

## ARTIFICIAL INTELLIGENCE

# Learning agile and dynamic motor skills for legged robots

Jemin Hwangbo<sup>1\*</sup>, Joonho Lee<sup>1</sup>, Alexey Dosovitskiy<sup>2</sup>, Dario Bellicoso<sup>1</sup>, Vassilios Tsounis<sup>1</sup>, Vladlen Koltun<sup>3</sup>, Marco Hutter<sup>1</sup>

Copyright © 2019  
The Authors, some  
rights reserved;  
exclusive licensee  
American Association  
for the Advancement  
of Science. No claim  
to original U.S.  
Government Works

Legged robots pose one of the greatest challenges in robotics. Dynamic and agile maneuvers of animals cannot be imitated by existing methods that are crafted by humans. A compelling alternative is reinforcement learning, which requires minimal craftsmanship and promotes the natural evolution of a control policy. However, so far, reinforcement learning research for legged robots is mainly limited to simulation, and only few and comparably simple examples have been deployed on real systems. The primary reason is that training with real robots, particularly with dynamically balancing systems, is complicated and expensive. In the present work, we introduce a method for training a neural network policy in simulation and transferring it to a state-of-the-art legged system, thereby leveraging fast, automated, and cost-effective data generation schemes. The approach is applied to the ANYmal robot, a sophisticated medium-dog-sized quadrupedal system. Using policies trained in simulation, the quadrupedal machine achieves locomotion skills that go beyond what had been achieved with prior methods: ANYmal is capable of precisely and energy-efficiently following high-level body velocity commands, running faster than before, and recovering from falling even in complex configurations.

## INTRODUCTION

Legged robotic systems are attractive alternatives to tracked/wheeled robots for applications with rough terrain and complex cluttered environments. The freedom to choose contact points with the environment enables them to overcome obstacles comparable to their leg length. With such capabilities, legged robots may one day rescue people in forests and mountains, climb stairs to carry payloads in construction sites, inspect unstructured underground tunnels, and explore other planets. Legged systems have the potential to perform any physical activity humans and animals are capable of.

A variety of legged systems are being developed in the effort to take us closer to this vision of the future. Boston Dynamics introduced a series of robots equipped with hydraulic actuators (1, 2). These have advantages in operation because they are powered by conventional fuel with high energy density. However, systems of this type cannot be scaled down (usually >40 kg) and generate smoke and noise, limiting them to outdoor environments. Another family of legged systems is equipped with electric actuators, which are better suited to indoor environments. Massachusetts Institute of Technology's (MIT) Cheetah (3) is one of the most promising legged systems of this kind. It is a fast, efficient, and powerful quadrupedal robot designed with advanced actuation technology. However, it is a research platform optimized mainly for speed and has not been thoroughly evaluated with respect to battery life, turning capability, mechanical robustness, and outdoor applicability. Boston Dynamics' newly introduced robot, SpotMini, is also driven by electric actuators and is designed for both indoor and outdoor applications. Although details have not been disclosed, public demonstrations and media releases (4) are convincing evidence of its applicability to real-world operation. The platform used in this work, ANYmal (5), is another promising quadrupedal robot powered by electric actuators. Its bioinspired actuator design makes it robust against impact while allowing accurate torque measurement at the joints. However, the complicated actuator design increases cost and compromises the power output of the robot.

Designing control algorithms for these hardware platforms remains exceptionally challenging. From the control perspective, these robots are high-dimensional and nonsmooth systems with many physical constraints. The contact points change over the course of time and depending on the maneuver being executed and, therefore, cannot be prespecified. Analytical models of the robots are often inaccurate and cause uncertainties in the dynamics. A complex sensor suite and multiple layers of software bring noise and delays to information transfer. Conventional control theories are often insufficient to deal with these problems effectively. Specialized control methods developed to tackle this complex problem typically require a lengthy design process and arduous parameter tuning.

The most popular approach to controlling physical legged systems is modular controller design. This method breaks the control problem down into smaller submodules that are largely decoupled and therefore easier to manage. Each module is based on template dynamics (6) or heuristics and generates reference values for the next module. For example, some popular approaches (7–10) use a template-dynamics-based control module that approximates the robot as a point mass with a massless limb to compute the next foothold position. Given the foothold positions, the next module computes a parameterized trajectory for the foot to follow. The last module tracks the trajectory with a simple proportional-integral-derivative (PID) controller. Because the outputs of these modules are physical quantities, such as body height or foot trajectory, each module can be individually hand-tuned. Approaches of this type have achieved impressive results. Kalakrishnan *et al.* (11) demonstrated robust locomotion over challenging terrain with a quadrupedal robot: To date, this remains the state of the art for rough-terrain locomotion. Recently, Bellicoso *et al.* (12) demonstrated dynamic gaits, smooth transitions between them, and agile outdoor locomotion with a similar controller design. Yet, despite their attractive properties, modular designs have limitations. First, limited detail in the modeling constrains the model's accuracy. This inherent drawback is typically mitigated by limiting the operational state domain of each module to a small region where the approximations are valid. In practice, such constraints lead to substantial compromises in performance, such as slow acceleration, fixed upright pose of

<sup>1</sup>Robotic Systems Lab, ETH Zurich, Zurich, Switzerland. <sup>2</sup>Intelligent Systems Lab, Intel, Munich, Germany. <sup>3</sup>Intelligent Systems Lab, Intel, Santa Clara, CA, USA.

\*Corresponding author. Email: jhwangbo@ethz.ch

the body, and limited velocity of the limbs. Second, the design of modular controllers is extremely laborious. Highly trained engineers spend months to develop a controller and to arduously hand-tune the control parameters per module for every new robot or even for every new maneuver. For example, running and climbing controllers of this kind can have markedly different architectures and are designed and tuned separately.

More recently, trajectory optimization approaches have been proposed to mitigate the aforementioned problems. In these methods, the controller is separated into two modules: planning and tracking. The planning module uses rigid-body dynamics and numerical optimization to compute an optimal path that the robot should follow to reach the desired goal. The tracking module is then used to follow the path. In general, trajectory optimization for a complex rigid-body model with many unspecified contact points is beyond the capabilities of current optimization techniques. Therefore, in practice, a series of approximations are used to reduce complexity. Some methods approximate the contact dynamics to be smooth (13, 14), making the dynamics differentiable. Notably, Neunert *et al.* (13) demonstrated that such methods can be used to control a physical quadrupedal robot. Other methods (15) prespecify the contact timings and solve for sections of trajectories where the dynamics remain smooth. A few methods aim to solve this problem with little to no approximation (16, 17). These methods can discover a gait pattern (i.e., contact sequence) with hard contact models and have demonstrated automatic motion generation for two-dimensional (2D) robotic systems, but, like any other trajectory optimization approach, the possible contact points are specified *a priori*. Although more automated than modular designs, the existing optimization methods perform worse than state-of-the-art modular controllers. The primary issue is that numerical trajectory optimization remains challenging, requires tuning, and in many cases, can produce suboptimal solutions. Besides, optimization has to be performed at execution time on the robot, making these methods computationally expensive. This problem is often solved by reducing precision or running the optimization on a powerful external machine, but both solutions introduce their own limitations. Furthermore, the system still consists of two independent modules that do not adapt to each other's performance characteristics. This necessitates hand-tuning of the tracker; yet, accurately tracking fast motion by an underactuated system with many unexpected contacts is nearly impossible.

Data-driven methods, such as reinforcement learning (RL), promise to overcome the limitations of prior model-based approaches by learning effective controllers directly from experience. The idea of RL is to collect data by trial and error and automatically tune the controller to optimize the given cost (or reward) function representing the task. This process is fully automated and can optimize the controller end to end, from sensor readings to low-level control signals, thereby allowing for highly agile and efficient controllers. On the down side, RL typically requires prohibitively long interaction with the system to learn complex skills—typically weeks or months of real-time execution (18). Moreover, over the course of training, the controller may exhibit sudden and chaotic behavior, leading to logistical complications and safety concerns. Direct application of learning methods to physical legged systems is therefore complicated and has only been demonstrated on relatively simple and stable platforms (19) or in a limited context (20).

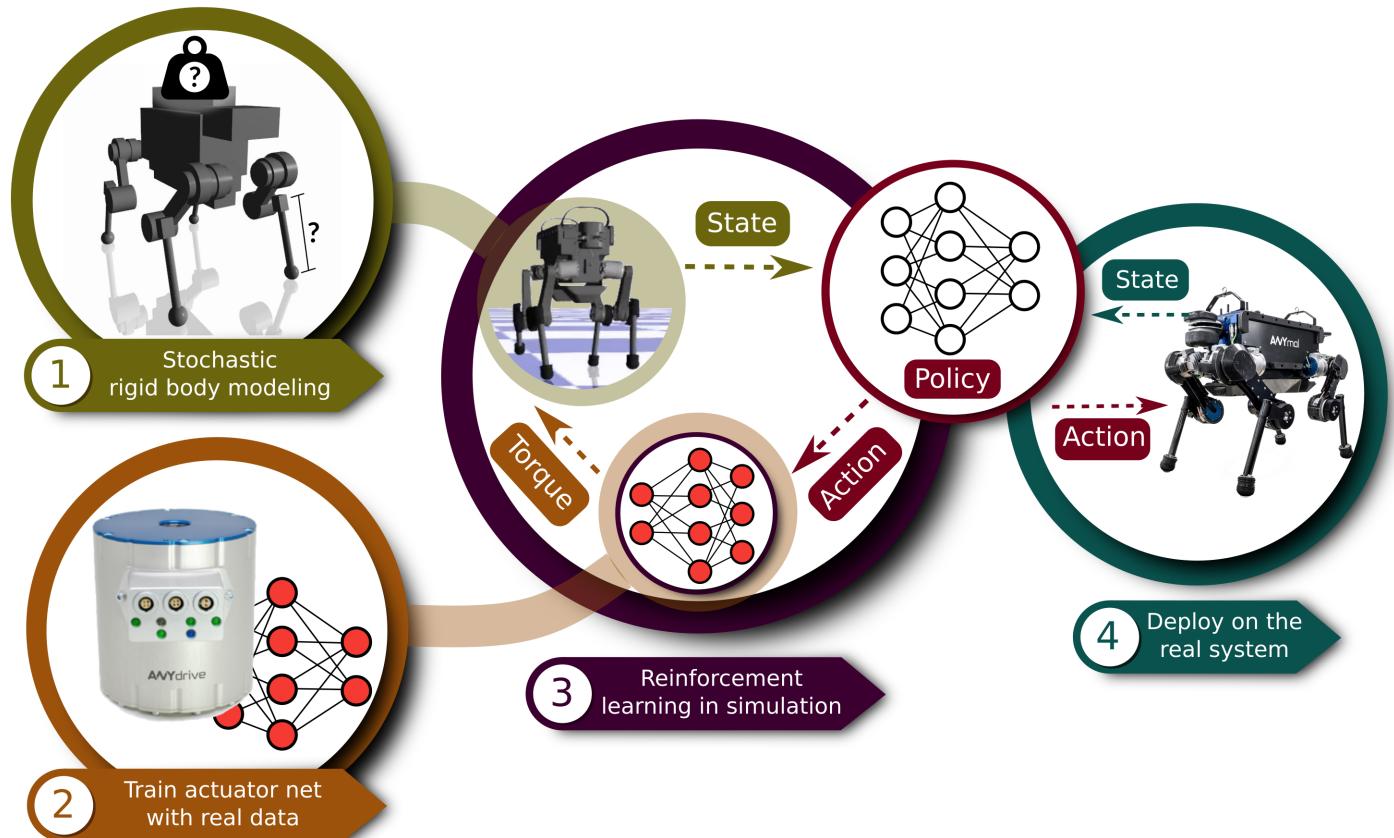
Because of the difficulties of training on physical systems, most advanced applications of RL to legged locomotion are restricted to

simulation. Recent innovations in RL make it possible to train locomotion policies for complex legged models. Levine and Koltun (21) combined learning and trajectory optimization to train a locomotion controller for a simulated 2D walker. Schulman *et al.* (22) trained a locomotion policy for a similar 2D walker with an actor-critic method. More recent work obtained full 3D locomotion policies (23–26). In these papers, animated characters achieve remarkable motor skills in simulation.

Given the achievements of RL in simulated environments, a natural question is whether these learned policies can be deployed on physical systems. Unfortunately, such simulation-to-reality transfer is hindered by the reality gap—the discrepancy between simulation and the real system in terms of dynamics and perception. There are two general approaches to bridging the reality gap. The first is to improve simulation fidelity either analytically or in a data-driven way; the latter is also known as system identification (27–32). The second approach is to accept the imperfections of simulation and aim to make the controller robust to variations in system properties, thereby allowing for better transfer. This robustness can be achieved by randomizing various aspects of the simulation: using a stochastic policy (33), randomizing the dynamics (34–37), adding noise to the observations (38), and perturbing the system with random disturbances. Both approaches lead to improved transfer; however, the former is cumbersome and often impossible, and the latter can compromise the performance of the policy. Therefore, in practice, both are typically used in conjunction. For instance, the recent work of Tan *et al.* (35) demonstrated successful sim-to-real transfer of locomotion policies on a quadrupedal system called Minitaur via the use of an accurate analytical actuator model and dynamic randomization. Although it achieved impressive results, the method of Tan *et al.* (35) crucially depended on accurate analytical modeling of the actuators, which is possible for direct-drive actuators (as used in Minitaur) but not for more complex actuators, such as servomotors, series elastic actuators (SEAs), and hydraulic cylinders, which are commonly used in larger legged systems.

In this work, we developed a practical methodology for autonomously learning and transferring agile and dynamic motor skills for complex and large legged systems, such as the ANYmal robot (5). Compared with the robots used in (35), ANYmal has a much larger leg length relative to footprint, making it more dynamic, less statically stable, and therefore more difficult to control. In addition, it features 12 SEAs, which are difficult to control and for which sufficiently accurate analytical models do not exist. Gehring *et al.* (39) have attempted analytical modeling of an SEA, but, as we will show, their model is insufficiently accurate for training a high-performance locomotion controller.

Our approach is summarized in Fig. 1. Our key insight on the simulation side is that efficiency and realism can be achieved by combining classical models representing well-known articulated system and contact dynamics with learning methods that can handle complex actuation (Fig. 1, steps 1 and 2). The rigid links of ANYmal, connected through high-quality ball bearings, closely resemble an idealized multibody system that can be modeled with well-known physical principles (40). However, this analytical model does not include the set of mechanisms that map the actuator commands to the generalized forces acting on the rigid-body system: the actuator dynamics, the delays in control signals introduced by multiple hardware and software layers, the low-level controller dynamics, and compliance/damping at the joints. Because these mechanisms are nearly impossible



**Fig. 1. Creating a control policy.** In the first step, we identify the physical parameters of the robot and estimate uncertainties in the identification. In the second step, we train an actuator net that models complex actuator/software dynamics. In the third step, we train a control policy using the models produced in the first two steps. In the fourth step, we deploy the trained policy directly on the physical system.

to model accurately, we learned the corresponding mapping in an end-to-end manner—from commanded actions to the resulting torques—with a deep network. We learned this “actuator net” on the physical system via self-supervised learning and used it in the simulation loop to model each of the 12 joints of ANYmal. Crucially, the full hybrid simulator, including a rigid-body simulation and the actuator nets, runs at nearly 500,000 time steps per second, which allows the simulation to run roughly a thousand times faster than real time. About half of the run time was used to evaluate the actuator nets, and the remaining computations were efficiently performed via our in-house simulator, which exploits the fast contact solver of Hwangbo *et al.* (41), efficient recursive algorithms for computing dynamic properties of articulated systems (composite rigid-body algorithm and recursive Newton-Euler algorithm) (40), and a fast collision-detection library (42). Thanks to efficient software implementations, we did not need any special computing hardware, such as powerful servers with multiple central processing units (CPUs) and graphics processing units (GPUs), for training. All training sessions presented in this paper were done on a personal computer with one CPU and one GPU, and none lasted more than 11 hours.

We used the hybrid simulator for training controllers via RL (Fig. 1, step 3). The controller is represented by a multilayer perceptron (MLP) that took as input the history of the robot’s states and produced as output the joint position target. Specifying different reward functions for RL yielded controllers for different tasks of interest.

The trained controller was then directly deployed on the physical system (Fig. 1, step 4). Unlike the existing model-based control approaches, our proposed method is computationally efficient at run time. Inference of the simple network used in this work took 25  $\mu$ s on a single CPU thread, which corresponds to about 0.1% of the available onboard computational resources on the robot used in the experiments. This is in contrast to model-based control approaches that often require an external computer to operate at sufficient frequency (13, 15). Also, by simply swapping the network parameter set, the learned controller manifested vastly different behaviors. Although these behaviors were trained separately, they share the same code base: Only the high-level task description changed depending on the behavior. In contrast, most of the existing controllers are task-specific and have to be developed nearly from scratch for every new maneuver.

We applied the presented methodology to learning several complex motor skills that were deployed on the physical quadruped. First, the controller enabled the ANYmal robot to follow base velocity commands more accurately and energy-efficiently than the best previously existing controller running on the same hardware. Second, the controller made the robot run faster, breaking the previous speed record of ANYmal by 25%. The controller could operate at the limits of the hardware and push performance to the maximum. Third, we learned a controller for dynamic recovery from a fall. This maneuver is exceptionally challenging for existing methods because it involves multiple



**Movie 1. Summary of the results and the method.**

unspecified internal and external contacts. It requires fine coordination of actions across all limbs and must use momentum to dynamically flip the robot. To the best of our knowledge, such recovery skill has not been achieved on a quadruped of comparable complexity.

## RESULTS

Movie 1 summarizes the results and the method of this work. In the following subsections, we describe the results in detail.

### Command-conditioned locomotion

In most practical scenarios, the motion of a robot is guided by high-level navigation commands, such as the desired direction and the speed of motion. These commands can be provided, for instance, by an upper-level planning algorithm or by a user via teleoperation. Using our method, we trained a locomotion policy that could follow such commands at runtime, adapting the gait as needed, with no prior knowledge of command sequence and timing. A command consists of three components: forward velocity, lateral velocity, and yaw rate.

We first qualitatively evaluated this learned locomotion policy by giving random commands using a joystick. In addition, the robot was disturbed during the experiment by multiple external pushes to the main body. The resulting behavior is shown in movie S1. The video shows about 40 s of robust command following. We also tested the policy for 5 min without a single failure, which manifests the robustness of the learned policy.

The trained policy performed stably within the command distribution that it was trained on, with any random combination of the command velocities. Although the forward command velocity was sampled from  $U(-1, 1)$  m/s during training, the trained policy reached 1.2 m/s of measured forward velocity reliably when the forward command velocity was set slightly higher (1.23 m/s), and the other command velocities were set to zero.

Next, we quantitatively evaluated this learned locomotion policy by driving the robot with randomly sampled commands. The commands were sampled as described in section S2. The robot received a new command every 2 s, and the command was held constant in between. The test was performed for 30 s, and a total of 15 random transitions were performed, including the initial transition from zero velocity. The base velocity plot is shown in fig. S1. The average linear velocity error was 0.143 m/s, and the average yaw rate error was 0.174 rad/s.

We next compared the learned controller with the best-performing existing locomotion controller available for ANYmal (12). For this experiment, we used a flying trot gait pattern (trot with full flight phase) because this is the only gait that stably reached 1.0 m/s forward velocity. We used the same velocity command profile, which resulted in the base velocity shown in fig. S2. The average linear velocity error

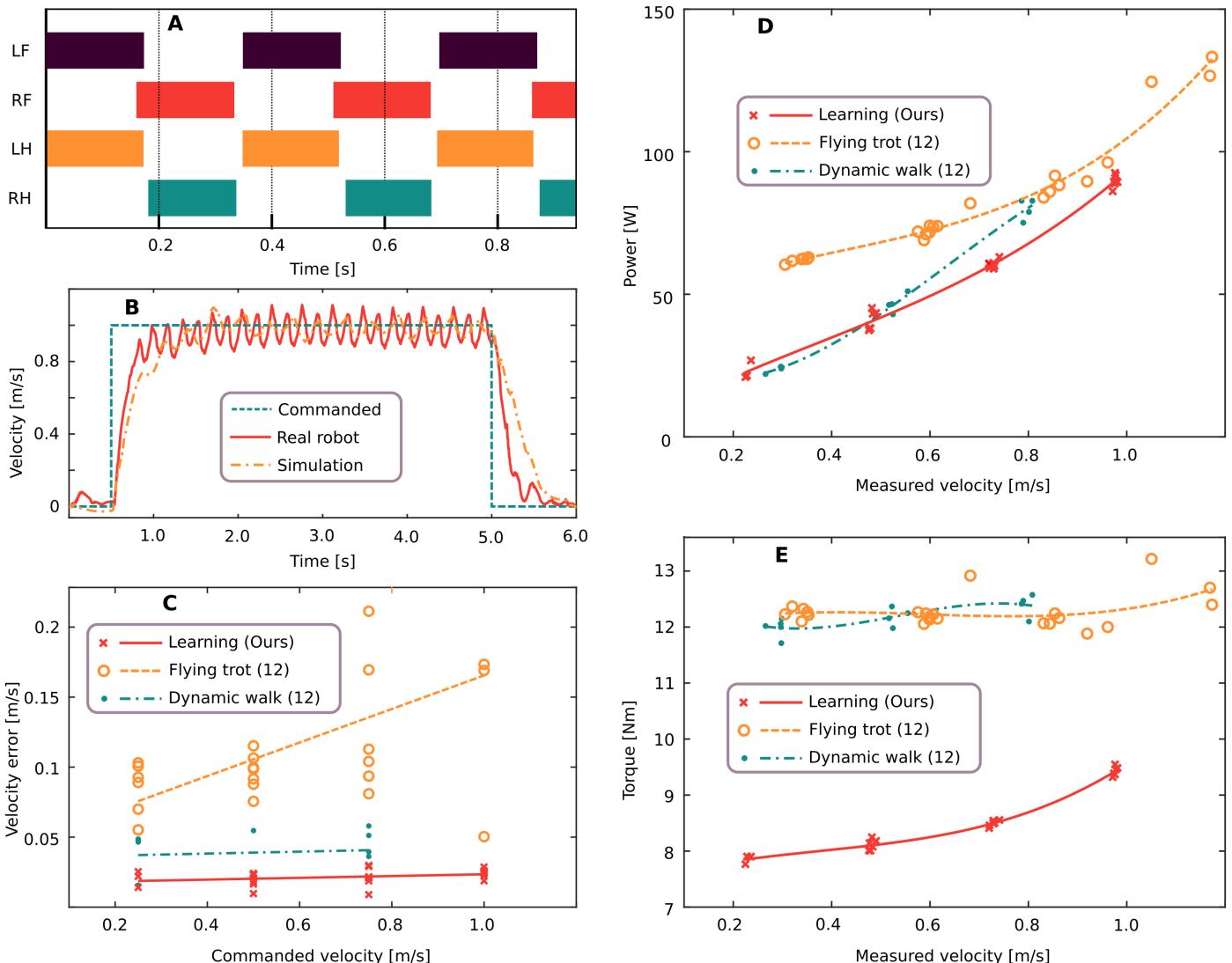
was 0.231 m/s, and the average yaw rate error was 0.278 rad/s. Given the same command profile, the tracking error of the model-based controller is about 95% higher than our learned controller with respect to linear velocity and about 60% higher with respect to yaw rate. In addition, our learned controller used less torque (8.23 N·m versus 11.7 N·m) and less mechanical power (78.1 W versus 97.3 W) on average. Movie S2 illustrates the experiments for both the learned policy and the model-based policy.

The control performance was also evaluated and compared in forward running. To this end, we sent a step input of four different speed commands (0.25, 0.5, 0.75, and 1.0 m/s) for 4.5 s each. The results, including a comparison to the prior method (12), are shown in Fig. 2. Figure 2A shows the flying trot pattern discovered by the learned controller. Note that this flight phase disappeared for low-velocity commands, and ANYmal displayed walking trot as shown in movie S1. Even without specifying the gait pattern, the learned policy manifested trot, a gait pattern that is commonly observed in quadrupedal animals. Figure 2B shows the velocity tracking accuracy of the policy both in simulation and on the real robot. Note that the oscillation of the observed velocity around the commanded one is a well-known phenomenon in legged systems, including humans (43). In terms of average velocity, the learned policy has an error of 2.2% on the real robot, 1.1% higher than in a simulation.

Figure 2 (C to E) compares the performance of the learned controller with the approach of Bellicoso *et al.* (12) in terms of accuracy and efficiency. We used two gaits from (12) for the comparison: flying trot, the only gait that can achieve 1 m/s, and dynamic lateral walk, the most efficient gait. First, we compared the velocity error at various commanded velocities in Fig. 2C. The learned controller is more accurate than the prior controller for all commanded velocities: by a factor of 1.5 to 2.5 compared with the dynamic lateral walk and by a factor of 5 to 7 compared with the flying trot, depending on the speed. Figure 2D shows the mechanical power output as a function of the measured velocity. The learned controller performed similarly to the dynamic lateral walk and more efficiently than the flying trot by a factor of 1.2 to 2.5, depending on the speed. Last, Fig. 2E plots the average measured torque magnitude against the measured velocity. The learned controller is more efficient in this respect than both prior gaits, using 23 to 36% less torque depending on the velocity. This large improvement in efficiency is possible because the learned controller walks with a nominal knee posture that is 10° to 15° straighter than prior gaits. The nominal posture cannot be adjusted to this level in the approach of Bellicoso *et al.* because this would markedly increase the rate of failure (falling).

Next, we compared our method with ablated alternatives: training with an ideal actuator model and training with an analytical actuator model. The ideal actuator model assumes that all controllers and hardware inside the actuator have infinite bandwidth and zero latency. The analytical model uses the actual controller code running on the actuator in conjunction with identified dynamic parameters from experiments and computer-aided design (CAD) tools. Some parameters, such as latency, damping, and friction, were hand-tuned to increase the accuracy of predicted torque in relation to data obtained from experiments. The policy training procedure for each method was identical to ours.

Neither alternative method could make a single step without falling. The resulting motions are shown in movies S3 and S4. We observed violent shaking of the limbs, probably due to not accounting for various delays properly. Although the analytical model contained multiple delay



**Fig. 2. Quantitative evaluation of the learned locomotion controller.** (A) The discovered gait pattern for 1.0 m/s forward velocity command. LF, left front leg; RF, right front leg; LH, left hind leg; RH, right hind leg. (B) The accuracy of the base velocity tracking with our approach. (C to E) Comparison of the learned controller against the best existing controller, in terms of power efficiency, velocity error, and torque magnitude, given forward velocity commands of 0.25, 0.5, 0.75, and 1.0 m/s.

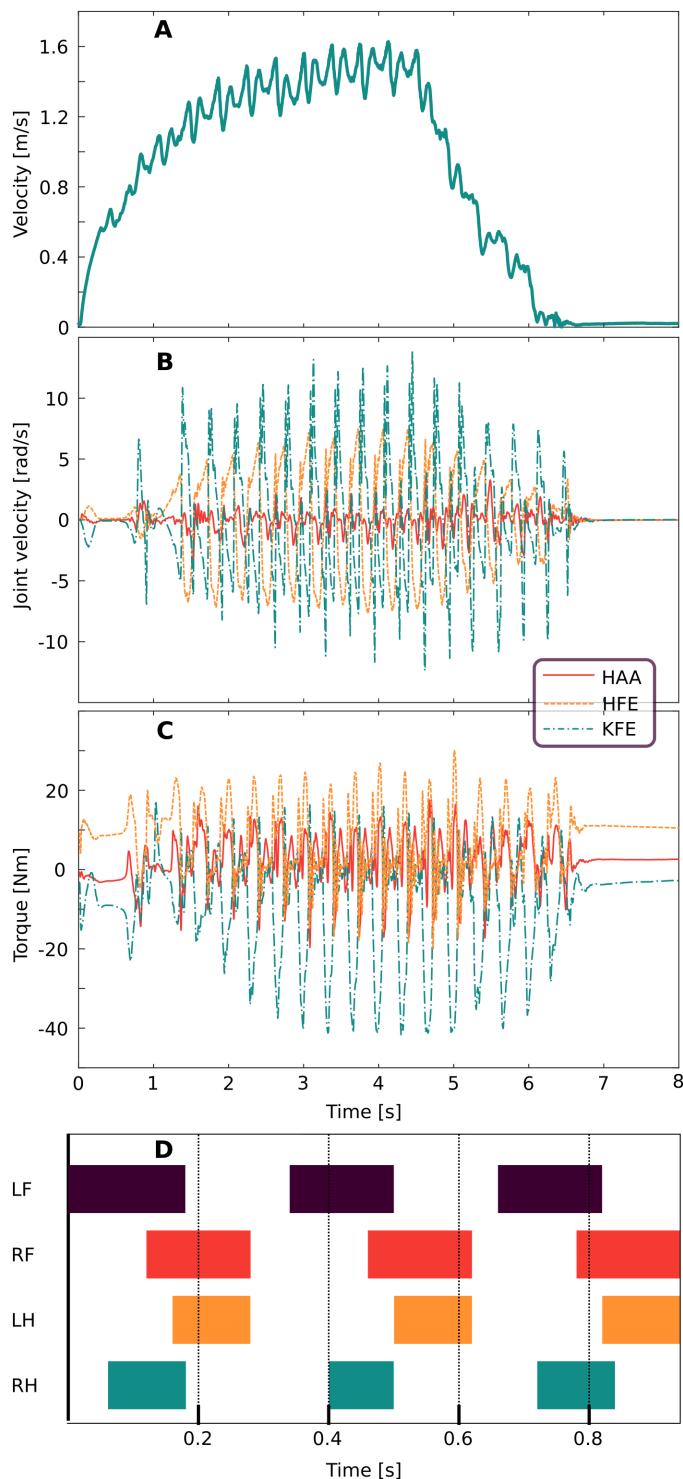
sources that were tuned using real data, accurately modeling all delay sources is complicated when the actuator has limited bandwidth. SEA mechanisms generate amplitude-dependent mechanical response time, and manual tuning of latency parameters becomes challenging. We tuned the analytical model for more than a week without much success.

### High-speed locomotion

In the previous section, we evaluated the generality and robustness of the learned controller. Here, we focus on operating close to the limits of the hardware to reach the highest possible speed. The notion of high speed is, in general, hardware dependent. There are some legged robots that are exceptional in this regard. Park *et al.* (44) demonstrated full 3D legged locomotion at over 5.0 m/s with the MIT Cheetah. The Boston Dynamics WildCat has been reported to reach 8.5 m/s (45). These robots are designed to run as fast as possible, whereas ANYmal is designed to be robust, reliable, and versatile. The current speed record on ANYmal is 1.2 m/s and was set using the flying trot gait

(12). Although this may not seem high, it is 50% faster than the previous speed record on the platform (39). Such velocities are challenging to reach via conventional controller design while respecting all limits of the hardware.

We used the presented methodology to train a high-speed locomotion controller. This controller was tested on the physical system by slowly increasing the commanded velocity to 1.6 m/s and lowering it to zero after 10 m. The forward speed and joint velocities/torques are shown in Fig. 3. ANYmal reached 1.58 m/s in simulation and 1.5 m/s on the physical system when the command was set to 1.6 m/s. All speed values were computed by averaging over at least three gait cycles. The controller used both the maximum torque (40 N·m) and the maximum joint velocities (12 rad/s) on the physical system (Fig. 3, B and C). This shows that the learned policy can exploit the full capacity of the hardware to achieve the goal. For most existing methods, planning while accounting for the limitations of the hardware is very challenging, and executing the plan on the real



**Fig. 3. Evaluation of the trained policy for high-speed locomotion.** (A) Forward velocity of ANYmal. (B) Joint velocities. (C) Joint torques. (D) Gait pattern.

system reliably is harder still. Even state-of-the-art methods (12, 46) cannot limit the actuation during planning owing to limitations of their planning module. Modules in their controllers are not aware of the constraints in the later stages; consequently, their outputs may not be realizable on the physical system.

The gait pattern produced by our learned high-speed controller (Fig. 3D) is distinct from the one exhibited by the command-conditioned locomotion controller. It is close to a flying trot but with notably longer flight phase and asymmetric flight phase duration. This is not a commonly observed gait pattern in nature, and we suspect that it is among multiple near-optimal solution modes for this task. The behavior of the policy is illustrated in movie S5.

### Recovery from a fall

Legged systems change contact points as they move and are thus prone to falling. If a legged robot falls and cannot autonomously restore itself to an upright configuration, then a human operator must intervene. Autonomous recovery after a fall is thus highly desirable. One possibility is to represent recovery behaviors as well-tuned joint trajectories that can simply be replayed—an approach that has been taken in some commercial systems (47). Such trajectories have required laborious manual tuning. They also take a very long time to execute because they do not take dynamics into account in the motion plan or the control. Some robots are designed such that recovery is either unnecessary or trivial (48, 49). However, such a design may not be possible for bigger and more complex machines. Morimoto *et al.* (50) demonstrated that a standing-up motion can be learned on a real robot. However, a simple three-link chain was used for demonstration, and the method has not been scaled to realistic systems.

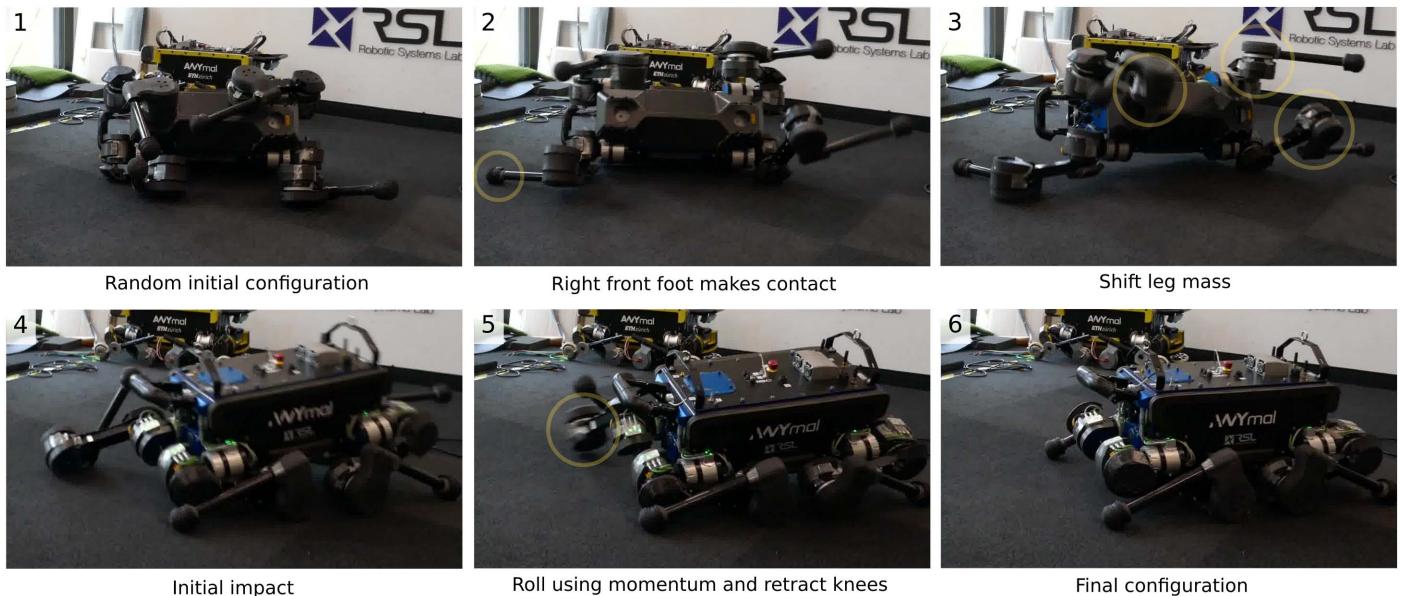
Fast and flexible recovery after a fall, as seen in animals, requires dynamic motion with multiple unspecified contact points. The collision model for our quadruped is highly complicated: It consists of 41 collision bodies, such as boxes, cylinders, and spheres (Fig. 1, step 1). Planning a feasible trajectory for such a model is extremely complicated. Even simulating such a system is challenging because there are many internal contacts. We used the approach of Hwangbo *et al.* (41) owing to its ability to handle such simulation in numerically stable fashion.

Using the presented methodology, we trained a recovery policy and tested it on the real robot. We placed ANYmal in nine random configurations and activated the controller as shown in movie S6. Many challenging configurations were tested, including a nearly entirely upside-down configuration (pose 8) and more complex contact scenarios where ANYmal was resting on its own legs (poses 2 and 4). In all tests, ANYmal successfully flipped itself upright. An example motion is shown in Fig. 4. These agile and dynamic behaviors demonstrate that our approach is able to learn performant controllers for tasks that are difficult or impossible to address with prior methods.

### DISCUSSION

The learning-based control approach presented in this paper achieved a high level of locomotion skill based purely on training in simulation and without tedious tuning on the physical robot. The system achieved more precise and energy-efficient motions than the prior state of the art. It outperformed the previous speed record by 25% and learned to consistently restore the robot to an operational configuration by dynamically rolling over its body.

Existing controllers are created by engineers. A model with adequate complexity has to be designed, and a control strategy has to be developed, tested, and tuned. This process typically takes months and has to be repeated for every distinct maneuver. In contrast, the simulation and learning framework used in this work are applicable to any rigid-body system. For applications to new tasks, our method



**Fig. 4. A learned recovery controller deployed on the real robot.** The learned policy successfully recovers from a random initial configuration in less than 3 s.

only requires a task description, which consists of the cost function, the initial state distribution, and randomization.

In our method, learned actuator dynamics effectively reduce the reality gap, whereas stochastic modeling guides the policy to be sufficiently conservative. The recovery task was successful on the very first attempt on the hardware. We then further improved the success rate to 100% by relaxing the joint velocity constraints. The results presented here were obtained on the second day of experiments on the physical system. In contrast, because of many model abstraction layers, which are necessary to make the computation tractable, prior methods often cannot exploit a sophisticated actuator model in controlling a complex legged system. Consequently, they often compromise performance or rely on well-tuned low-level controllers. For example, low-level controllers (e.g., the tracking controllers and the whole-body controller) have to be extensively tuned in the tested model-based controller (12) to mitigate imperfections of the actuators.

The learned policies are also robust to changes in hardware, such as those caused by wear and tear. All control policies have been tested for more than 3 months on the real robot without any modification. Within this period, the robot was heavily used with many controllers, including the ones presented here. Many hardware changes were introduced as well: different robot configurations, which roughly contribute 2.0 kg to the total weight, and a new drive which has a spring three times stiffer than the original one. All of the policies presented in this paper have performed robustly even under such conditions.

In terms of computational cost, our approach has an advantage over prior methods. Although it requires several hours of training with an ordinary desktop PC, the inference on the robot requires less than 25  $\mu$ s using a single CPU thread. Our method shifts nearly all computational costs to the training phase, where we can use external computational resources. Prior controllers often require two orders of magnitude more onboard computation. These demanding requirements limit the level of sophistication and thus the overall performance of the controller.

Using a policy network that directly outputs a joint-level command brings another advantage to our method. In contrast to many prior

methods that have numerical issues at singular configurations of the robot, our policies can be evaluated at any configuration. Consequently, our method is free from using ad hoc methods (e.g., branching conditions) in resolving such issues.

Although our approach allows for largely automated discovery of performant policies, it still requires some human expertise. A cost function and an initial state distribution have to be designed and tuned for each task. For a person with good understanding of both the task and RL, this process takes about 2 days for the locomotion policies presented in this work. Although this is still substantial amount of time, all the necessary tuning happens in simulation. Therefore, the development time will keep decreasing as computational technology evolves. In contrast, the prior controllers that use model abstractions inevitably require more development time and often extensive tuning on the real systems. Developing the recovery policy took about a week largely owing to the fact that some safety concerns (i.e., high impacts, fast swing legs, collisions with fragile components, etc.) are not very intuitive to embed in a cost function. Achieving a stand-up behavior was as simple as other tasks. However, for achieving the safe and robust behaviors that are demonstrated in this work, the cost function had to be tweaked several times. Longer development time was also attributed to the fact that it was trained by a person who had no previous experience with any real robot.

To train policies for a new robot, necessary modeling effort has to be made. This includes rigid-body modeling using the CAD model and actuator modeling using an actuator network. The former is often automated by modern CAD software, and the latter is easy if all necessary software/hardware infrastructures (e.g., logging, regression, and torque measurements) are in place. If not, it will also take a substantial portion of the development time. In addition, there are a few actuation types that manifest coupled dynamics (e.g., hydraulic actuators sharing a single accumulator). Learning actuators independently might not result in a sufficient accuracy for these systems. With a good understanding on the actuator dynamics, an appropriate history configuration can be estimated a priori and tuned further with respect to the validation error. In contrast, constructing an analytical actuator model

for ANYmal takes at least 3 weeks even if there is a very similar model studied in literature (39). The model also has many more parameters, many of which cannot be accurately obtained from measurements or the data sheet. Consequently, it requires more tuning than constructing an actuator network.

Another limitation of our approach was observed over the course of this study. A single neural network trained in one session manifests single-faceted behaviors that do not generalize across multiple tasks. Introducing hierarchical structure in the policy network can remedy this and is a promising avenue for future work (25).

The presented approach is not fundamentally limited to known and simple environments. We see the results presented in this paper

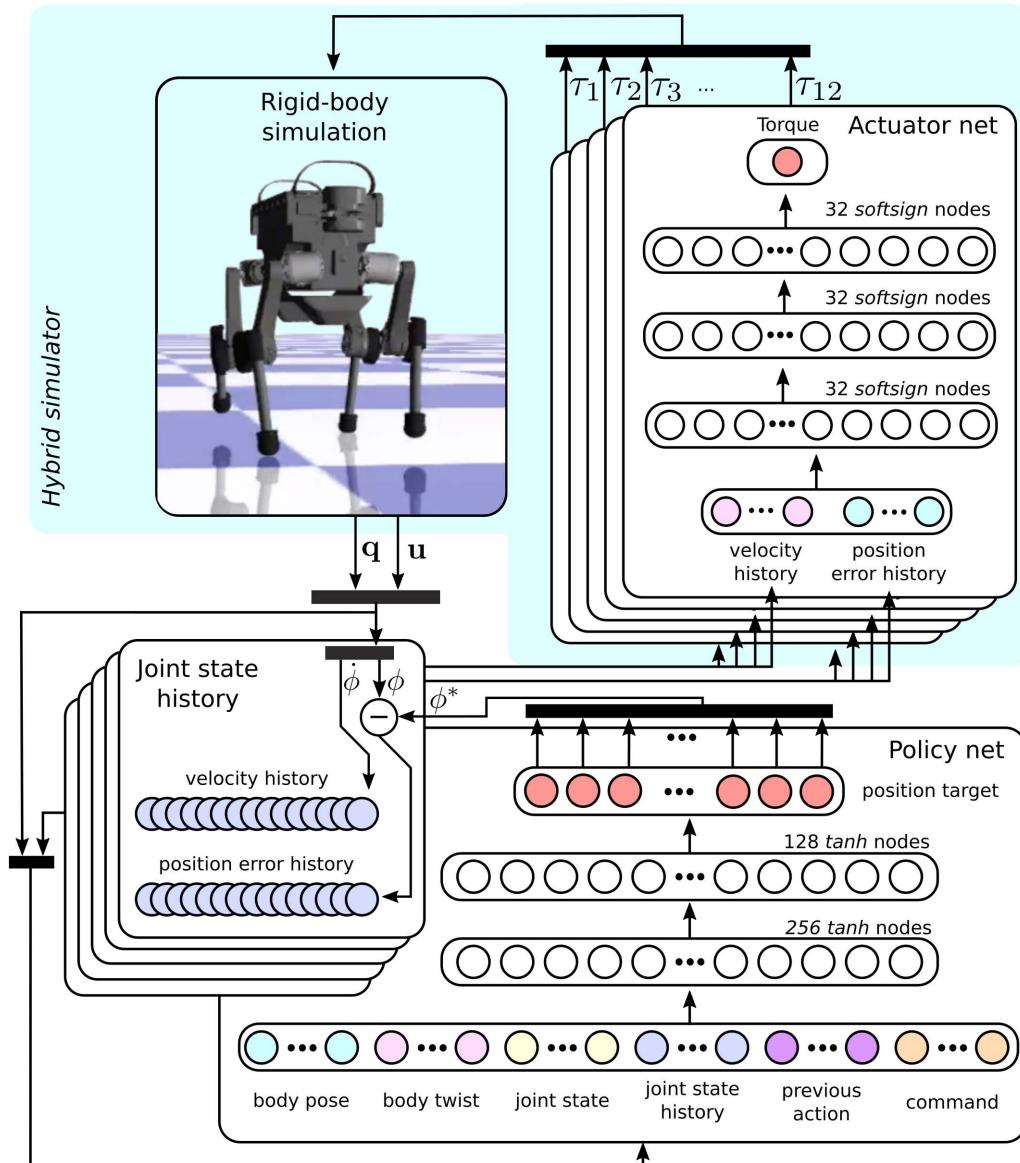
as a step toward comprehensive locomotion controllers for resilient and versatile legged robots.

## MATERIALS AND METHODS

This section describes in detail the simulation environment, the training process, and the deployment on the physical system. An overview of our training method is shown in Fig. 5. The training loop proceeds as follows: The rigid-body simulator outputs the next state of the robot given the joint torques and the current state. The joint velocity and the position error are buffered in a joint state history within a finite time window. The control policy, implemented by an MLP with two hidden layers, maps the observation of the current state and the joint state history to the joint position targets. Last, the actuator network maps the joint state history and the joint position targets to 12 joint torque values, and the loop continues. In what follows, we describe each component in detail.

**Modeling rigid-body dynamics**  
To efficiently train a complex policy within a reasonable time and transfer it to the real world, we needed a simulation platform that is both fast and accurate. One of the biggest challenges with walking robots is the dynamics at intermittent contacts. To this end, we used the rigid-body contact solver presented in our previous work (41). This contact solver uses a hard contact model that fully respects the Coulomb friction cone constraint. This modeling technique can accurately capture the true dynamics of a set of rigid bodies making hard contacts with their environment. The solver is not only accurate but also fast, generating about 900,000 time steps per second for the simulated quadruped on an ordinary desktop machine.

The inertial properties of the links were estimated from the CAD model. We expected up to about 20% error in the estimation due to unmodeled cabling and electronics. To account for such modeling inaccuracies, we robustified the policy by training with 30 different ANYmal models with stochastically sampled inertial properties. The center of mass positions, the masses of links, and joint positions were randomized by adding a noise sampled from  $U(-2, 2)$  cm,  $U(-15, 15)\%$ , and  $U(-2, 2)$  cm, respectively.



**Fig. 5. Training control policies in simulation.** The policy network maps the current observation and the joint state history to the joint position targets. The actuator network maps the joint state history to the joint torque, which is used in rigid-body simulation. The state of the robot consists of the generalized coordinate  $q$  and the generalized velocity  $u$ . The state of a joint consists of the joint velocity  $\dot{\phi}$  and the joint position error, which is the current position  $\phi$  subtracted from the joint position target  $\phi^*$ .

## Modeling the actuation

Actuators are an essential part of legged systems. Fast, powerful, lightweight, and high-accuracy actuators typically translate to dynamic, versatile, and agile robots. Most legged systems are driven by hydraulic actuators (51) or electric motors with gears (3), and some even include dedicated mechanical compliance (5, 52). These actuators have one thing in common: They are extremely difficult to model accurately. Their dynamics involve nonlinear and nonsmooth dissipation, and they contain cascaded feedback loops and a number of internal states that are not even directly observable. Gehring *et al.* (39) extensively studied SEA actuator modeling. The model of Gehring *et al.* includes nearly 100 parameters that have to be estimated from experiments or assumed to be correct from data sheets. This process is error prone and time consuming. In addition, many manufacturers do not provide sufficiently detailed descriptions of their products; consequently, an analytical model may not be feasible.

To this end, we used supervised learning to obtain an action-to-torque relationship that included all software and hardware dynamics within one control loop. More precisely, we trained an actuator network that output an estimated torque at the joints given a history of position errors (the actual position subtracted from the commanded position) and velocities. In this work, we assumed that the dynamics of the actuators are independent to each other such that we could learn a model for each actuator separately. This assumption might not be valid for other types of actuation. For example, hydraulic actuators with a single common accumulator might manifest coupled dynamics, and a single large network, representing multiple actuators together, might be more desirable.

The states of the actuators are only partially observable because the internal states of the actuators (e.g., states of the internal controllers and motor velocity) cannot be measured directly. We assumed that the network could be trained to estimate the internal states given a history of position errors and velocities, because otherwise the given information is simply insufficient to control the robot adequately. The actuator used in this work is revolute and radially symmetric, and the absolute angular position is irrelevant given the position error. We use a history consisting of the current state and two past states that correspond to  $t - 0.01$  and  $t - 0.02$  s. Note that too-sparse input configuration might not effectively capture the dynamics at high frequency ( $>100$  Hz). This issue was partially mitigated by introducing a smoothness cost term, which penalizes abrupt changes in the output of the policy. Too-dense history can also have adverse effects: It is more prone to overfitting and computationally more expensive. The length of the history should be chosen such that it is sufficiently longer than the sum of all communication delays and the mechanical response time. In practice, the exact input configuration is tuned with respect to the validation error. This tuning process often takes less than a day because the network is very small.

To train the network, we collected a dataset consisting of joint position errors, joint velocities, and the torque. We used a simple parameterized controller that generates foot trajectories in the form of a sine wave; the corresponding joint positions were computed using inverse kinematics. The feet constantly made or broke a contact with the ground during data collection so that the resulting trajectories roughly mimicked the trajectories followed by a locomotion controller. To obtain a rich set of data, we varied the amplitude (5 to 10 cm) and the frequency (1 to 25 Hz) of the foot trajectories and disturbed the robot manually during data collection. We found that the excitation must cover a wide range of frequency spectra; otherwise, the trained model

generated unnatural oscillation even during the training phase. Data collection took less than 4 min because the data could be collected, in parallel, from the 12 identical actuators on ANYmal. Data were collected at 400 Hz; therefore, the resulting dataset contains more than a million samples. About 90% of the generated data were used for training, and the rest were used for validation.

The actuator network is an MLP with three hidden layers of 32 units each (Fig. 5, actuator net box). After testing with two common smooth and bounded activation functions—tanh and softsign (53)—we chose the softsign activation function because it is computationally efficient and provides a smooth mapping. Evaluating the actuator network for all 12 joints took 12.2  $\mu$ s with softsign and 31.6  $\mu$ s with tanh. As shown here, the tanh activation function resulted in a higher computational cost and is therefore less preferred. The two activation functions resulted in about the same validation error [0.7 to 0.8 N·m in root mean square (RMS)]. The validation result with the softsign function is shown in Fig. 6. The trained network nearly perfectly predicted the torque from the validation data, whereas the ideal actuator model failed to produce a reasonable prediction. Here, the ideal actuator model assumes that there is no communication delay and that the actuator can generate any commanded torque instantly (i.e., infinite actuator bandwidth). The trained model has an average error of 0.740 N·m on the validation set, which is not far from the resolution of the torque measurement (0.2 N·m) and much smaller than the error of the ideal actuator model (3.55 N·m). Its prediction error on test data (i.e., collected using the trained locomotion policies) is notably higher (0.966 N·m) but still far less than that of the ideal model (5.74 N·m).

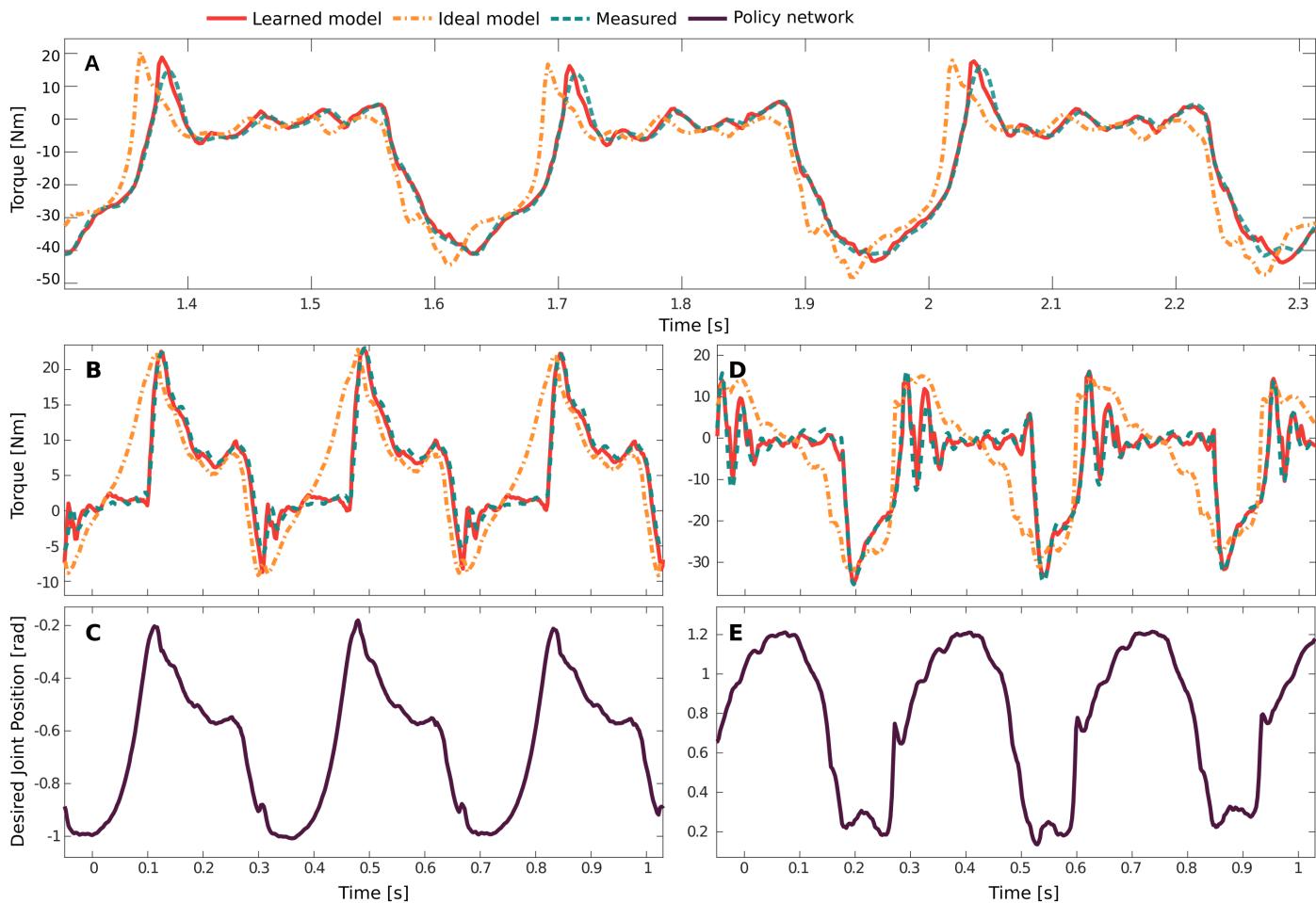
## Reinforcement learning

We represent the control problem in discretized time. At every time step  $t$ , the agent obtains an observation  $o_t \in \mathcal{O}$ , performs an action  $a_t \in \mathcal{A}$ , and achieves a scalar reward  $r_t \in \mathcal{R}$ . We refer to reward and cost interchangeably, with cost being the negative of the reward. We denote by  $\mathbf{O}_t = \langle o_t, o_{t-1}, \dots, o_{t-h} \rangle$  the tuple of recent observations. The agent selects actions according to a stochastic policy  $\pi(a_t | \mathbf{O}_t)$ , which is a distribution over actions conditioned on the recent observations. The aim is to find a policy that maximizes the discounted sum of rewards over an infinite horizon:

$$\pi^* = \arg \max_{\pi} \mathbb{E}_{\tau(\pi)} \left[ \sum_{t=0}^{\infty} \gamma^t r_t \right] \quad (1)$$

where  $\gamma \in (0, 1)$  is the discount factor, and  $\tau(\pi)$  is the trajectory distribution under policy  $\pi$  (the distribution depends on both the policy and the environment dynamics). In our setting, the observations are the measurements of robot states provided to the controller, the actions are the position commands to the actuators, and the rewards are specified so as to induce the behavior of interest.

A variety of RL algorithms can be applied to the specified policy optimization problem. We chose Trust Region Policy Optimization (TRPO) (22), a policy gradient algorithm that has been demonstrated to learn locomotion policies in simulation (54). It requires almost no parameter tuning; we used only the default parameters [as provided in (22, 54)] for all learning sessions presented in this paper. We used a fast custom implementation of the algorithm (55). This efficient implementation and fast rigid-body simulation (41) allowed us to generate and process about a quarter of a billion state transitions in roughly 4 hours. A learning session terminates if the average performance of



**Fig. 6. Validation of the learned actuator model.** The measured torque and the predicted torque from the trained actuator model are shown. The “ideal model” curve is computed assuming an ideal actuator (i.e., zero communication delay and zero mechanical response time) and is shown for comparison. (A) Validation set. Data from (B) a command-conditioned policy experiment with 0.75 m/s forward command velocity and (C) its corresponding policy network output. Data from (D) a high-speed locomotion policy experiment with 1.6 m/s forward command velocity and (E) its corresponding policy network output. Note that the measured ground truth in (A) is nearly hidden because the predicted torque from the trained actuator network accurately matches the ground-truth measurements. Test data were collected at one of the knee joints.

the policy does not improve by more than a task-specific threshold within 300 TRPO iterations.

### Observation and action

The observations in our method should be observable (i.e., can be inferred from measurements) on the real robot and relevant for the task. The joint angles, velocities, and body twists are all observable and highly relevant. Measuring the body orientation is not straightforward because only two degrees of freedom in the orientation are observable with an inertial measurement unit (IMU). The set of observable degrees in the orientation is in bijection with  $S^2$ , or with a unit vector, which can be interpreted as the direction of the gravity vector expressed in the IMU frame. We denote this unit vector as  $\phi^g$ . The height of the base is not observable, but we can estimate it from the leg kinematics, assuming the terrain is flat. A simple height estimator based on a 1D Kalman filter was implemented along with the existing state estimation (56). However, this height estimator cannot be used when the robot is not on its feet, so we removed the height observation when training for recovery from a fall. The whole observation at  $t = t_k$  is

defined as  $o_k = \langle \phi^g, r_z, v, \omega, \phi, \dot{\phi}, \Theta, a_{k-1}, C \rangle$ , where  $r_z$ ,  $v$ , and  $\omega$  are height, linear, and angular velocities of the base,  $\phi$  and  $\dot{\phi}$  are positions and velocities of the joints,  $\Theta$  is a sparsely sampled joint state history,  $a_{k-1}$  is the previous action, and  $C$  is the command. The joint state history is sampled at  $t = t_k - 0.01$  s and  $t = t_k - 0.002$  s.

The joint state history was essential in training a locomotion policy. We hypothesize that this is due to the fact that it enables contact detection. An alternative way to detect contacts is to use force sensors, which give a reliable contact state estimate. However, such sensors increase the weight of the end effectors and consequently lower the energy efficiency of the robot. The exact history configuration was found empirically by analyzing the final performance of the policy.

Our policy outputs low-impedance joint position commands, which we find to be very effective in many tasks. Peng and van de Panne (57) found that such a controller can outperform a torque controller in both training speed and final control performance. Although there is always a bijective map between them, the two action parameterizations have different smoothness and thus different training difficulty. In addition, a position policy has an advantage in training

because it starts as a standing controller, whereas a torque controller initially creates many trajectories that result in falling. Thus, we use the policy network as an impedance controller. Our network outputs a single position reference, which is converted to torque using fixed gains ( $k_p = 50 \text{ N}\cdot\text{m rad}^{-1}$  and  $k_d = 0.1 \text{ N}\cdot\text{m rad}^{-1} \text{ s}^{-1}$ ) and zero target velocity. The position gain is chosen to be roughly the nominal range of torque ( $\pm 30 \text{ N}\cdot\text{m}$ ) divided by the nominal range of motion ( $\pm 0.6 \text{ rad}$ ). This ensures that the policy network has similar output range for torque and position. The velocity gain is chosen to be sufficiently high to prevent unwanted oscillation on the real robot. From our experience, the final locomotion performance is robust against a small change in gains. For instance, increasing the position gain to  $80 \text{ N}\cdot\text{m rad}^{-1}$  does not noticeably change the performance.

Note that the position policy we use here is different from position controllers commonly used in robotics. Position controllers are sometimes limited in performance when the position reference is time indexed, which means that there is a higher-level controller that assumes that the position plan will be followed at high accuracy. This is the main reason that torque controllers have become popular in legged robotics. However, as in many other RL literature, our control policy is state indexed and does not suffer from the limitations of common PD controllers. The policy is trained to foresee that position errors will occur and even uses them to generate acceleration and interaction forces. In addition, thanks to kinematic randomization, a trained policy does not solely rely on kinematics: The policy inevitably has to learn to exert appropriate impulse on the environment for locomotion. This makes our policy more robust because impulse-based control approaches are known to be more robust against system changes and model inaccuracies (44).

### Policy training details

The control policies presented in this work were trained only in simulation. To train performant policies using only simulated data, we followed both standard and problem-specific training procedures. Here, we describe them in detail and explain the rationale behind them.

Training control policies for locomotion have been demonstrated multiple times in literature. (22, 24, 25). However, many of the trained policies do not manifest natural motions, and it is highly questionable whether they will work on physical systems. Some researchers have noticed that naive methods cannot generate natural-looking and energy-efficient locomotion behaviors (58). Low penalty on joint torque and velocity results in unnatural motions, whereas high penalty on them results in a standing behavior. The main reason for the standing behavior is that such a behavior is already a good local minimum when there is high penalty associated with motion.

We solved this problem by introducing a curriculum: Using a curriculum, we shape the initial cost landscape such that the policy is strongly attracted to a locomotion policy and then later polish the motion to satisfy the other criteria. A simple curriculum was generated by modulating the coefficients of the cost terms and the disturbance via a multiplicative curriculum factor. We define a curriculum factor that describes the progression of the curriculum:  $k_c = k_0 \in (0, 1)$  corresponds to the start of the curriculum and  $k_c = 1$  corresponds to the final difficulty level. The intermediate values are computed as  $k_{c,j+1} \leftarrow (k_{c,j})^{k_d}$ , where  $k_d \in (0, 1)$  is the advance rate, which describes how quickly the final difficulty level is reached, and  $j$  is the iteration index of RL training. The sequence of curriculum factors is monotonically increasing and asymptotically converging to 1 within the given

parameter intervals. We suspect that many other update rules adhering to these criteria will result in similar learning performance. All of cost terms are multiplied by this curriculum factor, except the cost terms related to the objective (i.e., base velocity error cost in the command-conditioned and high-speed locomotion task and base orientation cost in recovery task). This way, the robot first learns how to achieve the objective and then how to respect various constraints. This technique is related to curriculum learning introduced by Bengio *et al.* (59), which incrementally introduces samples of more difficulties. Instead of altering the samples, we alter the objective to control the training difficulty. For all training sessions, we use  $k_0 = 0.3$  and  $k_d = 0.997$ . The parameter  $k_0$  should be chosen to prevent the initial tendency to stand still. It can be easily tuned by observing the first 100 iterations of the RL algorithm. The parameter  $k_d$  is chosen such that the curriculum factor almost reaches 1 (or  $\sim 0.9$ ) at the end of training. Although the required number iterations are not known *a priori*, there are sufficient publications on RL applications (including this one) to provide necessary insights to the users.

We tuned the discount factor  $\gamma$  (Eq. 1) separately for each task based on the qualitative performance of the trained controllers in simulation. For training the command-conditioned controller and the high-speed controller, we used  $\gamma = 0.9988$ , which corresponds to a half-life of 5.77 s. We also successfully trained almost equally performant policies with a lower half-life (2 s), but they manifest a less natural standing posture. For training the recovery controller, we used  $\gamma = 0.993$ , which corresponds to a half-life of 4.93 s. A sufficiently high discount factor shows more natural standing posture owing to the fact that it penalizes standing torque more than motion (torque, joint velocities, and other quantities incurring due to motion). However, a too-high discount factor might result in a slow convergence, so it should be tuned appropriately depending on the task. For training command-conditioned and high-speed locomotion, TRPO finished training in 9 days of simulated time, which corresponds to 4 hours of computation in real time. For training for recovery from a fall, TRPO took 79 days of simulated time, which corresponds to 11 hours of computation in real time.

For command-conditioned and high-speed locomotion, we represent a command by three desired body velocity values: forward velocity, lateral velocity, and the turning rate. During training, the commands are sampled randomly from predefined intervals (see tables S1 and S2 for details), and the cost defined in section S3 is used. The initial state of the robot is sampled either from a previous trajectory or a random distribution, shown in table S3, with equal probability. This initialization procedure generates data containing complicated state transitions and robustifies the trained controller. Each trajectory lasts 6 s unless the robot reaches a terminal state earlier. There are two possibilities for termination: violating joint limits and hitting the ground with the base. Upon termination, agent receives a cost of 1 and is reinitialized. The value of the termination cost is not tuned: Because only the ratio between the cost coefficients is important for the final performance, we tune other cost terms to work with this terminal value.

For training recovery from a fall, the collision bodies of the ANYmal model are randomized in size and position. Samples that result in unrealistic internal collisions are removed. The cost function and the initial state distribution are described in section S4 and fig. S3, respectively. The special initialization method in section S4 is needed to train for this task, because naive sampling often results in interpenetration and the dynamics become unrealistic. To this end, we

dropped ANYmal from a height of 1.0 m with randomized orientations and joint positions, ran the simulation for 1.2 s, and used the resulting state as initialization.

Another crucial detail is that joint velocities cannot be directly measured on the real robot. Rather, they are computed by numerically differentiating the position signal, which results in noisy estimates. We modeled this imperfection by injecting a strong additive noise [ $U(-0.5, 0.5)$  rad/s] to the joint velocity measurements during training. This way, we ensured that the learned policy is robust to inaccurate velocity measurements. We also added noise during training to the observed linear velocity [ $U(-0.08, 0.08)$  m/s] and angular velocity [ $U(-0.16, 0.16)$  m/s] of the base. The rest of the observations were noise free. Removing velocities from the observation altogether led to a complete failure to train, although in theory, the policy network could infer velocities as finite differences of observed positions. We explain this by the fact that nonconvexity of network training makes appropriate input preprocessing important. For similar reasons, input normalization is necessary in most learning procedures.

We implemented the policy with an MLP with two hidden layers, with 256 and 128 units each and tanh nonlinearity (Fig. 5). We found that the nonlinearity has a strong effect on performance on the physical system. Performance of two trained policies with different activation functions can be very different in the real world even when they perform similarly in simulation. Our explanation is that unbounded activation functions, such as rectified linear unit, can degrade performance on the real robot, because actions can have very high magnitude when the robot reaches states that were not visited during training. Bounded activation functions, such as tanh, yield less aggressive trajectories when subjected to disturbances. We believe that this is true for softsign as well, but it was not tested in policy networks owing to an implementation issue in our RL framework (55).

## Deployment on the physical system

We used the ANYmal robot (5), shown in step 4 of Fig. 1, to demonstrate the real-world applicability of our method. ANYmal is a dog-sized quadrupedal robot weighing about 32 kg. Each leg is about 55 cm long and has three actuated degrees of freedom, namely, hip abduction/adduction, hip flexion/extension, and knee flexion/extension.

ANYmal is equipped with 12 SEAs (60, 61). An SEA is composed of an electric motor, a high gear ratio transmission, an elastic element, and two rotary encoders to measure spring deflection and output position. In this work, we used a joint-level PD controller with low feedback gains on the joint-level actuator module of the ANYmal robot. The dynamics of the actuators contain multiple components in succession, as follows. First, the position command is converted to the desired torque using a PD controller. Subsequently, the desired current is computed using a PID controller from the desired torque. The desired current is then converted to phase voltage using a field-oriented controller, which produces the torque at the input of the transmission. The output of the transmission is connected to an elastic element whose deflection finally generates torque at the joint (39). These highly complex dynamics introduce many hidden internal states that we do not have direct access to and complicate our control problem.

After acquiring a parameter set for a trained policy from our hybrid simulation, the deployment on the real system was straightforward. A custom MLP implementation and the trained parameter set were ported to the robot's onboard PC. This network was evaluated at 200 Hz for command-conditioned/high-speed locomotion and at 100 Hz for recovery from a fall. We found that performance was un-

expectedly insensitive to the control rate. For example, the recovery motion was trained at 20 Hz but performance was identical when we increased the control rate up to 100 Hz. This was possible because the flip-up behaviors involve low joint velocities (mostly below 6 rad/s). More dynamic behaviors (e.g., locomotion) often require a much higher control rate to have an adequate performance. A higher frequency (100 Hz) was used for experiments because it made less audible noise. Even at 100 Hz, evaluation of the network uses only 0.25% of the computation available on a single CPU core.

## SUPPLEMENTARY MATERIALS

[robotics.sciencemag.org/cgi/content/full/4/26/eau5872/DC1](http://robotics.sciencemag.org/cgi/content/full/4/26/eau5872/DC1)

Section S1. Nomenclature

Section S2. Random command sampling method used for evaluating the learned command-conditioned controller

Section S3. Cost terms for training command-conditioned locomotion and high-speed locomotion tasks

Section S4. Cost terms for training recovery from a fall

Fig. S1. Base velocity tracking performance of the learned controller while following random commands.

Fig. S2. Base velocity tracking performance of the best existing method while following random commands.

Fig. S3. Sampled initial states for training a recovery controller.

Table S1. Command distribution for training command-conditioned locomotion.

Table S2. Command distribution for training high-speed locomotion.

Table S3. Initial state distribution for training both the command-conditioned and high-speed locomotion.

Movie S1. Locomotion policy trained with a learned actuator model.

Movie S2. Random command experiment.

Movie S3. Locomotion policy trained with an analytical actuator model.

Movie S4. Locomotion policy trained with an ideal actuator model.

Movie S5. Performance of a learned high-speed policy.

Movie S6. Performance of a learned recovery policy.

## REFERENCES AND NOTES

- M. Raibert, K. Blankespoor, G. Nelson, R. Playter, BigDog, the rough-terrain quadruped robot. *IFAC Proc. Vol.* **41**, 10822–10825 (2008).
- G. Nelson, A. Saunders, N. Neville, B. Swilling, J. Bondaryk, D. Billings, C. Lee, R. Playter, M. Raibert, PETMAN: A humanoid robot for testing chemical protective clothing. *J. Robot. Soc. Japan* **30**, 372–377 (2012).
- S. Seok, A. Wang, M. Y. Chuah, D. Otten, J. Lang, S. Kim, Design principles for highly efficient quadrupeds and implementation on the MIT Cheetah robot, in *2013 IEEE International Conference on Robotics and Automation* (IEEE, 2013), pp. 3307–3312.
- Spotmini autonomous navigation, [https://youtu.be/Ve9kWv\\_KXUs](https://youtu.be/Ve9kWv_KXUs). [accessed 11 August 2018].
- M. Hutter, C. Gehring, D. Jud, A. Lauber, C. D. Bellicoso, V. Tsounis, J. Hwangbo, K. Bodie, P. Fankhauser, M. Bloesch, R. Diethelm, S. Bachmann, A. Melzer, M. Hoepflinger, ANYmal - a highly mobile and dynamic quadrupedal robot, in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (IEEE, 2016), pp. 38–44.
- R. J. Full, D. E. Koditschek, Templates and anchors: Neuromechanical hypotheses of legged locomotion on land. *J. Exp. Biol.* **202**, 3325–3332 (1999).
- M. H. Raibert, J. J. Craig, Hybrid position/force control of manipulators. *J. Dyn. Sys. Meas. Control* **103**, 126–133 (1981).
- J. Pratt, J. Carff, S. Drakunov, A. Goswami, Capture point: A step toward humanoid push recovery, in *2006 6th IEEE-RAS International Conference on Humanoid Robots* (IEEE, 2006), pp. 200–207.
- A. Goswami, B. Espiau, A. Keramane, Limit cycles in a passive compass gait biped and passivity-mimicking control laws. *Auton. Robots* **4**, 273–286 (1997).
- W. J. Schwind, Spring loaded inverted pendulum running: A plant model, thesis, University of Michigan (1998).
- M. Kalakrishnan, J. Buchli, P. Pastor, M. Mistry, S. Schaal, Fast, robust quadruped locomotion over challenging terrain, in *2010 IEEE International Conference on Robotics and Automation* (IEEE, 2010), pp. 2665–2670.
- C. D. Bellicoso, F. Jenelten, C. Gehring, M. Hutter, Dynamic locomotion through online nonlinear motion optimization for quadrupedal robots. *IEEE Robot. Autom. Lett.* **3**, 2261–2268 (2018).
- M. Neunert, F. Farshidian, A. W. Winkler, J. Buchli, Trajectory optimization through contacts and automatic gait discovery for quadrupeds. *IEEE Robot. Autom. Lett.* **2**, 1502–1509 (2017).

14. I. Mordatch, E. Todorov, Z. Popović, Discovery of complex behaviors through contact-invariant optimization. *ACM Trans. Graph.* **31**, 43 (2012).
15. F. Farshidian, M. Neunert, A. W. Winkler, G. Rey, J. Buchli, An efficient optimal planning and control framework for quadrupedal locomotion, in *2017 IEEE International Conference on Robotics and Automation (ICRA)* (IEEE, 2017), pp. 93–100.
16. M. Posa, C. Cantu, R. Tedrake, A direct method for trajectory optimization of rigid bodies through contact. *Int. J. Robot. Res.* **33**, 69–81 (2014).
17. J. Carius, R. Ranftl, V. Koltun, M. Hutter, Trajectory optimization with implicit hard contacts. *IEEE Robot. Autom. Lett.* **3**, 3316–3323 (2018).
18. S. Levine, P. Pastor, A. Krizhevsky, J. Ibarz, D. Quillen, Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *Int. J. Robot. Res.* **37**, 421–436 (2018).
19. R. Tedrake, T. W. Zhang, H. S. Seung, Stochastic policy gradient reinforcement learning on a simple 3d biped, in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (IEEE, 2004), pp. 2849–2854.
20. J. Yosinski, J. Clune, D. Hidalgo, S. Nguyen, J. C. Zagal, H. Lipson, Evolving robot gaits in hardware: The hyperNEAT generative encoding vs. parameter optimization, in *Proceedings of the 20th European Conference on Artificial Life* (Springer MIT Press, 2011), pp. 890–897.
21. S. Levine, V. Koltun, Learning complex neural network policies with trajectory optimization, in *Proceedings of the 31st International Conference on Machine Learning, PMLR* (PMLR, 2014), pp. 829–837.
22. J. Schulman, S. Levine, P. Abbeel, M. Jordan, P. Moritz, Trust region policy optimization, in *International Conference on Machine Learning (ICML)* (Omnipress, 2015), pp. 1889–1897.
23. J. Schulman, F. Wolski, P. Dhariwal, A. Radford, O. Klimov, Proximal policy optimization algorithms, arXiv:1707.06347 (2017).
24. N. Heess, S. Sriram, J. Lemmon, J. Merel, G. Wayne, Y. Tassa, T. Erez, Z. Wang, A. Esfandiari, M. Riedmiller, D. Silver, Emergence of locomotion behaviours in rich environments, arXiv:1707.02286 (2017).
25. X. B. Peng, G. Berseth, K. Yin, M. Van De Panne, Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning. *ACM Trans. Graph.* **36**, 41 (2017).
26. Z. Xie, G. Berseth, P. Clary, J. Hurst, M. van de Panne, Feedback control for Cassie with deep reinforcement learning, arXiv:1803.05580 (2018).
27. T. Lee, F. C. Park, A geometric algorithm for robust multibody inertial parameter identification. *IEEE Robot. Autom. Lett.* **3**, 2455–2462 (2018).
28. M. Neunert, T. Boaventura, J. Buchli, *Why Off-The-Shelf Physics Simulators Fail in Evaluating Feedback Controller Performance—A Case Study for Quadrupedal Robots*. *Advances in Cooperative Robotics* (World Scientific, 2017), pp. 464–472.
29. J. Bongard, V. Zykov, H. Lipson, Resilient machines through continuous self-modeling. *Science* **314**, 1118–1121 (2006).
30. D. Nguyen-Tuong, M. Seeger, J. Peters, Model learning with local gaussian process regression. *Adv. Robot.* **23**, 2015–2034 (2009).
31. D. Nguyen-Tuong, J. Peters, Learning robot dynamics for computed torque control using local gaussian processes regression, in *2008 ECSIS Symposium on Learning and Adaptive Behaviors for Robotic Systems (LAB-RS)* (IEEE, 2008), pp. 59–64.
32. K. Nikzad, J. Ghaboussi, S. L. Paul, Actuator dynamics and delay compensation using neurocontrollers. *J. Eng. Mech.* **122**, 966–975 (1996).
33. R. S. Sutton, A. G. Barto, *Reinforcement Learning: An Introduction* (MIT Press, 1998), vol. 1.
34. I. Mordatch, K. Lowrey, E. Todorov, Ensemble-cio: Full-body dynamic motion planning that transfers to physical humanoids, in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (IEEE, 2015), pp. 5307–5314.
35. J. Tan, T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez, V. Vanhoucke, Sim-to-real: Learning agile locomotion for quadruped robots. *Proc. Robot. 10.15607/RSS.2018.XIV.010* (2018).
36. X. B. Peng, M. Andrychowicz, W. Zaremba, P. Abbeel, Sim-to-real transfer of robotic control with dynamics randomization. arXiv:1710.06537 (2017).
37. N. Jakobi, P. Husbands, I. Harvey, *Noise and The Reality Gap: The Use of Simulation in Evolutionary Robotics*, *European Conference on Artificial Life* (Springer, 1995), pp. 704–720.
38. A. Dosovitskiy, V. Koltun, Learning to act by predicting the future, in *International Conference on Learning Representations (ICLR)* (PMLR, 2017).
39. C. Gehring, S. Coros, M. Hutter, C. D. Bellicoso, H. Heijnen, R. Diethelm, M. Bloesch, P. Fankhauser, J. Hwangbo, M. Hoepflinger, R. Siegwart, Practice makes perfect: An optimization-based approach to controlling agile motions for a quadruped robot. *IEEE Robot. Autom. Mag.* **23**, 34–43 (2016).
40. R. Featherstone, *Rigid Body Dynamics Algorithms* (Springer, 2014).
41. J. Hwangbo, J. Lee, M. Hutter, Per-contact iteration method for solving contact dynamics. *IEEE Robot. Autom. Lett.* **3**, 895–902 (2018).
42. R. Smith, Open dynamics engine (2005).
43. D. A. Winter, *The Biomechanics and Motor Control of Human Gait: Normal, Elderly and Pathological* (University of Waterloo Press, ed. 2, 1991).
44. H.-W. Park, S. Park, S. Kim, Variable-speed quadrupedal bounding using impulse planning: Untethered high-speed 3d running of MIT cheetah 2, in *2015 IEEE International Conference on Robotics and Automation (IEEE, 2015)*, pp. 5163–5170.
45. Introducing wildcat, <https://youtu.be/wE3fmFTtP9g> [accessed 6 August 2018].
46. A. W. Winkler, F. Farshidian, D. Pardo, M. Neunert, J. Buchli, Fast trajectory optimization for legged robots using vertex-based zmp constraints. *IEEE Robot. Autom. Lett.* **2**, 2201–2208 (2017).
47. S. Shamsuddin, L. I. Ismail, H. Yusoff, N. I. Zahari, S. Bahari, H. Hashim, A. Jaffar, Humanoid robot Nao: Review of control and motion exploration, in *2011 IEEE International Conference on Control System, Computing and Engineering (IEEE, 2011)*, pp. 511–516.
48. U. Saranli, M. Buehler, D. E. Koditschek, Rhex: A simple and highly mobile hexapod robot. *Int. J. Robot. Res.* **20**, 616–631 (2001).
49. E. Ackerman, Boston dynamics sand flea robot demonstrates astonishing jumping skills. *IEEE Spectrum Robotics Blog* **2**, (2012).
50. J. Morimoto, K. Doya, Acquisition of stand-up behavior by a real robot using hierarchical reinforcement learning. *Robot. Auton. Syst.* **36**, 37–51 (2001).
51. C. Semini, N. G. Tsagarakis, E. Guglielmino, M. Focchi, F. Cannella, D. G. Caldwell, Design of HyQ – a hydraulically and electrically actuated quadruped robot. *Proc. Inst. Mec. Eng. Part I J. Syst. Control Eng.* **225**, 831–849 (2011).
52. N. G. Tsagarakis, S. Morfey, G. M. Cerda, L. Zhibin, D. G. Caldwell, COMpliant huMANoid COMAN: Optimal joint stiffness tuning for modal frequency control, in *2013 IEEE International Conference on Robotics and Automation (IEEE, 2013)*, pp. 673–678.
53. J. Bergstra, G. Desjardins, P. Lamblin, Y. Bengio, Quadratic polynomials learn better image features. Technical report, 1337 (2009).
54. J. Schulman, P. Moritz, S. Levine, M. Jordan, P. Abbeel, High-dimensional continuous control using generalized advantage estimation, in *Proceedings of the International Conference on Learning Representations (ICLR)* (2016).
55. J. Hwangbo, I. Sa, R. Siegwart, M. Hutter, Control of a quadrotor with reinforcement learning. *IEEE Robot. Autom. Lett.* **2**, 2096–2103 (2017).
56. M. Bloesch, M. Hutter, M. A. Hoepflinger, S. Leutenegger, C. Gehring, C. D. Remy, R. Siegwart, State estimation for legged robots—consistent fusion of leg kinematics and imu. *Robotics* **17**, 17–24 (2013).
57. X. B. Peng, M. van de Panne, Learning locomotion skills using deeprl: Does the choice of action space matter? in *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation (ACM, 2017)*, p. 12.
58. W. Yu, G. Turk, C. K. Liu, Learning symmetric and low-energy locomotion. *ACM Trans. Graph.* **37**, 144 (2018).
59. Y. Bengio, J. Louradour, R. Collobert, J. Weston, Curriculum learning, in *Proceedings of the 26th annual International Conference on Machine Learning (ACM, 2009)*, pp. 41–48.
60. G. A. Pratt, M. M. Williamson, Series elastic actuators, in *Proceedings 1995 IEEE/RSJ International Conference on Intelligent Robots and Systems. Human Robot Interaction and Cooperative Robots* (IEEE, 1995), pp. 399–406.
61. M. Hutter, K. Bodie, A. Lauber, J. Hwangbo, Joint unit, joint system, robot for manipulation and/or transportation, robotic exoskeleton system and method for manipulation and/or transportation EP16181251 (2016).

**Acknowledgments:** We thank ANYbotics for responsive support on ANYmal. **Funding:** The project was funded, in part, by the Intel Network on Intelligent Systems and the Swiss National Science Foundation (SNF) through the National Centre of Competence in Research Robotics and Project 200021-166232. The work has been conducted as part of ANYmal Research, a community to advance legged robotics. **Author contributions:** J.H. conceived the main idea of the train and control methods, set up the simulation, and trained networks for the command-conditioned locomotion and the high-speed locomotion. J.L. trained a network for recovery from a fall. J.H., A.D., M.H., and V.K. refined ideas and contributed in the experiment design. J.H. and J.L. performed experiments together. D.B. and V.T. helped setting up the hardware for the experiments. J.H., A.D., M.H., and V.K. analyzed the data and prepared the manuscript. **Competing interests:** The authors declare that they have no competing interests. **Data and materials availability:** All data needed to evaluate the conclusions in the paper are present in the paper and/or the Supplementary Materials. Other materials can be found at [https://github.com/junja94/anymal\\_science\\_robotics\\_supplementary](https://github.com/junja94/anymal_science_robotics_supplementary).

Submitted 3 September 2018

Accepted 12 December 2018

Published 16 January 2019

10.1126/scirobotics.aau5872

**Citation:** J. Hwangbo, J. Lee, A. Dosovitskiy, D. Bellicoso, V. Tsounis, V. Koltun, M. Hutter, Learning agile and dynamic motor skills for legged robots. *Sci. Robot.* **4**, eaau5872 (2019).

## Learning agile and dynamic motor skills for legged robots

Jemin Hwangbo, Joonho Lee, Alexey Dosovitskiy, Dario Bellicoso, Vassilios Tsounis, Vladlen Koltun and Marco Hutter

*Sci. Robotics* **4**, eaau5872.  
DOI: 10.1126/scirobotics.aau5872

### ARTICLE TOOLS

<http://robotics.sciencemag.org/content/4/26/eaau5872>

### SUPPLEMENTARY MATERIALS

<http://robotics.sciencemag.org/content/suppl/2019/01/14/4.26.eaau5872.DC1>

### RELATED CONTENT

<http://robotics.sciencemag.org/content/robotics/5/46/eabb9764.full>  
<http://robotics.sciencemag.org/content/robotics/5/49/eabb2174.full>

### REFERENCES

This article cites 25 articles, 2 of which you can access for free  
<http://robotics.sciencemag.org/content/4/26/eaau5872#BIBL>

### PERMISSIONS

<http://www.sciencemag.org/help/reprints-and-permissions>

Use of this article is subject to the [Terms of Service](#)

---

*Science Robotics* (ISSN 2470-9476) is published by the American Association for the Advancement of Science, 1200 New York Avenue NW, Washington, DC 20005. The title *Science Robotics* is a registered trademark of AAAS.

Copyright © 2019 The Authors, some rights reserved; exclusive licensee American Association for the Advancement of Science. No claim to original U.S. Government Works