

©2019 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

Robust Recovery Controller for a Quadrupedal Robot using Deep Reinforcement Learning

Joonho Lee, Jemin Hwangbo, and Marco Hutter

Robotic Systems Lab, ETH Zurich

E-mails: {jolee, jhwangbo, mahutter}@ethz.ch

Abstract—The ability to recover from a fall is an essential feature for a legged robot to navigate in challenging environments robustly. Until today, there has been very little progress on this topic. Current solutions mostly build upon (heuristically) predefined trajectories, resulting in unnatural behaviors and requiring considerable effort in engineering system-specific components. In this paper, we present an approach based on model-free Deep Reinforcement Learning (RL) to control recovery maneuvers of quadrupedal robots using a hierarchical behavior-based controller. The controller consists of four neural network policies including three behaviors and one behavior selector to coordinate them. Each of them is trained individually in simulation and deployed directly on a real system. We experimentally validate our approach on the quadrupedal robot ANYmal, which is a dog-sized quadrupedal system with 12 degrees of freedom. With our method, ANYmal manifests dynamic and reactive recovery behaviors to recover from an arbitrary fall configuration within less than 5 seconds. We tested the recovery maneuver more than 100 times, and the success rate was higher than 97 %.

I. INTRODUCTION

In case of a fall, animals show the remarkable ability to recover from any posture by pushing against their surroundings and swinging limbs to gain momentum. Having similar abilities in legged robots would significantly improve their robustness against failure and extend their applicability in harsh environments. We address this topic in the present work by developing a control strategy for the robust recovery maneuver of quadrupedal robots. By recovery maneuver, we mean the maneuver of returning to a typical operating state (standing or walking) from a fall as shown in the Fig. 1. For such maneuver, the robot needs to produce motions that make good use of the interactions with the ground and swinging motion of the legs while avoiding self-collisions. Optimization-based methods [17] have a hard time to solve such a task as they usually require analytic dynamic models and often predefined contact sequences [1], which are both hard to find as the system can interact at multiple uncertain contact points or patches. Moreover, all control methods that are based on simplified template models are not valid anymore in such fall configuration. Existing methods in recovery controllers simplify the problem by using handcrafted control sequences [24], [25] or using simplified models [21], or even adding mechanisms such as a tail or extra limbs [5], [7]. Consequently, they exhibit predictable behavioral patterns, which limit their robustness in corner cases (e.g., when a robot’s legs get stuck below its base). They also require a considerable engineering effort.

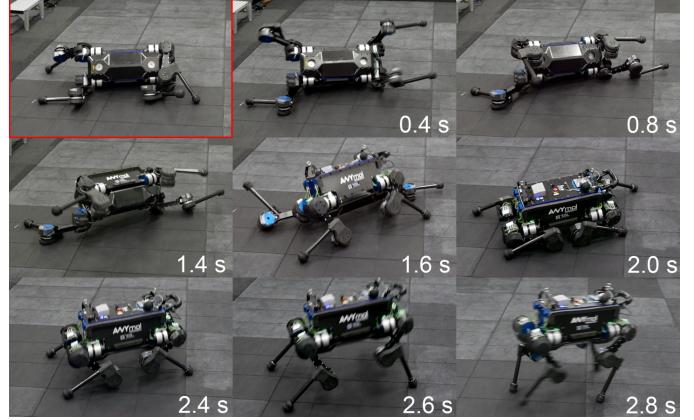


Fig. 1: A recovery maneuver of ANYmal. (Top left) ANYmal is initialized at a fall configuration. (Top row) ANYmal swings its legs to gain momentum, (Middle row) pushes the ground to regain the upright and stable posture, and then (Bottom row) stands up and walks.

A promising alternative for generating self-righting behaviors is model-free Deep Reinforcement Learning (RL). In this paradigm, an agent interacts with its environment and learns the dynamics and a control policy from the experience. By using model-free methods, we can generate control policies without any abstraction in modeling complex dynamics. Unfortunately, existing model-free algorithms typically require an excessive number of trial-and-errors to obtain a performant policy, and hence it often becomes impractical to train a policy on sophisticated hardware. This is particularly true for dynamic systems like legged robots where bad policies can be fatal. To overcome these limitations, existing works leverage simulations where one can generate massive data at no cost and with high consistency. Very recent and promising research results in the field of legged robotics have demonstrated that learned locomotion policies can be transferred from simulation to reality [13], [26]. In order to realize this transfer and overcome the so-called reality gap, it was important to use high-fidelity simulations. In [26] this was achieved by model parameter estimation, while [13] proposed a method to learn parts of the simulated model from real data. Beside traditional locomotion, the latter work also demonstrated a sophisticated self-righting policy that can be trained in a few hours in the simulation. However, it is still challenging to train a

single policy that can manifest multiple behaviors including self-righting, standing up, locomotion or others, because the present Deep RL algorithms have limited capability in learning multiple skills. It often shows the degradation of the performance in individual tasks in case of learning multiple skills at once (e.g., Multitasker in [2]). A simple yet powerful solution to incorporate multiple behaviors is to divide-and-conquer. In this approach, a control problem is decomposed into several behaviors, and then the behaviors are coordinated either by hand [6] or learning a rule for behavior selection. The behaviors and the selection rule can be learned separately [16], [14] or simultaneously [8], [19]. For separate learning, a high-level behavior selector is trained using a set of pre-learned behaviors. This is a widely adopted approach in behavior-based robotics [15] and has shown success in high-dimensional continuous control problems recently. Merel et al. [16] and Liu and Hodgins [14] demonstrated human-like behaviors in simulation with a hierarchically structured controller that consists of multiple low-level controllers and an agent to organize them in a task-oriented way.

In this paper we present a hierarchically structured controller consisting only of neural networks that is able to generate complex combined maneuvers like recovering from a fall. we build up the complex skill set from individual behaviors including self-righting, standing up, and locomotion. The self-righting and locomotion behaviors are introduced in the previous work [13]. We reproduced the self-righting behavior with different action representation and reused existing policy for the locomotion behavior. These behaviors are trained using Trust Region Policy Optimization (TRPO) [22] using the simulation framework presented in the existing work [13]. We take inspiration from [16] and [14], and learn the complex behavior selection subsequently. Moreover, we introduce a novel learning-based state estimator that is learned in parallel to the behavior selection and which even works in degenerated conditions (i.e., when fallen on the ground and being in a complex, unobservable contact condition).

With these elements, we are able to generate a recovery controller of unprecedented robustness. The system was tested on the quadrupedal robot ANYmal [10] in more than 100 trials with a success rate higher than 97 %. Thereby, we showed that our method can cope with all kinds of corner cases for which previous solutions failed. As the propose controller is not based on any heuristics, it has the potential to be applicable for a wide variety of complex skill sets and hence bring our robots as step closer to their natural counterparts.

II. METHOD

In this section, we first provide an overview of our method and then describe the details of the designs and implementation of each component.

A. Overview

Fig. 2 shows an overview of the control framework. The behavior selector and three behaviors form a hierarchical behavior-based controller. Each behavior is a control policy

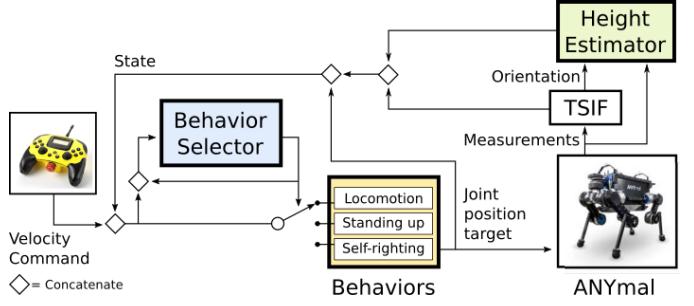


Fig. 2: Control architecture for the recovery controller. TSIF refers to the Two State Implicit Filter [4].

for individual behaviors. The behavior selector selects the most appropriate behavior for the current situation depending on the recent observations, commands, previously chosen behavior type and the previous action. At each time step, the control policy for the chosen behavior sends commands to the actuators. The height estimator is a neural network regression module for estimating base height during the deployment on the real system.

We decomposed the control task into three behaviors: self-righting, standing up, and locomotion. This decomposition is to cope with the difficulty of training a single policy that can manifest all of the necessary behaviors. From our experience, a policy trained to perform multiple tasks shows ungraceful behaviors such as frequent slippages and highly conservative postures. It also requires a lot more effort to come up with an effective cost function that leads to natural motions because the desired controller has to cover larger state space. Learning three behaviors separately simplifies the cost function design and enables us to troubleshoot each control policy separately on the real system.

During the deployment on the real system, the estimated states from the Two State Implicit Filter (TSIF) [4] are used. Additionally, we used a neural network named height estimator to estimate the base height (z -position of the base in the world frame) to resolve the drift issues in linear position. The definition of the state spaces and the characteristics of TSIF are provided in the section II-B.

Each behavior is separately trained and tested on the real robot, whereby training is done only in simulation. Subsequently, the behavior selector and a height estimator are trained using the set of pretrained behaviors. The simulated environments for learning consists of the data-driven actuator model [13] and the fast contact solver presented in [12], which efficiently generates high-fidelity samples. Trust Region Policy Optimization algorithm [22] with the Generalized Advantage Estimator [23] (TRPO+GAE) is used for learning. Although stochastic policies are used during training, the variances are reduced to 0 during deployment to ensure a more consistent behavior.

B. Feature Selection for the State Spaces

For each behavior, we select the most representative and reliable set of states as given in table I. The existing state

Function	Data
Self-Righting Policy	Gravity vector (e_g)
	Base angular velocity in body frame (ω_{IB}^B)
	Joint positions (ϕ_j)
	Joint velocities ($\dot{\phi}_j$)
	History of joint position error & velocity
Standing Up Policy	Previous joint position targets (a_{t-1})
	Base linear velocity in body frame (v_{IB}^B)
Locomotion Policy	State space of the Self-Righting policy
	Velocity commands
	Estimated base height (h_e)
Behavior Selector	State space of the Standing up policy
	Previous action (one-hot vector)
	State space of the Locomotion policy
Height Estimator	Gravity vector (e_g)
	Joint positions
	Joint velocities
Actuator Model	History of joint position errors & velocities
	Desired joint position
	History of joint position errors & velocities

TABLE I: Definition of the State Spaces

estimation framework of ANYmal relies on the Two State Implicit Filter (TSIF) [4] for the base pose and twist, and angular encoders for the joint states. TSIF estimates the base twist and the base position in the inertial frame by recursively fusing kinematics and measurements from the Inertial Measurement Unit (IMU). It makes use of the positions of feet touching the ground to incorporate kinematic contact constraints. When a foot slips or all four feet lose contacts with the ground, which are likely to happen when ANYmal falls, the estimated base states from TSIF becomes unreliable as the position and linear velocity drift over time (see section III-B). To maximize reliability, the linear position and linear velocity are excluded from the state space of the self-righting controller.

A unit vector pointing in the direction of the gravity expressed in the base frame (e_g) is used to represent the orientation of the base because an IMU can observe only this two dimensional subspace of the orientation (yaw angle is unobservable). The x, y positions of the base in the inertial frame are excluded from the state spaces because they are always unobservable with IMU [3].

On the other hand, the z position of the base (base height) can be accurately estimated using kinematics while walking. We kept this state in the locomotion policy as it is critical for fast learning and estimate the height using a new estimation network presented in section II-F.

C. Cost terms

The cost terms are presented in the table II. We defined cost functions for each behavior by a linear combination of cost terms. For joint position cost, we used the minimum angle difference between two angular positions denoted by $d_\phi(\cdot, \cdot) : \mathbb{R} \times \mathbb{R} \rightarrow [0, \pi]$.

It is important to define a bounded cost function because otherwise an agent often finds it more rewarding to terminate than to explore its environment. To this end, we used a logistic kernel function $K : \mathbb{R} \rightarrow [-0.25, 0)$ defined as $K(e|\alpha) = -1/(e^{\alpha e} + 2 + e^{-\alpha e})$, $\alpha \in \mathbb{R}_{>0}$ where e represents an error

Symbols	
$\dot{\phi}_{j,slim}$	maximum joint speed
I_c	index set of the contact points
$I_{c,f}$	index set of the foot contact points
$I_{c,in}$	index set of the self-collision points
$i_{c,n}$	impulse of the n th contact
g_i	gap function of the i th contact
$p_{f,i}$	position of the i th foot
τ	vector of joint torques
$ \cdot $	cardinality of a set or l_1 norm
$\ \cdot\ $	l_2 norm
$\hat{\cdot}$	target value
Cost Terms	
Angular velocity	$c_\omega = K(\omega_{IB}^B - \hat{\omega}_{IB}^B , \alpha_a)$
Linear velocity	$c_v = K(v_{IB}^B - \hat{v}_{IB}^B , \alpha_l)$
Height	$c_h = 1.0$ if $h < 0.35$, otherwise 0
Joint position	$c_{jp} = d_\phi(\phi_j, \hat{\phi}_j)$
Orientation	$c_o = \ [0, 0, -1]^T - e_g\ $
Torque	$c_\tau = \ \tau\ ^2$
Power	$c_{pw} = \sum_{i=0}^{12} \max(\dot{\phi}_{j,i} \tau_i, 0)$
Joint acceleration	$c_a = \sum_{i=0}^{12} \ \ddot{\phi}_i\ ^2$
Joint speed	$c_{js} = \sum_{i=0}^{12} \max(\dot{\phi}_{j,slim} - \dot{\phi}_i , 0)^2$
Body impulse	$c_{bi} = \sum_{n \in I_c \setminus I_{c,f}} \ i_{c,n}\ / (I_c - I_{c,f})$
Body slippage	$c_{bs} = \sum_{n \in I_c} \ v_{c,n}\ ^2 / I_c $
Foot slippage	$c_{fs} = \sum \ v_{f,i}\ $ $\forall i \text{ s.t. } g_i = 0, i \in I_{f,c}$
Foot clearance	$c_{fc} = \sum (p_{f,i} - 0.07)^2 \ v_{f,i}\ $ $\forall i \text{ s.t. } g_i > 0, i \in I_{f,c}$
Self collision	$c_{cin} = I_{c,in} $
Action difference	$c_{ad} = \ a_{t-1} - a_t\ ^2$

TABLE II: Cost Terms

term. This kernel comes handy in tuning because it enables us to leverage relative importance between different cost terms and it enables us to adjust an agent's sensitivity to e by adjusting α .

D. Behaviors

The policies for self-righting, standing up and locomotion are individually trained to achieve different tasks.

1) *Tasks*: Each behavior is learned based on a different cost function, initial state distributions, and termination conditions.

- **Self-righting behavior** is to regain upright base pose from an arbitrary configuration (Fig. 3a) and re-position joints to the sitting configuration (Fig. 3b) which is designed such that ANYmal has all feet on the ground for a safe stand-up maneuver. The cost function is defined as $k_o c_o + k_{jp} c_{jp} + k_a c_a + k_{bi} c_{bi} + k_{bs} c_{bs} + k_{c,in} c_{c,in} + k_{ad} c_{ad} + k_{j,slim} c_{j,slim} + k_\tau c_\tau$, where $k_{(.)}$ is a scaling factor. Each cost terms are explained in table. II. The weight for the orientation cost (k_o) was set to be the highest such that ANYmal recovers up-right base pose as soon as possible. The magnitudes of contact impulses are penalized to avoid violent motions. The joint accelerations are penalized to generate smooth motions.

To sample initial states for training in simulation, we dropped ANYmal from 0.5 m above the ground with random joint positions. The termination condition is only

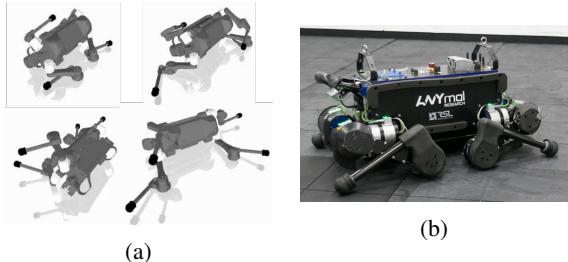


Fig. 3: (a) Sampled initial states and (b) the target configuration of the self-righting task.

the time limit.

- **Standing up behavior** is to stand up from arbitrary sitting configurations. The cost function is similar to the cost of the self-righting but a height cost is additionally introduced: $k_{jp}c_{jp} + k_{oc_o} + k_h c_h + k_{ac_a} + k_{ad}c_{ad} + k_{jslim}c_{jslim} + k_\tau c_\tau$. The target joint configuration is the standing configuration.

We use the same strategy for sampling initial states as the self-righting task but with a near-upright pose and do not specify any termination condition except for the time limit.

- **Locomotion behavior** is to follow a given velocity command composed of desired forward velocity, lateral velocity, and turning rate (or yaw rate) which are sampled from the uniform distribution, $U(-1, 1)$ m/s, $U(-0.4, 0.4)$ m/s, and $U(-1.2, 1.2)$ rad/s respectively. They are defined concerning the capabilities of an existing controller [1]. The cost function penalizes the velocity tracking errors (c_ω and c_v), foot motions (c_{fc} and c_{fs}), and constraint violation. It is defined as $k_\omega c_\omega + k_v c_v + k_{oc_o} + k_{fc}c_{fc} + k_{fs}c_{fs} + k_{ad}c_{ad} + k_{jslim}c_{jslim} + k_\tau c_\tau$.

The initial states are sampled from a multivariate Gaussian distribution centered at the standing configuration. An episode terminates when the joint limit is violated or ANYmal falls.

2) *Action Representation*: The output of a policy for a behavior is a 12-dimensional real-valued vector and each dimension is mapped to a position target of the low-impedance Proportional-Derivative (PD) controllers running at each joint actuator. Peng and van de Panne [18] showed that using this type of action representations for learning motor skills achieves better final performance, faster learning, and higher robustness compared to other representations such as target joint torques and target joint velocities.

The output of a policy network is mapped to joint position targets differently depending on the task. For locomotion, the desired joint position ϕ_d is defined as $\phi_d = k o_t + \phi_n$ where k is a scaling parameter, o_t is the output, and ϕ_n is a nominal joint configuration (standing). It is designed such that the distribution of the target positions has a standard deviation of approximately 1 and mean at the nominal configuration at the beginning of training. It accelerates the learning because the agent explores trajectories near the standing configuration

more frequently. For the self-righting and the standing up, we define $\phi_d = k o_t + \phi_t$, where ϕ_t is a vector of current joint positions. It promotes exploration in joint spaces and results in faster learning of self-righting.

3) *Architecture*: The policies are parameterized by a two-layered feed-forward neural network with tanh units in the hidden layers. The self-righting and standing-up policies have 128 units in each hidden layer and the locomotion policy has 128 and 256 units, respectively.

4) *Training*: Each policy is trained separately with TRPO+GAE. For the natural and smooth motions, we penalized joint torque, velocity, acceleration, and action difference. We employed Curriculum Learning (CL) in a way that these terms are near zero at the first iteration and increase as the training proceeds [13]; otherwise a learning agent converged to a local minima where it stops moving.

E. Behavior Selector

The behavior selector learns to determine which behavior to execute depending on the current situation.

1) *Task*: A behavior selector has to choose an appropriate behavior such that ANYmal returns to a nominal operating state every time it loses balance. By a nominal operating state, we mean states where it can locomote. To this end, the cost function is defined as $k_\omega c_\omega + k_v c_v + k_{oc_o} + k_h c_h + k_{pw} c_{pw} + k_{ad} c_{ad} + k_{jslim} c_{jslim} + k_\tau c_\tau$, which is similar to that of the locomotion task. Additionally, the use of c_{pw} makes the resulting behaviors power efficient, while c_{ad} and c_τ ensure smooth transitions between different behaviors.

The initial states are sampled from the initial state distributions of a randomly selected behaviors and the termination condition is the same as that of self-righting.

2) *State and Action Spaces*: The state space of the behavior selector consists of the union of the state spaces of behaviors and the index of previously chosen behavior.

A behavior selector maps a state s to a categorical distribution over the behaviors. It is denoted as $\pi_\theta(a|s)$ with $a \in \{0, 1, 2\}$ and the output is a three dimensional real-valued vector $\{p_0, p_1, p_2\}$. Each dimension represents the probability for choosing each behavior.

3) *Architecture*: The behavior selector is parameterized by a two-layered feed-forward neural network with 128 tanh units in the hidden layers. Softmax function is used at the output such that $\sum_{i=0}^3 \pi_\theta(i|s) = 1$ for any s . The height estimator is of the same structure but without Softmax. Moreover it uses the softsign unit which is computationally more efficient than tanh.

4) *Training*: We use a set of pre-trained behaviors, which are regarded as a part of the environment, to train the behavior selector using TRPO+GAE. The height estimator, which is explained in section II-F, is trained concurrently as outlined in the algorithm 1. The reasoning behind this strategy is to match their state visitation frequency.

F. Height Estimation

The estimated base height becomes unreliable when ANYmal falls. We could observe huge errors from the base position

Algorithm 1 Training Behavior Selector

```

Initialize  $\theta, \psi$  randomly
for  $i = 0, 1, \dots, N$  do
    for  $t = 0, 1, \dots, T$  do
        if  $i > N_w$  then  $\triangleright N_w =$  Warm-up period
            Use the estimated height  $h_\psi(s_t)$ 
        Sample action  $a_t \sim \pi_\theta(a|s_t)$ 
        Execute the corresponding behavior
        Collect state  $s_t$ , action  $a_t$ , and reward  $r_t$ 
        Collect true height  $h_t$ .
        Append a  $s_t$ - $h_t$  pair into the replay memory
    Sample  $K$  pairs from the replay memory
    Update  $\psi$  by minimizing  $\sum_{j=0}^K \|h_j - h_\psi(s_j)\|^2$ 
    Update  $\theta$  using TRPO [22]

```

estimates during the experiments when ANYmal lies on the ground, which can lead to undesired behaviors. To resolve this issue, we trained a neural network to estimate the base height. It is denoted as h_ψ with a set of parameters ψ . The output is calculated from the body orientation (IMU) and joint positions (encoders), which are states that do not have a drift issue. The base height can be computed using forward kinematics under an assumption that ANYmal is on the flat ground and the geometric properties of the links are known.

The linear velocity estimate also had the same issue but the errors were not significant.

G. Handcrafted Behavior Selector

We introduce a traditional approach that we considered before using the neural network behavior selector. Finite State Machine (FSM) is a widely adopted method for controlling hybrid systems. It is defined by a set of states and transitions between them, which are usually designed by a domain expert. The proposed controller can be seen as an FSM if we regard behaviors as states and the behavior selector as a learned transition rule. As the task is straightforward, we could handcraft it by going through trial-and-error (Fig. 4). To maximize the success rate, it waits for TSIF to converge for 0.5 seconds after it conducts the self-righting behavior.

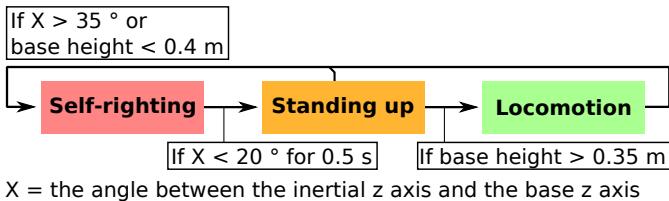


Fig. 4: FSM for behavior selection.

H. Simulating ANYmal

The structure of our simulator is depicted in Fig. 5. It has actuator models and stochastic components that are designed to account for modeling errors and to robustify the resulting policies [13]. We used RaiSim [12] as rigid-body simulator

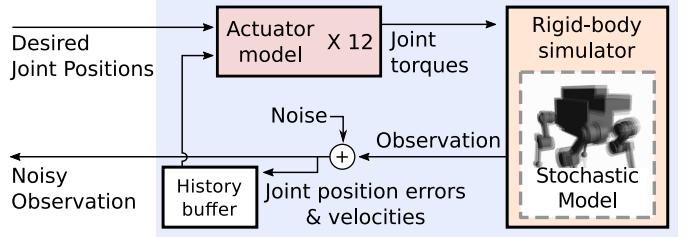


Fig. 5: Simulation for ANYmal.

together with learned networks that represent the actuator dynamics.

1) *Randomized Physical Properties:* To make the solution robust against modeling errors and to avoid tedious parameter estimation for each link as done in [26], we directly use physical properties computed from the CAD model including link lengths and inertial properties, but randomize the simulation to overcome modeling errors. It has been shown in several works that randomization enhances the robustness and increases the success rate of a sim-to-real transfer [13], [26], [11]. The link masses are randomized with additive noises up to 10 % of the original value, and the COM of the base is randomly translated up to 3 cm in x, y, z directions respectively every episode.

We approximated the collision geometry of ANYmal with collision primitives such as a box, a cylinder, and a sphere. The positions and shapes of these collision bodies are also randomized. The coefficient of friction between the objects is uniformly sampled between 0.8 to 2.0 every time step because we cannot accurately simulate the material properties.

2) *Actuator Model:* ANYmal's joints are Series Elastic Actuators (SEA) [20]. An SEA consists of multiple components including a spring, gears, encoders, and an electric motor, which results in highly complex dynamics. It is essential to simulate actuators accurately and fast to efficiently train a sim-to-real transferable policy because it substantially improves the simulation accuracy. Developing an analytic model requires a large number of parameters to be estimated or assumed to be accurately provided by a manufacturer and thus often results in an inaccurate model [9]. Instead, we use a data-driven model introduced in [13]. The actuator model is a neural network that outputs a joint torque conditioned on the position command and a history of joint position errors and velocities. It is a two-layered feed-forward neural network with 32 softsign units and trained with real data.

3) *Additive Noise to the Observation:* To replicate the noisy observation from the real robot, we added up to 0.2 m/s noise to the linear velocity, 0.25 rad/s to the angular velocity, 0.5 rad/s to the joint velocities, and 0.05 rad to the joint positions during training in simulation. Additionally, in order to replicate the behavior of TSIF, we increase the magnitude of the noise for the linear velocity and position of the base when all four legs lose contact.

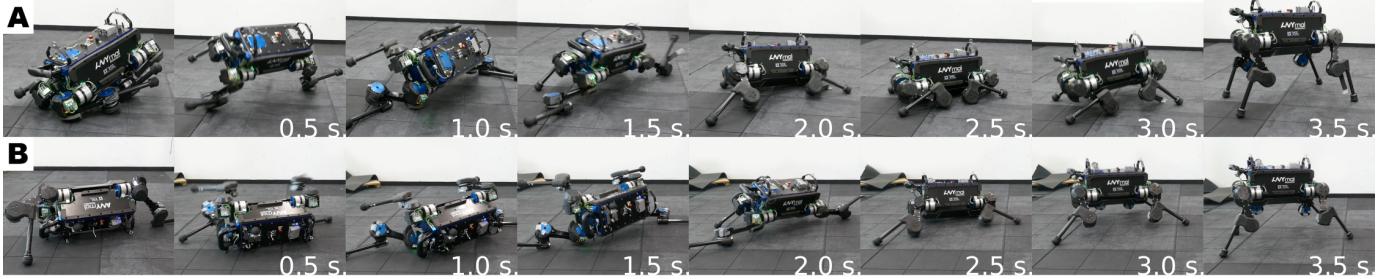


Fig. 6: ANYmal recovering from arbitrary configurations.

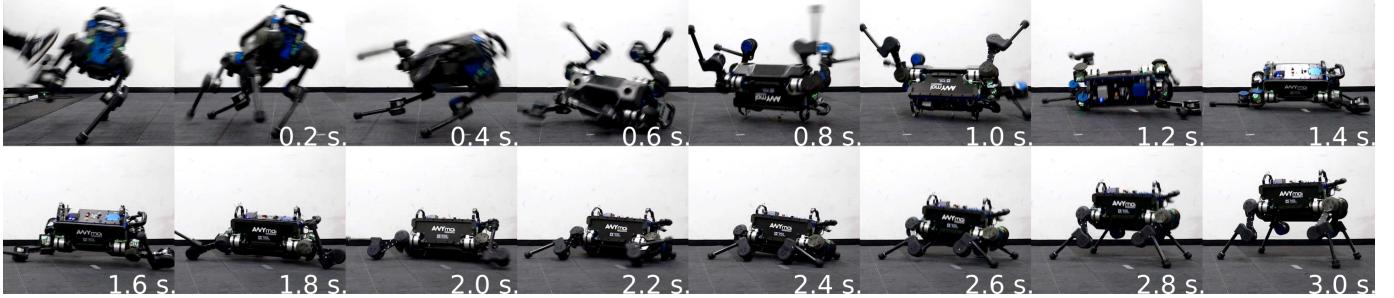


Fig. 7: ANYmal reacting to a kick.

I. Implementation Details

We used the Robotics Artificial Intelligence [11] framework together with the rigid-body simulator [12], which are both written in C++.

Temporal attributes of an RL task such as the control frequency, the time limit, and the discount factor are regarded as hyper-parameters. The policies for self-righting, standing up, and locomotion run at 20 Hz, 100 Hz, and 200 Hz respectively and the behavior selector runs at 50 Hz. The height estimator runs synchronously with TSIF, which runs at 400 Hz. When ANYmal switches to a different behavior, the output of the chosen behavior is computed immediately. As a result, the time step of the self-righting is often not preserved because it runs at the lowest frequency.

As the policies require a history of joint measurements as input, we implemented a history buffer that saves states for 0.05 seconds in 400 Hz. ANYmal waits in freeze mode to fill the history buffer before running the controller.

III. RESULT AND DISCUSSION

The experimental results are provided in this section. The behaviors for standing-up and locomotion are not assessed in detail in this paper. We refer the readers to [13] for a comprehensive analysis of the locomotion behavior.

A. Robustness

We conducted two kinds of experiments to verify the robustness of the policies. Firstly, we started the proposed controller while ANYmal lies on the ground at an arbitrary configuration. We tested 50 different configurations and the self-righting policy could recover up-right base pose within 5 seconds in all experiments. ANYmal can recover even when

its legs are stuck beneath its base (Fig. 6.A). It flips to its side to free the legs and then quickly regains up-right base pose. It recovers when its base is almost upside-down (Fig. 6.B).

Secondly, we applied external disturbances while ANYmal is walking or standing. An example is provided in Fig. 7. It smoothly switches to self-righting behavior or standing up behavior without any noticeable delay.

Both experiments are conducted 50 times and ANYmal fell more than 100 times in total. The recovery maneuver failed only three times. Consequently, The success rate was higher than 97 %. Self-righting failed when a joint position $\geq 2\pi$. It fails because a self-righting policy hardly experiences such a high position during the training. This is very unlikely to happen when ANYmal falls while walking and can be easily fixed by applying modulo operation to the joint positions with 2π .

B. Comparison to Simulation

To qualitatively analyze the accuracy of the simulation, we ran the same controller with the same initial state and velocity command. As shown in Fig. 8, the switch in behaviors appears in simulation (top) and reality (bottom) almost at the same time, and the behaviors are visually identical (Fig. 8).

The estimated heights for the maneuver above is provided in Fig. 9. Unfortunately, due to a lack of motion capture system, we could not measure the height in the experiment but have only data from simulation. The output of the TSIF shows a huge error when ANYmal lies on the ground (as shown by the initial value) and fluctuates during the self-righting maneuver. On the other hand, the output of the neural network is stable and accurately matches the simulated data (the RMS error is less than 1 cm). When deployed without the height estimator,

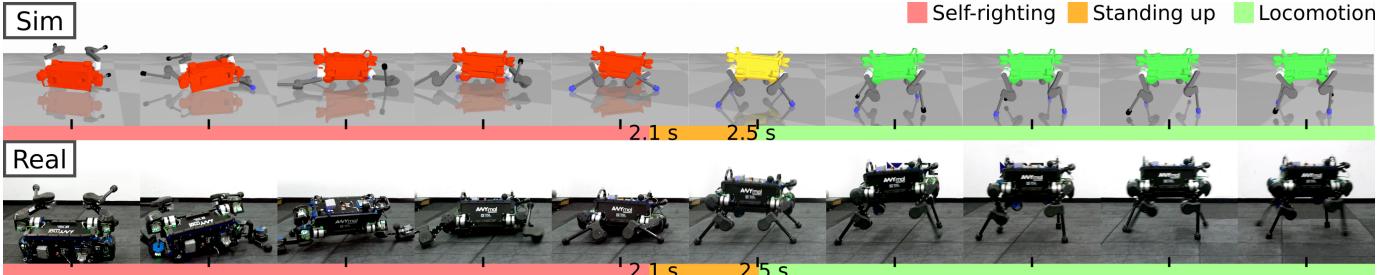


Fig. 8: Comparison between simulation and experiments. ANYmal trots in 1 m/s after standing up. The snapshots are taken every 0.5 seconds and the color bars represent the active behavior. The numbers represent the time of the switching and rounded to the first decimal place. (**Sim**) Simulated ANYmal with the initial state obtained from the experiment. (**Real**) Deploying the recovery controller on the real robot. We used the same noises and randomized dynamic properties as in simulation and did not hand-tune any number to matched the motions of the simulation and the experiment.

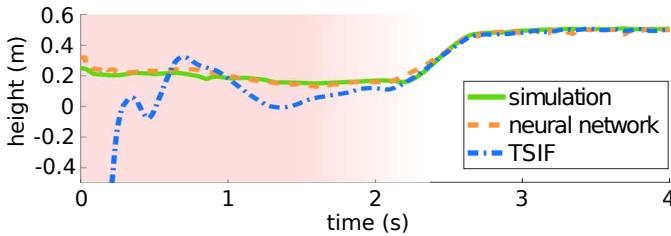


Fig. 9: Estimated base height from different sources. Self-righting policy runs in the shaded region. (**neural network & TSIF**) Output of the height estimator network and TSIF during the experiment in Fig. 8.Real. (**simulation**) Computed height from the simulation in Fig. 8.Sim.



Fig. 10: ANYmal showing undesirable behavior without the height estimator. ANYmal jumps up and hits itself (red circle). The behavior selector switches to locomotion policy while ANYmal is lying on the ground.

the error in the base height sometimes results in an undesirable behavior switch as shown in the Fig. 10.

C. FSM behavior selector

We discuss the FSM behavior selector introduced in section II-G. The simplicity of FSM did not allow us to capture corner cases (even with significant tuning in simulation and on the real system) as the one shown in Fig. 11. Besides, the transitions between the behaviors are often unsatisfactory. The FSM behavior selector is still robust enough to be used in the field. However, we did not examine the success rate of it because the result can be manipulated if we experiment with the corner cases more. The performance can be improved by adding more states and transitions. For example, the corner case in Fig. 11 can be resolved if we check joint positions before standing up. However, the fundamental problem is that

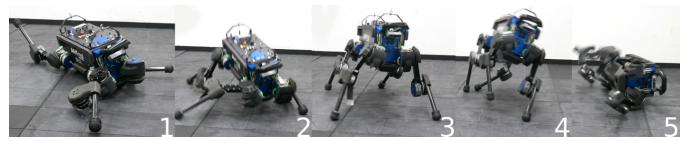


Fig. 11: A corner case of our FSM. It switches to standing up policy at a bad timing and falls.

it requires a number of design iterations and experiments on the real robot.

IV. CONCLUSION

This paper presents a hierarchically structured controller for the quadruped robot ANYmal that can autonomously recover from a fall and locomote on flat terrain. The control task is decomposed into three behaviors: standing, self-righting and walking. This strategy made it easier to design well-defined RL tasks and troubleshoot on the real robot. The policies for behaviors are individually trained to achieve a distinct behavior, and a behavior selector is trained to coordinate them. Additionally, a height estimator is learned, which turned out to be a crucial part for reliable maneuvers. All the policies are trained in simulation and deployed on the real robot.

The proposed controller exhibits dynamic recovery maneuvers involving multiple ground contacts and the resulting motions are consistent with the simulated ones. ANYmal can recover from multiple random fall configurations within 5 seconds and switches seamlessly between three behaviors in response to disturbances. The robustness of our recovery controller is evaluated by testing the controller more than 100 times on the physical system.

The current limitation of the proposed method is that it has only been trained and tested on flat ground and it will fail in case of large inclinations or rough ground. We plan to overcome these limitations in future work by randomizing the environment in simulation and by estimating its properties.

REFERENCES

- [1] C Dario Bellicoso, Fabian Jenelten, Christian Gehring, and Marco Hutter. Dynamic Locomotion Through Online Nonlinear Motion Optimization for Quadrupedal Robots. *IEEE Robotics and Automation Letters*, 3(3):2261–2268, 2018.
- [2] Glen Berseth, Cheng Xie, Paul Cernek, and Michiel Van de Panne. Progressive reinforcement learning with distillation for multi-skilled motion control. *arXiv preprint arXiv:1802.04765*, 2018.
- [3] Michael Bloesch et al. State estimation for legged robots—consistent fusion of leg kinematics and IMU. *Robotics*, 17:17–24, 2013.
- [4] Michael Bloesch et al. The Two-State Implicit Filter Recursive Estimation for Mobile Robots. *IEEE Robotics and Automation Letters*, 3(1):573–580, 2018.
- [5] BostonDynamics. Introducing SpotMini, June 2016. URL <https://youtu.be/tf7IEVTDjng?t=91>. Accessed Jan. 18, 2019.
- [6] Rodney Brooks. A robust layered control system for a mobile robot. *IEEE journal on robotics and automation*, 2(1):14–23, 1986.
- [7] Dian-sheng Chen, Jun-mao Yin, Yu Huang, Kai Zhao, and Tian-miao Wang. A hopping-righting mechanism analysis and design of the mobile robot. *Journal of the Brazilian Society of Mechanical Sciences and Engineering*, 35(4):469–478, 2013.
- [8] Kevin Frans, Jonathan Ho, Xi Chen, Pieter Abbeel, and John Schulman. Meta learning shared hierarchies. *arXiv preprint arXiv:1710.09767*, 2017.
- [9] Christian Gehring et al. Practice makes perfect: An optimization-based approach to controlling agile motions for a quadruped robot. *IEEE Robotics & Automation Magazine*, 23(1):34–43, 2016.
- [10] Marco Hutter et al. Anymal—a highly mobile and dynamic quadrupedal robot. In *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*, pages 38–44. IEEE, 2016.
- [11] Jemin Hwangbo, Inkyu Sa, Roland Siegwart, and Marco Hutter. Control of a quadrotor with reinforcement learning. *IEEE Robotics and Automation Letters*, 2(4):2096–2103, 2017.
- [12] Jemin Hwangbo, Joonho Lee, and Marco Hutter. Per-Contact Iteration Method for Solving Contact Dynamics. *IEEE Robotics and Automation Letters*, 3(2):895–902, 2018.
- [13] Jemin Hwangbo, Joonho Lee, Alexey Dosovitskiy, Dario Bellicoso, Vassilios Tsounis, Vladlen Koltun, and Marco Hutter. Learning Agile and Dynamic Motor Skills for Legged Robots. *Science Robotics*, 4(26):eaau5872, 2019.
- [14] Libin Liu and Jessica Hodgins. Learning to schedule control fragments for physics-based characters using deep Q-learning. *ACM Transactions on Graphics (TOG)*, 36(3):29, 2017.
- [15] Pattie Maes and Rodney A Brooks. Learning to Coordinate Behaviors. In *AAAI*, volume 90, pages 796–802, 1990.
- [16] Josh Merel et al. Hierarchical visuomotor control of humanoids. *arXiv preprint arXiv:1811.09656*, 2018.
- [17] Igor Mordatch, Emanuel Todorov, and Zoran Popović. Discovery of complex behaviors through contact-invariant optimization. *ACM Transactions on Graphics (TOG)*, 31(4):43, 2012.
- [18] Xue Bin Peng and Michiel van de Panne. Learning locomotion skills using deeplrl: does the choice of action space matter? In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, page 12. ACM, 2017.
- [19] Xue Bin Peng, Glen Berseth, and Michiel Van de Panne. Terrain-adaptive locomotion skills using deep reinforcement learning. *ACM Transactions on Graphics (TOG)*, 35(4):81, 2016.
- [20] Gill A Pratt and Matthew M Williamson. Series elastic actuators. In *Intelligent Robots and Systems 95.'Human Robot Interaction and Cooperative Robots', Proceedings. 1995 IEEE/RSJ International Conference on*, volume 1, pages 399–406. IEEE, 1995.
- [21] Uluc Saranli, Alfred A Rizzi, and Daniel E Koditschek. Model-based dynamic self-righting maneuvers for a hexapedal robot. *The International Journal of Robotics Research*, 23(9):903–918, 2004.
- [22] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International Conference on Machine Learning*, pages 1889–1897, 2015.
- [23] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.
- [24] Claudio Semini et al. DESIGN OVERVIEW OF THE HYDRAULIC QUADRUPED ROBOTS. In *The Fourteenth Scandinavian International Conference on Fluid Power*, 2015.
- [25] Jörg Stückler, Johannes Schwenk, and Sven Behnke. Getting Back on Two Feet: Reliable Standing-up Routines for a Humanoid Robot. In *IAS*, pages 676–685, 2006.
- [26] Jie Tan, Tingnan Zhang, Erwin Coumans, Atil Iscen, Yunfei Bai, Danijar Hafner, Steven Bohez, and Vincent Vanhoucke. Sim-to-Real: Learning Agile Locomotion For Quadruped Robots. *arXiv preprint arXiv:1804.10332*, 2018.