking and recovery control. These policies account for changes in physical parameters and external perturbations. We train and evaluate our framework on a complex quadrupedal system, ANYmal version B, and demonstrate transferability to a larger and heavier robot, ANYmal C, without requiring retraining.

Index Terms—AI-based methods, deep learning in robotics and automation, legged robots, robust/adaptive control of robotic systems.

 NOMENCLATURE

| Notation | Definition |
|----------|------------|
| $W^T W_B$ or $t_B$ | Translation of frame $B$ represented frame $W$. |
| $R_B W_B$ or $R_B$ | Rotation matrix of $B$ represented in $W$. |
| $e_W^W$ | $z$-axis of $W$. |
| $q_j$ | Joint positions. |
| $w V W_B$ or $v_B$ | Linear velocity of $B$ measured and expressed in $W$. |
| $w W B_B$ or $w_B$ | Angular velocity of $B$ measured and expressed in $W$. |
| $q_j$ | Joint velocities. |
| $R_{B_z}$ | $z$ decomposition of $R_B$. |
| $\tau_j$ | Joint torques. |
| $c_F$, $a_f$, $r_F$ | Feet lift-off, touch-down, and clearance positions. |
| $s_r$, $a_r$, $r_R$ | Feet contact state. |
| $\pi_{\theta_s}$, $\pi_{\theta_d}$, $\pi_{\theta_r}$ | Footstep planner state, action, and reward. |
| $S_m$ | Domain adaptive tracker state, action, and reward. |
| $\Phi$, $r_f^d$, $r_f^h$, $r_f$ | Desired velocity command. |
| $\Phi$, $r_f^d$, $r_f^h$, $r_f$ | Robotic terrain elevation represented in frame $H$. |
| $M_H$ | Latent embedding of $M_H$. |
| $Z_{M_H}$ | 2-D convolution operator. |
| $C$ | Footstep planning, domain adaptive tracking, and recovery control policies. |
| $\Phi$ | Stability margin function. |
| $\Phi$ | Terrain cost map operator. |
| $B_{M_H}$ | Elevation map update frame. |

I. INTRODUCTION

LEGGED robot locomotion research has been driven by the goal of developing robust and efficient control solutions with the potential to exhibit a level of agility that matches their biological counterparts. Their advanced mobility is characterized by the ability to form support contacts with the environment at discontinuous locations as required. This makes such systems suitable for traversal over terrains that are inaccessible for wheeled or tracked robots of equivalent size. This maneuverability, however, comes at the expense of increased control complexity. Such systems often need to maintain dynamic stability, select appropriate contact sequences, and actively balance during locomotion.

In this work, we design, train, and evaluate a control framework for legged locomotion over uneven terrain using the ANYmal B [1] and ANYmal C [2] quadrupedal robots. ANYmal version B is a 30 kg rugged robot developed with a focus on high mobility and the ability to perform dynamic locomotion behaviors for rough terrain operations. It features a large workspace that enables a wide range of leg motions and is suitable for a broad domain of applications, for example, inspection tasks on offshore platforms [3] and sewer systems [4].
Fig. 1 shows the ANYmal B robot used for testing our proposed framework on different uneven terrains. We additionally highlight the generality of our control framework by testing our approach with the newer ANYmal, version C [2]—a 50 kg quadrupedal robot with a similar kinematic structure but different dynamic characteristics—and by evaluating its performance without retraining any of our planning and control policies. Experiments performed with ANYmal C are shown in Fig. 2.

The control of such underactuated robotic systems is an active area of research.

Approaches based on whole-body trajectory optimization such as contact-implicit optimization [5] can generate complex multicontact motions but often remain too computationally expensive to be used online. More recent works based on differential dynamic programming (DDP) algorithms [6], [7] enable efficient computation of whole-body motions suitable for online execution. These methods, however, are still under development, and their effects on physical systems and complex environments are yet to be demonstrated.

A rich body of work has focused on using simplified dynamic models which capture only the main dynamics of the system so as to achieve a balance of computational complexity [8]. Such models are also known as templates and are typically lower dimensional than the full (whole-body) model and also linear with respect to the states of the system; the mechanism that relates them back to the whole-body model is instead usually called the anchor. Examples of templates include the linear inverted pendulum (LIP) [9] and the spring-loaded inverted pendulum (SLIP) [10] and their derived variants [11].

For the development of robust model-based controllers, the synthesis of stability criteria based on ground reference points (two-dimensional (2-D) points on a flat ground plane), such as the zero moment point (ZMP) [12] and the instantaneous capture point [13], is also introduced into the control problem. The templates can be seen as a way to constrain the admissible domain of such ground reference points. In this regard, the balancing problem for legged robots can be translated into an inclusion problem that requires the reference point to belong to its admissible region, i.e., the support region, for the robot to be stable. In this way, a measure of robustness, the stability margin, can be defined as the distance between the reference point and the edges of its admissible region.

Often, simplified models and ground reference points are limited to flat terrain and their descriptive accuracy deteriorates when applied to environments with complex geometry. For such multicontact conditions, the centroidal dynamics model [14] is widely used. It represents the system in a compact manner such that it focuses on the inputs (contact wrenches) and the outputs (center of mass (CoM) accelerations) in the task space of the robot discarding the intermediate joint-space physical quantities. In this case, the reference point for evaluating stability can be described by a 6-D wrench. Therefore, for the evaluation of stability, templates that illustrate suitable 6-D admissible regions need to be defined [15]. Despite their low dimensionality, such simplified models are still typically too numerically inefficient to be employed in the robot’s real-time control loop. An alternative approach is to evaluate the stability of the robot directly using the contact forces [16]. This method, however, necessitates introduction of the contact force variables in the locomotion problem, thereby adding to computational delays which result in reduced reactive control.

Recent research on model-predictive control-based strategies has demonstrated perceptive locomotion over uneven terrain [17], [18]. These controllers, however, often employ system model approximations, due to which the potential of the physical robot is not fully exploited.

Moreover, these methods require exhaustive tuning of planning and control parameters, requiring considerable development time.

Alternatively, an entirely different approach has focused on investigating solutions that can perform a direct mapping between sensory information to desired control signals with minimal computational overhead. Model-free data-driven techniques, such as reinforcement learning (RL), have introduced promising substitutes to model-based control methods and achieved highly dynamic and robust locomotion skills on complex legged robot systems, such as ANYmal B [19]–[21], Laikago [22], and Cassie [23]. The major challenge of using RL lies in the fact that training these methods is extremely sample inefficient, especially for high-dimensional problems, such as legged locomotion. When applied to a robot such as ANYmal B, the vast exploration space associated with an RL policy, not only necessitates considerable amount of training samples but also implies that without precise reward function tuning, the RL agent could converge toward an undesired control behavior. As an alternative, a guided policy exploration approach, such as guided policy search [24], which relies on utilizing model-based control solutions in conjunction with data-driven policy optimization, can be employed to direct policy convergence and increase sample efficiency.

The possibility of damaging the robotic system itself and the environment of operation during policy exploration necessitates the use of simulation platforms. Ease of access to physics

---

1 Overview video can be found at: https://youtu.be/GTI-0gl6Hg0; https://youtu.be/GTI-0gl6Hg0
To address the reality gap problem, more accurate models of the training environment can be directly introduced into the simulators. For example, Hwangbo et al. [19] introduced an actuator network to model the dynamics of the physical actuators and further performed ablation studies to justify the necessity for accurate models to achieve simulation-to-real transfer. Moreover, for a training environment, it is very likely that not all the dynamics can be precisely modeled. As a solution, domain randomization [27] techniques have been used to train policies that are robust to unaccountable factors [19], [21].

Most of the above-mentioned RL research focuses on quadrupedal locomotion on flat ground, overlooking the primary purpose of quadrupedal systems—mobility over rough terrain.

Tsounis et al. [28] presented an approach for terrain-aware gait planning and control using RL for the ANYmal B quadruped. Tsounis et al. used a hierarchical approach to plan gaits suitable for navigation to the desired goal position and then performed whole-body control, both using RL. However, different gait planning policies were trained for different terrains while sim-to-real transfer was not addressed. Lee et al. [29] presented an RL-based approach for quadrupedal locomotion over challenging terrains; however, this work only focused on the use of proprioceptive feedback for locomotion without explicitly processing terrain information. Recently, Miki et al. [30] demonstrated robust and perceptive locomotion skills in unstructured environments. Their control framework exhibited impressive terrain-adaptive behavior even with invalid exteroceptive feedback. This work, however, did not demonstrate significant robustness to variations in kinematic and dynamic properties. For transfer to a different robot not introduced during training, their proposed approach is expected to require further retraining.

In this work, we propose a unified RL and optimal control (OC) based terrain-aware legged locomotion framework, as illustrated in Fig. 3. We train a footstep planning RL policy which maps the robot state and terrain information to desired feet positions. In order to track these positions, we extend the dynamic gaits motion controller of Bellicoso et al. [31] to enable foothold tracking over nonflat terrain. This forms the core of our locomotion framework. In order to account for changes in physical parameters and actuation dynamics, we train a domain adaptive tracking policy which introduces corrective feed-forward joint torques for accurate tracking of the trajectories generated by the motion controller. Furthermore, to enable recovery to a stable state upon perturbations or unexpected events, such as foot slippage, we train an RL policy to generate low-level joint position commands such that during unstable motions, the RL policy stabilizes the robot and then the control switches back to the footstep planning and tracking modules. A video overview of our proposed framework is presented in the Appendix (Movie 1).

Our main contribution to this work is to realize a modular framework that combines RL- and OC-based approaches to obtain a system-adaptive control solution for perceptive and dynamic robotic locomotion. The adaptive characteristic further facilitates robustness to variations in dynamic and kinematic properties of the system. The key elements of this work thus include 1) presenting a training approach for obtaining a perceptive footstep planning policy that employs a model-based dynamic motion controller. This setup enables us to generate valid dynamic motion plans while ensuring the obtained policy adapts to and exploits the domain largely described by the characteristics of the motion controller. 2) Obtaining a domain adaptive tracking policy which can be utilized to address system modeling inaccuracies during whole-body motion tracking. This allows us to use a model-based controller on a robot where the modeled characteristics differ from the actual kinematic and dynamic properties of the system. This further enables us to perform dynamics randomization while training the footstep...
the orientation $q_W B \in SO(3)$ is represented using Hamiltonian unit quaternions with a corresponding rotation matrix $R_W B \in SO(3)$. We assume that the $z$-axis of $W e^W$ aligns with the gravity axis. The angular positions of the rotational joints in each of the limbs are described as a vector $q_j \in \mathbb{R}^{n_j}$. For the ANYmal robots, $n_j = 12$. The linear and angular velocities of the base $w.r.t.$ the global frame are written as $W v W B \in \mathbb{R}^3$ and $W \omega W B \in \mathbb{R}^3$, respectively. The generalized coordinates and velocities are, thus, stacked as vectors $q$ and $u$ where

$$\begin{bmatrix}
W r W B \\
q_W B \\
q_j
\end{bmatrix} \in SE(3) \times \mathbb{R}^{n_f}, \quad u = \begin{bmatrix}
W v W B \\
W \omega W B
\end{bmatrix} \in \mathbb{R}^{6+n_j}.$$ (1)

We denote positions of the feet in the global reference frame as $W r W F \in \mathbb{R}^{3 \times n_f}$, where $n_f = 4$ is the number of feet. We additionally define $W r W F_{xy} \in \mathbb{R}^{2 \times n_f}$ as a vector of $x$ and $y$ components of the foot positions.

We introduce a horizontal frame $H$ such that $W r W F = W r W B$ and $R_W H = R_W B$, where $R_W B_j$ is the $z$-decomposition of the base rotation matrix, which can also be expressed as $R_W B = R_W B_j R_W B_j R_W B_j$. The frame $H$ can, thus, be interpreted as having the same translation as the robot base frame $B$ w.r.t. the global inertial reference frame, but rotation in the frame $H$ is only permissible along $e_z^H$, implying that the gravity axis always aligns with the $z$-axis of frame $H$. Additionally, the yaw rotation of frame $H$ also always aligns with the yaw rotation of frame $B$. We then represent the local robocentric terrain elevation in the horizontal frame as $M_H : \mathbb{R}^2 \times \mathbb{R} \to \mathbb{R}^{3 \times 91 \times 91}$, an elevation map corresponding to an area of $1.8 \times 1.8$ m$^2$ and a resolution of $0.02$ m.

The quadrupedal system is actuated using the joint control torques $\tau_j \in \mathbb{R}^{n_j}$. These torques are computed at the actuator level using an impedance control model written as

$$\tau_j = K_p(q_j^* - q_j) + K_d(\dot{q}_j^* - \dot{q}_j) + \tau_{jFF}$$ (2)

where $K_p$ and $K_d$ refer to the position and velocity tracking gains, respectively, $q_j^*$ is the vector representing desired joint positions, $\dot{q}_j^*$ the desired joint velocities, and $\tau_{jFF}$ refers to the feed-forward joint torques.

Note that for concise presentation, we will assume that all of the translations and rotations are measured and expressed in the global reference frame $W$, unless otherwise stated. In this regard, for the rest of this article, the notation $W r W x$ will be shortened to $r_B$, $R_W B$ to $R_B$, and similarly for all the other introduced notations.

### B. Motion Parameterization

The motion of the quadrupedal system is expressed in terms of the CoM and feet motions. In consistency with the definitions introduced in [31], we represent the CoM motion plan in the form of a sequence of quintic splines where the $j$th spline for planning policy without requiring retuning of the motion controller. 3) Demonstrating zero-shot generalization to a domain not explicitly modeled during training. For this, we show transfer of our framework to the physical ANYmal C robot despite having been trained in a simulator with ANYmal B.

Our framework allows for high-level tracking of reference base velocity commands over complex uneven terrain, enabling the robot to be operated either by a user (through a remote control) or a higher level goal planner. We evaluate the performance of RLOC on a wide range of simulated terrains and present the results for the ANYmal B and C quadrupeds. We further transfer RLOC on to the real robots.

## II. PRELIMINARIES

### A. System Model

A general quadrupedal robotic system can be modeled as a floating base $B$ with four actuated limbs attached to it. The robot state can be described $w.r.t.$ a global inertial reference frame $W$, where the base position is expressed as $W r W B \in \mathbb{R}^3$ and

![Diagram](image_url)
the $x$ component is represented as
\[
x(t) = \alpha_1 x_1 t^5 + \alpha_2 x_4 t^4 + \alpha_3 x_3 t^3 + \alpha_4 x_2 t^2 + \alpha_0 x_0 \\
= \begin{bmatrix} t^5 & t^4 & t^3 & t^2 & t & 1 \end{bmatrix} \\
\cdot \begin{bmatrix} \alpha_1 & \alpha_2 & \alpha_3 & \alpha_4 & \alpha_0 \end{bmatrix} \\
= \eta(t) \alpha_j.
\]

The spline $j$ corresponds to a duration of $\Delta t_j$ such that $t \in [\tilde{t}, \tilde{t} + \Delta t_j]$. Here $\tilde{t}$ refers to the total elapsed time of the motion described by $j - 1$ splines. We, therefore, express the CoM position $p_{\text{CoM}} \in \mathbb{R}^3$ as $p_{\text{CoM}} = K(t) \alpha_j$, where
\[
K(t) = \begin{bmatrix} \eta^T(t) & 0 \\
0 & \eta^T(t) \\
0 & \eta^T(t) \end{bmatrix}
\]
and $\alpha_j = [\alpha_j^T \alpha_j^T \alpha_j^T]^T$. The CoM velocity is thus, represented as $p_{\text{CoM}}' = \dot{K}(t) \alpha_j$ and the acceleration as $p_{\text{CoM}}'' = \ddot{K}(t) \alpha_j$.

We parameterize the feet motions based on their contact state given by the vector $c_F \in \{0, 0.5, 1\}^4$ where an open-contact is represented by 0 and close-contact (with zero foot velocity) by 1. Depending on the terrain, there remains a possibility of foot slippage, i.e., a close contact with nonzero foot velocity. This contact-uncertainty state is, therefore, denoted by 0.5. For the purpose of motion planning, we only consider the open (0) and close (1) contact states. The contact-uncertainty state is instead utilized for adapting the joint PD control gains during motion tracking. This is detailed in Section III-B.

We define the swing phase $\Phi_{s,i}$ for foot $i \in \{0, 1, 2, 3\}$ to correspond to the foot motion where $c_{F,i} = 0$. Conversely, the stance phase $\Phi_{c,i}$ corresponds to the motion where $c_{F,i} = 1$. Both the swing and stance phases can be described by the tuple
\[
\Phi = (r_F^i, r_p^i, r_h^i, c_F, t_F^i, t_p^i, t_h^i, \psi_F, \psi_p, \psi_h)
\]
where $r_F^i$, $r_p^i$, and $r_h^i$ refer to the vectors describing the feet lift-off, touch-down, and clearance positions, respectively. The foot clearance position represents the maximum $r_{F,i}$ during the given phase. For $\Phi_{s,i}$, $t_p^i$ is the time elapsed since the previous lift-off and $t_h^i$ is the time until touch-down. For $\Phi_{c,i}$, $t_p^i$ refers to the time until lift-off and $t_h^i$ is the time elapsed since the previous touch-down. $\psi_F$, $\psi_p$, and $\psi_h$ refer to the feet lift-off and touch-down velocities, respectively. Additionally, for $\Phi_{c,i}$, we consider $r_p^i = r_h^i = r_F^i$.

C. Reinforcement Learning

The problem of sequential decision making can be formulated in the framework of a discrete time Markov decision process (MDP) [32]. An MDP is defined by the tuple $(S, A, R, P, \mu)$, where $S$ represents a set of states, $A$ a set of actions, $R : S \times A \times S \rightarrow \mathbb{R}$ the reward function, $P : S \times A \times S \rightarrow [0, 1]$ the state transition probability, and $\mu$ the initial state distribution. In the context of deep RL, as utilized in our work, an RL agent described in this MDP framework interacts with an environment based on a policy $\pi : S \rightarrow \mathcal{P}(A)$ which is modeled as a neural network.

This policy can be defined as a function mapping states to probability distributions over actions such that $\pi(\cdot | s)$ denotes the probability of selecting action $a$ in state $s$. The objective of the RL agent is to then obtain a policy $\pi_\theta$ for parameters $\theta$ in order to maximize the expected cumulative discounted return
\[
J(\pi) = \mathbb{E}_{T \sim \pi_\theta} \left[ \sum_{t=0}^{\infty} \gamma^t R(s_t, a_t, s_{t+1}) \right]
\]
where $\gamma \in [0, 1)$ is the discount factor and $T$ denotes a trajectory dependent on $\pi_\theta$.

III. METHODOLOGY

We present a modular terrain-aware quadrupedal planning and control framework for locomotion over uneven terrain. We propose a footstep planner (FP), which utilizes the proprioceptive and exteroceptive information to generate the desired feet touch-down positions $r_F^d$ that are tracked during the immediate swing phases. The FP is sequentially evaluated after each of the previous four swing phases are executed, completing a gait stride. The desired footholds are, thus, updated after each gait stride at the instance when each of the feet is in stance phase, $c_F = 1$ for which $\tilde{t}_F \in \mathbb{R}^4$ is a vector of ones. The dynamic gaits motion controller, which serves as a motion planner and whole-body controller, is then employed to generate and track the desired CoM and feet motion plans. The whole-body tracking of motion plans occurs at 400 Hz, while the motion planning optimization is performed in parallel to the control thread and a new optimization problem is run every time a solution is found.

The domain adaptive tracker (DAT), also executed at 400 Hz, is trained to introduce additive deviations to the feed-forward joint torques generated by the motion controller $r_{j,F}$. These additive deviations $\delta r_{j,F}$ are included in the impedance controller, shown in (2), and account for discrepancies in the modeled and actual system dynamics. Finally, the recovery controller (RC), running at 400 Hz, is activated to perform joint position control when a robot state is encountered such that the motion controller becomes unstable and is unable to execute a state recovery behavior.

We employ this modular control architecture for the following reasons.

1) The individual trained modules can be precisely tuned to adapt to different tasks. For example, transferring the framework for use on a quadruped with a significantly larger mass would only require adapting either the motion controller or the DAT. This, in fact, also allows us to perform a zero-shot transfer from the simulated ANYmal B training domain to the physical ANYmal C.

2) In our preliminary experiments, we observed that training a terrain-aware locomotion policy, which outputs desired joint position commands, required a comprehensive tuning of the RL environment parameters, without which the network often failed to learn, resulting in behavior where the robot would not respond to velocity commands, instead
would stand in place. This observation significantly contributed toward our modular control architecture design. A modular framework enabled us to perform easier tuning of individual components. The contributions of each of the modules are detailed in Section V.

3) Distributing the planning and control tasks between the RL FP and the model-based motion controller enabled us to obtain an FP, which was able to generalize to a wide range of terrains in less than 32 h of total training. In contrast, the gait planner of DeepGait [28] required 82 h of training time for each of the terrains. Moreover, the gait controller of DeepGait required a further 116 h of training time for each of the uneven terrains and an additional 58 h of training for locomotion on flat-ground, totaling the training time to approximately 454 h for locomotion over three different terrains: flat-world, temple-ascent, and random-stairs (see DeepGait [28]).

An overview of the control framework is presented in Fig. 3. This section details each of the modules. We also describe the MDPs for the FP, DAT, and the RC in this section. The detailed RL training setups are presented in Section IV.

A. Elevation Map Encoding

We extract robocentric terrain elevation \( M_H \) using the elevation mapping framework presented in [33] and embed it to obtain a latent representation \( Z_{M_H} \in \mathbb{R}^{96} \). This representation is generated by an encoding network \( E_{M_H} \), which is trained using a denoising convolutional autoencoding strategy. Our motivation for performing this embedding (including the encoding approach) is based on the following reasons.

1) Three of the modules constituting our RLOC framework utilize exteroceptive terrain information. Preprocessing the elevation map to extract terrain features eliminates repeated processing of the elevation map by each of the modules limiting further computational delays.

2) The denoising property of \( E_{M_H} \) is relevant to perform a sim-to-real transfer. The elevation map extracted on the physical system is often noisy and contains occlusions that cannot be precisely modeled. An example of this is shown in Fig. 4. This has further been studied in [30]. A nearest-neighbors interpolation technique is employed to obtain a filtered elevation map which often results in distortions (e.g., the edges of the steps are blurred and shifted). The denoising autoencoding strategy enables extraction of the most relevant terrain features, such as edges and slopes even with distorted terrain information.

3) A pretrained terrain feature extractor implies that during training of RL policies, the RL agent does not need to learn the parameters for the terrain feature extraction block, thereby making the training faster while also adapting to a pretrained encoder, which can be shared with multiple modules.

4) Even though different strategies, such as fast Fourier transform, can be employed to compress the elevation map, our work focuses on utilizing deep learning strategies for feature extraction. This enables us to adapt the previously trained network to a diverse class of terrains by retraining the encoder. Additionally, strategies, such as transfer learning, can be utilized to reduce training complexity. The feature extractor can also be retrained along with an RL policy to adapt to different tasks not introduced during prior training.

The architecture of the denoising convolutional autoencoder is presented in Fig. 5. Note that while extracting the elevation map, we clip \( M_H \) in the range \([-2, 2]\) m. For network training and evaluation stability, we further normalize \( M_H \) by performing a scalar division by 2 to obtain \( \hat{M}_H \) with elements in the range \([-1, 1]\).

As part of training, our objective is then to learn the weights of the convolutional encoder \( E_{M_H} \) such that

\[
Z_{M_H} = E_{M_H}(M_H^0 + N_M(M_H^0))
\]

where \( N_M \) represents the artificial noise introduced in \( M_H^0 \), and

\[
\hat{M}_H^n = D_{M_H}(Z_{M_H})
\]

where \( D_{M_H} \) is the decoder network which maps the latent embedding of \( M_H^n \) to its approximate reconstruction \( \hat{M}_H^n \). Both \( E_{M_H} \) and \( D_{M_H} \) are trained together using supervised learning to minimize the reconstruction loss

\[
L_{M_H} = (M_H^0 - \hat{M}_H^n)^2.
\]

The artificial noise \( N_M : \mathbb{R}^{91 \times 91} \rightarrow \mathbb{R}^{91 \times 91} \) is expressed as

\[
N_M(M) = n_A N_A \times 91(0, 1) + (I(M, p_T) - M) + n_C (C(M, k_0) - M)
\]

where \( M \in \mathbb{R}^{91 \times 91} \) is a matrix similar to \( M_H^n \), \( N_A \) is the additive Gaussian noise, \( I \) is the impulse noise, also known as salt-and-pepper noise, \( p_T \sim \mathcal{U}(0, 0.05) \) is the probability of introduced impulses sampled uniformly in the range \([0, 0.05]\), \( C \) is the
2-D convolution function, and $k_b \in \mathbb{R}^{3 \times 3}$ is the convolutional blurring kernel written as

$$k_b = \frac{1}{9} \times \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}.$$  \hspace{1cm} (11)

$n_N \sim \mathcal{U}(0,0.05)$ and $n_C \sim \mathcal{U}(0,1)$ are the coefficients for weighting the noise functions.

To train the denoising convolutional autoencoder, we use our terrain generator tool, described in Section IV-A1, to generate $20 \times 20$ m$^2$ terrains, where each terrain is represented as $T \in \mathbb{R}^{1001 \times 1001}$. These terrains contain objects such as stairs, wave, bricks, planks, and unstructured ground. We slice a $91 \times 91$ matrix from $T$ at randomly selected positions and further use data-augmentation techniques such as rotation, mirroring, and modifying contrast during training of the autoencoder online. This augmentation is done in parallel to the main training session in a separate thread. Fig. 6 shows examples of regeneration of noisy terrain inputs using the trained autoencoder.

B. Motion Controller

Rudin et al. [34] propose an architecture that allows for direct mapping of proprioceptive and exteroceptive state information to desired joint positions. This, however, often requires exhaustive reward tuning to achieve a desired locomotion behavior [19], [21]. As an alternative, Miki et al. [30] utilize central pattern generators to achieve an underlying oscillatory behavior for each of the quadrupedal limbs. This architecture enables online adaptation of the locomotion gait while also making the RL training less demanding [35]. In this work, we are driven by the motivation that the proposed framework should be robust to variations in system kinematics and dynamics. For this, we utilize a motion controller to generate and track base and feet motion plans for traversal over uneven terrain. The desired feet positions are instead generated by the RL FP described in Section III-C. This eliminates the need for the RL agent to perform motion control, thereby simplifying training. Additionally, the model-based motion control architecture ensures that safety-critical constraints, such as joint kinematic limits and peak joint torques, are still enforced by the low-level controller. The system-adaptive property is then induced by the DAT presented in Section III-D.

In this work, we extend the dynamic gaits motion controller of Bellicoso et al. [31] to generate CoM and feet motion plans for traversal over uneven terrain. The CoM motion plan is generated by solving a nonlinear optimization problem to track the velocity command $e^*$ for a time horizon $T_H$. This time horizon corresponds to the period of the active locomotion gait. We introduce a control frame $C$ such that $r_C = r_{B_z}$, $R_{C_z} = R_{B_z}$, and $e^C_z$ aligns with the normal of the estimated terrain plane $e^C_T$. The velocity command is, therefore, expressed in the control frame as

$$e^* = [v^C_x e^C_y \omega^C_z]^T$$  \hspace{1cm} (12)

where $v^C_x$ and $v^C_y$ are the desired heading and lateral velocities expressed along $e^C_x$ and $e^C_y$, respectively, and $\omega^C_z$ is the desired yaw rate around the vertical axis $e^C_z$.

The motion controller employs a quadratic programming (QP) based foothold planning module to generate the desired feet touch down positions $r^d_{F_z}$. This is replaced by the RL FP in our proposed framework. The vertical component of the desired touch down positions $r^d_{F_z}$ is directly extracted from the terrain plane estimate for each swing phase. The contact scheduler module of the motion controller manages the feet contact events for a full stride of an active gait. It generates the lift-off $t^l_{F_z}$ and touch-down $t^d_{F_z}$ timings for the feet swing and stance phases. The $\Phi_o$ and $\Phi_e$, thus, obtained are used to generate feet motion plans to track $r^d_{F_z}$.
A detailed description of the modifications introduced in the original motion controller is provided in the Appendix.

The generated reference motion plans are tracked using a whole-body controller [36] based on a hierarchical QP optimization framework to optimize for the generalized accelerations $\ddot{q}$ and feet contact forces to generate the feed-forward torques $\tau_{FF}$. The impedance control torques are then computed by extracting the desired joint positions $q^*_j$ and joint velocities $\dot{q}^*_j$ from the feet motion plans using inverse kinematics.

The work of Jenelten et al. [37] on locomotion over slippery surfaces proposes the use of impedance gains depending on the contact-uncertainty state $c_{F,i}$. The high gains used for joints in the leg containing the foot $i$ prevent further leg slippage due to increased stiffness. For detailed description, we defer the reader to the original work. We use this approach of adaptive impedance for whole-body tracking of the reference motion plans in RLOC.

### C. Footstep Planner

Approximating the perceptive footstep planning policy using a neural network implies that we can directly map the exteroceptive and proprioceptive robot state information to desired feet positions. This rids us of the need for computationally exhaustive task of combinatorial search among a large space of feasible footstep plans. Additionally, as presented in Section V-A, training the FP using deep RL enables natural emergence of planning behavior learned through interactions with the environment of operation. This offers better performance for traversal over unstructured terrains compared to a baseline sampling-based perceptive FP.

Computing new set of desired feet positions after each gait stride, specifically after each of the previous four feet swing phases have been executed, allows us to formulate the problem of footstep planning as a sequential MDP. In particular, we represent the problem as an infinite-state space and infinite-action space MDP.

1) State and Action: We define the MDP state as a tuple $s_f := (s_R, s_v, s_j, s_c, s_M)$ and the action $a_f := (\tau_{FF}, \gamma_H)$. The state $s_f \in \mathbb{R}^{204}$ comprises terms representing robot base orientation $s_R \in \mathbb{R}^3$, base linear and angular velocities $s_v \in \mathbb{R}^6$, sparse representation of history of joint positions and velocities $s_j \in \mathbb{R}^{96}$, desired base velocity $s_c \in \mathbb{R}^3$, and an embedded representation of the robonectric elevation map $s_M \in \mathbb{R}^{90}$. The exact definitions of the FP state terms are provided in Table I. The action $a_f \in \mathbb{R}^8$ comprises the desired foot locations $\gamma_H$ for each of the four feet represented in the horizontal frame. The z components of the desired feet positions $\gamma_H$ are directly extracted from $M_H$. Therefore, the desired feet positions $\gamma_H$ measured and expressed in horizontal frame can be transformed into the world frame representation using the transformation

$$r_{FP,i} = (R_H)(\gamma_H) + r_B$$

for foot $i$. The feet positions generated by the FP $r_{FP}$ are then forwarded to the dynamic gaits motion controller.

Note that we introduce the history of joint states in the state space to infer the active locomotion gait. The different gaits result in different stability margin [38], [39], a metric introduced in the reward function to assess the dynamic stability of the robot. This implies that the desired feet positions generated by the FP depend on the gait in use. The state space is, thus, required to contain information relating to the robot locomotion gait.

2) Reward Function: The reward function in the MDP formulation encourages stable dynamic locomotion over uneven terrain in order to track a desired base velocity command while minimizing energy consumption. During execution of the sequential footstep planning MDP in an episodic framework, the planning state transition $P_{f,n} := (s_{f,n}, a_{f,n}, s_{f,n+1})$ is only observed after the motion plans for each of the feet are executed. The tracking of these motion plans is instead performed at each control step executed at 400 Hz. Therefore, each $P_f$ in the footstep planning MDP comprises several control steps. The control behavior observed within these state transitions is still relevant to assess the performance of the FP, especially pertaining to velocity tracking and foot slippage. In this regard, our reward function includes a running reward term $r_{FP} \in \mathbb{R}$ accumulated over the control horizon for each $P_f$, and a final reward term $r_{FP} \in \mathbb{R}$ computed after a state transition is observed. The reward function can then be expressed as

$$r_F = r_{FP} + \frac{1}{(n+1)\mu + (n_t + 1)} \sum_{t=n_t+1}^{(n+1)\mu} r_{FP,t}$$

where $n_t$ corresponds to the control step at which state $s_{f,n}$ is observed. The duration between $t = n_t + 1$ and $t = (n + 1)\mu$ is not fixed and depends upon the touch-down and lift-off timings. We express the running reward at control step $t$ as

$$r_{FP,t} = 8r_{vel} - 0.016d_f r_{tau} - 2.5dr_{slip} + 4r_{mnt}$$

comprising the velocity tracking reward $r_{vel}$, control joint torque penalty $r_{tau}$, feet slippage penalty $r_{slip}$, and the stability margin reward $r_{mnt}$. The weighting factor $d_f \in [0,1]$ corresponds to the training curriculum and is increased as training progresses to fine tune the behavior of the RL agent. We update the curriculum factor after each policy iteration by the rule $d_{fp,i} \leftarrow d_{fp,i} \cdot 0.998$, where $k$ refers to the policy iterations and $d_{fp} = 0.02$. The final
2916
IEEE TRANSACTIONS ON ROBOTICS, VOL. 38, NO. 5, OCTOBER 2022

Fig. 7. Overview of the approach used to train the RL policies as part of our framework. We introduce the denoising convolutional encoder to embed the elevation map into a lower dimensional representation. This, along with the robot states and desired velocity commands, becomes the observation space for each of the RL policies. Since we train the RL policies in simulation, we use an actuator network to approximately model the actuation dynamics of the real robot to make sim-to-real transfer feasible.

| TABLE II |
| REWARD TERM DEFINITIONS FOR THE FP MDP |

| FP Rewards | |
| --- | --- |
| $r_v = K \left( \| e^* - e \|^2 \right)$ | |
| $r_e = \| \tau_j \|^2$ | |
| $r_\mu = \sum \| \tau_{\mu,j} \|$, $\forall i \in \{0, 1, 2, 3\}$ | |
| $r_m = S_m(q_i, \dot{u}_F, \dot{s}_F, \dot{F}_B, \dot{T}_B, \dot{T}_p)$ | |
| $r_a = \| K_P \dot{F}_H - H \dot{r}_{\mu,F} \|^2$ | |
| $r_n = \sum S_n(\tau_{n,j})$, $\forall i$ | |
| $r_h = K(\| \dot{r}_{\mu,F} \|^2)$ | |

Note: Here $K$ refers to the logistic Kernel defined as $K(x) := \left( e^x + 2 + e^{-x} \right)^{-1}$. $S_m$ is the function mapping robot state information to the robot stability margin. $N_F$ represents the feet contact normals, $F_B$ represents the external force on base, $\tau_B$ is the external torque applied on base, $\tau_{\mu,F}$ is the terrain friction coefficient, and $C_M$ maps the foot position to a cost describing the distance of the foot from the terrain edges. The functions $S_m$ and $C_M$ are detailed in the Appendix. $\dot{r}_{\mu,F,z}$ is the $z$ component of the nominal feet positions.

The reward is represented as

$$r_F^f = -2d_f \tau_n - 1d_f \tau_e - 2.5d_f \tau_m + 4r_m + 2r_h$$ (17)

comprising the penalty for deviation from nominal footholds $r_n$, reward for distance from terrain edges $r_e$, feet slippage penalty $r_\mu$, the robot stability margin reward $r_m$ based on [39], and the feet height reward relating to the closeness to the nominal feet height $r_h$. The definitions for the reward terms are presented in Table II.

3) Episode Termination Criteria: We execute the footstep planning MDP in an episodic framework employing the following termination criteria while training the FP:

1) angle between the terrain normal and base vertical axis exceeds a threshold, $|\cos^{-1}(e_B^T e_C^T)| > \pi/4$;

2) any of the robot links except for the feet collide with the terrain;

3) robot self-collision.

Upon termination, the robot state is reset and a new episode is executed.

4) Policy Parameterization: We parameterize the footstep planning policy as a bounded Gaussian distribution $\pi_{\theta_f}(a|s_f) = \text{tanh}(N(\mu_{\theta_f}(s_f), \sigma_{\theta_f}(s_f)))$, where both the mean $\mu_{\theta_f}(s_f)$ and the log standard deviations $\log(\sigma_{\theta_f}(s_f))$ are state-dependent outputs of a neural network. The footstep planning network architecture representing the function mapping state to the action mean is shown in Fig. 7. The log standard deviation is extracted using a linear mapping of the latent vector obtained as an output of the last hidden layer. This state-dependent stochasticity is used to encourage exploration during training. We utilize a custom implementation of soft actor critic (SAC) [40], heavily based on RLlib [41], to train the footstep planning policy $\pi_{\theta_f}$ by performing entropy-regularized RL. For a slow environment setup of footstep planning, which includes the prediction of desired footholds and further includes tracking of these feet positions using the motion controller, SAC offers better sample efficiency in comparison to the widely used on-policy alternatives, trust region policy optimization [42], and proximal policy optimization (PPO) [43], and has further been demonstrated to perform efficiently on quadrupedal locomotion tasks [44]. The training is detailed in Section IV-A.

D. Domain Adaptive Tracker

The model-based characteristics of the dynamic gaits motion controller implies that in order to exhibit robust motion tracking, significant tuning of the controller parameters is required. This is especially relevant for computing $\tau_{j,FP}$ [see (2)], which is written as

$$\tau_{j,FP} = M_j \ddot{u}_r + h_j - J^T_{\lambda,j} \lambda^*$$ (18)
where $M_j$ represents the mass matrix relative to the joints, $h_j$ comprises Coriolis, centrifugal, and gravity terms, $J_{sj}$ corresponds to the support Jacobian, $\dot{u}^*$ is the desired generalized accelerations, and $\lambda^*$ represent the desired contact forces. This is detailed in [36]. Executing such a controller on a system where the robot dynamics differ from the modeled dynamics will result in failure to perform motion plan tracking. In this regard, we introduce a DAT to generate corrective joint torques $\delta \tau_j$ for impedance control, which can then be rewritten as

$$\tau_j^c = K_p(\dot{q}_j^* - \dot{q}_j) + K_d(\ddot{q}_j^* - \ddot{q}_j) + \tau_{jF} + \delta \tau_j.$$  \hspace{1cm} (19)

This reformulation offers us the ability to deploy the motion controller on systems with different dynamics without the need for retuning the control parameters. The experimental evaluation presented in Section V-B includes ablation studies, which support this statement. This is further illustrated in Figs. 10 and 12. Our motivation for introducing the DAT is, therefore, twofold.

1) When deployed on a physical system, RLOC can be made robust to system disturbances, encouraging accurate motion plan tracking even with change in dynamic properties, such as added mass, without the need for explicit modeling of system inaccuracies.

2) Introducing the DAT in an RL environment setup employing a model-based motion controller allows us to perform dynamics randomization without retuning the controller. This enables us to utilize the DAT for training the footstep planning policy with a wide domain of kinematic and dynamics properties eventually letting us perform a zero-shot transfer of the FP trained with ANYmal B to ANYmal C.

1) State and Action: We formulate the problem of domain adaptive tracking in the framework of MDP such that given the robot state and the history of tracking errors, the DAT implicitly models the deviation in system dynamics from the modeled dynamics to generate corrective joint torques at each control step. We define the DAT state as

$$s_d := \langle s_r, s_\delta r, s_R, s_{RV}, s_{V}, s_{\delta V}, s_j, s_F, S_M, s_h \rangle$$ \hspace{1cm} (20)

where $s_d \in \mathbb{R}^{330}$ comprises vector representing the history of robot base position $s_r \in \mathbb{R}^6$, history of base position tracking error $s_{\delta r} \in \mathbb{R}^6$, history of robot orientation $s_R \in \mathbb{R}^9$, history
of desired orientation $s_r \in \mathbb{R}^3$, history of robot velocities $s_v \in \mathbb{R}^{18}$, history of velocity tracking errors $s_{\Delta v} \in \mathbb{R}^{18}$, history of joint state $s_j \in \mathbb{R}^{72}$, history of desired feet positions and velocities $s_f \in \mathbb{R}^{72}$, previous corrective torques $s_c \in \mathbb{R}^{12}$, embedded elevation map $s_M \in \mathbb{R}^{96}$, robot base position relative to the elevation map update frame $s_b \in \mathbb{R}^3$, and robot orientation in the elevation map update frame $s_o \in \mathbb{R}^6$. It is important to note that unlike in the case of footstep planning where each call to the policy occurs after each gait stride, the domain adaptive tracking policy is executed at 400 Hz. Since the perception sensors on the robot are significantly slower, the elevation map on the physical robot cannot be updated at 400 Hz. In order to account for the asynchronous updates of the exteroceptive and proprioceptive feedback, we store the reference frame of the robot’s base during each update of the elevation map and include the current state of the robot in this frame in the state tuple. The elevation map update frame is represented as $B_{M, t}$. The definitions of the state terms are provided in Table III.

The DAT action can be defined as $a_d := (\delta \tau_j)$, where $\delta \tau_j \in \mathbb{R}^{12}$ is the corrective joint torque introduced in the impedance controller, as shown in (19).

2) Reward Function: Domain adaptive tracking can be expressed as an imitation learning problem where the DAT policy tracks the reference motion plan by generating corrective feed-forward joint torques. The reward function is then expressed in terms of tracking errors as

$$r_D = r_b + 0.125 r_v + 1.5 r_j + 0.4 r_j$$

where $r_b$ promotes minimizing base position tracking errors, $r_v$ encourages base velocity tracking, $r_j$ relates to joint position tracking, and $r_j$ is the joint velocity tracking reward. The reward definitions are given in Table IV.

3) Episode Termination Criteria: The termination criteria include the following:

1) any of the robot links except for the feet collide with the terrain;
2) robot self-collision.

4) Policy Parameterization: Unlike in the case of footstep planning which utilizes a stochastic policy, we parameterize the DAT policy in the form of a discrete probability distribution, wherein for a particular action $a_{d, t} \in \mathcal{A}$, the distribution $\pi_{\theta_d}(a_{d, t}) := \delta_{a_{d, t}}$ $\forall a_{d, t} \in \mathcal{A}$, where $\delta$ refers to Kronecker delta. Since the probability is nonzero for only a particular action in a given state, we can represent the policy as a deterministic mapping between a state to action. We, therefore, rewrite $\pi_{\theta_d}(s) : \mathcal{S} \to \mathcal{A}$. We model the policy using a neural network. The architecture is presented in Fig. 7. Additionally, we clip

| Table III: State Term Definitions for the Domain Adaptive Tracking MDP |
|------------------|------------------|
| **DAT State**    | $s_t = [\alpha r_{b,t}, \alpha r_{f,t}]^T$ |
| $s_r = (\alpha r_{b,t} - \alpha r_{f,t})^T$ |
| $s_c = (\alpha r_{b,t} - \alpha r_{f,t})^T$ |
| $s_j = [\alpha r_{b,t}^j, \alpha r_{f,t}^j]^T$ |
| $s_f = (\alpha r_{b,t}^j - \alpha r_{f,t}^j)^T$ |
| $s_b = (\alpha r_{b,t} - \alpha r_{f,t})^T$ |
| $s_o = (\alpha r_{b,t} - \alpha r_{f,t})^T$ |

**Note:** Here, subscript notation $t$ on a frame indicates the state of the frame at control step $t$, for example, $B_{M, t-4}$ represents the state of frame $E$ at control step $t-4$. If no subscript exists, the frame is considered to be in the current state $t$. $B_{M, t}$ is the desired base position at control step $t$ expressed in the current measured base frame, $b_{\Delta v}$ is the desired orientation, $v_{\Delta v}$ is the desired feet positions, $v_{\Delta v}$ is the desired feet velocity, $v_{\Delta v}$ is desired linear velocity, and $\omega_{\Delta v}$ is the desired angular velocity. The desired robot state at control step $t$ is extracted from the motion plans generated by dynamic gaits.

| Table IV: Reward Term Definitions for the Domain Adaptive Tracking MDP |
|------------------|------------------|
| **DAT Rewards**  | $r_b = K(10||\alpha r_{b,t}^2||)$ |
| $r_v = K(6||\alpha r_{b,t} - \alpha r_{f,t}||^2)$ |
| $r_j = K(20||q_j - q_{j,t}||^2)$ |
| $r_j = K(8||\alpha r_{b,t}^j - \alpha r_{f,t}^j||^2)$ |

Fig. 11. Velocity tracking error observed for the domain adaptive tracking policy when running the motion controller on ANYmal C with the description parameters of ANYmal B.

Fig. 12. Absolute mean corrective torques generated by the DAT to minimize motion plan tracking errors for ANYmal B base mass (left) and link (right) scaling. The color bands represent one standard deviation.
the action in the range $[-40, 40]$ Nm. We use the twin delayed deep deterministic policy gradient (TD3) [45] algorithm to train our domain adaptive tracking policy. Exploration is performed using Gaussian noise regularization as opposed to exploration through Ornstein–Uhlenbeck process. For this, we used the RLlib TD3 implementation. Note that for the FP, the stochasticity of the footstep planning policy was leveraged to perform exploration during training. In the case of the deterministic DAT policy, however, we perform exploration by introducing noise during execution of an action. We detail upon the training in Section IV-B.

E. Recovery Controller

To address system disturbances and external perturbations, the motion controller additionally employs an inverted pendulum model-based stabilizing cost term in its foothold optimizer. The cost, $w_i c^* - c_{hip}$, for hip $i$ with height $h$ and acceleration due to gravity $g$, encourages the foothold optimizer to minimize the error between the desired velocity $c^*$ and the hip velocity $c_{hip}$. This results in generation of a desired foothold in the direction of the perturbation. Depending on the weighting $w_i$, the foothold optimizer either prioritizes reference velocity tracking or stability. This, however, relates to the motion controller adapting to the external disturbances as opposed to resisting them. Therefore, for perturbations of large magnitude, adapting to the disturbances by executing feet swing motions to track footholds in the direction of the perturbations may not be responsive enough, especially, considering the joint velocity limits utilized in the optimization problem. We, therefore, introduce the RC for the following reasons.

1) The motion controller executes motion plans to track the desired feet positions generated by the RL FP. These feet positions are only updated after each gait stride. With system disturbances, the CoM position may drift while the desired feet positions are still not updated. This would result in the ZMP drifting significantly from the support region eventually resulting in failure if no recovery motion is executed.

2) Perturbations of large magnitude cannot be fully addressed by adapting the motion plans. The ability to resist these disturbances allows us to better address such perturbations and also reduce the velocity command tracking errors while only adapting to the disturbances when the perturbations are extremely large.

In this regard, the RC, operating at 400 Hz, is activated when any of the empirically tuned criteria relating to the robot state, described by $q$ and $u$, exceed the limits. These are provided in the Appendix. RC is deactivated and the control switches back to the motion controller when each of the robot state criteria lies within its deactivation limit (also provided in the Appendix).

An evaluation of the RC’s robustness to external perturbations is provided in Section V-C. As shown in Fig. 13, the RC is able to resist lateral forces of more than twice the magnitude compared to the dynamic gaits motion controller.

We express the recovery control problem in the framework of MDP directly building upon our prior work [21].

![Fig. 13. Bivariate KDE representing the success probability of the control policies to stabilize upon external perturbations along the frontal (x) and the lateral (y) axes. Note that for (e), (f) the x-axis scale of the plots is different. (a) RL—ANYmal B. (b) RL RC—ANYmal B. (c) Model-based controller only. (d) Model-based controller only. (e) RL RC—ANYmal C. (f) RL RC—ANYmal C.](image)
term \( r_\delta = \| \max(|\dot{q}_j| - 75, 0) \|^2 \), feet acceleration reward term \( r_f = \| \dot{r}_f \|^2 \), and \( d_r \) is the reward scaling term increased as training progresses, updated by the rule \( d_{r_k+1} = d_{r_k}^0 \cdot 0.9 \), where \( k \) refers to the policy iterations and \( d_{r_0} = 0.3 \). The rest of the reward term definitions are consistent with FP.

3) Episode Termination Criteria: We utilize the same termination criteria as employed for DAT.

4) Policy Parameterization: We parameterize the recovery control policy as a Gaussian distribution \( \pi_\theta(a|s_r, r) = \mathcal{N}(\mu_\theta(s_r), \sigma_\theta). \) The mean \( \mu_\theta(s_r) \) is generated using a neural network as a function of the state, while the standard deviation \( \sigma_\theta \), is independent of the state and is utilized during training for exploration. The recovery control network architecture representing the function mapping state to the action mean is shown in Fig. 7. We use the RLlib implementation of PPO [43] to train the recovery control policy. The training is detailed in Section IV-C.

### IV. Training and Evaluation

This section details upon the training, testing, and deployment setups for each of the RL policies. We trained these policies using the RaiSim [46] physics simulator for its extremely fast contact dynamics computation. Moreover, RaiSim supports online reloading of heightmaps enabling the user to simulate terrains making it suitable for training terrain-aware RL policies. We defer the reader to our previous work [21] for a detailed analysis on the preference for RaiSim over other widely accessible simulation platforms. Fig. 7 represents an overview of the training approach.

#### A. Footstep Planner

1) Terrain Generation: Similar to [29], we introduce a large set of terrains, 10 k in our case where each terrain is represented as \( T \in \mathbb{R}^{1001 \times 1001} \) into our simulation environment to obtain robust policies that can generalize to a wide domain of terrains. Our terrain generation procedure is similar to the one introduced in [47], and is further detailed in the Appendix.

2) Actuation Dynamics: As introduced in [19], we train an actuator network to model the actuation dynamics of the physical system to reduce the reality gap by emulating the impedance controller response as observed on the real robot. However, unlike in the case of [19] and [21] where the actuators were modeled only for joint position targets with a predefined set of PD gains given by

\[
\tau_j = j_G(q^*_j, q_j, \dot{q}_j)
\]

in this work, we trained the actuator network using supervised learning to obtain the effective actuation torques

\[
\tau_j = j_G(q^*_j, q_j, \dot{q}_j, \dot{q}_j, \tau_{j,FF}, K_p, K_d).
\]

Since the motion controller utilizes different sets of PD gains for joints of legs with foot in close, open, and contact-uncertainty states, we further include the PD gains as an input to the approximated actuator model. Note that, even for the same contact states, the actuators located on the joints of the same limb can have different PD gains. The actuator network, thus, maps the input \( j_s \in \mathbb{R}^{21} \) to an approximation of the joint torque \( \tau_j \in \mathbb{R} \) observed on the physical system. The network architecture and training details are provided in the Appendix.

3) Velocity Command Generation: In order to reduce computational overhead, our terrain generation tool does not include collision detection between objects. As a result, these terrains contain overlapping objects. In some cases, the terrains also have significant deviations in height in small regions which cannot be traversed due to the limited size of the robots. In this regard, we generate velocity commands with the assumption that the robot tracks the velocity commands perfectly. We then check for height deviations in the terrain for the next 400 ms along the robot’s expected trajectory. In case the height deviation exceeds the threshold of 0.4 m, we generate new velocity commands until the height deviation along the expected trajectory remains less than the threshold. This height deviation test is performed every 100 ms. The described velocity command generation approach significantly helps boost the performance of the footstep planning and domain adaptive tracking policies during training. Without introducing a strategic velocity command generator, we observed that our policies failed to converge, often resulting in reward collapse.

4) Domain Randomization: Our FP is trained to adapt to the behavior of the motion controller which significantly depends on the dynamic properties of the system. In order to achieve a locomotion behavior robust to change in system properties, such as added mass, we further utilize domain randomization strategies in addition to terrain randomization. These are given in Table V. Note that the sampling distributions presented in Table V relate to the limits which the motion controller is robust to. We observed that increasing the range of the distributions caused motion tracking failure. In this regard, we perform two training sessions for the FP. Session 1 employs the distributions presented in Table V, while Session 2 utilizes the DAT to enhance the robustness of the motion controller and, therefore, employs the distributions presented in Table VI. Furthermore, since the perception sensors present on the real robot cannot

---

### Table V

| Property                                | Sampling Distribution |
|-----------------------------------------|-----------------------|
| Gravity, \( g = 9.81 \text{m/s}^2 \)   | \( U(0.96g, 1.04g) \) |
| Actuation Torque Scaling                | \( U(0.85, 1.15) \)   |
| Robot Link Mass Scaling                 | \( U(0.9, 1.1, 0) \)  |
| Robot Link Length Scaling               | \( U(0.85, 1.15) \)   |
| Actuation Damping Gain \( G \) such that | \( \tau_j = G \tau_j^* + (1-G) \tau_j \) |
|                                        | \( U(0.8, 1) \)       |

Note: Here \( \tau_j^* \) is the expected torque output of the impedance controller.

### Table VI

| Property                                | Sampling Distribution |
|-----------------------------------------|-----------------------|
| Gravity, \( g = 9.81 \text{m/s}^2 \)   | \( U(0.96g, 1.04g) \) |
| Actuation Torque Scaling                | \( U(0.85, 1.15) \)   |
| Robot Link Mass Scaling                 | \( U(0.9, 1.1, 0) \)  |
| Robot Link Length Scaling               | \( U(0.85, 1.15) \)   |
| Actuation Damping Gain \( G \) such that | \( \tau_j = G \tau_j^* + (1-G) \tau_j \) |
|                                        | \( U(0.8, 1) \)       |
be used to perceive the elevation along regions occluded by an obstacle, we apply blurring filter to the elevation map, as shown in (10), to emulate interpolation around unobservable terrain features.

We also apply external perturbations along $e_x^B$ and $e_y^B$. However, since the motion controller is not robust to perturbations of high magnitude, we sample the external forces from a normal distribution $\mathcal{N}(0, 10) \text{N}$ clipped between $[-30, 30] \text{N}$ for a duration in the range $[1, 4] \text{s}$. We also perform gait randomization, where we select the trot gait with a probability of 0.7, crawl with a probability of 0.2, and amble and pace with a probability of 0.05 each.

5) Reinforcement Learning: For Session 1, we employ an interactive direct policy learning approach to perform guided updates of our footstep planning policy $\pi_{\theta_f}$ in order to increase sample efficiency during RL policy optimization. We initialize policy parameters $\theta_f$ using Xavier initialization [48]. We consider a fixed-length episode of 1024 desired feet position samples, implying 1024 gait strides are executed in each episode. During training, we also perform locomotion over flat terrain. In this case, we use the motion controller to track the feet position generated by the policy and store it in a buffer $\mathcal{D}$. Additionally, we generate the desired feet position using the foothold optimizer of the dynamic gaits motion controller for the same robot state $s_f$ and add it to $\mathcal{D}$. After the desired number of episodes used for locomotion over flat ground, we perform backpropagation through the footstep planning policy using the mean-squared error loss between the predicted and the expert samples. For uneven terrain, we utilize SAC to perform policy exploration so as to learn to generate feet position that avoid edges and slopes while ensuring the stability of the robot. We alternate between policy optimization through exploration and the interactive guided policy learning during training. The ratio of episodes used to perform guided learning to the number of episodes used to perform policy exploration is reduced as training progresses. We refer to this approach as interactive guided policy optimization (IGPO). Algorithm 1 describes the IGPO approach used for training the footstep planning policy.

It is important to note that since we use a curriculum-based reward transition given by the factor $d_f$, as shown in (16) and (17), we continuously push new state transition tuples to the SAC replay buffer while discarding the older samples to ensure that the action-value function approximations utilized in SAC remain relevant to the latest objective weighted by $d_f$. Moreover, while performing guided learning, we set the target value of the state-dependent stochasticity based on $\sigma_{\theta_j}(s_f)$ to zero.

As training progresses, we increase the complexity of the RL environment by scaling the maximum terrain elevation. As introduced in Section IV-A1, we represent the elements of $T$ in the range of $[0, h_{\text{max}}]$, where $h_{\text{max}} = 2 \text{m}$. During training, we scale $h_{\text{max}}$ using $d_{hf}$ which is updated by the rule $d_{hf,t+1} \leftarrow d_{hf}^{0.992}$, where $k$ refers to the policy iterations and $d_{hf,0} = 0.01$.

For Session 2, we do not perform guided learning and only utilize SAC to make footstep planning more robust to the changes in modeled system dynamics. We also set each of the curriculum factors to one. This additional training allows us to utilize the same FP trained with ANYmal B for deployment on ANYmal C. The training hyperparameters and policy convergence iterations are provided in the Appendix.

6) Testing: For the purpose of evaluation, we utilize the stochastic policy in a deterministic manner. Therefore, we consider the $\mu_{\theta_j}(s_f)$ as the desired action $H_{HF,y}^{\*}$ and discard $\sigma_{\theta_j}(s_f)$. We detail the test results in Section V-A.

7) Deployment: The observations utilized for the FP are accessible on the physical robots. During deployment on the real robots, we observe a computational delay of approximately 35 ms during elevation map extraction and encoding. The elevation map processing is, therefore, executed in a parallel thread, while the motion tracking controller runs in the main thread at 400 Hz. The robot maintains the previous desired footholds while the forward pass through the elevation map encoder and the footstep planning policy is performed, after which, the new motion plans for the updated footholds are generated and tracked. This methodology remains consistent for both ANYmal B and C. A forward pass through the FP policy network requires approximately 195 $\mu$s for both the robots.
B. Domain Adaptive Tracker

We utilize the terrain generator, actuator network, and velocity command generator introduced in Section IV-A for training the DAT policy. In addition, we use the footstep planning policy obtained after training Session 1 to train our domain adaptive tracking policy.

1) Domain Randomization: The randomization parameters and the corresponding sampling distributions representing the limits are provided in Table VI. During training of the DAT policy, the parameter randomization is approached using a curriculum learning setup where initial limits of the distributions are set to 1. The lower and upper limits are then linearly expanded after every policy iteration \( k \) based on the rule \( l_{t,k} = \max(\frac{k}{2000}(l_{\text{min}} - 1) + 1, l_{\text{min}}) \) and \( l_{u,k} = \min(\frac{k}{2000}(l_{\text{max}} - 1) + 1, l_{\text{max}}) \), respectively, where the \( l_{\text{min}} \) and \( l_{\text{max}} \) are the lower and upper limits presented in Table VI. We also introduce external perturbations and gait randomization as done for the FP.

2) Reinforcement Learning: We employ TD3 for training the DAT policy allowing us to perform constrained exploration. We observed that the stochasticity associated with policies trained using SAC and PPO required exhaustive hyperparameter tuning. Without significant tuning, the exploration resulted in the introduction of large corrective torques. This caused the whole-body controller to significantly drift from the generated motion plans causing aggressive recovery and eventual failure. This inhibited learning since the corrective torques mostly resulted in episode termination. We, therefore, used TD3 to initially perform constrained exploration using uncorrelated Gaussian sampling. During training, we scale the standard deviation of the standard Gaussian distribution according to the rule \( a_{t+1} = a_t + \delta_T \), where \( \delta_T \) by a factor of 40 before adding it to the feed-forward torque generated by the whole-body controller.

As in the case of training the FP, we use the same curriculum approach of scaling the maximum terrain elevation. We used the standard RLlib TD3 implementation for RL. The hyperparameters are provided in the Appendix.

3) Testing: Since we obtain a deterministic policy, we do not perform any modifications to the neural network and directly employ it as a function mapping state to action. The test results are presented in Section V-B.

4) Deployment: The DAT is executed along with the whole-body controller in the main control loop at 400 Hz. The generated corrective torques are then added to the output of the whole-body controller, after which the desired joint states are forwarded to the actuator. The DAT remains active when the RC is deactivated. A forward pass through the DAT policy network requires approximately 66 \( \mu s \).

C. Recovery Controller

Our RC training setup is directly built on top of our previous work [21]. However, unlike in the case of our previous work, we do not employ a guided policy optimization strategy since the guided learning focused on velocity command tracking as opposed to recovery control and instead utilize the training approach presented in [19] performing RL using PPO.

For training the RC policy, we employ the terrain generator introduced in Section IV-A1. Since we perform joint position control with the recovery policy, we also use the actuator network to model the actuation dynamics for fixed \( K_p = 50 \) and \( K_d = 0.1 \) with zero velocity target. This is described in (23). We defer the reader to the original work [19] for detailed description of the joint position target actuator network.

1) Domain Randomization: We use the same randomization strategy, including curriculum learning, as used for DAT for training the RC.

2) Reinforcement Learning: Since our objective is to learn an RC that can stabilize to perturbations in addition to performing state recovery, we introduce external perturbations in our training environment. However, in order to encompass a wide possibility of robot states when the RC is activated, we initially train the RC policy to track reference velocity commands. We then transition from this objective to that of recovery control. For this, we initially learn to locomote over flat ground after which we introduce larger external perturbations while also scaling up the terrain elevation. The approach is detailed in Algorithm 2.

3) Testing: As in the case of FP, we utilize the stochastic policy in a deterministic manner. In this regard, the action \( a_\tau = \mu_\theta(s_\tau) \). The desired joint positions \( q_{\tau}^d = a_\tau \) are then forwarded to the actuator to generate the effective actuation torques. The results are presented in Section V-C.

4) Deployment: We query the robot state at each control step executed at 400 Hz to activate RC according to the limits described in Section III-E. When active, RC is run in the main control loop to generate the desired joint positions. While the RC remains active, the FP, motion controller, and DAT are deactivated. A forward pass through the RC policy network requires approximately 42 \( \mu s \).

V. RESULTS AND DISCUSSION

This section details the results obtained for each of the RL policies introduced in our framework. Note that, unless otherwise stated, the results presented here relate to the ANYmal B quadruped.
Algorithm 2: Recovery Control Training Objective Transition.

Input: $n_{eps}, l_{eps}, H$

Initialize $e_{max} = 1e - 4, f_{max} = 0.01, t_{fmax} = 0.01, t_f = 0, t_a = 0$

1: for $k = 0, 1, 2 \ldots H$ do
2: for $i = 0$ to $n_{eps} - 1$ do
3: $t_{vel}, s = RL_{ENVIRONMENT}\text{RESET}(e_{max})$
4: for $t = 0$ to $n_{eps} - 1$ do
5: if $t \geq t_{vel}$ then
6: Generate new velocity command
7: Sample $t_{vel} \sim \mathcal{U}(1,3)$
8: end if
9: Sample action $a \sim \pi_{\theta_i}(a|s)$
10: if $t \geq t_a$ and $t < t_a + t_f$ then
11: Apply external force $f$
12: else if $t > t_a + t_f$
13: Sample external force, $f \sim \mathcal{N}(0, 45f_{max})$
14: Sample $t_a \sim \mathcal{U}(t(t + 2))$ and $t_f \sim \mathcal{U}(0, 4tf_{max})$
15: end if
16: $s, r = RL_{ENVIRONMENT}\text{STEP}(a)$
17: if $s$ is a terminal state then
18: $t_{vel}, s = RL_{ENVIRONMENT}\text{RESET}(e_{max})$
19: end if
20: end for
21: end for
22: Update $\pi_{\theta_i}$ using PPO
23: $e_{max} \leftarrow e_{max} + 0.009996, f_{max} \leftarrow f_{max} + 0.00984, t_{fmax} \leftarrow t_{fmax} + 0.0997$
24: end for
25: function RL_{ENVIRONMENT}\text{RESET}(e_{max})
26: Sample terrain and scale elevation by $e_{max}$
27: Randomize robot pose
28: Randomize $q_j$ where $\max(|q_j - q_j^0|) < 0.3$
29: Randomize $\dot{q}_j$ where $\max(|\dot{q}_j|) < 1.0$
30: Update state $s$ and set $t_{vel} = 0$
31: return $t_{vel}, s$
32: end function

this optimization, we compute the cost by sampling $\hat{r}_{F_{xy,i}}$ along the polar radial coordinate with a resolution of 0.005 m and a polar angular resolution of $\pi/18$ rad. We test the performance over 1000 different terrains of varying dimensions for each of the four object types—stairs, bricks, wave, and unstructured ground, as shown in Fig. 8. Each of these objects are located at the center of a 5 x 5 m$^2$ terrain, with the length of the object, along the direction of the robot’s frontal axis, varying between 2.0 m and 3.6 m. We consider the following terrain properties.

1) The amplitude for the unstructured terrain is varied between $[0.0125, 0.025]$ m.
2) For stairs, the number of steps range between $[3, 8]$. The maximum terrain elevation is varied between $[0.25, 0.8]$ m using clipped normal distribution, $\mathcal{N}(0.3, 0.1)$ m.
3) The wave terrain is generated using a sine function along its length with an amplitude range of $[0.05, 0.1]$ m. The period of the sine function ranges between $\pi/2$ and $\pi$.

for brick terrain, the bricks are made of deformed $10 \times 10$ cm$^2$ patches with extrusions $\{-h, 0, h\}$, where $h \in [0.02, 0.05]$ m.

The results obtained for traversal over these terrains are presented in the Appendix (Movie S1).

For each run, we set the robot at the origin and generate a forward velocity command of 0.3 m/s. We denote the run as a success if the robot traverses a distance of 4 m, and regard it as a failure if any of the FP termination criteria is observed. The SRs obtained for each of the experiments are represented in Table VII. Fig. 9 presents the SR for traversing each terrain for each of the tested controllers with two gaits.

The RL FP was observed to exhibit preference for wider stance. This directly corresponds to a higher stability margin, a term introduced in the reward function during training. We measured $H_{RF}$ after every gait stride. For traversal over stairs, the absolute mean foot position in frame $H$ was 0.35 m and 0.20 m along its x and y axes, similar to the expected nominal stance value of 0.35 m and 0.20 m. However, in the case of unstructured terrain, we computed the mean feet position to be (0.34, 0.23)m, (0.35, 0.21)m for the wave terrain, and (0.34, 0.22)m for the bricks terrain.

Fig. 9(e) represents the SR as a function of step height and depth evaluated using kernel density estimation (KDE).

The extrusions in the brick terrains often caused leg blockages. This was prominent for the blind planner. The ability of the perceptive planner to minimize distance from the edges in the terrain significantly reduced the problem with leg blockage. However, the difference in the feet height resulted in the robot toppling over in most cases. With the RL planner, we observed a wide stance and preference for foot placement along regions which minimized the deviation in feet height. For the case of the blind planner, we computed a mean deviation between the maximum and minimum foot height to be 0.074 m. For the perceptive planner, the mean height deviation was 0.083 m, and 0.057 m with the RL planner. Fig. 9(f) illustrates the SR computed for different brick heights. In this case, the brick height only corresponds to the extrusion of the terrain implying that the maximum deviation in feet height is twice the height of the bricks. We observed that the RL planner fails to traverse brick terrains with maximum elevation of higher than 0.12 m (corresponding to a maximum height deviation of 0.24 m). This mainly occurs due to the limitations of the physical dimensions of the quadruped.

We also compared the generalization capability of the FP obtained after training Sessions 1 and 2. We performed the experiments discussed above using the ANYmal C quadruped, adapting the motion controller for the parameter description
TABLE VIII

SRs Obtained for Locomotion Over Different Terrains Using the ANYmal C Quadruped Comparing the RL FP Performance After Training Sessions 1 and 2

| Terrain   | Crawl Session 1 | Crawl Session 2 | Trot Session 1 | Trot Session 2 |
|-----------|-----------------|-----------------|----------------|----------------|
| Unstructured | 99.9            | 99.9            | 99.4           | 99.5           |
| Stairs    | 93.0            | 93.4            | 90.5           | 91.9           |
| Wave      | 85.7            | 91.2            | 82.4           | 88.4           |
| Bricks    | 76.3            | 85.8            | 71.6           | 82.2           |

Note that, while running experiments on the simulated ANYmal C, we increased the joint torque limit from $\pm 40 \text{Nm}$ for the case of ANYmal B to a limit of $\pm 80 \text{Nm}$. The tracking error measured for varying base mass of the ANYmal C quadruped is represented in Fig. 10(b). With an original base mass of 26.3732 kg, the motion controller could track the forward velocity command of $0.3 \text{m/s}$ even when we scaled this mass in the range $[0.91, 1.10]$. When we introduced the domain adaptive tracking policy, we were further able to extend this range to $[0.81, 1.19]$.

We performed another test on the simulated ANYmal C robot. However, in this case, we used the parameter description of the ANYmal B quadruped. Therefore, the motion controller generated motion plans and control behavior for the ANYmal B quadruped even though the controller was used on ANYmal C. Note that the domain adaptive policy was never trained using ANYmal C. Fig. 11 represents the velocity tracking error observed in 1 M simulation samples for randomly generated desired base velocity commands in the range of $[-0.3, 0.3] \text{m/s}$ and $[-0.3, 0.3] \text{rad/s}$ for the linear and angular velocities, respectively. Note that in this case of domain adaptive tracking, we did observe an oscillatory behavior in the limbs while tracking the base velocity commands. This was largely due to the whole-body controller outputting desired joint states which were then aggressively corrected by the domain adaptive tracking policy. Moreover, the DAT failed to stably track absolute velocity commands beyond $\pm 0.4 \text{m/s}$ for heading velocity, $\pm 0.3 \text{m/s}$ for lateral velocity, and $\pm 0.5 \text{rad/s}$ for yaw rate.

Fig. 12 represents the mean and standard deviation of the absolute corrective joint torque generated by DAT to address ANYmal B base mass and link length scaling for locomotion on flat ground for a heading velocity command of $0.5 \text{m/s}$.

We have also provided a set of examples employing the DAT in the Appendix (Movie S2).

C. Recovery Controller

We tested the behavior of the RL recovery control policy and the motion controller for a range of external forces applied to the robot’s base along the heading and lateral axes for duration between $[0.5, 2.5] \text{s}$. The observations representing the KDE of the recovery rate (RR) as a function of the magnitude and duration of the external force for each of the controllers are as shown in Fig. 13. The color-bar labels represent the success probability of the robot to stabilize following external perturbations. We measure this for both ANYmal B and ANYmal C and observe that compared to the motion controller that can handle perturbations of up to approximately 100 N along the $x$-axis for the ANYmal B quadruped, the RL policy can handle external force of up to approximately 225 and 400 N for ANYmal B and ANYmal C, respectively. It is important to note that for the case of ANYmal C, we increased the position tracking gain to 60 compared to 45 for ANYmal B and also increased the joint torque limits to $\pm 80 \text{Nm}$ compared to $\pm 40 \text{Nm}$ for ANYmal B. The results obtained are also demonstrated in the Appendix (Movie S3).

ANYmal C. The corresponding SRs are presented in Table VIII. As expected, we observe a better ability to generalize a different platform with the policy trained using significant domain randomization.

B. Domain Adaptive Tracking

In our experiments, the tracking error associated with the whole-body controller was negligible when the robot’s base mass, of 18.3546 kg, was scaled in the range $[0.95, 1.05]$.

However, when we decreased the mass by a factor below 0.94, or increased it beyond 1.06, the motion controller became unstable resulting in increased oscillations. This was caused due to the whole-body controller aggressively correcting for imprecise motion tracking. Introducing DAT helped stabilize whole-body motion tracking. We were able to scale the base mass in the range of $[0.84, 1.17]$ without increasing the amplitude of the oscillations. Fig. 10(a) illustrates the $p_{\text{COM}}$ tracking error observed before episode termination for different robot base masses. Here, the episode termination criteria include traversing a distance of 5 m or failure of whole-body control.

We also tested our domain adaptive tracking policy, without retraining, on the simulated model of the ANYmal C quadruped.
D. RLOC

Fig. 14 represents the tracking error observed for the RLOC framework for traversal over different types of terrains. The improved tracking performance over the motion controller was a direct result of introduction of DAT. Removing DAT resulted in a very similar performance as that of the motion controller.

The experimental setups for each of these terrains included the generation of a velocity command for a duration of 5 s. We performed 100 runs for each command.

In order to quantify the effects of the recovery control policy, we performed experiments using the bricks and wave terrains for the trotting gait, as introduced for evaluation of the FP. This is further presented in the Appendix (Movie S3). Introduction of the RC resulted in improved SRs. The results are summarized in Table IX. We evaluated the velocity tracking performance of our framework in a slower but higher resolution Gazebo simulator [49], employing the ODE physics engine [50], as presented in Fig. 14(d). Fig. 1 presents examples of the experiments performed on the real ANYmal B quadruped. The corresponding video clips are included in the Appendix (Movie 1). Fig. 15 represents the snapshots of the robot ascending and descending a staircase. In our experiments with the real robot, we observed motion behavior similar to the behavior in RaiSim, especially for tests on simpler terrains such as a staircase. In the case of bricks, however, the state-estimation drift, sensor noise, and the fast but less accurate nearest-neighbor interpolation technique used to obtain the elevation map along occluded regions sometimes resulted in the elevation map being distorted, especially along the edges, and also slightly shifted with respect to the robot. The distortion of the elevation map is illustrated in Figs. 4 and 16 and the Appendix (Movie S4), which represent the real ANYmal B quadruped tested on a terrain comprising bricks and the corresponding visualization of the robot state and the elevation map. This blurring and slight shifting of the elevation map caused the robot to step close to the edges, especially in environments composed of a large number of small objects.

We compared the performance of RLOC on simulated ANYmal C with an open-sourced implementation of [34] as baseline. The SRs (for 1000 runs each) obtained for traversal over different terrains are provided in Table X. While the two control frameworks performed similarly for locomotion over unstructured and stairs terrain, the baseline controller was unable to stably traverse the brick terrain. This is mostly due to the significantly larger terrain elevation sampling resolution of 0.1 m utilized in [34] compared to 0.02 m as used in RLOC. Reducing the sampling resolution for the baseline is expected to require significant retuning of the training environment and the policy architecture. Although the baseline controller offered slightly better performance for traversing stairs, we observed that the controller was unable to reliably avoid edges while often staggering. In contrast, the motions exhibited by RLOC were a lot smoother. We suspect this behavioral comparison is also valid for [30]. This is attributed to the complex reward functions employed for training RLOC policies, which encourage stable maneuvers (stability margin reward) while avoiding terrain edges (terrain cost map).

We also transferred the RLOC framework to the real ANYmal C quadruped for both indoor and outdoor locomotions, as shown in Fig. 2. We were able to locomote outdoors over grassy and muddy terrain and slopes. We observed a similar behavior as in the case of ANYmal B; however, the ANYmal C offered an
advantage of higher joint torque limits enabling us to ascend steeper slopes. For this transfer, the only changes included the robot parameter description for the motion controller and the impedance controller gains. The experiments performed with the real ANYmal B and ANYmal C are presented in the Appendix (Movie 1 and Movie 6).

We observed that ascending steep staircases where the ANYmal-B base pitch exceeds \( \pi/4 \) rad with respect to the horizontal frame result in failure to track heading velocity commands where the robot is unable to move forward. This is partially attributed to the joint torque limits while also resulting from the CoM motion planner that uses ZMP, implicitly assuming constant base height and attitude. For ANYmal C, we observe velocity command tracking failure beyond a pitch angle of approximately \( \pi/3 \) rad. This was the case for both the simulated and real robots.

VI. CONCLUSION AND FUTURE WORK

We presented an approach which bridges the gap between policies trained using RL and controllers developed using model-based optimization methods. Our modular framework allowed us to greatly improve the success of traversal over complex terrains. For traversal over the bricks environment, as presented in Section V, we showed that our hybrid approach, RLOC, was able to increase the SR from 71.5% (see Table VII) to 90.2% (see Table IX) in comparison to our baseline perceptive controller. We showed that we were able to traverse reliably (i.e., with a SR of 90% or above over 4 m) and with a dynamic gait (trot at an effective velocity of approximately 0.25 m/s on the real robot) over uneven terrain even though the model-based locomotion controller was primarily developed and tuned for quasi-flat terrains.

To address ascending steep stairs and slopes, in our future work, we aim to learn a CoM and foot motion planner. Another direction for future work includes obtaining a hybrid long and short horizon planning policy to avoid obstacles and perform CoM and feet motion adaptations for traversal over extremely complex and nonstationary terrains.

REFERENCES

[1] M. Hutter et al., “ANYmal—A highly mobile and dynamic quadrupedal robot,” in Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst., 2016, pp. 38–44.
[2] E. Ackerman, “ANYbots introduces sleek new ANYmal C quadruped,” IEEE Spectr., 2019. [Online]. Available: https://spectrum.ieee.org/robotics/industrial-robots/anybots-introduces-sleek-new-anymal-c-quadruped
[3] E. Ackerman, “North sea deployment shows how quadruped robots can be commercially useful,” IEEE Spectr., Nov. 2018. [Online]. Available: https://spectrum.ieee.org/robotics/industrial-robots/north-sea-deployment-shows-how-quadruped-robots-can-be-commercially-useful
[4] H. Kolvenbach et al., “Towards autonomous inspection of concrete deterioration in sewers with legged robots,” J. Field Robot., vol. 37, no. 8, pp. 1314–1327, 2020. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.21964
[5] I. Mordatch, E. Todorov, and Z. Popović, “Discovery of complex behaviors through contact-invariant optimization,” ACM Trans. Graph., vol. 31, no. 4, pp. 1–8, Jul. 2012. [Online]. Available: https://doi.org/10.1145/2185520.2185539
[6] C. Mastalli et al., “Crocodile: An efficient and versatile framework for multi-contact optimal control,” in Proc. IEEE Int. Conf. Robot. Automat., 2020, pp. 2536–2542.
[7] R. Grandia, F. Farshidian, R. Ranfil, and M. Hutter, “Feedback MPC for torque-controlled legged robots,” in Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst., 2019, pp. 4730–4737.
[8] R. Full and D. Koditschek, “Templates and anchors: Neuromechanical hypotheses of legged locomotion on land,” J. Exp. Biol., vol. 202, no. 23, pp. 3325–3332, 1999. [Online]. Available: http://jeb.biologists.org/content/202/23/3325
[9] S. Kajita, F. Kanehiro, K. Kaneko, K. Yokoi, and H. Hirukawa, “The 3D linear inverted pendulum mode: A simple modeling for a biped walking pattern generation,” in Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst., 2001, vol. 1, pp. 239–246.
[10] I. Poulakakis, E. Papadopoulos, and M. Buehler, “On the stability of the passive dynamics of quadrupedal running with a bounding gait,” Int. J. Robot. Res., vol. 25, no. 7, pp. 669–687, 2006. [Online]. Available: https://doi.org/10.1177/0278364906066768
[11] I. Poulakakis and J. W. Grizzle, “The spring loaded inverted pendulum as the hybrid zero dynamics of an asymmetric spring,” IEEE Trans. Autom. Control, vol. 54, no. 8, pp. 1779–1793, Aug. 2009.
[12] M. Vukobratović and D. Juricic, “Contribution to the synthesis of biped gait,” IEEE Trans. Biomed. Eng., vol. BME-16, no. 1, pp. 4–16, 1969.
[13] J. Pratt, J. Carff, S. Drakunov, and A. Goswami, “Capture Point: A step toward humanoid push recovery,” in Proc. 6th IEEE-RAS Int. Conf. Humanoid Robots, 2007, pp. 206–207.
[14] D. E. Orin, A. Goswami, and S.-H. Lee, “Centroidal dynamics of a humanoid robot,” Auton. Robots, vol. 35, no. 2/3, pp. 161–176, Oct. 2013. [Online]. Available: http://dx.doi.org/10.1007/s10514-013-9341-4
[15] H. Dai and R. Tedrake, “Planning robust walking motion on uneven terrain via convex optimization,” in Proc. IEEE-RAS 16th Int. Conf. Humanoid Robots, 2016, pp. 579–586.
[16] J. Carpentier and N. Mansard, “Multicontact locomotion of legged robots,” IEEE Trans. Robot., vol. 34, no. 6, pp. 1441–1460, Dec. 2018.
[17] F. Jenelten, T. Miki, A. E. Vijayan, M. Bjelicion, and M. Hutter, “Perceptive locomotion in rough terrain—Online foothold optimization,” IEEE Robot. Automat. Lett., vol. 5, no. 4, pp. 5370–5376, Oct. 2020.
[18] D. Kim et al., “Vision aided dynamic exploration of unstructured terrain with a small-scale quadruped robot,” in Proc. IEEE Int. Conf. Robot. Automat., 2020, pp. 2464–2470.
[19] J. Hwangbo et al., “Learning agile and dynamic motor skills for legged robots,” Sci. Robot., vol. 4, no. 26, 2019, Art. no. eaau5872. [Online]. Available: https://robotics.sciencemag.org/content/4/26/eaau5872
[20] J. Lee, J. Hwangbo, and M. Hutter, “Robust recovery controller for a quadrupedal robot using deep reinforcement learning,” 2019, arXiv:1901.07517.
[21] S. Gangapurwala, A. Mitchell, and I. Havoutis, “Guided constrained policy optimization for dynamic quadrupedal robot locomotion,” IEEE Robot. Automat. Lett., vol. 5, no. 2, pp. 3642–3649, Apr. 2020.
[22] X. B. Peng, E. Coumans, T. Zhang, T.-W. Lee, J. Tan, and S. Levine, “Learning agile robot locomotion skills by imitating animals,” in Robot. Sci. Syst., Jul. 2020, doi: 10.15607/RSS.2020.XVI-064
[23] Z. Xie, G. Berseth, P. Clary, J. Hurst, and M. van de Panne, “Feedback control for Cassie with deep reinforcement learning,” IEEE/RSJ Int. Conf. Intell. Robots Syst., 2018, pp. 1241–1246.
[24] S. Levine and V. Koltun, “Guided policy search,” in Proc. Int. Conf. Mach. Learn., 2013, pp. 2244–2252.
[25] T. Haarnoja, S. Ha, A. Zhou, J. Tan, G. Tucker, and S. Levine, “Learning to walk via deep reinforcement learning,” in Robot. Sci. Syst., 2019.
[26] Y. Yang, K. Caluwaerts, A. Iscen, T. Zhang, J. Tan, and V. Sindhwani, “Data efficient reinforcement learning for legged robots,” in Proc. Conf. Robot Learn., 2020, pp. 1–10.
[27] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, “Domain randomization for transferring deep neural networks from simulation to the real world,” in Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst., 2017, pp. 23–30.
[28] V. Tsounis, M. Alge, J. Lee, F. Farshidian, and M. Hutter, “Deepgait: Planning and control of quadruped gaits using deep reinforcement learning,” IEEE Robot. Automat. Lett., vol. 5, no. 2, pp. 3699–3706, Apr. 2020.
[29] J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter, “Learning quadrupedal locomotion over challenging terrain,” Sci. Robot., vol. 5, no. 47, 2020, Art. no. eabc5986.
[30] T. Miki, J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter, “Learning robust perceptive locomotion for legged robots in the wild,” Sci. Robot., vol. 7, no. 62, 2022, Art. no. eabc2822.
[31] C. D. Bellicoso, J. 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, Jul. 2018.
Siddhant Gangapurwala received the B.E. degree in electronics from the University of Mumbai, Mumbai, India, in 2016. In 2017, he joined the AIMS program as a Doctoral candidate with the University of Oxford, Oxford, U.K. The following year, he joined the Dynamic Robot Systems Group of the Oxford Robotics Institute, Oxford, to working toward the D.Phil. degree with focus on machine learning and optimal control based approaches for robotic locomotion over uneven terrain. He is currently a Postdoctoral Researcher on locomotion, manipulation, and locomoplanning.

Mathieu Geisert received the M.Eng. degree in aerospace from Institut Supérieur de l'Aéronautique et de l'Espace SUPAERO, Toulouse, France, in 2013. He then joined the Humanoid Robotics Team, GEPETTO, LAAS-CNRS, Toulouse, France, where he worked for five years in different positions: as an Intern, Engineer, Ph.D. student, and finally as a Postdoctoral Researcher. In 2018, he joined the DRS Group as a Postdoctoral Researcher, where his work focused on planning, optimal control, and machine learning for legged robots. In 2021, he joined Arrival, London, U.K., as a Robotics Research Engineer.

Romeo Orsolino received the B.Sc. degree in mechanical engineering from the University of Genova, Genoa, Italy, in 2013, and the M.Sc. degree in robotics engineering from the Ecole Centrale de Nantes, Nantes, France, in 2015. Since 2015, he has been working toward the Ph.D. degree in motion planning for legged locomotion in rough terrains from Dynamic Legged Systems, Istituto Italiano di Tecnologia, Genova, Italy. In October 2019, he joined the DRS Group as a Postdoctoral Researcher where he pursued his research on multicontact motion planning, optimal control, dynamics, and perception. In December 2020, he joined Arrival Ltd., where he continues to work on perceptive motion planning and control of dynamical systems.

Maurice Fallon (Senior Member, IEEE) studied electronic engineering with University College Dublin, Dublin, Ireland. He received the Ph.D. degree in acoustic source tracking from the Department of Engineering, University of Cambridge, Cambridge, U.K., in 2009. After the Ph.D. degree, he joined MIT, Cambridge, MA, USA, as a Postdoctoral Researcher and later as a Research Scientist with the Marine Robotics Group (2008–2012). From 2012 to 2015, he was the perception lead of MIT’s team in the DARPA Robotics Challenge. In 2015, he was a Lecturer with the University of Edinburgh, Edinburgh, U.K., where he led research in collaboration with NASA’s humanoid robotics program. He moved to Oxford in April 2017. Dr. Fallon was the recipient of the Royal Society University Research Fellowship in October 2017.

Ioannis Havoutis received the M.Sc. and Ph.D. degrees (with distinction) in 2007 and 2011, respectively, from the University of Edinburgh, Edinburgh, U.K., where he worked on machine learning for motion planning and control of articulated robots. He then joined the Dynamic Legged Systems Lab, Advanced Robotics Department, Istituto Italiano di Tecnologia, Genova, Italy, where he led the Locomotion Group within the HyQ team. He was a Postdoctoral Researcher with the Robot Learning and Interaction Group, Iidap Research Institute, where he worked on learning complex skills from demonstration. He is currently a Departmental Lecturer with the Oxford Robotics Institute, Oxford, U.K. His research focuses on whole-body motion planning and control with machine learning, focusing on robots with arms and legs.