NAVIGATION

ANYmal parkour: Learning agile navigation for quadrupedal robots

David Hoeller[1,2]*†, Nikita Rudin[1,2]*†, Dhionis Sako[1], Marco Hutter[1]

Performing agile navigation with four-legged robots is a challenging task because of the highly dynamic motions, contacts with various parts of the robot, and the limited field of view of the perception sensors. Here, we propose a fully learned approach to training such robots and conquer scenarios that are reminiscent of parkour challenges. The method involves training advanced locomotion skills for several types of obstacles, such as walking, jumping, climbing, and crouching, and then using a high-level policy to select and control those skills across the terrain. Thanks to our hierarchical formulation, the navigation policy is aware of the capabilities of each skill, and it will adapt its behavior depending on the scenario at hand. In addition, a perception module was trained to reconstruct obstacles from highly occluded and noisy sensory data and endows the pipeline with scene understanding. Compared with previous attempts, our method can plan a path for challenging scenarios without expert demonstration, offline computation, a priori knowledge of the environment, or taking contacts explicitly into account. Although these modules were trained from simulated data only, our real-world experiments demonstrate successful transfer on hardware, where the robot navigated and crossed consecutive challenging obstacles with speeds of up to 2 meters per second.

INTRODUCTION

Parkour, also known as free running, is a discipline originating in the late 1980s that has gained popularity with the advent of the internet. Free runners perform acrobatic stunts where the goal is to attain a hard-to-reach location in the most elegant and efficient manner. It involves navigating through the environment by walking, running, climbing, and jumping over obstacles, and the athlete must coordinate these agile skills in a precisely timed sequence. This discipline requires years of practice to develop the necessary competencies, intuitions, and reflexes.

Although legged robots aspire to be as nimble and agile as humans or other animals, we are still far from fully exploiting robotic capabilities to achieve similar behaviors. By aiming to match the agility of free runners, we can better understand the limitations of each component in the pipeline from perception to actuation, circumvent those limits, and generally increase the capabilities of our robots, which in return paves the road for many new applications, such as search and rescue in collapsed buildings or complex natural terrains.

The complexity of the parkour task exacerbates many of the challenges commonly faced by mobile robots. The robot must sense its environment to develop an understanding of the rapidly changing surrounding scene and select a feasible path and sequence of motions based on its set of skills. In the case of large and challenging obstacles, it has to perform dynamic maneuvers at the limits of actuation while accurately controlling the motion of the base and limbs. All of the above must be achieved in real time with limited onboard computing and using its exteroceptive sensors' partial and noisy information.

This work aims to solve the abovementioned challenges and proposes a method to perform agile navigation with a quadrupedal robot in parkour-like settings (Fig. 1). We split the pipeline into three interconnected components trained and deployed sim-to-real: a perception module, a locomotion module, and a navigation module (Fig. 2). The perception module receives point cloud measurements from the onboard cameras and the LiDAR (light detection and ranging) and computes an estimate of the terrain around the robot. The locomotion module contains a catalog of locomotion skills that can overcome specific terrains. The navigation module guides the locomotion module in the environment by selecting which skill to activate and providing intermediate commands. Each of these learning-based modules was trained in simulation.

In our experimental validation, we demonstrate the system's ability to solve the problem autonomously, resulting in behaviors not shown before with such platforms. The robot could cross difficult terrains with speeds of up to 2 m/s and make the right navigation decisions to reach the target in time. The locomotion controllers performed precise and agile movements, sometimes on narrow boxes barely the size of the robot's footprint, and leveraged the system's full range of motion to pass higher obstacles. The mapping pipeline, which provides updates at a high frequency, correctly reconstructed the scene despite state estimation and sensing noise stemming from the robot's fast speeds. Last, the planner used the available information and its intrinsic knowledge of each skill's capabilities to guide the robot around the course on a feasible path. All of these components were designed with efficiency in mind. They scaled properly when training with thousands of agents in simulation and operated in real time on the real robot. We show that the complete pipeline could be deployed sim-to-real and achieve high agility despite the harsh conditions of the real world.

Related work

This work builds on state-of-the-art methods in locomotion, planning, and perception. Legged locomotion has been tackled by multiple approaches ranging from model-based to fully learned techniques. Model predictive control can be used to traverse challenging terrains requiring precise foot placement (*1*–*3*), but the approach is limited by

[1] Robotic Systems Lab, ETH Zurich, Zurich, Switzerland. [2] NVIDIA, Zurich, Switzerland.
*Corresponding author. Email: dhoeller@nvidia.com (D.H.); rudinn@ethz.ch (N.R.)
†These authors contributed equally to this work.

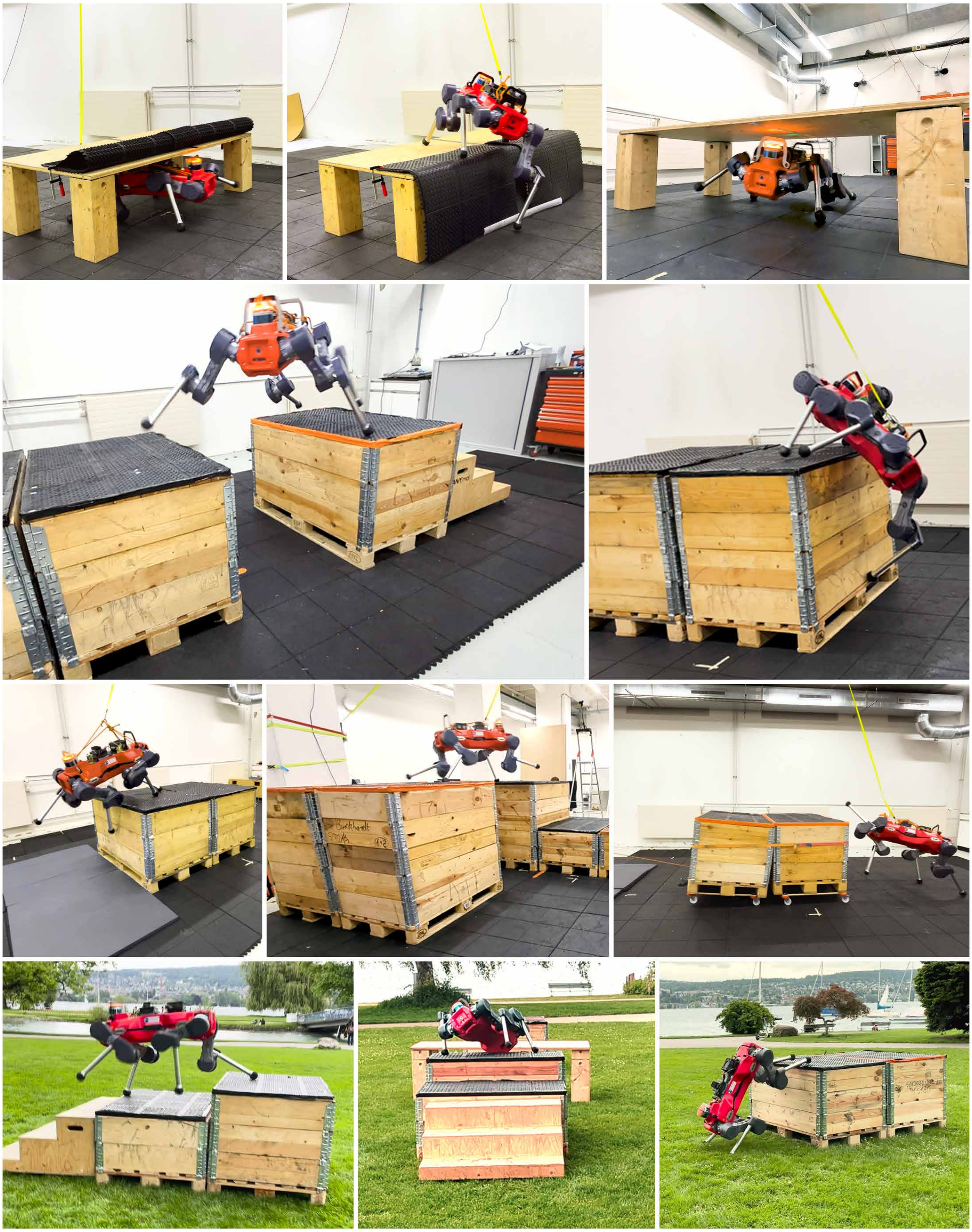


Fig. 1. Deployment of the pipeline on the quadrupedal robot ANYmal D. The robot performs highly dynamic maneuvers and makes contact with its limbs where necessary.

the simplified underlying model and the strong assumptions about the contact schedule. Deep reinforcement learning has proven to be an effective solution for robust perceptive locomotion (*4*–*6*). Nevertheless, such approaches are still far from exploiting the full potential of the robot. Agile locomotion has been of strong interest since the first legged robots (*7*). In recent years, with the combined democratization of commercially available quadrupedal platforms and openly available deep reinforcement learning frameworks, various new tasks have been demonstrated. Notable examples include jumping and climbing (*8*–*11*), performing cat-like landing motions (*12*, *13*), recovering from

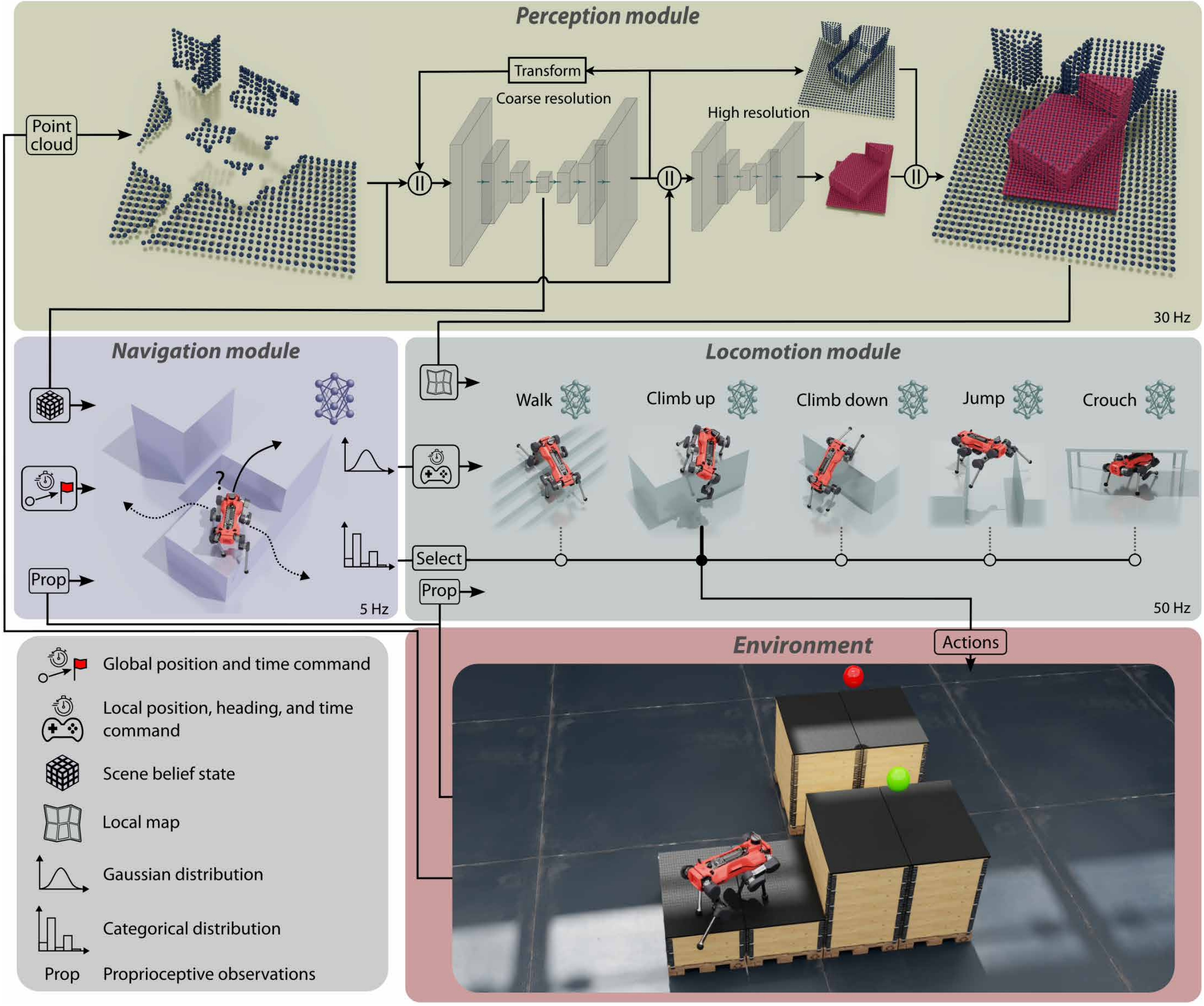


Fig. 2. Description of our approach. We decompose the problem into three components: The perception module receives the point cloud measurements to estimate the scene's layout and produces a latent tensor and a map. The locomotion module contains several low-level skills that can overcome specific scenarios. The navigation module is given a target goal and uses the latent to plan a path and select the correct skill.

falls (*14*, *15*), and dribbling with a football (*16*, *17*). In parallel, bipedal robots have also demonstrated their agile capabilities by walking blindly on rough terrain (*18*) and jumping on obstacles (*19*).

Navigation is typically achieved with a hierarchical setup, where a planner computes a feasible and collision-free path, which a controller then tracks. Although sampling-based methods are commonly used to create such a path (*20*), using these techniques with legged robots is challenging because of the system's hybrid nature. The robot must constantly make and break contact with the environment to influence its motion, which leads to the combinatorial explosion of the set of possible solutions. As a result, the problem is usually simplified to keep it tractable (*21*, *22*). Arguably, learning-based methods can break down such complexity and provide a more straightforward way to guide the robot from point A to B. Previous works have trained navigation policies from expert demonstration (*23*, *24*), using reinforcement learning (*25*–*27*), fully self-supervised (*28*), or by combining sampling-based planning with learned motion costs (*29*).

Hierarchical reinforcement learning has gained attention in the field of robotics because it enables robots to acquire, combine, and reuse versatile skills to solve complex tasks. Pretraining low-level skills with imitation learning and then controlling them through latent actions have been proposed for both character animation (*30*) and robotics (*16*). Combining multiple expert policies has also been explored by switching between policies trained to imitate fragments of motions (*31*), fusing locomotion policies with gating neural networks (*32*), or distillation (*33*).

To perceive the environment, navigation and locomotion pipelines for legged robots heavily rely on elevation maps (*1*, *6*, *34*), which are susceptible to noise and inaccurate state estimation. Such perceptive representations, however, cannot represent the full three-dimensional (3D) configuration of the world and cannot extrapolate beyond visible data, which is necessary to pass below obstacles or to reconstruct the top surfaces of higher obstacles. For navigation, signed distance fields (*35*) are commonly used because they can easily be integrated into the problem formulation to avoid elements in the scene. The exteroceptive measurements can also be directly provided as input to the policy (*5*, *36*). These methods, however, involve multiple stages that provide direct supervision to the perceptive part.

In this work, we trained locomotion skills using the position-based task formulation of (*11*). Similar to (*31*), the navigation module then learned to steer and switch between those skills. For the perception module, we took inspiration from (*37*) to reconstruct the environment in 3D from point cloud data. We augmented the method with a multi-resolution scheme to combine a higher-resolution map near the robot with a coarser larger-scale map further away.

RESULTS

We deployed the pipeline on the quadrupedal robot ANYmal D. It weighs around 55 kg, has 12 series elastic actuators capable of producing a torque of 85 N m each, and is equipped with a total of six Intel RealSense depth cameras (two in the front, two in the back, one left, and one right) and a Velodyne Puck LiDAR. The whole system was implemented in several ROS nodes across different onboard computers. The locomotion and navigation modules operated synchronously in a single node on the onboard computer. The perception module was implemented on an NVIDIA Jetson Orin and operated asynchronously with the rest of the system. The navigation and locomotion policies used the last received message from the perception module to infer their respective networks. More details on the setup are provided in Supplementary Methods "Implementation details." Movie 1 summarizes the proposed approach and shows indoor and outdoor experiments on the real robot.

The three learning-based modules operated without expert demonstration, offline computation, or a priori knowledge of the environment and enabled the robot to reliably reach a target across different arrangements of randomized obstacles. Figure 3 shows two trajectories and the corresponding profiles of the robot's speed, the selected skills, and the joint positions and torques for one of the leg's hip flexion-extension (HFE) and knee flexion-extension (KFE) motors. The robot crossed the terrain swiftly and chose suitable skills at every time step. It reached speeds of up to 2 m/s and underwent fast accelerations and decelerations [Fig. 3, A(i) and B(i)]. The system leveraged a large portion of the motor's range and often reached maximum torque. Along trajectory A, the HFE motor deflected by more than 160° [Fig. 3A(ii)], which was necessary for the leg to reach the other side of the gap and catch the fall of the robot during the climb-down maneuver. In trajectory B, the policy saturated the motor during the climb to propel the robot onto the 0.95-m-high platform [Fig. 3B(iii)].

Movie 1. Summary of experimental design and results.

The system was able to control the robot precisely despite the high speeds. In scenario A, the robot reached the leftmost box after the stairs with a speed of 1.5 m/s. With a width of 0.8 m, the box was smaller than the robot's footprint in standing configuration. At this location, it had to perform precise foothold placement to pass the last step and prepare for the jump, despite the out-of-distribution scenario for the jumping skill, which had been trained with boxes double the size. This shows that the low-level skills could cope with more intricate scenes than what they had been trained on.

In scenario B, the skill selection scheme of the navigation module was nontrivial. At several locations along the path, it chose skills that had not been designed for the specific setting at hand. It favored the jumping skill to quickly turn the robot on the spot in the narrow passages after the first step down or before the climb. This can be explained by the jumping skill's training setup, where it had to jump from one box to another, and the initial and target headings were randomized. The policy learned to turn on the spot in tight spaces and was more capable in such scenarios compared with other skills. The navigation module was capable of discovering such strengths during its training process and exploited them on deployment.

Although the skills were trained separately, the switches along trajectories were unnoticeable on the real robot. This smoothness was not enforced directly and can be explained by the following reasons. First, a harsh transition would lead to a risky motion that the navigation module learned to avoid. Second, all of the skills were also trained on flat ground without obstacles. They showed very similar motions in these scenarios, which resulted in a skill overlap during switches.

The system could also recover from disturbances or crashes; see movie S1. The robot stood up and completed the course after falling from a box, and it could pass a table after heavily slipping due to low ground friction. Moreover, the robot quickly readapted its trajectory when obstacles were pulled away during execution, even though all the components were trained with static environments only. This arose from the rapid response times exhibited by each component, coupled with the perception module's adeptness at promptly rectifying discrepancies between its internal state of belief and current sensor measurements.

In the following analysis, we delve into each component of our proposed approach, revealing how such behaviors can be effectively achieved. All statistical results were computed by averaging over 1000 roll-outs in simulation.

Locomotion module

First, we analyzed the performance and emerging behavior of each locomotion policy separately for a range of difficulties (Fig. 4). Movie S2 shows the deployment of each skill.

Jumping

The robot started on a box and had to jump to a neighboring box separated by a gap of up to 1 m. To perform a successful jump, the robot approached the gap sideways and carefully placed its feet as close as possible to the edge before using the full actuation power to leap to the other side. It used three legs to propel itself, whereas the fourth was extended to land on the other side. The robot then transferred two diagonal legs before bringing the last leg across the gap. Because of randomization, the policy kept the feet at a safe distance from the edge and could recover from missteps and slippage by transferring the robot's weight between the non-leaping legs.

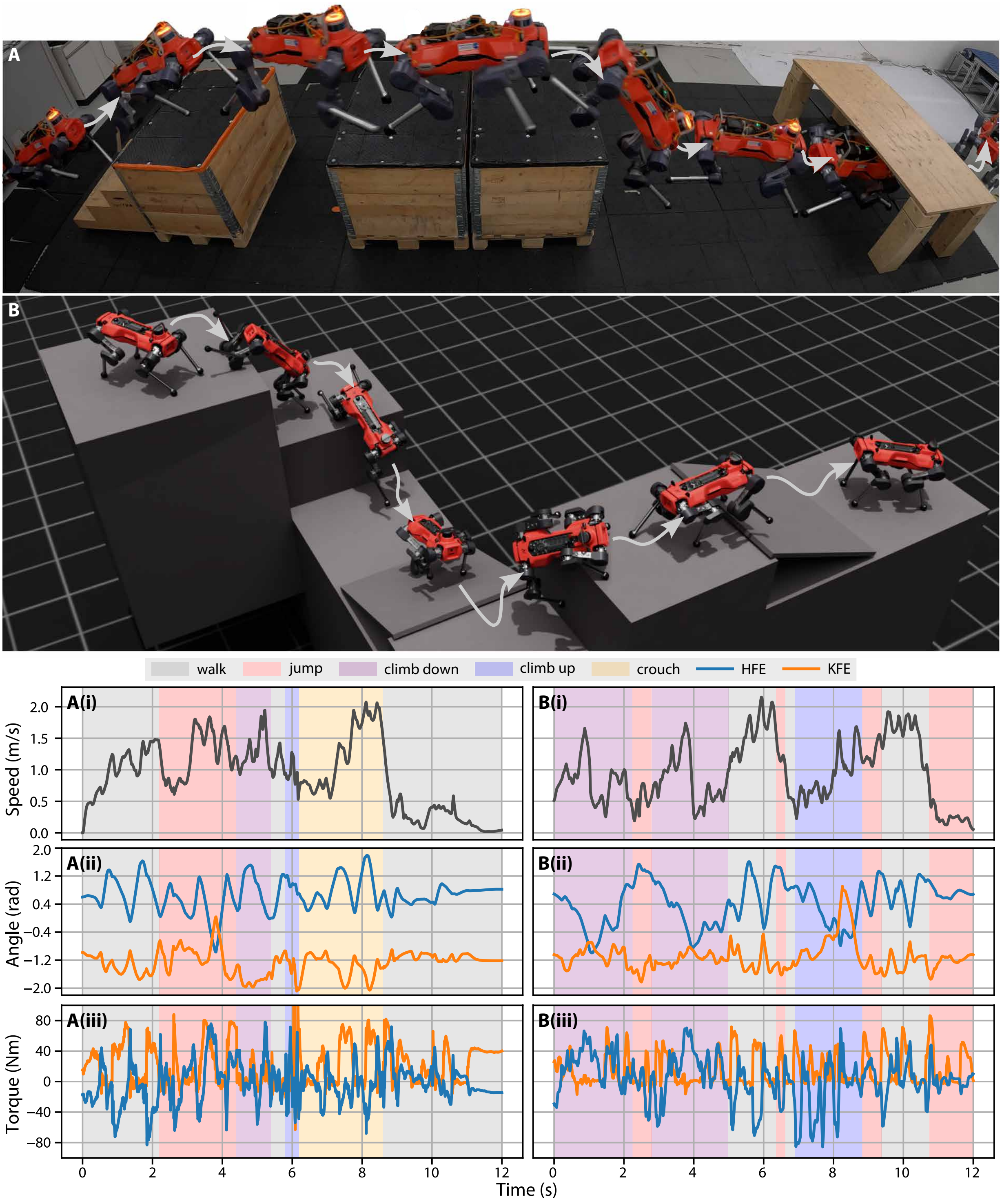
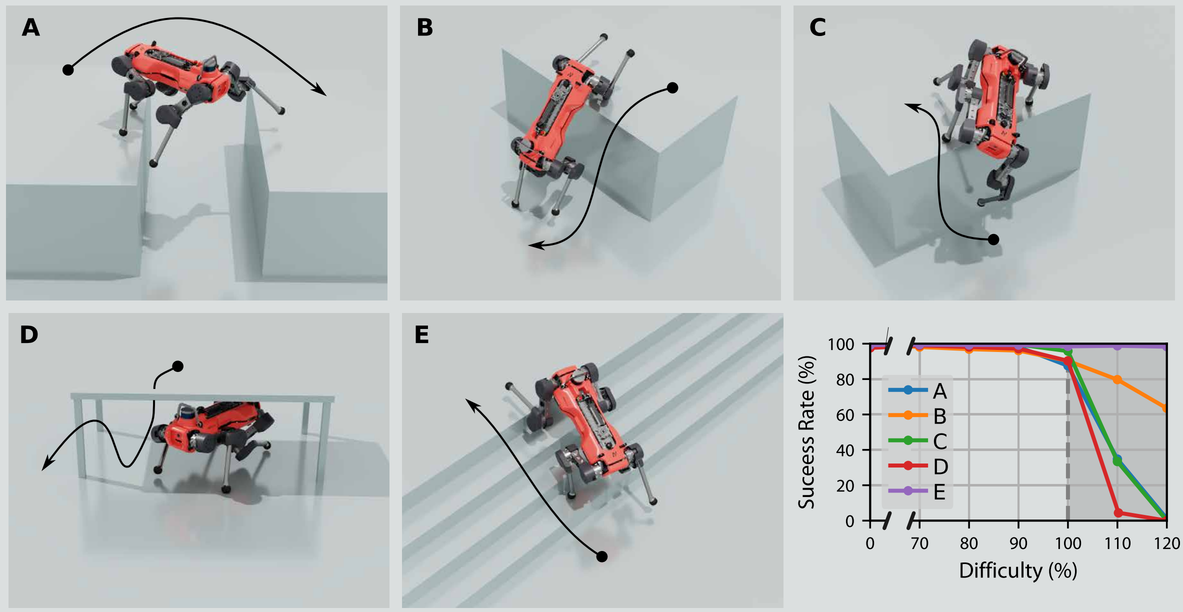


Fig. 3. Deployment of the pipeline on the quadrupedal robot ANYmal D. (**A**) Trajectory on the real robot. (**B**) Trajectory in simulation. A(i to iii) and B(i to iii) depict the profiles of the robot's speed, the selected skills, and two joint angles and torques corresponding to (A) and (B), respectively.

Climbing down

The robot started on a box with a height of up to 1 m and had to climb down to reach a target on the ground. Because we penalized high impacts on its feet to prevent motor damage, the robot first dropped to its knees on the edge and brought its center of gravity as low as possible. It then jumped down to land on its front legs while holding its weight with the back knees on top of the box. It then took a few steps forward on the front legs to reposition itself and allow



	A	B	C	D	E		
Parameter	gap size (m)	box height (m)	box height (m)	table height (m)	slope (°)	step height (m)	obstacle height (m)
Difficulty Range	[0.1, 1.2]	[0.1, 1.0]	[0.1, 1.0]	[0.8, 0.45]	[0, 45]	[0.05, 0.2]	[0.1, 0.4]

Fig. 4. Training scenarios of the locomotion skills with the resulting behaviors. (**A**) Jumping. (**B**) Climbing down. (**C**) Climbing up. (**D**) Crouching. (**E**) Walking. (**F**) The success rate of each skill for obstacles of varying difficulty. (**G**) Ranges of parameters used during training [0 to 100% in (F)].

the back legs to come down gently. The policy learned to be robust to small shifts in the perceived terrain by slowly pushing its feet over the edge until it made contact with its knees. It then used the conveniently L-shaped shank and knees of the robot as a hook on the edge of the box.

Climbing up

The robot started on the ground and had to climb on top of a box with a height of up to 1 m. To climb to the top, the robot put one of its front feet on the top surface and used it to lift itself to an upright configuration. It then repositioned itself before jumping to land with a hind leg on the top while balancing by pushing against the vertical surface with the fourth leg. Last, it propelled the whole body up and brought the fourth leg on top. The robot only used its feet and shanks when possible, but it also learned to use its knees when needed. For example, when the third leg slipped or missed the edge, the robot could use its knee to recover without falling back to the ground.

Crouching

The robot had to reach a target located on the other side of a narrow passage with a minimum height of 0.4 m. When it crossed the table, the robot lowered its base while walking in the desired direction. With a low base height, it adapted its gait and used both hip motors to lift its feet off the ground.

Walking

The robot had to traverse various irregular terrains consisting of stairs and slopes and randomly placed small obstacles. These diverse terrains were similar to the ones widely used in perceptive-legged locomotion works (*4*, *38*). The policy could scale and descend short slopes of 40°, climb steps of 0.25-m step height, and run on flat ground at 2 m/s. Because of the diversity of training scenarios, this policy generalized well to unseen terrains, such as narrow stairs, slopes, or combinations of different obstacles.

Figure 4F shows the success rate of each skill across a range of corresponding obstacles with increasing difficulty. The displayed range covers 0 to 120% of the maximum obstacle difficulty during training. All skills performed well up to 90% of their respective difficulty. After that, the crouching skill's performance dropped the quickest when the passage became narrower than the height of the robot. The performance of the jumping, climbing, and climbing down skills also dropped sharply because of the physical limits of the robot. Last, the walking skill extrapolated well beyond the training range of difficulties because these terrains were less challenging.

Navigation module

The navigation module was trained in simulation on three different terrain types presented in Fig. 5. Scenario C was introduced because of the complexity of constructing scenario B in the real world. For that setup, we had to add additional penalties to prevent the robot from going around the obstacle course. Because of the different formulations, we deployed two separate navigation policies, one trained on scenarios A and B and a separate one trained on scenario C only.

We examined the emerging behaviors of the navigation module (Figs. 3 and 6). It selected appropriate subgoals on the basis of its instantaneous measurement of the terrain by extracting 3D information

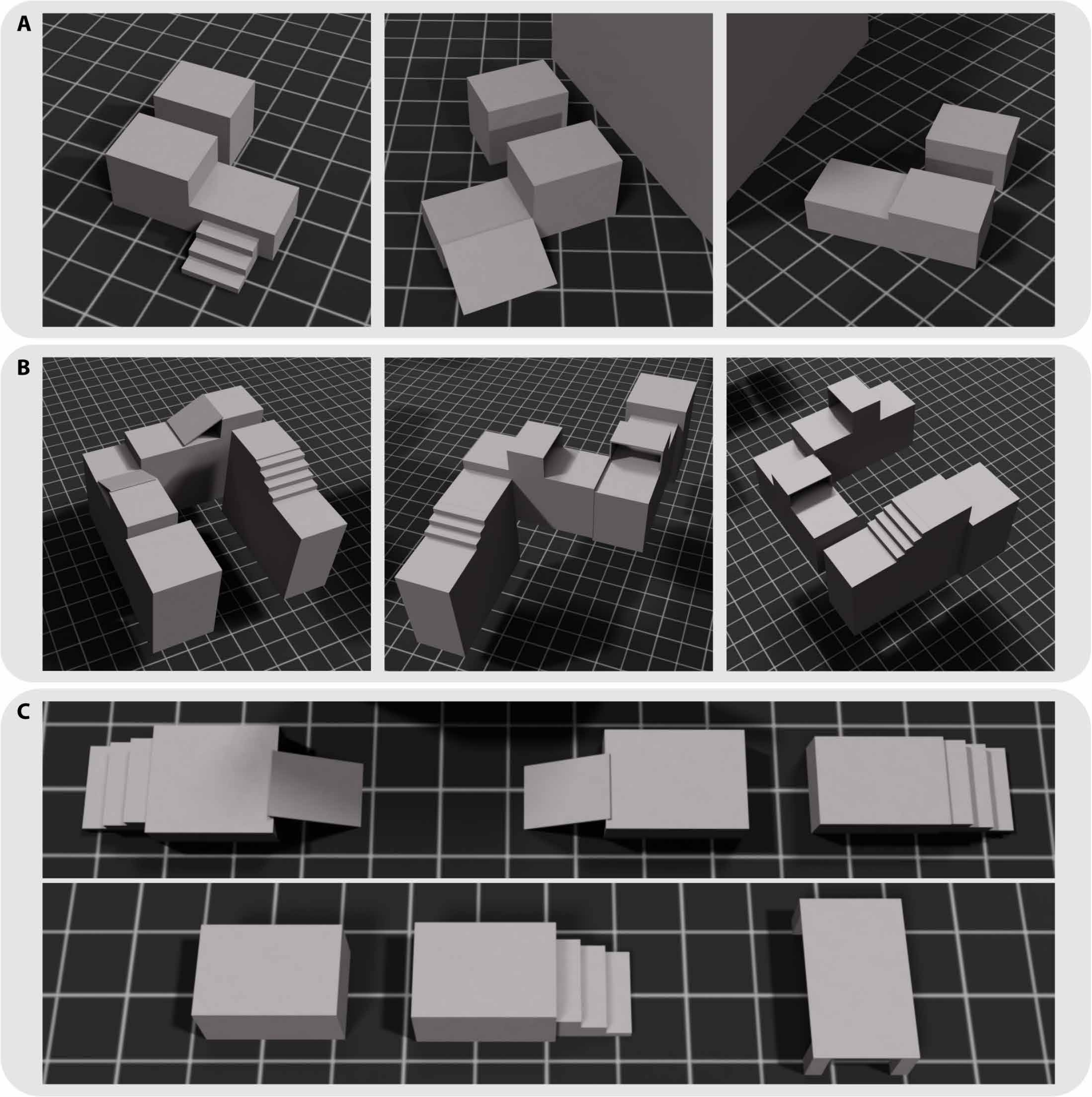


Fig. 5. Types of environments used for training. The dimensions of the individual obstacles and the arrangements were randomized. (**A**) Different arrangements of boxes, where the robot might have to climb and jump over a gap to reach the target. The robot had to reach any of the boxes starting from the ground or reach a target on the ground starting from one of the boxes. (**B**) A parkour line consisting of a long winding platform with multiple obstacles on the way. (**C**) A simplified version of the parkour line due to the complexity of implementing (B) in the real world.

from the latent space of the perception module. For similar environment configurations, it adapted the path depending on the obstacles' dimensions. It also selected the most appropriate policy based on the terrain and sent the right commands to control the robot's trajectory. Upon convergence, the navigation policy could fully control the five locomotion skills across the course to solve the problem. This task was not trivial because of the position-based formulation these policies were trained with. Each low-level policy could modulate the robot's movement freely within the allocated time and only had to comply with the position and heading commands when the time was over. For example, it could comply with the orientation command at any time along the trajectory. The navigation policy learned how to properly combine the position, heading, and timing commands for each skill to achieve the desired motion of the robot. This was particularly important when the robot arrived at high speeds on a narrow obstacle. There, it had to decelerate quickly and turn on the spot to get to the next obstacle.

The navigation module took the capabilities and limitations of each skill into account to adapt the trajectory. This was primarily visible with the climb up, climb down, and crouch skills, where depending on the configuration of the obstacle, it modified its output (movies S3 and S4). When a box was too high, the policy did not go up or down directly because it would have failed. For tables that were too low, it climbed over them rather than crouching underneath. Such adaptation is depicted in Fig. 6, where the robot started on the ground, was commanded to reach the target box in the back

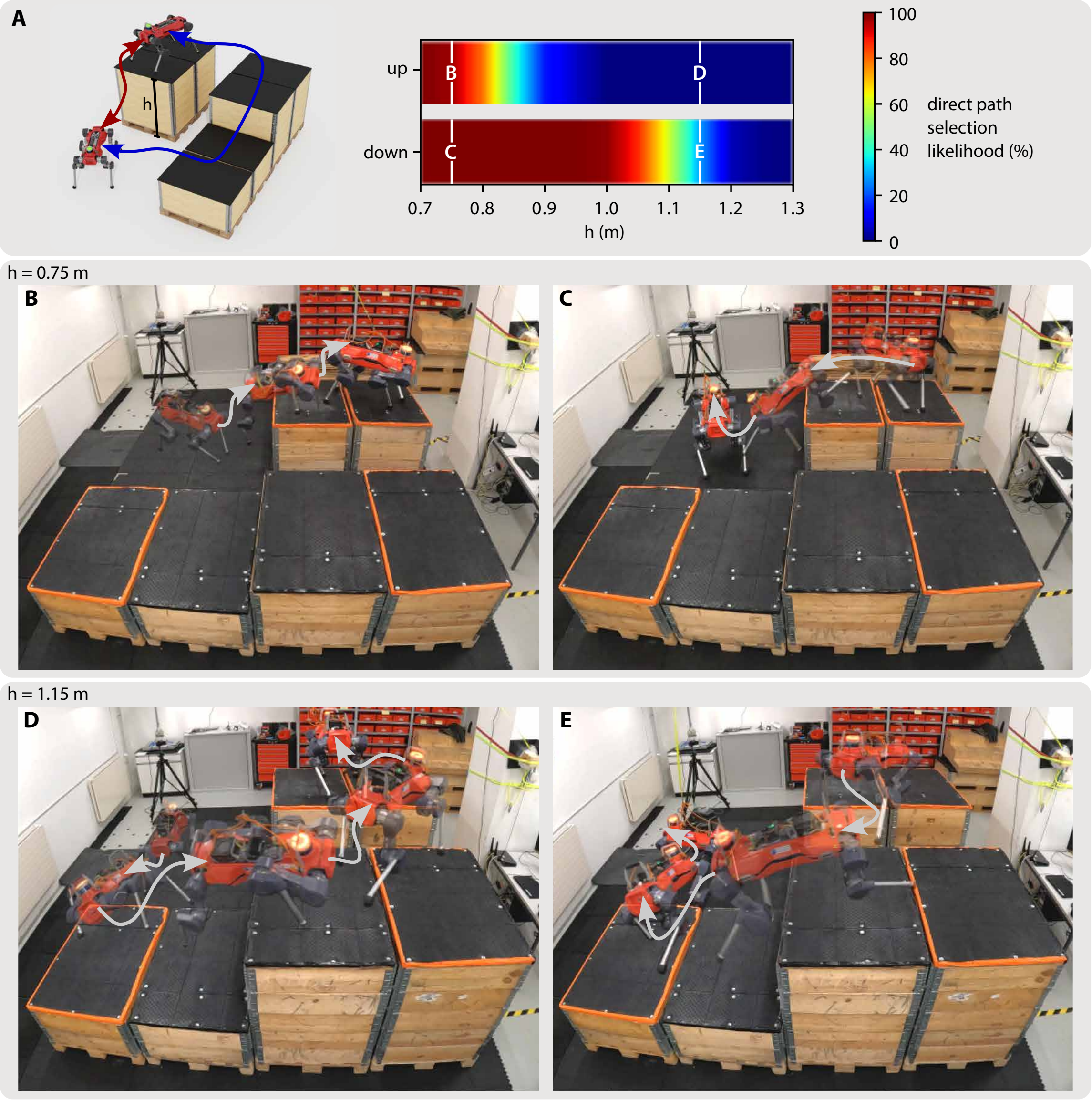


Fig. 6. Adaptive path selection. The robot started on the ground, was given a target on top of the box in the back, and then commanded back to the initial position. (**A**) Likelihood of going up and down along the direct path (red line) as a function of the height of the box. (**B** and **C**) Deployment on the robot for $h = 0.75$ m. (**D** and **E**) Deployment on the robot for $h = 1.15$ m. The trajectories (B to E) are shown in movie S3.

(up), and then go back to the starting position (down). In Fig. 6A, we show how likely the robot was to take the direct path as a function of the height of the box. As the height of the box increased, the policy was more likely to choose a longer but safer path. The policy sent the robot down directly until a height of 1 m, after which it chose the longer route. On the other hand, it switched much more quickly to the longest route when going up, because disturbances and noise have a higher impact on performance for this maneuver.

Figure 6 (B and C) shows the resulting trajectories on the real robot for $h = 0.75$ m, and Fig. 6 (D and E) for $h = 1.15$ m. Another example where the robot had to distance itself from the target to reach distant goals is described in Supplementary Results "Navigation across long ranges."

We compared the performance of our method in simulation against a manually designed trajectory (Table 1) for the different terrains depicted in Fig. 5. For the manual trajectory, we used the known sequence of obstacles to place suitable waypoints along the way and set the most appropriate skill to connect neighboring waypoints. The robot was controlled to reach these waypoints in sequence. The obstacles' difficulty was close to the maximum defined during low-level training (100% in Fig. 4). The table shows that manually placing targets performed well in certain scenarios but failed in other cases where the locomotion policies required finer-grained control. Moreover, our high-level policy learned to dynamically adjust the targets by placing them further away to increase the speed of the robot. Manual demonstrations with targets at key locations (such as in the middle of obstacles) led to lower speeds, thus

Table 1. Comparison of the navigation policy's performance against a manually hard-coded trajectory.

Terrain	Ours	Manual
Fig. 5A	98.2%	95.3%
Fig. 5B	96.3%	60.9%
Fig. 5C	97.6%	75.3%

requiring a longer time to reach the target. Human demonstrations did not scale well when randomizing the terrain because they required hand labeling of each new case.

Perception module

The perception module processed the noisy and occluded point cloud measurements to produce a meaningful latent for the navigation module and a clean reconstruction for the locomotion module. As mentioned earlier, the module operated asynchronously with the rest of the system on deployment. This differed from our training setup, where the latent was available to the navigation policy at the exact time of inference. However, because the induced delay was smaller than the update period of the policy, the performance was unaffected.

We analyzed the reconstructions and compared them against an elevation mapping baseline (*39*) that ran alongside our network. This method provides several improvements to the commonly used framework described in (*40*), making it a stronger contender for the parkour task. It can detect drift in the *z* direction to realign the map to the correct height and perform additional visibility checks to remove outliers. We mainly evaluated the reconstruction performance qualitatively and refer the reader to (*27*) for a quantitative analysis of a similar approach.

Several outputs of the network for scenarios Figs. 3A and 6D (real-world data) are presented in Fig. 7 and in movie S5. The first column corresponds to the measurements, the second to the baseline map visualized as a point cloud, and the last to our reconstruction. Because the baseline was an elevation map, the corresponding point cloud did not contain vertical surfaces. For better distinction, the high-resolution (refinement process around the robot) and low-resolution outputs of our approach are colored in red and blue, respectively. The coarse-resolution outputs (blue points) within the red regions are only shown for comprehension and were not used by the rest of the pipeline.

From the various outputs, it can be seen that the network could cope with sparse measurements and correctly estimate the layout of the scene. The points falling on the edge of the boxes were used as evidence to reconstruct the upper parts at the right height (Fig. 7A). The surface on the right of the robot was correctly identified as a wall and reconstructed accordingly. On the other hand, the baseline did not consider the regions on top of the higher boxes because no measurements were available at these locations.

The coarse network produced less precise reconstructions further away from the robot because of the lower resolution of the voxels and noisy measurements along some of the obstacles' edges. In Fig. 7A, for example, the estimated height of the box to the left of the robot was correct, but the width of approximately 8 cm was too large. However, near the robot, the refiner could deal with such inaccuracies and further enhance the reconstruction. This can be seen in Fig. 7D, where the refiner produced cleaner stairs than the coarse map. The importance of the refiner is further described in Supplementary Results "Ablation of the perception refinement."

The auto-regressive feedback played a key role when the robot crouched under the table in Fig. 7C. Despite the sparsity of the measurements on the top surface, the network remembered this region because it could be seen during the approach in previous time steps. The baseline method was not designed to handle such scenarios with overhangs. It produced a mix containing the top surface at some locations and the ground at others, resulting in an erroneous map.

The robustness of state estimation drift can be seen in Fig. 7 (B and D) by comparing it with the baseline. In Fig. 7B, the robot's position estimate suddenly jumped to the left. Our network detected such situations and immediately corrected the map. The elevation map, on the other hand, could not cope with the drift, and the knees of the robot and the hind leg were inside the map. Similarly, the hind leg was inside the elevation map when climbing stairs (Fig. 7D).

DISCUSSION

This work aims to extend the capabilities of legged robots on highly challenging terrains. We have presented a complete pipeline for robotic parkour, including specially developed low-level locomotion skills, a high-level navigation module, and a perception module. The proposed approach allows the robot to move with unprecedented agility. It can now evolve in complex scenes where it must climb and jump on large obstacles while selecting a nontrivial path toward its target location. The dynamic nature of the task poses multiple challenges that render existing approaches unsuitable. It requires nonstandard locomotion skills at the actuation limit, a planner with an intrinsic understanding of the locomotion capabilities with respect to the surrounding obstacles, and a perception module capable of inferring the 3D topology of the terrain on the basis of the partial observations provided by the sensors.

We propose a fully learned approach where each module uses one or multiple neural networks. The networks are trained in simulation and transferred to the real world. We demonstrate that our task can be solved without premapping or offline planning, and all required computations can happen on board the robot in real time. Using learning-based modules is advantageous for real-world deployment. The complexity of solving the task is shifted to the learning stage. Once the relatively small networks are trained, they display complex behaviors at almost no cost compared with optimization or sampling-based methods, without resorting to limiting assumptions or simplifications.

Our hierarchical approach with discrete skills differs from works where multiple skills are distilled into a single low-level policy either by direct imitation (*33*) or by encoding the skills into a latent space (*16*). The distillation process could potentially lead to better generalization and performance, but this remains to be shown. In the Barkour benchmark (*33*), for example, the distilled version performs worse than the nondistilled one. By keeping the skills separate, our navigation module is able to detect and exploit the strengths of each individual skill, for example, when it uses the jumping policy to turn the robot on the spot in narrow passages. This would not be possible with a distilled policy without performing a careful dataset curation. Also, in our current setup with height scans for locomotion policies,

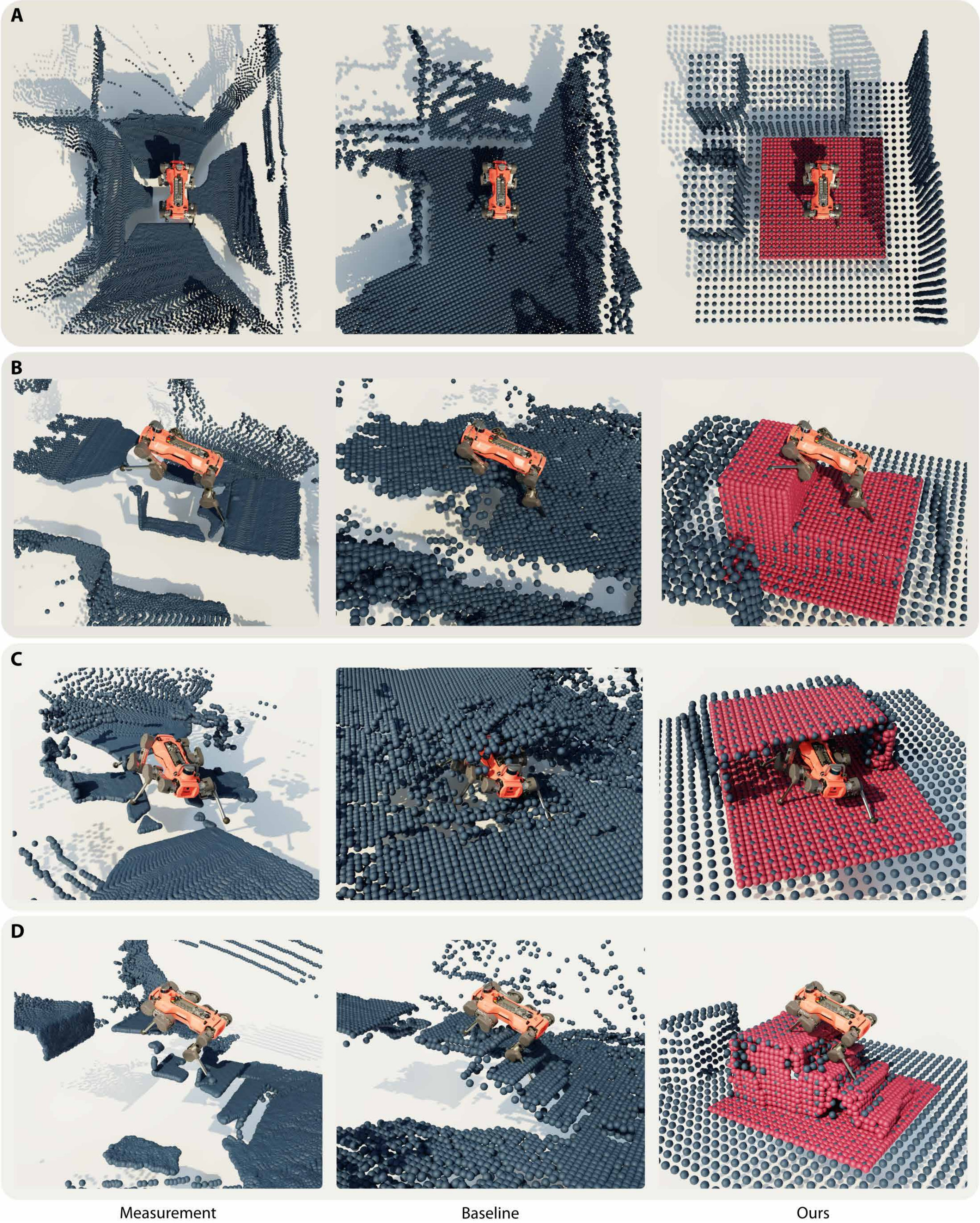


Fig. 7. Terrain reconstructions for different scenarios (real-world data). (**A** to **D**) The first column shows the point cloud measurements, the second shows the baseline elevation map (*39*) viewed as a point cloud, and the last corresponds to the reconstruction with our method. For our method, we show the coarse-resolution output in blue and the high-resolution output (refinement process) in red.

a distilled policy could not distinguish between a box and a table and thus would not be able to learn the crouching skill. A more sophisticated perceptive input, such as depth cameras, or the latent tensor of the perception module could be used, but this would substantially increase training times and require a delay compensation mechanism as described in Supplementary Results "Ablation of the measurement delay compensation." However, combining different skills to generate new types of motions is an important venue for future research that could address some of the limitations that we raise in the next section.

Current limitations

The pipeline has some limitations that remain to be tackled for deployment in realistic and unstructured scenarios. First, the scalability of the method to more diverse scenarios remains to be tested, for example, in unstructured scenes. We showcase the system's capabilities in a limited range of scenarios, using a handful of distinct modules within the environment. To scale complex environments, such as a collapsed building or even a real parkour course, the robot would be required to perceive, navigate, and cross a wider variety of obstacles. We can always train more low-level skills, provide more data to the perception module, and train the navigation module in more diverse scenarios, but it remains to be seen how well these different modules can generalize to completely new scenarios.

Furthermore, training the whole pipeline can be time consuming because it uses a total of eight neural networks, each requiring separate tuning. Some of them are interdependent, meaning that modifying one requires retraining the others. For instance, the navigation module can only receive the latent tensor of the specific perception module it was trained on and has to use the same locomotion policies. In turn, the perception module needs to be retrained if a skill adopts a different motion or if a new obstacle is introduced. A possible solution would be to train the different components simultaneously. This would also probably improve performance because the skills could be fine-tuned on the commands issued by the navigation module, and the skills could further train on the parkour terrains that are more complicated than the ones used during locomotion training. However, it is not straightforward to achieve such fine-tuning with our delayed position-based rewards because the pose commands of the locomotion policies and their corresponding timer are constantly overridden by the navigation module.

Last, because the navigation module must make a series of correct decisions to reach the goal, with many possibilities leading to failure, the algorithm requires many iterations to converge. We developed a specific curriculum to overcome this limitation. Without this step, the robot struggles to discover the correct behaviors and gets stuck in front of larger obstacles. A possible solution would be to pretrain the navigation module using expert demonstrations, for example, by finding candidate solutions with a brute-force search. Although further research is needed to address these limitations, our approach serves as a promising foundation for future work, demonstrating substantial progress in legged robot capabilities on challenging terrains.

MATERIALS AND METHODS

Overview

The goal of the agent is to navigate and locomote in an environment to reach a specific target location within a short amount of time. We constrained the task to different configurations of pallet-sized boxes (see Fig. 5), allowing us to keep the main challenges of agile navigation in a feasible, structured, and repeatable scenario.

Each scenario showcases different capabilities of the pipeline. Scenario A demonstrates the general applicability to realistic but relatively constrained scenarios. The navigation module has to understand the capabilities of the locomotion skills and choose the path accordingly. Although the obstacle arrangements are fairly constrained, the robot can start anywhere on the terrain and must choose different paths depending on the target location and obstacle parameters. On the other hand, scenario B shows generalization to more randomized scenarios with different platform shapes and obstacle arrangements. The sequence of obstacles leads to various cases that the navigation and perception modules must learn to handle correctly. Last, scenario C allows us to force the robot to climb on the obstacles without having to recreate a high winding platform with gaps on either side for real-world deployment.

Pipeline

The pipeline consists of three learning-based modules, which are described in the following subsections. Supplementary Methods define the observations, actions, and rewards of the locomotion and navigation policies and provide further implementation details.

Perception module

The perception module operates at 30 Hz and endows the robot with scene understanding. The navigation and locomotion modules both use its output to make path planning, policy selection, foothold placement, and contact decisions. It ingests noisy and heavily occluded point clouds of the scene coming from depth cameras and LiDAR to produce a 3D estimate of the terrain centered around the robot as well as a compact latent representation of the scene. Similar to (*37*), the point cloud data are spatially and temporally processed using a fully convolutional encoder-decoder network architecture. The encoder takes in the point cloud and compresses it into a compact representation that is used by the decoder to complete the missing information and filter out noise.

To prepare the input, the measurements were first converted to a voxel grid around the robot. In each occupied voxel, a feature describes the position of the centroid of the points that fall within that voxel, and the features of unoccupied voxels were set to 0. Despite the sparse implementation used in (*37*), the library did not scale well to the typical batch sizes required for reinforcement learning. Unexpectedly, a dense formulation can handle such a large batch size with satisfactory speeds, but this comes at the cost of high memory requirements (approximately 45 GB of GPU memory for a batch size of 4096). The decoder outputs the voxel occupancy probability and the position of the centroid for each cell. The reconstructed point cloud can then be recovered by pruning the cells whose occupancy probability is below a user-defined threshold. Contrary to (*37*), no skip connections were used to produce a more informative latent that the navigation module can directly use. Although this might limit the generalization performance, we found that it worked well for our task with randomized parkour worlds.

To balance the trade-off between reconstruction accuracy and map size, we used a multi-resolution scheme consisting of two such networks at different scales (see Fig. 2). First, a coarse-resolution network operates on a 32-by-32-by-32 grid with a voxel size of 12.5 cm (4-m map size), allowing for a broad view of the scene. This network also benefits from an auto-regressive feedback, where the previous output is transformed into the current frame and concatenated with the measurement. This allows the module to accumulate evidence over time and reconstruct parts of the scene that are no longer visible. For example, when the robot passes below a table, the module can use the aggregated information from previous frames to estimate the layout of the table and reconstruct the top surface, even if it is currently not visible to the sensors. This is also necessary with certain maneuvers, such as climbing, where the robot's limbs often block a large portion of the left and right cameras; see Supplementary Methods "Measurement blind spots." The resulting latent tensor of dimension 2 by 2 by 2 by 64 is directly fed to the navigation module's policy as a flattened input. It represents the belief state of the scene because it is trained to contain all the information to reconstruct the partially observable world. Second, a high-resolution network operates on a 32-by-32-by-32 grid with a voxel size of 6.25 cm (2-m map size), allowing for better quality reconstruction near the robot because this region is essential for locomotion and proper foothold placement. Rather than using an auto-regressive feedback, this network uses the features of the coarse-resolution network's last layer as input along with the point cloud measurements. The point cloud output of this network is converted to an elevation map and sent to the locomotion module.

We trained these networks in an unsupervised fashion from simulated data on a total of 2000 trajectories with 100 time steps each. We equally split the dataset across the different parkour scenarios. The occupancy and centroid outputs were trained using a binary cross-entropy loss and the Euclidean distance to the ground truth, respectively. We followed the same data augmentation procedure described in (*37*). It consists of perturbing the position of the points, adding random blobs, removing patches of points, and noisifying the robot's position. We found experimentally that it was necessary to add randomized objects (walls and additional boxes) around the terrains during training to enhance reconstruction performance and to allow deployment in closed rooms and cluttered environments.

Locomotion module

The locomotion module is an interface that exposes the low-level skills to the rest of the pipeline and operates at 50 Hz. It contains a catalog of policies that can be activated by the navigation module, each trained for a specific locomotion skill: walking, climbing up, climbing down, crouching, and jumping. These skills were trained using reinforcement learning and output joint position commands for the motors.

As input, the policies receive the current proprioceptive state, a local map of the surrounding terrain, a position and heading command, a timer, and output position commands to the motors. The training setup resembles (*11*) and uses position-based commands. Instead of tracking velocity commands, the robot has to reach a target position and heading within a given time. The timer input indicates how much time is left to reach the commanded position and adopt the right heading. The tracking reward is only activated when the timer is over, meaning that the robot can modulate its motion freely along the trajectory. The skills are trained separately and share the observation and action spaces but require different flavors of rewards and termination conditions to be trained efficiently. Furthermore, we implemented symmetry augmentations and found that they solved the asymmetry issues reported in (*11*) and led to more robust policies. We describe this procedure in Supplementary Methods "Symmetric data augmentation for locomotion training."

The navigation module receives a full 3D representation of the scene, i.e., the latent tensor, but such a representation is impractical for the locomotion policies because of their high update rate and the corresponding computational cost during training. Instead, they perceive the environment using a set of height measurements around the robot computed from the elevation map coming from the perception module. During training, these height measurements can be efficiently computed using ray-casting, bypassing the need to render all the cameras and infer the perception module, which substantially improves training times, as described in Supplementary Methods "Implementation details." In addition, delays in the perception pipeline, which negatively influence the skill's performance (see Supplementary Results "Ablation of the measurement delay compensation"), can be directly compensated for. The locomotion module registers the maps it receives in the world frame, and between map updates, the height scans for the skills are computed at the robot's estimated location in the world-registered map. Such compensation would not be possible to achieve if the skills were to use the latent directly.

To bridge the reality gap, we perturbed the height measurements during training by adding noise to individual points and shifting the map up to 7.5 cm in all directions. This forces the policies to adopt safer behavior in critical situations. During a climb-down motion, for example, the robot first slides down on its shanks until the knees hit the edge of the box. The motion is less dependent on accurate foothold placement, which improves robustness to slight imperfections in the map reconstructions.

Navigation module

The navigation module guides the robot around the scene to reach the target within the allocated time. It was trained in a hierarchical reinforcement learning setup consisting of an outer loop running the navigation policy at 5 Hz and an inner loop running the locomotion module at 50 Hz. The locomotion policies of the inner loop were frozen throughout training. At every high-level time step, the navigation policy receives the relative position of the goal to reach, the remaining time to accomplish the task, the robot's base velocity, orientation, and the flattened latent tensor of the perception module. The input to the policy is formed by concatenating these values. It then selects a locomotion skill and guides the latter with a local position, heading, and timer command. As mentioned in the previous section, the timer command informs the skill when it has to comply with the position and heading command. By changing the timer output, the navigation policy can give the skill a sense of urgency. The navigation policy must carefully combine and adjust the time, position, and heading commands to achieve the desired motion. Similar to the skills' training setup, we used the time-dependent command formulation described in (*11*). The agent is given a fixed time to reach the goal, and the distance-to-goal penalty is only activated when the time is over. This sparse formulation allows the policy to explore the terrain to find safer paths and take its time where needed. The episode is also terminated if the robot falls or the contact forces are too high. To speed up convergence, we used a curriculum where we first placed the global targets close to the robots' starting positions and then moved them farther away on the terrain as the reward increased.

To accommodate for the formulation, we modified the PPO algorithm and augmented the actor's multilayer perceptron with a hybrid output. The last layer's features are split to form a Gaussian distribution for the commands and a categorical distribution for skill activation. The categorical distribution assigns a selection probability for each of the low-level skills. During training, the actions were sampled from the respective distributions to enable exploration. On deployment, we use the mean of the Gaussian and select the policy with the highest assigned probability. Although this hybrid formulation adds a new hyperparameter to the algorithm to scale the entropy coefficient of each distribution, we found empirically that this does not make training more complex.

Compared with other approaches such as (*27*), which deploy simplified kinematic models in the inner loop, rolling out the actual low-level policies during training is necessary to perform agile navigation. The agent can make informed decisions taking into account the mode of operation, the capabilities, and the limitations of each low-level controller. It can infer when a box is too high to climb on and first move toward a lower one. It carefully places the target on narrow passages to enable fine-grained foot placement. It favors the climb-down policy on lower boxes, to step down to avoid high contact forces. Similar to the perception module, we add random perceptive distractors during training (walls and boxes) to improve the generalization of the policy to enclosed and cluttered rooms.

Supplementary Materials

This PDF file includes:
Methods
Results
Figs. S1 to S7
Tables S1 to S7
Legends for movies S1 to S5
References (*41*–*43*)

Other Supplementary Material for this manuscript includes the following:
Movies S1 to S5

REFERENCES AND NOTES

1. R. Grandia, F. Jenelten, S. Yang, F. Farshidian, M. Hutter, Perceptive locomotion through nonlinear model predictive control. arXiv:2208.08373 [quant-ph] (2022).
2. F. Jenelten, R. Grandia, F. Farshidian, M. Hutter, TAMOLS: Terrain-aware motion optimization for legged systems. *T-RO* **38**, 3395–3413 (2022).
3. D. Kim, D. Carballo, J. Di Carlo, B. Katz, G. Bledt, B. Lim, S. Kim, Vision aided dynamic exploration of unstructured terrain with a small-scale quadruped robot, in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, Paris, France, August 2020 (IEEE, 2020), pp. 2464–2470.
4. T. Miki, J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, M. Hutter, Learning robust perceptive locomotion for quadrupedal robots in the wild. *Sci. Robot.* **7**, eabk2822 (2022).
5. A. Loquercio, A. Kumar, J. Malik, Learning visual locomotion with cross-modal supervision, In *2023 International Conference on Robotics and Automation (ICRA)* (IEEE, 2023), pp. 7295–7302.
6. S. Gangapurwala, M. Geisert, R. Orsolino, M. Fallon, I. Havoutis, Rloc: Terrain-aware legged locomotion using reinforcement learning and optimal control. *T-RO* **38**, 2908–2927 (2022).
7. M. H. Raibert, *Legged Robots That Balance* (MIT Press, 1986), pp. 1–89.
8. D. Kim, J. D. Carlo, B. Katz, G. Bledt, S. Kim, Highly dynamic quadruped locomotion via whole-body impulse control and model predictive control. arXiv:1909.06586 (2019).
9. Q. Nguyen, M. J. Powell, B. Katz, J. D. Carlo, S. Kim, Optimized jumping on the MIT Cheetah 3 robot, in *2019 International Conference on Robotics and Automation (ICRA)*, Montreal, QC, Canada, May 2019 (IEEE, 2019), pp. 7448–7454.
10. H.-W. Park, P. M. Wensing, S. Kim, Jumping over obstacles with MIT Cheetah 2. *Robot. Auton. Syst.* **136**, 103703 (2021).
11. N. Rudin, D. Hoeller, M. Bjelonic, M. Hutter, Advanced skills by learning locomotion and local navigation end-to-end, in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Kyoto, Japan, October 2022 (IEEE, 2022), pp. 2497–2503.
12. N. Rudin, H. Kolvenbach, V. Tsounis, M. Hutter, Cat-like jumping and landing of legged robots in low gravity using deep reinforcement learning. *IEEE Trans. Robot.* **38**, 317–328 (2022).
13. S. H. Jeon, S. Kim, D. Kim, Real-time optimal landing control of the MIT Mini Cheetah. arXiv:2110.02799 [cs.RO] (2021).
14. 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).
15. Y. Ma, F. Farshidian, M. Hutter, Learning arm-assisted fall damage reduction and recovery for legged mobile manipulators. arXiv:2303.05486 [cs.RO] (2023).
16. S. Bohez, S. Tunyasuvunakool, P. Brakel, F. Sadeghi, L. Hasenclever, Y. Tassa, E. Parisotto, J. Humplik, T. Haarnoja, R. Hafner, M. Wulfmeier, M. Neunert, B. Moran, N. Siegel, A. Huber, F. Romano, N. Batchelor, F. Casarini, J. Merel, R. Hadsell, N. Heess, Imitate and repurpose: Learning reusable robot movement skills from human and animal behaviors. arXiv:2203.17138 [cs.RO] (2022).
17. Y. Ji, G. B. Margolis, P. Agrawal, Dribblebot: Dynamic legged manipulation in the wild, in *2023 IEEE International Conference on Robotics and Automation (ICRA)* (IEEE, 2023), pp. 5155–5162.
18. J. Siekmann, K. Green, J. Warila, A. Fern, J. Hurst, Blind bipedal stair traversal via sim-to-real reinforcement learning, in *Proceedings of Robotics: Science and Systems XVII*, D. A. Shell, M. Toussaint, M. A. Hsieh, Eds. (RSS, 2021); www.roboticsproceedings.org/rss17/p061.html.
19. Z. Li, X. B. Peng, P. Abbeel, S. Levine, G. Berseth, K. Sreenath, Robust and versatile bipedal jumping control through multi-task reinforcement learning. arXiv:2302.09450 [cs.RO] (2023).
20. S. Karaman, E. Frazzoli, Sampling-based algorithms for optimal motion planning. *Int. J. Robot. Res* **30**, 846–894 (2011).
21. S. Tonneau, A. Del Prete, J. Pettré, C. Park, D. Manocha, N. Mansard, An efficient acyclic contact planner for multiped robots. *T-RO* **34**, 586–601 (2018).
22. L. Wellhausen, M. Hutter, Rough terrain navigation for legged robots using reachability planning and template learning, in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Prague, Czech Republic, October 2021 (IEEE, 2021), pp.6914–6921.
23. M. Pfeiffer, M. Schaeuble, J. Nieto, R. Siegwart, C. Cadena, From perception to decision: A data-driven approach to end-to-end motion planning for autonomous ground robots, in *IEEE International Conference on Robotics and Automation (ICRA)*, Singapore, June 2017 (IEEE, 2017), pp. 1527–1533.
24. E. Kaufmann, A. Loquercio, R. Ranftl, A. Dosovitskiy, V. Koltun, D. Scaramuzza, Deep drone racing: Learning agile flight in dynamic environments, in *Proceedings of the 2nd Conference on Robot Learning (PMLR)*, A. Billard, A. Dragan, J. Peters, J. Morimoto, Eds. (MLResearchPress, 2018), pp. 133–145.
25. F. Sadeghi, S. Levine, CAD2RL: Real single-image flight without a single real image, in *Proceedings of Robotics: Science and Systems XIII*, N. Amato, S. Srinivasa, N. Ayanian, S. Kuindersma, Eds. (RSS, 2017); www.roboticsproceedings.org/rss13/p34.html.
26. F. Sadeghi, DIVIS: Domain invariant visual servoing for collision-free goal reaching, in *Proceedings of Robotics: Science and Systems XV*, A. Bicchi, H. Kress-Gazit, S. Hutchinson, Eds. (RSS, 2019); www.roboticsproceedings.org/rss15/p55.html.
27. D. Hoeller, L. Wellhausen, F. Farshidian, M. Hutter, Learning a state representation and navigation in cluttered and dynamic environments. *IEEE Robot. Autom. Lett.* **6**, 5081–5088 (2021).
28. G. Kahn, P. Abbeel, S. Levine, BADGR: An autonomous self-supervised learning-based navigation system. arXiv:2002.05700 (2020).
29. B. Yang, L. Wellhausen, T. Miki, M. Liu, M. Hutter, Real-time optimal navigation planning using learned motion costs, in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, Xi'an, China, June 2021 (IEEE, 2021), pp. 9283–9289.
30. X. B. Peng, Y. Guo, L. Halper, S. Levine, S. Fidler, ASE: Large-scale reusable adversarial skill embeddings for physically simulated characters. *TOG* **41**, 1–17 (2022).
31. J. Merel, A. Ahuja, V. Pham, S. Tunyasuvunakool, S. Liu, D. Tirumala, N. Heess, G. Wayne, Hierarchical visuomotor control of humanoids, paper presented at ICLR 2019: The Seventh International Conference on Learning Representations, New Orleans, USA, 6 to 9 May 2019.
32. C. Yang, K. Yuan, Q. Zhu, W. Yu, Z. Li, Multi-expert learning of adaptive legged locomotion. *Sci. Robot.* **5**, eabb2174 (2020).
33. K. Caluwaerts, A. Iscen, J. C. Kew, W. Yu, T. Zhang, D. Freeman, K.-H. Lee, L. Lee, S. Saliceti, V. Zhuang, N. Batchelor, S. Bohez, F. Casarini, J. E. Chen, O. Cortes, E. Coumans, A. Dostmohamed, G. Dulac-Arnold, A. Escontrela, E. Frey, R. Hafner, D. Jain, B. Jyenis, Y. Kuang, E. Lee, L. Luu, O. Nachum, K. Oslund, J. Powell, D. Reyes, F. Romano, F. Sadeghi, R. Sloat, B. Tabanpour, D. Zheng, M. Neunert, R. Hadsell, N. Heess, F. Nori, J. Seto, C. Parada, V. Sindhwani, V. Vanhoucke, J. Tan, Barkour: Benchmarking animal-level agility with quadruped robots. arXiv:2305.14654 [cs.RO] (2023).
34. R. O. Chavez-Garcia, J. Guzzi, L. M. Gambardella, A. Giusti, Learning ground traversability from simulations. *IEEE Robot. Autom. Lett.* **3**, 1695–1702 (2018).
35. H. Oleynikova, Z. Taylor, M. Fehr, R. Siegwart, J. Nieto, Voxblox: Incremental 3D Euclidean signed distance fields for on-board MAV planning, in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Vancouver, BC, Canada, September 2017 (IEEE, 2017), pp. 1366–1373.
36. A. Agarwal, A. Kumar, J. Malik, D. Pathak, Legged locomotion in challenging terrains using egocentric vision, in *Proceedings of the 6th Annual Conference on Robot Learning (PMLR)*, K. Liu, D. Kulic, J. Ichnowski, Eds. (MLResearchPress, 2022), pp. 403–415.
37. D. Hoeller, N. Rudin, C. Choy, A. Anandkumar, M. Hutter, Neural scene representation for locomotion on structured terrain. *IEEE Robot. Autom. Lett.* **7**, 8667–8674 (2022).
38. N. Rudin, D. Hoeller, P. Reist, M. Hutter, Learning to walk in minutes using massively parallel deep reinforcement learning, in *Proceedings of the 5th Conference on Robot Learning (PMLR)*, A. Faust, D. Hsu, G. Neumann, Eds. (MLResearch Press, 2022), pp. 91–100.
39. T. Miki, L. Wellhausen, R. Grandia, F. Jenelten, T. Homberger, M. Hutter, Elevation mapping for locomotion and navigation using GPU, in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (IEEE, 2022), pp. 2273–2280.
40. P. Fankhauser, M. Bloesch, M. Hutter, Probabilistic terrain mapping for mobile robots with uncertain localization. *IEEE Robot. Autom. Lett.* **3**, 3019–3026 (2018).

41. M. Macklin, Warp: A High-Performance Python Framework for GPU Simulation and Graphics (NVIDIA GPU Technology Conference, 2022); www.nvidia.com/en-us/on-demand/session/gtcspring22-s41599/.
42. V. Makoviychuk, L. Wawrzyniak, Y. Guo, M. Lu, K. Storey, M. Macklin, D. Hoeller, N. Rudin, A. Allshire, A. Handa, G. State, Isaac gym: High performance GPU based physics simulation for robot learning, in *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1 (Round 2)*, J. Vanschoren, S. Yeung, Eds. (NeurIPS, 2021).
43. F. Abdolhosseini, H. Y. Ling, Z. Xie, X. B. Peng, M. Van de Panne, On learning symmetric locomotion, in *Proceedings of the 12th ACM SIGGRAPH Conference on Motion*, *Interaction and Games* (ACM, 2019), pp. 1–10.

Acknowledgments

Funding: The project was funded by NVIDIA, the Swiss National Science Foundation (SNF) through the National Centre of Competence in Research Robotics, and the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program grant agreement nos. 852044 and 780883. The work has been conducted as part of ANYmal Research, a community to advance legged robotics. **Author contributions:** D.H. and N.R. developed and implemented the method. D.S., D.H., and N.R. conducted the experiments and designed and constructed the obstacle courses. D.H., N.R., and M.H. refined ideas and worked on the manuscript. **Competing interests:** The authors declare that they have no competing interests. **Data and materials availability:** All data needed to support the conclusions of this manuscript are included in the main text or Supplementary Materials. The datasets and codes to generate Fig. 3 and Fig. 4 are available at https://doi.org/10.5281/zenodo.10638996.

Submitted 30 May 2023
Accepted 16 February 2024
Published 13 March 2024
10.1126/scirobotics.adi7566