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

Siddhant Gangapurwala, Mathieu Geisert, Romeo Orsolino, Maurice Fallon and Ioannis Havoutis

***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 trained in simulation over a wide range of procedurally generated terrains. When ran online, the system tracks the generated footstep plans using a model-based controller. We evaluate the robustness of our method over a wide variety of complex terrains. It exhibits behaviors which prioritize stability over aggressive locomotion. Additionally, we introduce two ancillary RL policies for corrective 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*—Deep Learning in Robotics and Automation, AI-Based Methods, Legged Robots, Robust/Adaptive Control of Robotic Systems**

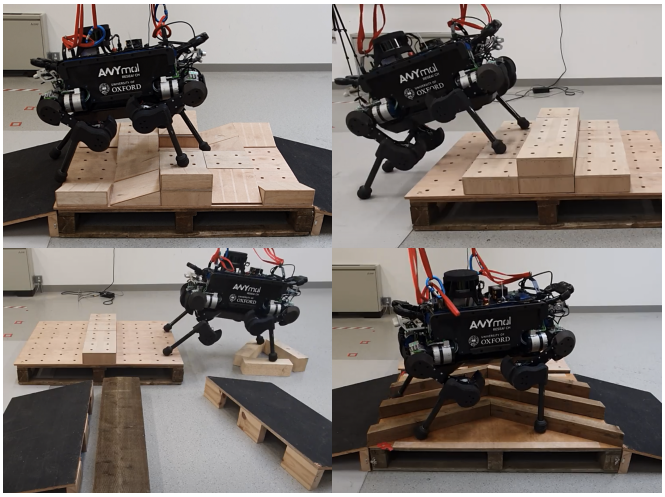


Fig. 1: ANYmal B quadruped used to perform experiments on different terrains using our RLOC framework. Overview video can be found at: https://youtu.be/k1WGNDeTGXU.

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 *ANYbotics ANYmal* [1] quadruped robot. ANYmal, version B specifically, 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 which enables a wide range of leg motions and is suitable for a broad domain of applications, for example, inspection tasks on offshore platforms [2] and sewer systems [3]. Figure 1 shows the ANYmal B robot used for testing our proposed framework on different uneven terrains. In addition, we highlight the generality of our control framework by testing our approach with the newest ANYmal, version C [4] –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.

The control of such underactuated robotic systems is an active area of research. 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 [5]. 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) [6] and the spring-loaded inverted pendulum (SLIP) [7], and their derived variants [8].

For the development of robust model-based controllers, the synthesis of stability criteria based on ground reference points (2D points on a flat ground plane) such as the zero moment point (ZMP) [9] and the instantaneous capture point (ICP) [10] is also introduced into the control problem. The templates can be seen as a way to constrain the admissible domain of such

This work was supported by the UKRI/EPSRC RAIN Hub [EP/R026084/1] and the EU H2020 Projects MEMMO and THING, the EPSRC grant 'Robust Legged Locomotion' [EP/S002383/1] and a Royal Society University Research Fellowship (Fallon). This work was conducted as part of ANYmal Research, a community to advance legged robotics. The authors are with Dynamic Robots Systems (DRS) group, Oxford Robotics Institute, University of Oxford, UK. Email: {siddhant, mathieu, rorsolino, mfallon, ioannis}@robots.ox.ac.uk

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, a 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 *multi-contact* conditions, the centroidal dynamics model [11] 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 linear and angular acceleration) in the task space of the robot discarding the intermediate joint-space physical quantities. In this case, the reference point for evaluating stability is a 6-dimensional wrench and, therefore, for evaluation of stability, templates that illustrate suitable 6D admissible regions need to be defined [12], [13]. Despite their low dimensionality, such simplified models are still typically too numerically inefficient to be employed in the robot's real-time control loop.

Alternatively, an entirely different approach has focused on investigating solutions which can perform a direct mapping between sensory information to desired control signals with minimal computational overhead. Model-free data-driven techniques such as 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 [14], [15], [16], Laikago [17] and Cassie [18]. 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 towards an undesired control behavior. As an alternative, a guided policy exploration approach, such as guided policy search (GPS) [19], 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 simulators, and the ability to execute simulations much faster than real-time, makes this approach appealing for training RL policies. However, the inaccuracy associated with the complex dynamics that are hard to model in simulation often affect the performance of RL policies when deployed onto real systems. Although there have been works where training was done on the real system itself [20], [21], this has been appropriate for simpler and inherently stable robots which cannot be employed for training a complex quadrupedal system such as the ANYmal B robot. 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.* introduced an actuator network [14] to model the dynamics of the real robot's motors 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 [22] techniques have been used to train policies which are robust to unaccountable factors [14], [16].

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. *Peng et al.* presented an approach for terrain-adaptive locomotion using RL [23], however, this work was limited to 2D character animation. *Tsounis et al.* presented an approach for terrain-aware gait planning and control using RL for the ANYmal B quadruped [24]. *Tsounis et al.* used a hierarchical approach to plan gaits suitable for navigation to a 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. Recently, *Lee et al.* presented an RL based approach for quadrupedal locomotion over challenging terrains [25], however, this work only focused on use of proprioceptive feedback for locomotion without explicitly processing terrain information.

A. Outline

We propose a unified RL and optimal control (OC) based terrain-aware legged locomotion framework as illustrated in Fig. 2. 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 whole-body dynamic motion controller from *Bellicoso et al.* [26] to enable foot tracking over non-flat terrain. This control architecture enables us to ensure that safety-critical constraints such as joint kinematic limits and peak joint torques are still enforced by the low-level controller. This forms the core of our locomotion framework. In order to account for changes in physical parameters and actuation dynamics on the real system, we introduce a domain adaptive RL 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 like leg 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 Movie 1.

B. Contributions

1) Procedural terrain generation - As detailed in our previous work [16], the behavior of an RL policy significantly depends on the setup of the environment. In order to obtain a robust policy which has the capability to generalize to disturbances, we incorporate a rigorous training setup. We *procedurally generate a large set of*

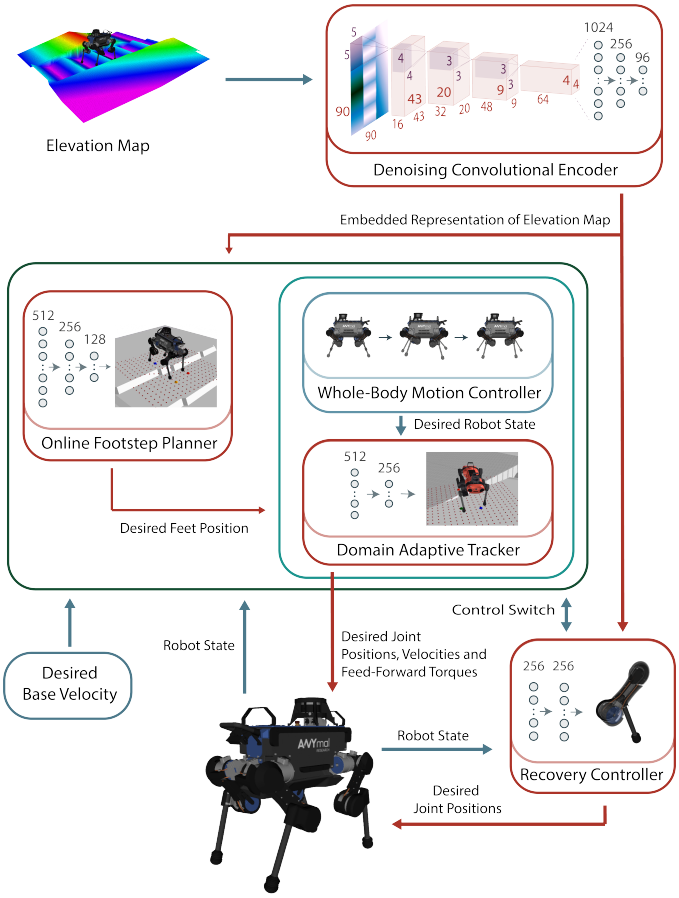


Fig. 2: Overview of the proposed control framework - RLOC. We use an online footstep planning RL policy to generate desired feet positions which are tracked using a model-based whole-body motion controller. In order to perform accurate tracking even in case of changes in robot dynamics, we introduce a domain adaptive tracking policy to generate feed-forward corrective torques. Furthermore, in order to exhibit a quick response to external perturbations, we introduce a recovery control policy which generates low-level joint position commands.

uneven terrains for training each of the RL policies and further introduce methods such as domain randomization. Moreover, in order to direct policy convergence to a desired optimum and to perform sample efficient training, we use an interactive direct policy learning approach to guide policy optimization.

2) Encoding elevation map - To perform terrain-aware locomotion, we introduce *latent representations of the terrain elevation*. This is local to the robot's base, and, together with the robot's state, is included in the observation space of the RL policies. We extract the latent space by training a denoising convolutional autoencoder and utilize the encoder network to generate embedded representations of these elevation maps as a form of exteroceptive feedback while training our RL policies. This step enables us to extract features from the elevation map (a matrix of size 91×91) and represent them in the form of a 96-dimensional vector. Moreover, unlike the case of proprioceptive sensing which can be estimated at frequencies as high as 400 Hz on the ANYmal B robot, the exteroceptive feedback relies on the sensors (cameras, LIDARs) which operate at much lower frequencies resulting in asynchronous updates of the robot's internal and external state information. Therefore, for the domain adaptive tracking and recovery control policies which run at 400 Hz, we include additional knowledge of the robot's relative pose in the frame of reference used for updating the perceptive information in the observation space of the RL policies.

3) Actuation modeling - To perform a successful transfer of the policy trained in simulations to the physical system, we *model the actuation dynamics* of the robot using an actuator network, a multi-layer perceptron (MLP) trained using data generated from the actuators present on the real robot. We extend the approaches in [14], [16], where an actuator network generated joint torques for target joint positions and for pre-defined PD gains, by implementing an actuator network that outputs joint torques for user-defined PD gains (that vary depending on the foot contact states - swing, in-contact and slip), desired target joint positions, velocities and feed-forward torques. Recording data for different PD gains is a difficult task, especially when performed on the real robot. This requires tuning the controller parameters for different sets of PD gains. In order to circumvent this problem, we utilize the gains from a smaller observation set and introduce a regressive resampling method, SMOGN [27], to make the actuator network generalize to a wider set of PD gains.

4) Stability margin network - In order to achieve more efficient learning, we introduce a dense reward function by incorporating a method to perform *stability analysis* represented by a *stability margin* metric. This method accounts for the delayed reward which may occur due to the robot becoming unstable earlier than certain termination criteria is reached. For the task of quadrupedal locomotion, we show that introducing the stability margin effects the performance of the RL policy by prioritizing lower base accelerations and operations along the nominal robot state. In order to perform extremely fast stability analysis, we train a neural network as a function approximator to map the robot state information to a stability margin metric.

C. Evaluation

We present a framework 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 our framework on a wide range of simulated terrains and present the results for the ANYmal B quadruped. We determine the success rate of traversal to a desired goal position while tracking reference base velocity using our footstep planning policy, and compare it against locomotion

using perceptive and blind planning as our baselines. For the domain adaptive tracking policy, we show that introduction of corrective feed-forward torque aids in more stable motions even with change in robot dynamics. We also show that our recovery controller is able to stably respond to external forces 5 times larger in magnitude than the model-based motion controller by itself. Additionally, we transfer our trained policies to ANYmal C and show that our control framework can be adapted for use on a different robot without the need for retraining the RL policies. We further transfer our control framework to the real ANYmal B quadruped and present a qualitative validation of its performance.

II. Approach

This section presents the approach used for training each of the RL policies and further introduces the model-based *Dynamic Gaits* controller used for whole-body motion tracking. An overview of our training approach is illustrated in Fig. 3.

A. Planning and Control Policies

We formulate the problems of footstep planning, domain adaptive tracking and recovery control in the framework of discrete time Markov decision processes (MDP) [28]. An MDP is defined by the tuple - (S, A, R, P, μ), where S represents a set of states, A a set of actions, $R : S \times A \times A \to \mathbb{R}$ the reward function, $P : S \times A \times S \to [0,1]$ the state transition probability, and μ the initial state distribution. In this work, we use a multilayer perceptron (MLP) as an approximation for each of the RL policies where an RL policy is defined as $\pi : S \to \mathcal{P}(A)$, a function mapping states to probability distributions over actions such that $\pi(a|s)$ denotes the probability of selecting action a in state s. We represent the expected cumulative discounted return,

$$J(\pi) \doteq \mathop{\mathrm{E}}_{\mathcal{T}\sim\pi}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t, s_{t+1})\right], \tag{1}$$

where $\gamma \in [0,1)$ is the discount factor and $\mathcal{T}$ denotes a trajectory dependent on π. As part of RL training, we aim to obtain a policy π which maximizes $J(\pi)$.

For the case of footstep planning, we employ a stochastic off-policy gradient based optimization method, soft actor critic (SAC) [29], to perform entropy-regularized reinforcement learning. For a slow environment setup of footstep planning, which includes prediction of desirable footholds and further includes tracking of these feet positions using the model-based controller, SAC offers better sample efficiency in comparison to the widely used on-policy alternatives, trust region policy optimization (TRPO) [30] and proximal policy optimization (PPO) [31], and has further been demonstrated to perform efficiently on quadrupedal locomotion tasks [32].

We use the successor to the continuous action-space deep deterministic policy gradient (DDPG) [33] algorithm - twin delayed deep deterministic policy gradient (TD3) [34] to train our domain adaptive tracking policy using Gaussian noise regularization as opposed to time-correlated exploration through Ornstein-Uhlenbeck process.

We train the recovery control policy using the method introduced in our previous work on guided constrained policy optimization (GCPO) [16] employing the PPO algorithm. We use the model-based motion controller, described in the following section, to generate reference trajectories used to perform guided policy updates (GPU) in order to direct policy convergence with reduced exploration for policy optimization.

It is important to note that, during evaluation, we use the stochastic policies trained using SAC and GCPO to generate deterministic actions. We do not sample the desired action from a Gaussian distribution, instead, only utilize the mean of the action probability distributions. In this regard, for the purpose of evaluation, we will refer to the RL policies as the functions mapping states to actions.

B. Motion controller

In order to follow the commanded robot velocity and the feet positions computed by the footstep planning policy, we use a model-based motion controller extending upon the work of *Bellicoso et al.*. This *Dynamic Gaits* controller [26] is included as part of the ANYmal robot software stack. For detailed description of the controller, we defer the reader to the original work [26] which presents the ability of the dynamic gaits controller to generate motion plans for a wide range of locomotion gaits and also demonstrates its robustness on the ANYmal B robot even enabling locomotion on quasi-flat terrains.

The Dynamic Gaits motion controller is a modular control framework which includes a (1) contact scheduler, (2) foothold optimizer, (3) center of mass (CoM) motion planner, (4) foot motion planner, and (5) whole-body controller. These modules are briefly described in Section A of the Appendix.

In the case of the base Dynamic Gaits controller, the contact scheduler, the foot motion planner and whole-body controller run at the same frequency as the main control loop (i.e. 400Hz here) while the foothold optimizer and the CoM motion planner run in separate threads and a new optimization problem is executed each time a solution is found. Therefore the frequency of those two controllers can vary depending on the difficulty of the task and the available computational resources. In order to reduce the computational overhead for training the RL policies, we execute the tasks of foothold optimization and CoM motion planning at a fixed frequency of 50Hz which corresponds to a conservative lower bound of the desirable planning frequency on the hardware.

Note that we use the foothold optimizer to perform interactive direct policy learning in order to direct policy exploration and convergence while training the footstep planning RL policy. During evaluation, however, we only utilize the footsteps generated by the RL policy.

III. Reinforcement Learning

This section details upon the RL environment setups and the training strategies used to obtain each of the footstep planning, domain adaptive tracking and recovery control policies.

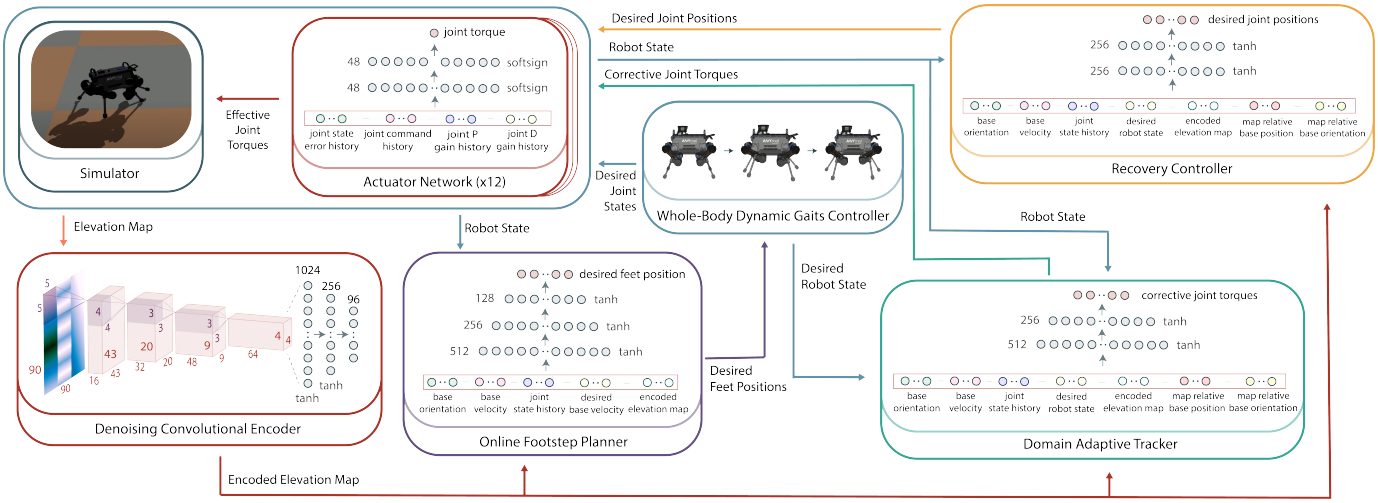


Fig. 3: Overview of the approach used to train the RL policies as part of our framework. We introduce the denoising convolutional encoder to embed the elevation map into a lower-dimensional representation. This, along with the robot states and desired velocity commands becomes the observation space for 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.

A. *Observation and action*

To capture terrain information, we include representation of exteroceptive feedback in the form of encoded local elevation maps along with the proprioceptive sensory information in the state vector s of our MDP formulation for each of the RL policies. In our RL environment setups, all of the state information utilized is accessible through on-board sensors and state-estimators of the physical system thereby making it feasible to successfully transfer the RL policies trained using a simulator to the real robot.

We introduce an embedded representation of an elevation map [35] local to the robot, in the state vector s, to capture terrain information. We use a 91×91-dimensional elevation map, along robot's frontal and lateral axes, orthogonal to the gravity aligned axis, corresponding to an area of $1.8 \times 1.8\text{m}^2$ centered at the robot's base position. We clip the elevation of these maps in the range of $[-2.0, 2.0]$m. We use this terrain information for each of our RL policies. In order to reduce the delays of processing the elevation map by each of these policies, we pre-train an encoding convolutional neural network to generate a latent representation of the elevation map. We perform feature extraction of our sparse elevation map representations using a denoising convolutional autoencoder network to learn an encoding $h = f(x + \mathcal{N}(0, \sigma^2))$ such that the decoder, approximated as a neural network g, is able to regenerate the input $x \approx g(h)$. We then use the encoder network f to encode our 91×91-dimensional height maps. The architecture of the autoencoder is shown in Fig. 4. With a compression factor of 86.26, we encode the extremely sparse elevation maps into 96-dimensional representations.

To train the denoising convolutional autoencoder, we use our terrain generator tool, described in Section III-C1, to procedurally generate $20 \times 20\text{m}^2$ terrains, represented as a 1001×1001-dimensional matrix. These terrains contain objects such as stairs, wave, bricks, planks and unstructured ground. We slice a 91×91 matrix from the terrain matrix 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 parallel to the main training session in a separate thread. This approach of data augmentation enables us to obtain an encoder network with high generalization capability. Moreover, training a denoising convolutional encoder enables us to extract the main terrain features such as edges and slopes while filtering noisy data. Figure 5 shows examples of regeneration of noisy terrain inputs using the trained autoencoder.

1) Footstep planning: The observation space of the footstep planning policy is represented by the 204-dimensional vector,

$$\left\{ \mathcal{O}, v, \omega, \mathcal{J}_{t,\, t-5,\, t-10,\, t-50}, \dot{\mathcal{J}}_{t,\, t-5,\, t-10,\, t-50}, \mathcal{V}, \mathcal{E}_z^{base} \right\},$$

where $\mathcal{O}$ represents the vertical axis of the robot, v is the linear velocity of the robot represented in the base frame, ω is the angular velocity of the robot represented in the base frame, $\mathcal{J}_t$ is the joint position at time t, $\dot{\mathcal{J}}_t$ is the joint velocity at time t, $\mathcal{V}$ is the user-generated gravity-axis aligned desired base velocity expressed in base frame and $\mathcal{E}_z^{base}$ is the gravity axis aligned embedded representation of the elevation map expressed in the base frame. The duration between $t-1$ to t corresponds to 2.5 ms. We introduce the history of joint states in observation space to infer the locomotion gait in use by the motion-controller. The different gaits result in different stability margins, a factor introduced in the reward function, which implies that the desired feet position generated by the policy depend on the gait in use hence necessitating the observation space to contain information relating to the robot locomotion gait. The footstep planning policy outputs an 8-dimensional vector comprising of gravity axis aligned orthogonal x and y-axis coordinates expressed in the robot's base frame for each of the 4 feet. The desired foot height is

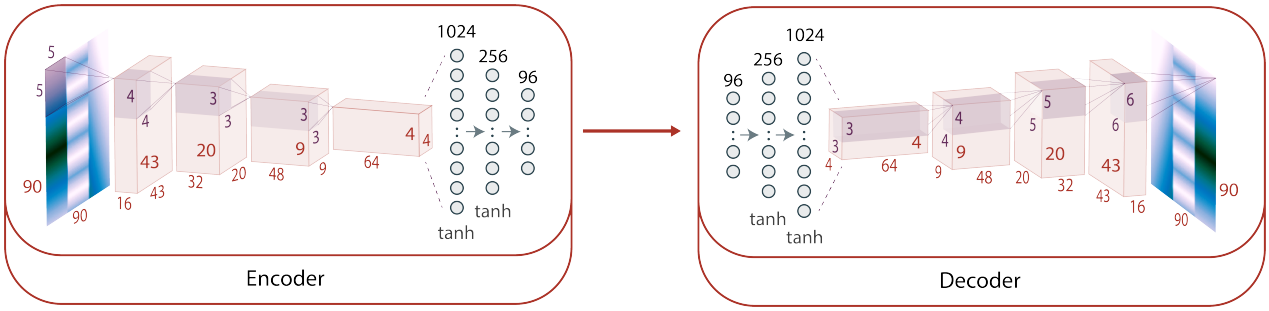


Fig. 4: Network architecture of the denoising convolutional autoencoder used to generate an embedded representation of the local elevation map. We use the *leaky ReLU* activation function for the convolutional layers and the *tan hyperbolic* activation function for the dense layers.

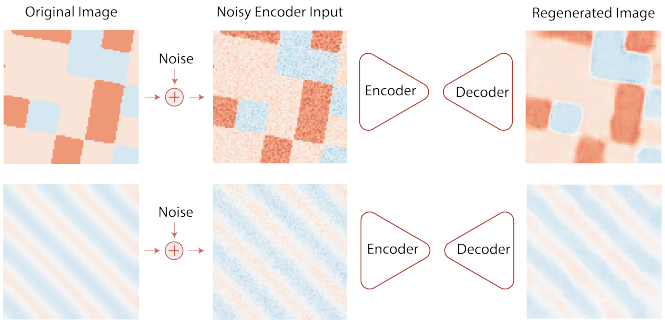


Fig. 5: Examples of regeneration of noisy images using our trained autoencoder - **top:** a patch from the bricks terrain and **bottom:** a patch from the sine wave terrain.

directly inferred from the elevation map for the generated x and y coordinates of the feet position.

2) Domain adaptive tracking: The domain adaptive tracking policy maps a history of tracking errors to corrective joint torques so as to reduce predicted tracking errors. The observation space is hence represented as a 275-dimensional vector,

$$\{c^{base}_{t-4,t-8}, \delta c_{t,t-4,t-8}, \mathcal{O}^{base}_{t-4,t-8}, \delta\mathcal{O}_{t,t-4,t-8}, v_{t,t-4,t-8}, \delta v_{t,t-4,t-8}, \omega_{t,t-4,t-8}, \delta\omega_{t,t-4,t-8}, \mathcal{J}_{t,t-4,t-8}, \dot{\mathcal{J}}_{t,t-4,t-8}, \hat{f}^{base}_{t,t-4,t-8}, \dot{\hat{f}}_{t,t-4,t-8}, \delta\tau_{t-1}, \mathcal{E}^m_z, c^m_t, \mathcal{O}^m_t\},$$

where c^{base}_t refers to the base position at time step t expressed in robot's current base frame, δc_t refers to the base position tracking error which is the difference between the desired base position as generated by the CoM planner and the current base position, $\mathcal{O}^{base}_t$ is the vertical axis of the robot at time step t expressed in the robot's current base frame, $\hat{f}^{base}_t$ is the desired feet position generated by the whole body motion controller and $\delta\tau_{t-1}$ is the corrective joint torques generated by the domain adaptive tracking policy. It is important to note that, unlike in the case of footstep planning where each call to the policy occurs after execution of every foot swing motion which requires at least 200 ms depending upon the base velocity and the locomotion gait, 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 express the gravity axis aligned elevation map in this stored reference frame denoted in the above observation vector as $\mathcal{E}^m_z$. We further include the current base position c^m_t and orientation $\mathcal{O}^m_t$ expressed in the stored reference frame in the observation vector so as to track the deviation from the stored reference frame.

The domain adaptive tracking policy outputs a 12-dimensional vector $\{\delta\tau\}$ representing the corrective joint torques. This output is added to the desired feed-forward torques generated by the whole-body motion controller and is then forwarded to the actuators along with the desired joint position and velocity commands, also generated by the motion controller.

3) Recovery control: The recovery control policy maps the current state of the robot to a 12-dimensional vector, $\{\hat{\mathcal{J}}\}$ comprising of the desired joint position commands in order to perform motions that stabilize the robot upon external perturbations. We define the observation space using a 197-dimensional vector,

$$\{\mathcal{O}, v, \omega, \mathcal{J}_t, \delta\mathcal{J}_{t-4,t-8}, \dot{\mathcal{J}}_{t,t-4,t-8}, \hat{\mathcal{J}}_t, \mathcal{V}, \mathcal{E}^m_z, c^m_t, \mathcal{O}^m_t\},$$

where $\delta\mathcal{J}_t$ refers to the joint position tracking error at time t, $\hat{\mathcal{J}}_t$ represents the desired joint positions output by the policy at the previous time step and $\mathcal{V}$ represents the desired reference base velocity introduced for use as a directional offset by a higher level mission planner.

B. Reward function

The behavior of an RL agent significantly depends on the reward function used during training. *Hwangbo et al.* [14] showed that introducing a sparse reward function for training an RL policy results in undesirable motions. We also illustrated this in our previous work [16] where the RL policy, without precise reward function tuning, learns to exhibit an undesired pronking locomotion behavior. Additionally, *Hwangbo et al.* observed that tuning a reward function significantly effects the performance of an RL policy - introducing higher penalty for joint torque usage, for example, results in a standing behavior whereas a lower penalty causes the RL agent to generate

unnatural and infeasible motions. As a solution to reward function tuning, a curriculum learning [36] setup could be leveraged such that the environment complexity is increased as training progresses [14]. We use this method to scale our reward function terms as training progresses in order to fine tune our RL policies.

In this work, we focus on obtaining RL policies which prioritize robot stability over aggressive locomotion behavior and therefore train the policies using a complex reward function which maximizes a stability metric represented by the *stability margin*. Furthermore, for the case of footstep planning policy we introduce terms in the reward function which encourage stepping farther from the terrain edges and slopes, and also minimize the height deviations. In order to perform domain adaptive tracking, our reward function comprises terms which penalize the trajectory tracking errors. The reward function used for training the recovery control policy is similar to that in our previous work [16], however, is augmented using the stability margin reward. The individual reward terms used for training are described in Section D of the Appendix.

1) Stability Margin: In our work, we define the stability margin as the distance between the Instantaneous Capture Point (ICP) [10] and the edges of the feasible region introduced in [37]. The margin is positive when the ICP belongs to the feasible region and negative when it lies outside of it. A positive margin corresponds to a stable robot state whereas a negative margin implies that the robot is unstable and will eventually fall unless a recovery action is introduced. If the margin is bounded, however, it means that the ICP does not diverge away from the ZMP and that the robot can still manage to recover to a stable configuration before the robot eventually falls. This is, for example, what happens for dynamic gait patterns such as trot and pace: in such cases the stability margin will be mostly negative but the robot will not fall if it manages to maintain a suitable lower bound value of stability margin at all the time.

It is therefore important for our proposed policy to maximize the stability margin to ensure the robot avoids unnecessary linear and angular accelerations and places its feet at the correct time instant and location. A maximization of the stability margin over a crawl gait (only one swing foot at the time), for example, will result in the policy trying to enlarge the support of the robot as much as possible corresponding to a larger feasible region.

The advantage of using the feasible region compared to the more common convex hull of the contact points is that the former is well defined also for multi-contact scenarios in presence of rough terrains. The feasible region, moreover, unlike the classical support region [38], does not just consider the stability problem as a mere satisfaction of the friction cone constraints but it also explicitly considers the joint-torque limits of the robot. A negative value of the stability margin, therefore, corresponds to a violation of either the friction cone constraints or the joint-torque limits (or both).

We extend the work of *Orsolino et al.* on the stability metric [37] such that the feasible region is suitable for single and double-point contact configurations. In such cases the feasible region, if exists, projects to a single point (in case of a single point-contact) and to a segment (for a double point-contact). The location of this point, or segment, might vary depending on the presence of external wrenches acting on the robot and it might not necessarily coincide with the contact points.

As detailed in Section B of the Appendix, the analytical computation of the stability margin requires more that 5 ms. For RL, which requires significantly large training samples, introducing the stability metric in the reward function would require computational time equivalent to 2x realtime for a controller operating at 400 Hz. In order to address this, we approximate the analytical solution using a feed-forward neural network trained using a supervised learning approach. Using this network, we are able to infer the stability margin of the robot in less than 4 µs.

2) Cost map: For the robot, stepping close to the edges of stairs or other polygonal obstacles, likely results in collision between the robot's shank and the obstacle eventually causing the robot to slip and fall. While collisions and slippage are already taken into account in the reward function, we further introduce explicit reward terms based on the terrain information in order to avoid the need for extensive sampling. In this regard, we explicitly penalize steps close to terrain edges computed using the 2^{nd} order derivative of the elevation map local to the desired feet position. This approach has been detailed in Section C of the Appendix.

C. Simulation

We performed training of our RL policies using the *RaiSim* [39] 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 [16] for a detailed analysis on the preference for RaiSim over other widely accessible simulation platforms. Some of the main components of our simulation setup for training the RL policies are described below.

1) Terrain generation: In order to perform training of an RL policy for terrain-aware quadrupedal locomotion, we introduce a large set of procedurally generated terrains into our simulation environment such that we obtain robust policies which can generalize to a wide domain of terrains.

To generate different terrains, we consider several objects such as a staircase, planks, bricks, unstructured sub-terrain and wave sub-terrain. Each terrain contains at least 1 and at most 5 objects. We position and rotate these objects within the terrain area (1001×1001-dimensional matrix) in order to limit height deviations around close regions and increase the occupancy over the entire terrain, and represent their corresponding heights as elevation in the range of $[0, 65535]$ where 0 corresponds to no elevation and 65535 corresponds to an elevation of 2 m. We automate this generation process using a Python tool which stores a wide range of terrains, 10k in our case, in the form of single channel 16-bit PNG files. These PNGs are then imported in RaiSim as heightmaps. Figure 6 represents the generated terrains for the unstructured ground,

Fig. 6: **(top-bottom)** represent the terrains comprising of unstructured ground, stairs, wave and brick objects respectively. The left images represent the height maps imported in the RaiSim simulator as shown on the right.

stairs, wave and brick objects along with their corresponding visualization in RaiSim.

2) Actuator Modeling: As introduced in [14], and also implemented in [16], we train an actuator network to model the actuation dynamics of the physical system. However, unlike in the case of [14], [16] where the actuators were modeled only for joint position targets with a predefined set of PD gains, in this work, we trained the actuator network using supervised learning to take as inputs the joint position and velocity targets along with the desired feed-forward torques. Since the motion controller utilizes different set of PD gains for joints of legs with foot in-contact, swing and contact-invariant (foot slippage) 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. This actuator network consists of a 21-dimensional input given by the vector,

$$\{\delta\mathcal{J}^i_{t,t-4,t-8}, \delta\dot{\mathcal{J}}^i_{t-1,t-5,t-9}, \delta\tau^i_{t-1,t-5,t-9}, \dot{\mathcal{J}}^{des,i}_{t,t-4,t-8}, \tau^{des,i}_{t,t-4,t-8}, K_{p,t,t-4,t-8}, K_{d,t,t-4,t-8}\},$$

TABLE I: Effective joint torque prediction error comparing the actuator network performance for training with and without re-sampling of the training dataset using SMOGN. Here K_p refers to the position tracking gain, K_d is the damping gain, μ_e is the mean prediction error and σ_e is the standard deviation of the prediction error observed over 50k test samples.

	K_p=8.0, K_d=0.1	K_p=35.0, K_d=0.4	K_p=160.0, K_d=0.3
Baseline	$\mu_e = 0.82,\ \sigma_e = 0.34$	$\mu_e = 0.77,\ \sigma_e = 0.31$	$\mu_e = 1.73,\ \sigma_e = 0.85$
SMOGN	$\mu_e = 0.83,\ \sigma_e = 0.21$	$\mu_e = 0.71,\ \sigma_e = 0.19$	$\mu_e = 0.96,\ \sigma_e = 0.39$

where $\delta\mathcal{J}^i_t$ represents the error between the measured and desired joint position for joint i at time t, $\delta\tau^i_t$ represents the error between the measured and desired feed-forward joint torque for joint i at time t, $\dot{\mathcal{J}}^{des,i}_t$ represents the desired joint velocity for joint i at time t, $\tau^{des,i}_t$ represents the desired feed-forward joint torque for joint i at time t, $K_{p,t}$ is the position tracking gain at time t and $K_{d,t}$ is the damping gain at time t. The network contains 2 hidden layers with 48 nodes in each of the layers and a 1-dimensional output layer representing the estimate of the joint torque observed on the physical system. We perform a forward pass through the network at each time step for each of the 12 joints of the quadrupedal robot.

To train the actuator network, we collected data from the real system using the dynamic gaits motion controller to track randomly sampled base reference velocities. We recorded the desired joint positions, velocities and feed-forward torques at 400 Hz for 12 actuators. As a result, we generated a dataset comprising of 8.64M samples in approximately 30 minutes. This was more than sufficient to model the actuation dynamics using a neural network. However, the PD gains used for tracking the actuator commands depended on the contact state of the robot. As such, we generated sufficient samples for the PD gains used for the foot contact state relating to the swing phase and in-contact with ground state. However, during operation, we obtained significantly less samples for the contact-invariant state. As a result, using the actuator network trained with the naive supervised learning approach failed to properly track actuation commands for contact-invariant states in the simulator causing the motion controller to generate unstable motion tracking commands eventually resulting in failure. To circumvent this problem. we utilized SMOGN [27] to perform regression over imbalanced data by re-sampling the dataset generated from the real robot. SMOGN utilizes a random undersampling approach for the majority samples and a synthetic sample generation method for the minority samples. Table I presents the mean absolute prediction error comparing the performance of networks trained using a naive supervised learning approach with resampled learning, tested on 50k samples for each of the different sets of PD gains. For the resampled training approach, we observe that the network prediction error is either comparable to or smaller than the prediction error of a network trained without resampling. Moreover, the standard deviation of error associated with the resampled training is less than that for the naive actuator network.

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 using the ANYmal B quadruped due to the limited size of the robot. 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: Similarly to our previous work [16], in order to obtain robust control and planning behavior, we perform domain randomization when training each of our RL policies using the following strategies.

- *Changing gravity* - We uniformly sample acceleration due to gravity between $[0.96g, 1.04g]$, where $g = 9.81\,\mathrm{m\,s^{-2}}$, to emulate inertial scaling.
- *Actuator torque scaling* - For training the recovery controller, we randomly scale the output torque of our actuator network with the scaling coefficient $s_t \in [0.9, 1.1]$ to account for differences between the real actuators and the approximated model. In case of training the domain adaptive tracking policy, we scale the outputs between $[0.85, 1.15]$.
- *Changing link mass and size* - To ensure the training does not converge to a local minimum, we scale the mass and size of each of the links by coefficients $s_m^{link} \in [0.93, 1.07]$, and $s_l^{link} \in [0.97, 1.05]$ respectively.
- *Adding actuator damping* - We use a complementary filter to emulate actuator damping given as $\mathcal{J}_t^{'} = K_{damp}\mathcal{J}_t + (1 - K_{damp})\mathcal{J}_{t-1}^{'}$ where the gain K_{damp} is randomized between $[0.8, 1]$.
- *Perturbing the robot base* - To ensure the recovery policy is able to perform state recovery, we apply external forces to the robot base along its frontal and lateral axes during trajectory tracking. We sample the external forces from a normal distribution with a mean 0 N and a standard deviation of 45 N. We apply these forces for a duration sampled randomly between $[1, 4]$ s. We also use this method while training the footstep planning and domain adaptive tracking policies. However, in these cases, we sample external forces from a normal distribution with a mean 0 N and a standard deviation of 10 N, with forces clipped between $[-30N, 30N]$.
- *Adding smoothing filters to elevation map* - Since the perception sensors present on the real robot cannot be used to perceive the elevation along regions occluded by an obstacle, we apply smoothing filters to the elevation map in order to emulate interpolation around unobservable terrain information.

D. Training

1) Footstep planning: We extend an interactive direct policy learning approach to perform guided updates of our footstep planning policy, a multilayer perceptron (MLP) comprising 3 hidden layers - $\{512, 256, 128\}$, in order to increase sample efficiency during RL policy optimization. We use the Xavier initialized [40] policy to generate desired feet position for traversal on uneven terrain for a fixed-length episode of 1024 desired feet position samples. 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 and add it to $\mathcal{D}$. After desired number of episodes used for locomotion over flat ground, we perform back-propagation through the footstep planning policy using the mean-squared error loss between the predicted and the expert samples. For rough terrain, we utilize the SAC algorithm 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.

As training progresses, we increase the complexity of the RL environment by scaling the maximum terrain elevation. As described in Section III-C1, we represent the terrain elevation matrix in the range of $[0, h_{max}]$ where h_{max} is considered to be 2 m during evaluation. During training, however, we increase h_{max} exponentially between 0 to 2 m to perform training using a curriculum learning setup.

2) Domain adaptive tracking: We use TD3 to obtain a deterministic domain adaptive tracking policy. We train this policy over 1.6B simulation samples incorporating a curriculum learning approach wherein we increase the deviations to the robot's physical parameters as training progresses. Moreover, we use the same approach of terrain elevation scaling as in the case of footstep planning to increase the environment difficulty.

3) Recovery control: The training of the recovery control policy is directly built on top of our previous work [16]. However, in our environmental setup, we prioritize external perturbations over velocity tracking. We additionally perform terrain elevation scaling during training as in the case of training the footstep planning policy.

IV. Results and Discussion

This section details the results obtained for each of the RL policies utilized by our framework. Note that, unless otherwise

Algorithm 1 Interactive Guided Policy Optimization

Input: λ_d, n_{eps}, l_{eps}, H
Initialize $\mathcal{D} = \{\}$, $\alpha = 1$
for $t = 0, 1, 2 \ldots H$ **do**
  $\theta, \mathcal{D} = \text{GUIDEDLEARNING}(\theta, \mathcal{D}, \alpha n_{eps}, l_{eps})$
  $\theta = \text{POLICYOPTIMIZATION}(\theta, (1-\alpha) n_{eps}, l_{eps})$
  $\alpha = e^{\lambda_d t}$
end for
function GUIDEDLEARNING(θ, $\mathcal{D}$, n_{eps}, l_{eps})
  for $i = 0$ **to** $n_{eps} - 1$ **do**
    for $t = 0$ **to** $l_{eps} - 1$ **do**
      Sample $a \sim \pi_\theta(a|s)$
      Generate a^* using DYNAMICGAITS(s)
      $\mathcal{D} = \mathcal{D} \cup \{(s, a, a^*)\}$
      $s, r = \text{RLENVIRONMENTSTEP}(a)$
    end for
    Sample batch of $\{(s, a_j, a_j^*)\}$ from $\mathcal{D}$
    Update θ by minimizing $\sum_j ||a_j - a_j^*||^2$
  end for
  return θ, $\mathcal{D}$
end function
function POLICYOPTIMIZATION(θ, n_{eps}, l_{eps})
  for $i = 0$ **to** $n_{eps} - 1$ **do**
    $\tau = \{\}$
    for $t = 0$ **to** $l_{eps} - 1$ **do**
      Sample $a \sim \pi_\theta(a|s)$
      $s, r = \text{RLENVIRONMENTSTEP}(a)$
      $\tau = \tau \cup (s, a, r)$
    end for
    Update θ using SAC
  end for
  return θ
end function

stated, the results presented here refer to the ANYmal B quadruped.

A. Footstep Planning

Our primary policy is a footstep planning policy trained on a large set of terrains to generate foot plans that are close to the nominal feet position of the robot while ensuring stability and minimizing foot slippage when in contact with the ground.

We evaluate our policy by measuring its success rate when traversing different terrains. We set up our simulation platform to perform 1000 runs for each of the 3 footstep planners navigating using 2 different gaits - trot and crawl. The 3 planners are our trained footstep planning RL policy, a baseline perceptive planner and a baseline blind planner. In the case of the baseline planners, we use the footholds generated by a model-based foothold planner to track a desired base reference velocity and adjust the height of the desired feet position by directly extracting the elevation of the terrain (including for the blind controller). For the case of perceptive controller we further adjust the desired feet position up to a radial deviation of 0.05 m so as to avoid stepping on an edge and minimize the terrain slope at the desired feet position. For evaluation, we generate 1000 different terrains of varying dimensions for each of the 4 object types - stairs, bricks, wave and unstructured. Each of these objects are located at the center of a $5 \times 5\text{m}^2$ terrain, with the length of the object, along the direction of the robot's frontal axis, varying between 2.0 m to 3.6 m.

1) The unstructured terrain is comprised of a sub-region of the flat terrain deformed using an introduction of random noise followed by Gaussian filter smoothing. The amplitude for the unstructured terrain is varied between 0.0125 m to 0.025 m.
2) For stairs, we generate a model with the number of steps ranging between 3 to 8. The overall height of the stair terrain is varied between 0.25 m to 0.8 m using clipped normal distribution of mean 0.3 m and standard deviation 0.1 m.
3) The wave terrain is generated using a sine function along its length with an amplitude ranging between 0.05 m to 0.1 m. We further vary the period of the sine function between $\pi/2$ to π.
4) In the case of bricks, the terrains are made of small $10 \times 10\text{cm}^2$ of random height $\{-h, 0, h\}$ where h ranges between 0.02 m to 0.05 m.

TABLE II: Success rates of policies measured for locomotion over different terrains generated for 1000 runs each.

	Crawl			Trot		
	Blind	**Perceptive**	**RL Policy**	**Blind**	**Perceptive**	**RL Policy**
Unstructured	99.6	99.4	**99.9**	**99.7**	99.6	99.6
Stairs	77.0	92.5	**93.6**	72.2	92.3	**92.7**
Wave	58.1	83.6	**94.4**	55.7	81.5	**93.0**
Bricks	52.1	71.5	**88.2**	51.1	70.9	**85.3**

The results obtained for traversal over these terrains are presented in 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 4.0 m, and regard it as a failure if (1) any of the robot's links collide with its body, (2) the base orientation with respect to the vertical gravity axis is more than 60° or (3) the robot base collides with the terrain. The success rates obtained for each of the experiments are represented in Table II. Fig. 7 presents the success rate for traversing each terrain for each of the tested controllers with two gaits.

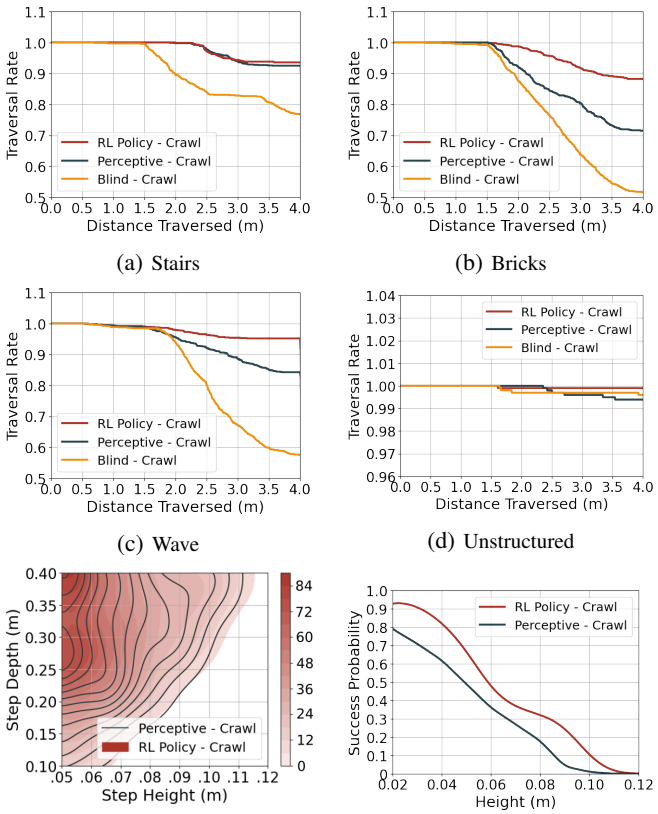
We observed that traversal over terrains using the crawl gait for each of the planners resulted in higher success rates compared to the trot gait. This was not surprising as the crawl gait is more conservative (stable 3-point contacts, as opposed to 2-point contacts). Moreover, as the test terrains became more complex, the value of the RL policy became more pronounced. Here, the whole-body controller used in our framework was mainly effective and robust to the unstructured terrain such that even the performance of the blind planner matched that of the RL policy and the perceptive planner. The RL planner, however, exhibited preference for wider stance which directly corresponds to a higher stability margin, a term introduced in the reward function during training of the RL policy. We observed a similar behavior of using a wider stance

for motion for the bricks and wave terrains. Increasing the stability margin, apart from enlarging its stance, also enables the robot to avoid foot slippage and to better handler external perturbations. To quantify the deviation from the nominal feet positions for footstep plans generated by the RL policy, we measured the feet position relative to the base frame after every cycle of 4 feet swing motions. We observed that for traversal over stairs, the mean foot position along the frontal axis of the robot was 0.35 m and 0.20 m along it's lateral, 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.

For traversal over stairs, the footstep plans generated by the RL policy were able to stably navigate the robot over each of the steps. Moreover, the perceptive planner was able to perform comparably to the RL policy due to the additional foot position adjustments introduced by the planner to step farther from an edge. With the blind controller, however, the robot often stepped close to the edges resulting in leg blockage eventually causing failure. We further investigated the success rate for traversal over stairs with different step heights and depths. This is illustrated in Fig. 7e which represents the success rate as a function of step height and depth (evaluated using *kernel density estimation* (KDE)). We observed that locomotion failed for larger step heights especially for steps with smaller depth.

Locomotion over the wave terrain was more difficult for the perceptive planner and the blind controller compared to the RL planner. In the case of the blind controller, the problem of foot slippage was significant. For the perceptive planner, however, we observed that the generated foot positions prioritized stepping near regions with lower terrain slope causing the robot to take larger steps. In this case, we observed self collision and collision of the base with the terrain resulting in failure. For the RL policy, we observed smaller foot deviations for forward motion and a wider stance enabling it to maintain its stability during motion.

The terrains comprising of bricks were the most difficult for the robot to traverse. The troughs and crests in the terrain often caused leg blockage resulting in failure which was prominent in the case of the blind controller. 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. This factor was also introduced in the reward function during training. 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, a behavior corresponding to the height deviation penalty introduced in the reward function during training. Moreover, Fig. 7f illustrates the success rate computed for different brick heights. In this case, the brick height only corresponds to the amplitude of the terrain implying that the maximum deviation in feet height is twice the



(a) Stairs (b) Bricks

(c) Wave (d) Unstructured

(e) Estimate of success rate for traversal over stairs with varying depth and height

(f) Estimate of success rate for terrains with varying brick height

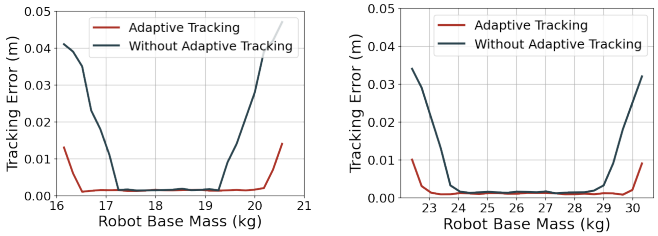
Fig. 7: **(a-d)** represent the traversal rate of each of the planning policies for the different types of terrains used for evaluation. This corresponds to the rate of success observed for the distance traversed. **(e)** is the plot representing the success rate for traversal over stairs with varying step height and depth. **(f)** represents the probability of success observed for brick terrain of varying brick heights.

height of the bricks. We observed that the RL planner fails to traverse brick terrains with maximum elevation of higher than 0.12 m. This mainly occurs due to the limitations of the physical dimensions of the quadruped.

B. Domain Adaptive Tracking

In our framework, we utilize the model-based motion controller of *Bellicoso et al.* [26] to track the generated reference foot plans. However, the gains for the motion controller are tuned specifically for the given mass of the robot, change of which would imply inaccuracies in the modeled dynamics. In this regard, the gravity compensation performed by the whole-body motion controller is inaccurate for precise motion tracking. To counteract this, the whole-body controller attempts to correct the tracking error which may result in an oscillatory behavior that can make the robot unstable.

In order to address inaccurate whole-body motion tracking during operation for factors such as variations in the robot's mass with respect to the model and actuator wear-and-tear, we train a domain adaptive tracking policy which generates a



(a) ANYmal B (b) ANYmal C

Fig. 8: The absolute mean tracking error observed before termination for varying physical parameters of the robot comparing the effects of the domain adaptive tracking policy.

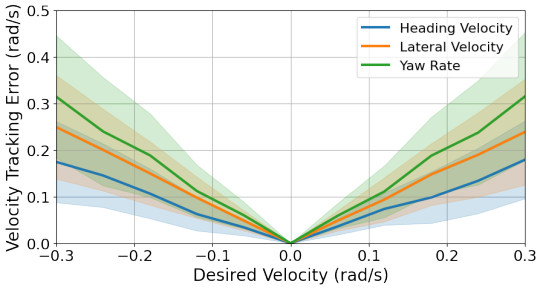


Fig. 9: Velocity tracking error observed for the domain adaptive tracking policy when running the motion controller on ANYmal C with the description parameters of ANYmal B.

corrective feed-forward torque based on a history of tracking errors. The results are presented in Movie S2.

In our experiments, the tracking error associated with the whole-body controller was not affected when the robot's base mass, of 18.3546 kg, was varied by a factor in the range of $[0.95, 1.05]$. The tracking error measured when operating across this range was negligible. However, when we decreased the mass to a factor of 0.94 or less, the motion controller eventually became unstable due to the increased oscillations observed during tracking. This was also the case when we increased the base mass by a factor of 1.06 and higher. However, introducing a domain adaptive tracking policy helped to further stabilize whole-body motion tracking. We were able to reduce the base mass down to a factor of 0.84 without increasing the amplitude of the oscillations. We also did not observe the increased oscillations when we increased the base mass up to a factor of 1.17. Figure 8a illustrates the frontal position tracking error observed before episode termination for different robot base masses. Here, the episode termination criteria include traversing an overall distance of 5.0 m or failure of whole-body control optimization.

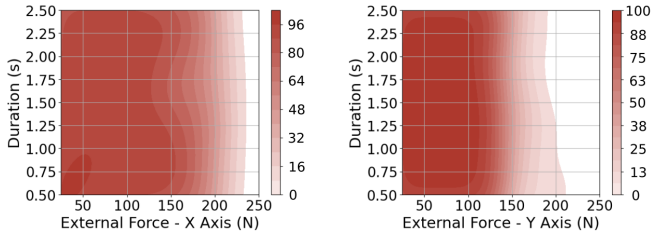
We also tested our domain adaptive tracking policy, without retraining, on the simulated model of the ANYmal C quadruped. Note that, while running experiments on the simulated ANYmal C, we increased the joint torque limit from $\pm 40\,\mathrm{N\,m}$ for the case of ANYmal B to a limit of $\pm 80\,\mathrm{N\,m}$. The tracking error measured for different base mass of the ANYmal C quadruped is represented in Fig. 8b. We observed that with an original base mass of 26.3732 kg, the whole-body motion controller could track the forward velocity command of $0.3\,\mathrm{m/s}$ even when we varied this mass with a factor included 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 also tested the performance of the domain adaptive tracking policy by transferring it to 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. Figure 9 represents the velocity tracking error observed in 1M simulation samples for randomly generated desired base velocity commands in the range of $[-0.3, 0.3]\mathrm{m\,s^{-1}}$ and $[-0.3, 0.3]\mathrm{rad\,s^{-1}}$ for the linear and angular velocities respectively. We observed that the domain adaptive policy managed to track base velocity commands up to a certain degree as opposed to the motion controller used without the domain adaptive policy, in which case the robot failed to even stand stably. Note that, in the 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 domain adaptive tracker failed to stably track the velocity commands beyond $\pm 0.4\mathrm{m\,s^{-1}}$ for heading velocity, $\pm 0.3\mathrm{m\,s^{-1}}$ for lateral velocity, and $\pm 0.5\mathrm{rad\,s^{-1}}$ for yaw rate.

C. Recovery Controller

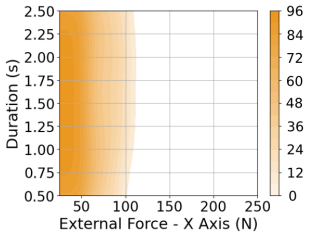
In order to account for external perturbations, we also trained an RL recovery control policy to map the robot states to low-level desired joint positions so as to withstand external forces or pushes and to maintain the robot's stability. In our experiments, we observed that despite the robustness of the model-based controller, the robot was unable to stabilize against external perturbations with forces exceeding 100 N of magnitude along the robot's base for a duration of 1 s. Furthermore, while walking over rough terrain, using a footstep planner and a whole-body controller to track the desired feet positions introduces delays in response to the disturbances. As a result, training the RL recovery control policy to stabilize over rough terrains further enables an active response to perturbations even on complex terrains.

We tested the behavior of the RL recovery control policy and the model-based controller for a range of external forces applied to the robot's base along the horizontal x-axis (frontal axis) and y-axis (lateral axis) for duration between 0.5 s and 2.5 s. The observations representing the KDE of the recovery rate as a function of the magnitude and duration of the external force for each of the controllers is as shown in Fig. 10. 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 whole-body controller which 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 N and 400 N for ANYmal B

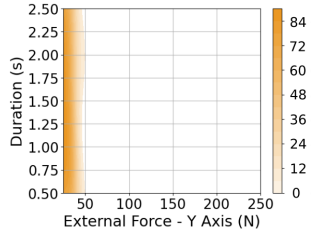


(a) RL recovery controller - ANYmal B

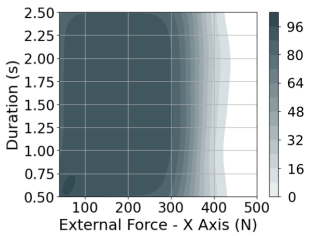
(b) RL recovery controller - ANYmal B



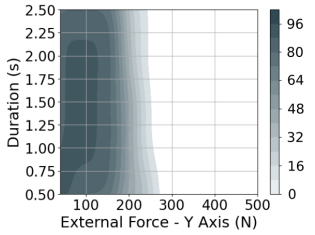
(c) model-based controller only



(d) model-based controller only



(e) RL recovery controller - ANYmal C



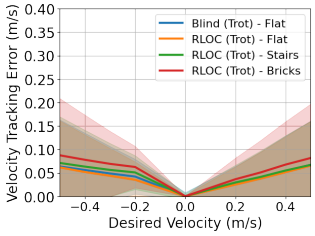
(f) RL recovery controller - ANYmal C

Fig. 10: Bivariate kernel density estimates representing the success probability of the control policies to stabilize upon external perturbations along the frontal (x) and the laeral (y) axes. Note that for **(e-f)** the x-axis scale of the plots is different.

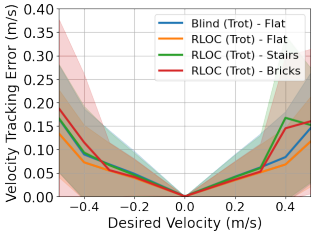
and ANYmal C respectively. It is important to note that for the case of ANYmal C, we increased the position tracking gain to 60 compared to 45 for ANYmal B and also increased the joint torque limits to $\pm 80\,\mathrm{N\,m}$ compared to $\pm 40\,\mathrm{N\,m}$ for ANYmal B. We also observed a significant increase in the ability to handle external forces laterally using the recovery control policy as compared to the model-based motion controller as is shown in Fig. 10, The results obtained are demonstrated in Movie S3.

D. RLOC

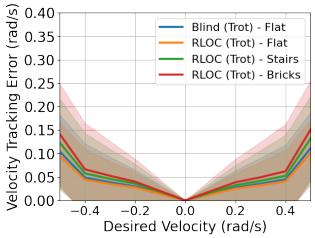
Our control framework, RLOC, offers the ability to closely track reference base velocity commands generated either by a user or a high-level goal planner. Figure 11 represents the tracking error observed for the RLOC framework for traversal over different types of terrains. In comparison to the whole-body motion controller, RLOC is better able to track the frontal and lateral velocity commands and also the yaw rate on flat terrain. This improvement in velocity tracking was a direct result of the introduction of the domain adaptive tracking policy. Removing the domain adaptive tracking policy resulted in a very similar tracking performance for both the model-based motion controller and RLOC.



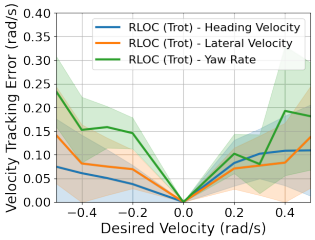
(a) Forward Velocity Tracking



(b) Lateral Velocity Tracking



(c) Yaw Rate Tracking



(d) Velocity Tracking - Gazebo

Fig. 11: **(a-c)** represent the velocity tracking error measured for locomotion over even and uneven terrain for the ANYmal B quadruped tested in the RaiSim simulator. **(d)** represents the velocity tracking error on flat ground observed for the RLOC framework in the Gazebo simulator (with different actuation dynamics).

The experimental setups for each of these terrains included the generation of a velocity command in the range of $[-0.5, 0.5]\mathrm{m/s}$ for frontal and lateral velocity tracking, and in the range of $[-0.5, 0.5]\mathrm{rad/s}$ for desired yaw rate tracking. Each episode comprised the following of a set velocity command for a duration of 5 s. For the case of tracking over stairs and bricks, we used a subset of the terrains generated for evaluation of the footstep planning policy and performed 100 runs for each velocity command. We initialized each episode by setting the robot to the center of the terrain. For bricks, we mirrored the terrains for negative velocity commands in order to be consistent with the evaluation of the control framework. Since the terrains used in the simulator are limited in dimensions, we also padded the terrain with flat surface to ensure that the robot does not fall off the edge of the uneven terrain.

In our experiments, for locomotion over bricks, the velocity tracking error was higher for each of the velocity commands compared to that for stairs. This occurred mainly due to the footstep planning policy generating feet positions closer to the robot base as described in Section IV-A. This directly manifests from our training setup to prioritize robot stability over aggressive locomotion behavior.

The switch between the dynamic gaits controller to the recovery control policy occurs when the empirically tuned limits on joint position, velocity and acceleration, and base orientation, velocity and acceleration are exceeded. We switch back to the dynamic gaits controller from the recovery control policy when the robot state is within another set of empirically tuned limits of the previously mentioned parameters. This is further presented in Movie S3. In order to quantify the effects of the recovery control policy, we performed experiments

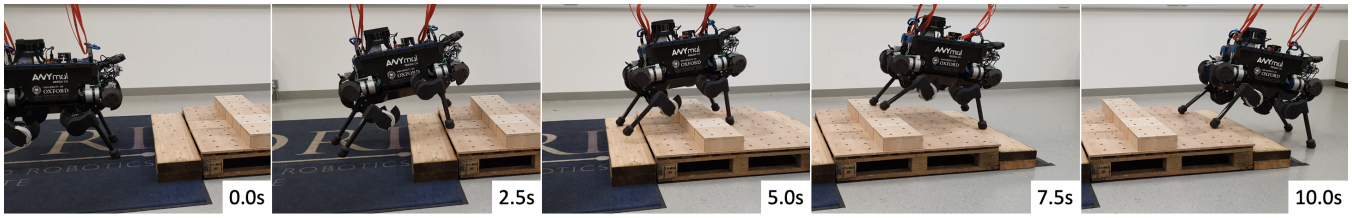


Fig. 12: Snapshots of the ANYmal B quadruped walking over a staircase using our RLOC framework.

TABLE III: Improvement in success rate (SR) observed after introducing the recovery controller (RC) for traversal over bricks and wave terrains. The episodes here refer to the number of episodes (out of 1000) in which the RC was called. Moreover, the recovery rate (RR) represents the successful recovery performed using RC. The RR and SR are expressed as %.

	RC Calls	Episodes	RR	SR w/o RC	SR with RC
Bricks	184	131	62.5	85.3	90.2
Wave	39	22	71.8	93.0	94.8

using the bricks and wave terrains for the trotting gait as introduced for evaluation of the footstep planning policy. For bricks, the recovery policy performed successful recovery at a rate of 62.5% with major failures occurring due to leg blockage. In this case, we also considered it a failure if the robot base collided with any of its legs or the terrain. We were able to increase the success rate for brick terrains from 85.3% to 90.2% with this recovery controller. Additionally, for wave terrain, introducing the recovery controller increased the success rate from 93.0% to 94.8%. These results are summarized in Table III.

As previously mentioned, we used the RaiSim simulator to train and evaluate the performance of our control framework. In order to be able to port it to the physical robot, we further evaluated the velocity tracking performance of our framework in a slower but higher resolution *Gazebo* [41] simulator as presented in Fig. 11d. The software stack for the ANYmal quadrupeds shipped by the manufacturer is based on the ROS/Gazebo framework. In this regard, ANYmal simulation has been extensively developed for Gazebo. This enables us to safely run tests which can be used to qualitatively validate the performance of our developed control framework before transferring it to the physical robot. Figure 1 presents examples of the experiments performed on the real ANYmal B quadruped. The corresponding video clips are included in Movie 1. Figure 12 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 unobservable regions sometimes resulted in the elevation map being smooth especially along the edges, and

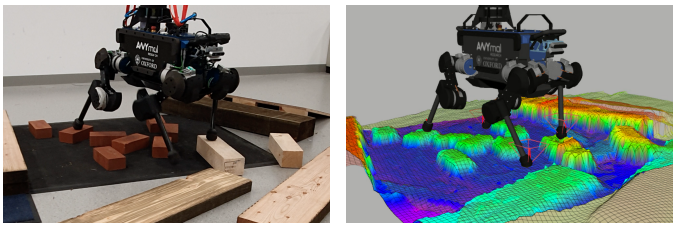


Fig. 13: ANYmal B quadruped tested on a terrain comprising ricks (left); visualization of the corresponding robot state and elevation map (right).

also slightly shifted with respect to the robot. The smoothing of the elevation map is illustrated in Fig. 13 which represents a snapshot of the real ANYmal B quadruped tested on a terrain comprising bricks, and the corresponding visualization of the robot state and the elevation map. Due to this smoothing and slight shifting of the elevation map, we observed that the robot stepped close to the edges especially in complex environments composed of a large number of smaller objects, such as bricks. The visualization of the elevation map for the test shown in Fig. 13 is also presented in Movie S4.

RLOC also enables the user to switch between locomotion gaits online. This is demonstrated in Movie S4.

V. 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. We substituted a high-level model-based footstep planner with our RL footstep planning policy trained using exploratory reward-feedback to better take into account the capabilities and limits of the robot and its controllers. Moreover, we enhanced the robustness of the model-based controllers by adding a corrective policy and a recovery policy, both trained using RL.

The relevance of our approach was demonstrated in the context of perceptive 3D locomotion where we trained perceptive planning and control policies which we incorporated into our modular control framework.

The integration of these different building blocks allowed us to greatly improve the success of traversal over complex terrains. For trotting over the bricks environment, as presented in Section IV, we showed that our hybrid approach, RLOC, was able to increase the success rate from 71.5% to 90.2% in comparison to our baseline perceptive controller. We showed that we were able to traverse reliably (i.e. with a success rate of 90% or above over 4m) and with a dynamic gait (trot at

an effective velocity of approximately $0.25\mathrm{m\,s^{-1}}$ on the real robot) over any terrain with elevation differences of 8cm or less, even though the model-based locomotion controller was only designed for quasi-flat terrains.

While those results are encouraging, we still find walking up standardized stairs challenging. This can partially be attributed to the physical limits of ANYmal B, but also could result from the CoM motion planner that uses *Zero Moment Point*, implicitly assuming constant base height and attitude. In future work we aim to use a more general planner that can provide examples of changing base pitch and roll for training.

Another direction for future work is adjusting the robot velocity together with the footstep policy. In the current approach, the policy tracks the commanded velocity directly, as used in the CoM motion planner. This means that the footsteps can only be adjusted locally and the policy cannot go around an obstacle even if it detects that the robot will not be able to walk over it.

APPENDIX

A. Motion Controller

The dynamic gaits motion controller [26] used in our work comprises of the following modules:

1) *Contact scheduler* which plans swing and stance phase timings of each leg based on predefined gait patterns. This scheduler also supports dynamic transitioning between locomotion gaits.
2) *Foothold optimizer* to compute the next foothold position of each leg using a linear inverted pendulum model. The problem is formulated as a quadratic problem which minimizes deviations from reference footholds (which follow the desired robot velocity), the previously computed footholds and a stabilization term based on the linear inverted pendulum model while ensuring that the footholds lie within the kinematic limits of the robot.
3) *Center of mass (CoM) motion planner* which plans the CoM motion over one period of the gait. This problem is formulated as a non-linear optimization problem minimizing accelerations, deviations from the reference trajectory and distances to the initial and final state of the motion plan while constraining the *Zero-Moment Point* (ZMP) [42] to stay inside the support polygon.
4) *Foot motion planner* which plans swing trajectories for each of the legs represented as splines where the start node is the current foot position, the end node is the next foothold and the middle node is the highest point of the swing. In our work, for traversal over non-flat terrain, we set the height of the middle nodes to be the maximum terrain height observed between the start and the end nodes, and additionally introduce a margin of 0.05 m to ensure that the foot does not collide with the terrain during swing motion in case of tracking errors.
5) *Whole-body controller* which computes the joint reference positions, velocities and torques to track the CoM and feet trajectories generated by the aforementioned planners. In this case, the whole-body control is also represented as a quadratic problem which minimizes the tracking and contact forces while respecting the floating base equation of motion, the contact motion, the friction cone, and the torque limits of the actuators.

B. Stability Margin

The computation of the stability margin requires the estimation of the feasible region, which can be done by means of an iterative projection algorithm up to an arbitrary tolerance [38]. This procedure requires the solution of multiple linear programs and, even for low tolerance values, in our current Python implementation, the estimation of the edges of the feasible region requires more than 5 ms on a normal desktop PC. Even for an optimized implementation in the C++ language, we estimate the computation of this region to be one order of magnitude smaller which is still not sufficiently fast to be used for training RL policies. For this reason we trained an MLP as an approximation of the analytical solution to perform stability estimation. In our C++ implementation, we can perform a forward pass through this MLP in less than 4 µs.

We use an MLP with 2 hidden layers of size $\{48, 48\}$ which maps a 47-dimensional input into a scalar stability margin. The input vector comprises of

$$\{\mathcal{O}_t^{base}, v_t, \omega_t, \dot{\omega}_t, \overrightarrow{F}, \overrightarrow{\tau}, {f_t}^{base}, f_c, N_c\}$$

where $\mathcal{O}_t^{base}$ represents the base orientation at time t, v_t is the linear base velocity, ω_t is the angular base velocity, $\dot{\omega}_t$ is the angular acceleration, $\overrightarrow{F}$ is the external force on base, $\overrightarrow{\tau}$ is the external torque applied on base, ${f_t}^{base}$ is the position of the feet in the base frame, f_c presents the contact state of each of the foot and N_c represents the direction of contact normals.

Upon training the stability margin network, we observed an absolute prediction error mean of 0.0051 m and a standard deviation of 0.0018 m when tested on a dataset comprising of 50k test samples.

In addition to efficiently obtaining an estimate of the stability margin, the use of a neural network, also enables computation of partial derivatives of the stability margin with respect to the robot's states thereby eliminating the need to compute the Jacobians by finite differences of the analytical solution.

This provides the possibility to formulate optimal control problems and gradient-descent based trajectory optimization problems that exploit the jacobian with respect to the optimization variables of costs and constraints based on this MLP in order to maximize the overall stability margin or to make sure that this quantity lies within some specified boundaries. Fig. 14 represents the values of the stability margin obtained for a straight walk of Anymal B on a flat terrain for 2s covering a distance of 0.5m. Both the motions were obtained using the TOWR [43] library, which employs a minimal parameterization of the state of the robot based on the single rigid body dynamics model. The red values refer to a motion plan obtained with the baseline library, which attempts to obtain a solution that satisfies all the feasibility constraints such as kinematic reachability and dynamic consistency without any minimization cost. The blue curve, instead, refers to the same

trajectory optimization problem where a cost has been added that maximizes the learned stability margin. As an effect of the added cost, the obtained solution presents smaller acceleration peaks and a larger stance configuration of the feet. These two elements together result in an increased value of the stability margin both during the triple and the quadruple support phases and in an average improvement of the stability margin of about 0.025 m. We observed a similar behavior while training our RL footstep planning policy. For a higher stability margin reward coefficient we observed that the footstep planning policy preferred a wider stance behavior. Upon increasing the reward coefficient for stepping close to the nominal feet position and lowering the stability margin coefficient, we observed that the footstep planning policy preferred a wider stance behavior only during unstable motion phases. This is demonstrated in Movie S5.

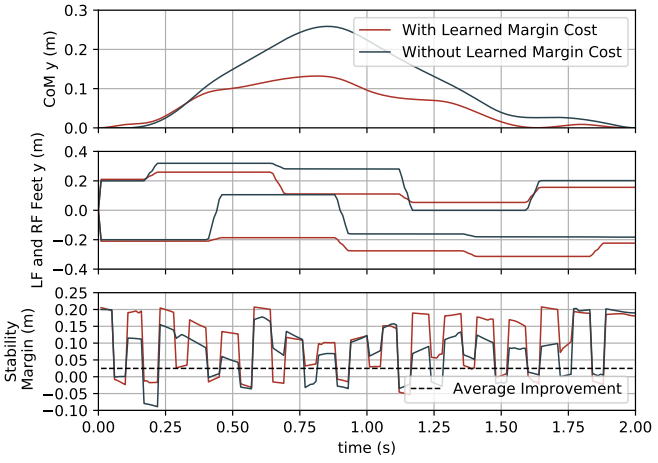


Fig. 14: Trajectories referring to two motion plans of Anymal B on flat terrain optimized using TOWR with (blue) and without (red) the cost on the learned stability margin. We can see that maximizing the learned stability margin leads to a decrease in unnecessary base motions (upper plot) and in a larger stance (mid plot). These two elements together lead to an average improvement of the stability margin of about 2.5 cm (lower plot).

In our preliminary experiments, we observed that introducing the stability margin reward term in an environment setup with a simple reward function results in reduced base accelerations. As shown in Movie 1, a locomotion policy obtained using the reward function $-(0.7 - v_x^{meas})^2$, where v_x^{meas} represents the robot's measured heading velocity, is significantly more aggressive compared to the reward function augmented using the stability margin. This reward function can further be tuned to obtain the desired locomotion behavior. In our RL environment setups, we use the stability margin in our reward function to encourage the RL agents to perform less aggressive maneuvers in order to maximize the stability metric.

C. Cost Map

In order to drive the RL footstep planning policy to generate desired feet positions which are away from the terrain edges,

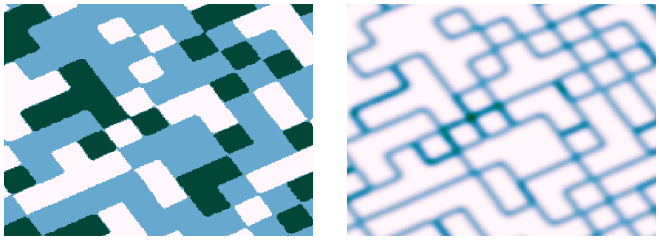


Fig. 15: Example of computation of the cost map. **left:** Elevation Map of a Bricks terrain (green color represents higher elevation while white color represents holes). **right:** Cost Map (dark blue represents position with a higher cost). The edges of the terrains have a high cost. Moreover, higher height differences (i.e. edges between green and white elevations) have a higher cost, over a wider surface.

we compute a penalty based on the cost map obtained using the 1st and 2nd order derivatives of the elevation map local to the feet positions with a radius of 0.05m. We compute these costs by performing convolutions in the following order:

1) We filter the elevation map using a smoothing kernel given as
$$k_s = \begin{bmatrix} 0.1 & 0.1 & 0.1 \\ 0.1 & 0.2 & 0.1 \\ 0.1 & 0.1 & 0.1 \end{bmatrix}$$
2) We use the Laplacian kernel as an approximation of the second order derivative given by the kernel
$$k_l = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$
3) We then use the absolute of the map obtained in order to represent the edges with the direction information.
4) We obtain the intensity of the change in heights along the edges using the blurring kernel
$$k_b = \frac{1}{9} \times \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$
5) We finally use a circular distance filter such that the edges at the center of the elevation map local to the desired foot position have the maximum weight for computing the cost penalty. This weight decreases with increase in the radial distance from the center of the feet.

Figure 15 shows an example of the cost map on a terrains with bricks and holes. In practice, the whole cost map does not need to be explicitly computed since only the cost at each foot position is needed so the transformation are only applied on small patches around the selected footholds.

D. Reward

1) Footstep planner: The reward function used for training the footstep planner encourages the RL agent to generate foot plans which are close to the robot's nominal feet position while tracking a reference velocity command. Additionally, the reward design prioritizes robot stability and further drives the

agent to step farther from edges and slopes. In order to evaluate each sample of the footstep planning policy, we perform four feet swing motions using the *Dynamic Gaits* motion controller. In this regard, we use a recurrent reward function to account for the behavior during the feet swing motions and a final reward function which represents the stability of the robot during the stance phase.

The recurrent reward terms used to train our footstep planning policy are presented in Table IV. The final reward terms are presented in Table V.

TABLE IV: Recurrent reward terms for the MDP formulation of the footstep planning policy. Here K refers to the logistic kernel defined as $K(x) = (e^x + 2 + e^{-x})^{-1}$, $\mathcal{V}^{lin}$ is the desired linear velocity in base frame, τ is the joint torque, $\mathcal{V}^{ang}$ is the desired angular velocity in base frame, v^{foot}_{world} is the foot velocity in world frame, and s_m is the robot's stability margin.

Term	Expression
Linear Velocity	$K\left(v - \mathcal{V}^{lin}\right)$
Torque	$\|\tau\|^2$
Angular Velocity	$K\left(\omega - \mathcal{V}^{ang}\right)$
Foot Slip	$\|\|v^{foot}_{world}\|\|^2$
Stability	s_m

TABLE V: Final reward terms for training of the footstep planning policy. Here $\hat{f}_p$ refers to the desired feet position generated by the footstep planner, f^n_p represents the nominal feet position, c^e_m represents the cost map term for a desired foot position, and $\hat{f}_h$ represents the nominal height of the foot in the base frame.

Term	Expression
Deviation from Nominal Feet Position	$\|\|\hat{f}_p - f^n_p\|\|^2$
Distance from Edges	$\sum c^e_m$
Foot Slip	$\|\|v^{foot}_{world}\|\|^2$
Stability	s_m
Foot Height Deviation	$K(\hat{f}_h - f_h)$

2) Domain adaptive tracker: For training the domain adaptive tracking policy, we use a comparatively simple reward function given by the error between the desired state of the robot as generated by the motion controller and the current robot state as estimated from the simulator. We also introduce the stability margin, to ensure oscillatory behavior observed during training due to change in robot's mass is minimized.

3) Recovery controller: We use the reward terms represented in Table VI to train our recovery control policy. Even though the reward function includes velocity tracking, we only use this as an offset for directing the motion of the robot while performing recovery. During training, we majorly focus on the response to external perturbations. We included the velocity tracking for a higher-level goal planner to further direct the recovery motion.

TABLE VI: Reward terms for used for training the recovery control policy. Here $\mathcal{J}_t$ represents the joint positions at time t and $\hat{f}_h$ is the desired height of foot when in swing.

Term	Expression
Linear Velocity	$K\left(v - \mathcal{V}^{lin}\right)$
Torque	$\|\tau\|^2$
Angular Velocity	$K\left(\omega - \mathcal{V}^{ang}\right)$
Foot Acceleration	$\|\|v^{foot}_{world,t} - v^{foot}_{world,t-1}\|\|^2$
Foot Slip	$\|\|v^{foot}_{world}\|\|^2$
Smoothness	$\|\|\mathcal{J}_t - \mathcal{J}_{t-1}\|\|^2$
Stability	s_m
Joint Speed	$\left\|\max(\|\dot{\mathcal{J}}_t\| - \dot{\mathcal{J}}^{limit}, 0)\right\|^2$
Joint Acceleration	$\left\|\max(\|\ddot{\mathcal{J}}_t\| - \ddot{\mathcal{J}}^{limit}, 0)\right\|^2$
Foot Clearance	$\sum_i \left(f_{h,i} - \hat{f}_h\right)^2 \left\|v_{f,i}\right\|^2$

ACKNOWLEDGEMENT

S.G. trained the actuator network, stability margin network and the denoising convolutional autoencoder. S.G. setup the RL environments and performed training of footstep planning, domain adaptive tracking and recovery control policies. M.G. extended the model-based motion controller for locomotion over uneven terrain. R.O. extended the stability margin analysis to dynamic motions with 2 feet contacts. S.G. and M.G. wrote the control software for the real robot. S.G., M.G and R.O. performed hardware experiments. I.H. and M.F. provided funding and supervised the project. S.G. took the lead in writing the manuscript. M.G., R.O., M.F. and I.H. provided critical feedback and helped shape the manuscript.

REFERENCES

[1] M. Hutter, C. Gehring, D. Jud, A. Lauber, C. D. Bellicoso, V. Tsounis, J. Hwangbo, K. Bodie, P. Fankhauser, M. Bloesch *et al.*, "Anymal-a highly mobile and dynamic quadrupedal robot," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2016, pp. 38–44.

[2] E. Ackerman, "North sea deployment shows how quadruped robots can be commercially useful," *IEEE Spectrum. https://spectrum.ieee.org/automaton/robotics/industrial-robots/north-sea-deployment-shows-how-quadruped-robots-can-be-commercially-useful*, November 2018.

[3] H. Kolvenbach, D. Wisth, R. Buchanan, G. Valsecchi, R. Grandia, M. Fallon, and M. Hutter, "Towards autonomous inspection of concrete deterioration in sewers with legged robots," *Journal of Field Robotics*, 2020. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.21964

[4] E. Ackerman, "Anybotics introduces sleek new anymal c quadruped," *IEEE Spectrum. https://spectrum.ieee.org/automaton/robotics/industrial-robots/anybotics-introduces-sleek-new-anymal-c-quadruped*, 2019.

[5] R. Full and D. Koditschek, "Templates and anchors: neuromechanical hypotheses of legged locomotion on land," *Journal of Experimental Biology*, vol. 202, no. 23, pp. 3325–3332, 1999. [Online]. Available: http://jeb.biologists.org/content/202/23/3325

[6] 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 *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, vol. 1, Oct 2001, pp. 239–246 vol.1.

[7] I. Poulakakis, E. Papadopoulos, and M. Buehler, "On the stability of the passive dynamics of quadrupedal running with a bounding gait," *The International Journal of Robotics Research*, vol. 25, no. 7, pp. 669–687, 2006. [Online]. Available: https://doi.org/10.1177/0278364906066768

[8] I. Poulakakis and J. W. Grizzle, "The spring loaded inverted pendulum as the hybrid zero dynamics of an asymmetric hopper," *IEEE Transactions on Automatic Control*, vol. 54, no. 8, pp. 1779–1793, 2009.

[9] M. Vukobratović and D. Juričić, "Contribution to the synthesis of biped gait," *Proc. IFAC Symp. Technical and Biological Problem on Control*, Erevan, USSR, 1968.

[10] J. Pratt, J. Carff, S. Drakunov, and A. Goswami, "Capture point: A step toward humanoid push recovery," pp. 200 – 207, 01 2007.
[11] 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
[12] H. Dai and R. Tedrake, "Planning robust walking motion on uneven terrain via convex optimization," in *Humanoids*, 2016.
[13] R. Orsolino, M. Focchi, C. Mastalli, H. Dai, D. G. Caldwell, and C. Semini, "Application of wrench based feasibility analysis to the online trajectory optimization of legged robots," *IEEE Robotics and Automation Letters (RA-L)*, 2018.
[14] J. Hwangbo, J. Lee, A. Dosovitskiy, D. Bellicoso, V. Tsounis, V. Koltun, and M. Hutter, "Learning agile and dynamic motor skills for legged robots," *Science Robotics*, vol. 4, no. 26, 2019. [Online]. Available: https://robotics.sciencemag.org/content/4/26/eaau5872
[15] J. Lee, J. Hwangbo, and M. Hutter, "Robust recovery controller for a quadrupedal robot using deep reinforcement learning," *arXiv preprint arXiv:1901.07517*, 2019.
[16] S. Gangapurwala, A. Mitchell, and I. Havoutis, "Guided constrained policy optimization for dynamic quadrupedal robot locomotion," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 3642–3649, 2020.
[17] X. B. Peng, E. Coumans, T. Zhang, T.-W. Lee, J. Tan, and S. Levine, "Learning agile robotic locomotion skills by imitating animals," *arXiv preprint arXiv:2004.00784*, 2020.
[18] Z. Xie, G. Berseth, P. Clary, J. Hurst, and M. van de Panne, "Feedback control for cassie with deep reinforcement learning," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 1241–1246.
[19] S. Levine and V. Koltun, "Guided policy search," in *International Conference on Machine Learning*, 2013, pp. 1–9.
[20] T. Haarnoja, S. Ha, A. Zhou, J. Tan, G. Tucker, and S. Levine, "Learning to walk via deep reinforcement learning," *arXiv preprint arXiv:1812.11103*, 2018.
[21] Y. Yang, K. Caluwaerts, A. Iscen, T. Zhang, J. Tan, and V. Sindhwani, "Data efficient reinforcement learning for legged robots," in *Conference on Robot Learning*, 2020, pp. 1–10.
[22] 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 *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 23–30.
[23] X. B. Peng, G. Berseth, and M. Van de Panne, "Terrain-adaptive locomotion skills using deep reinforcement learning," *ACM Transactions on Graphics (TOG)*, vol. 35, no. 4, pp. 1–12, 2016.
[24] V. Tsounis, M. Alge, J. Lee, F. Farshidian, and M. Hutter, "Deepgait: Planning and control of quadrupedal gaits using deep reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 3699–3706, 2020.
[25] J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter, "Learning quadrupedal locomotion over challenging terrain," *Science robotics*, vol. 5, no. 47, 2020.
[26] C. D. Bellicoso, F. Jenelten, C. Gehring, and M. Hutter, "Dynamic locomotion through online nonlinear motion optimization for quadrupedal robots," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 2261–2268, 2018.
[27] P. Branco, L. Torgo, and R. P. Ribeiro, "Smogn: a pre-processing approach for imbalanced regression," in *First International Workshop on Learning with Imbalanced Domains: Theory and Applications*, 2017, pp. 36–50.
[28] R. S. Sutton, A. G. Barto *et al.*, *Introduction to reinforcement learning*. MIT press Cambridge, 1998, vol. 135.
[29] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," *arXiv preprint arXiv:1801.01290*, 2018.
[30] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International conference on machine learning*, 2015, pp. 1889–1897.
[31] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.
[32] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel *et al.*, "Soft actor-critic algorithms and applications," *arXiv preprint arXiv:1812.05905*, 2018.
[33] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," in *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*, 2016.
[34] S. Fujimoto, H. Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *International Conference on Machine Learning*, 2018, pp. 1582–1591.
[35] P. Fankhauser, M. Bloesch, and M. Hutter, "Probabilistic terrain mapping for mobile robots with uncertain localization," *IEEE Robotics and Automation Letters (RA-L)*, vol. 3, no. 4, pp. 3019–3026, 2018.
[36] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," in *Proceedings of the 26th annual international conference on machine learning*, 2009, pp. 41–48.
[37] R. Orsolino, M. Focchi, S. Caron, G. Raiola, V. Barasuol, and C. Semini, "Feasible region: an actuation-aware extension of the support region," *IEEE Transactions on Robotics (TRO)*, 2020.
[38] T. Bretl and S. Lall, "Testing static equilibrium for legged robots," *IEEE Transactions on Robotics*, 2008.
[39] J. Hwangbo, J. Lee, and M. Hutter, "Per-contact iteration method for solving contact dynamics," *IEEE Robotics and Automation Letters*, vol. 3, no. 2, pp. 895–902, 2018.
[40] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, 2010, pp. 249–256.
[41] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, vol. 3. IEEE, 2004, pp. 2149–2154.
[42] M. Vukobratović and B. Borovac, "Zero-moment point—thirty five years of its life," *International journal of humanoid robotics*, vol. 1, no. 01, pp. 157–173, 2004.
[43] A. W. Winkler, C. D. Bellicoso, M. Hutter, and J. Buchli, "Gait and Trajectory Optimization for Legged Systems through Phase-based End-Effector Parameterization," *IEEE Robotics and Automation Letters (RA-L)*, pp. 1–1, 2018. [Online]. Available: http://ieeexplore.ieee.org/document/8283570/

Siddhant Gangapurwala completed his Bachelor of Engineering (B.E.) in Electronics from the University of Mumbai in 2016. In 2017, he joined the Autonomous Intelligent Machines and Systems (AIMS) program as a doctoral candidate at the University of Oxford. The following year, Siddhant joined the Dynamic Robot Systems (DRS) group of the Oxford Robotics Insitute (ORI) to pursue his DPhil degree with focus on machine learning and optimal control based approaches for robotic locomotion over uneven terrain where he continues to work as a doctoral researcher.

Mathieu Geisert received his M. Eng. 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, at LAAS-CNRS (Toulouse, France) where he worked for 5 years in different positions: as an intern, engineer, Ph.D. student and finally as a post doctoral researcher. Since 2018, He is a post doctoral researcher in the Dynamic Robot Systems (DRS) group of the Oxford Robotics Institute (ORI). His research focuses on Planning, Optimal Control and Machine Learning for legged robots.

Romeo Orsolino completed his B.Sc. in mechanical engineering from the University of Genova in 2013 and his M.Sc. in robotics engineering at the École Centrale de Nantes in 2015. In the same year, he then joined the DLS team at IIT for a Ph.D. focusing on motion planning for legged locomotion in rough terrains. After successfully defending his Ph.D. thesis in February 2019, Romeo remained at the DLS lab as a post doctoral researcher until September 2019. Since October 2019, he is a post doctoral researcher working as part of DRS where he continues to pursue his research on multi-contact motion planning, optimal control, dynamics and perception.

Maurice Fallon studied Electronic Engineering at University College Dublin. His Ph.D. research in the field of acoustic source tracking was carried out in the Engineering Department of the University of Cambridge.

Immediately after his Ph.D. he moved to MIT as a post doctoral researcher and later research scientist in the Marine Robotics Group (2008-2012). From 2012-2015 he was the perception lead of MIT's team in the DARPA Robotics Challenge – a multi-year competition developing technologies for semi-autonomous humanoid exploration and manipulation in disaster situations. The MIT DRC team competed in several phases of the international competition, finishing 3rd, 6th and finally 7th.

From 2015, he was a Lecturer at University of Edinburgh where he led research in collaboration with NASA's humanoid robotics program. He moved to Oxford in April 2017 and took up the Royal Society University Research Fellowship in October 2017.

Ioannis Havoutis is a Departmental Lecturer at the Oxford Robotics Institute. His research combines dynamic whole-body motion planning and control with machine learning, focusing on robots with arms and legs.

He received his Ph.D. (2011) and M.Sc. with distinction (2007) from the University of Edinburgh, where he worked on machine learning for motion planning and control of articulated robots.

After his studies, he joined the Dynamic Legged Systems Lab at the Advanced Robotics Department of the Italian Institute of Technology, where he worked on a hydraulically-actuated, fully torque-controlled quadruped robot. There he led the Locomotion Group within the HyQ team, while his work focused on dynamic motion planning and control for legged locomotion.

He was a postdoctoral researcher at the Robot Learning & Interaction Group of the Idiap Research Institute, where he worked on learning complex skills from demonstration. His work there mainly focused on skill representation, online Bayesian nonparametric learning, Riemannian manifold methods and optimal control for motion generation.