Science Robotics
AAAS

Supplementary Materials for

ANYmal parkour: Learning agile navigation for quadrupedal robots

David Hoeller *et al.*

Corresponding author: David Hoeller, dhoeller@nvidia.com; Nikita Rudin, rudinn@ethz.ch

Sci. Robot. **9**, eadi7566 (2024)
DOI: 10.1126/scirobotics.adi7566

The PDF file includes:

Other Supplementary Material for this manuscript includes the following:

Movies S1 to S5

Supplementary Methods

Observations, actions, and rewards definitions

Table S1: Symbols.

Symbol	Description
$\mathbf{r}$, $\mathbf{r}^*$	Current and local target base positions
ψ, ψ^*	Current and local target base headings
t^*	Remaining time to reach the local target
$\mathbf{r}_G^*$	Global target position
t_G^*	Remaining time to reach the global target
α	Angle between base z-axis and gravity
$\mathbf{v}_b$, $\boldsymbol{\omega}_b$	Base linear and angular velocities in base frame
$\mathbf{g}_b$	Gravity vector in base frame
$\mathbf{q}$, $\dot{\mathbf{q}}$, $\dot{\mathbf{q}}_{\text{lim}}$	Joint positions, velocities, and velocity limits
$\mathbf{q}^*$, $\mathbf{q}_d$	Desired and default joint positions
$\boldsymbol{\tau}$, $\boldsymbol{\tau}_{\text{lim}}$	Joint Torques and torque limits
$\mathbf{v}_f$, $\mathbf{F}_f$	Feet linear velocity and contact force
$\mathbf{h}$	$2\,\text{m} \times 1\,\text{m}$ grid of height measurements around the robot
$\mathbf{l}$	Scene belief state (perception module latent tensor)
s	Index of the selected locomotion skill
$\mathbb{S}_L$	Target reached (locomotion), $\mathbb{S}_L = \mathbb{1}_{\|\mathbf{r}_{xy} - \mathbf{r}_{xy}^*\| < 0.25} \mathbb{1}_{\|\psi - \psi^*\| < 0.5}$
$\mathbb{S}_N$	Target reached (navigation), $\mathbb{S}_N = \mathbb{1}_{\|\mathbf{r} - \mathbf{r}_G^*\| < 0.4}$

Table S2: Locomotion rewards.

Reward Term	Expression	Weight
Position tracking	$\mathbb{1}_{t^*<1}(1-0.5\|\mathbf{r}_{xy}-\mathbf{r}^*_{xy}\|)$	10
Heading tracking	$\mathbb{1}_{t^*<1}(1-0.5\|\psi-\psi^*\|)$	5
Joint velocity	$\|\dot{\mathbf{q}}\|^2$	-0.001
Torque	$\|\boldsymbol{\tau}\|^2$	-0.00001
Joint velocity limit	$\sum_{i=1}^{12}\max(\|\dot{\mathbf{q}}_i\|-\dot{q}_{\text{lim}},0)$	-1
Torque limit	$\sum_{i=1}^{12}\max(\|\boldsymbol{\tau}_i\|-\tau_{\text{lim}},0)$	-0.2
Base acc.	$\|\dot{\mathbf{v}}\|^2+0.02\|\dot{\boldsymbol{\omega}}\|^2$	-0.001
Feet acc.	$\sum_{f=1}^{4}\|\dot{\mathbf{v}}_f\|$	-0.002
Action rate	$\|\mathbf{q}^*_t-\mathbf{q}^*_{t-1}\|^2$	-0.01
Feet contact force	$\sum_{f=1}^{4}\max(\|F_f\|-700,0)^2$	-0.00001
Don't wait	$\mathbb{1}(\|\mathbf{v}_b\|<0.2)$	-1
Move in direction	$\cos\langle\mathbf{v}_b,\mathbf{r}^*-\mathbf{r}\rangle$	1
Stand at target	$\mathbb{S}_L\|\mathbf{q}-\mathbf{q}_d\|$	-0.5
Collision	$\mathbb{1}_{\textit{knee/shank collision}}$	-1
Stumble	$\mathbb{1}_{\|F_{f,xy}\|>2\|F_{f,z}\|}$	-1
Termination	$\mathbb{1}_{\textit{base collision}}+\mathbb{1}_{F_f>1500}$	-200

Table S3: Navigation rewards.

Reward Term	Expression	Weight
Position tracking	$\mathbb{1}_{t^*=0}(40\mathbb{S}_N-\|\mathbf{r}-\mathbf{r}^*_G\|)$	0.15
Termination	$\mathbb{1}_{\alpha<\pi/2}+\mathbb{1}_{F_f>2500}$	-0.5

Table S4: Locomotion and navigation observations.

Observation	Locomotion	Navigation
$\mathbf{v}_b$	$\times$	$\times$
$\boldsymbol{\omega}_b$	$\times$	
$\mathbf{g}_b$	$\times$	$\times$
$\mathbf{q},\dot{\mathbf{q}}$	$\times$	
$\mathbf{r}^*,t^*,\psi^*$	$\times$	
$\mathbf{r}^*_G,t^*_G$		$\times$
$\mathbf{h}$	$\times$	
$\mathbf{l}$		$\times$

Table S5: Navigation and locomotion actions

Module	Action
Locomotion	$\mathbf{q}^*$
Navigation	$s, \mathbf{r}^*, t^*, \psi^*$

Skill training details

Below, we describe the training set-ups of the various skills:

Walking The robot must traverse various irregular terrains consisting of stairs, slopes, and randomly placed small obstacles, similar to the ones commonly used in previous legged locomotion works (*4, 38*).

Jumping The robot starts on a box and must jump to a neighboring box separated by a gap of up to $1\,\mathrm{m}$. We use a curriculum on the size of the gap.

Climbing down The robot starts on a box with a height of up to $1\,\mathrm{m}$ and must climb down to reach a target on the ground. We use a curriculum on the height of the box. We add a termination condition on high impact forces on the feet. This termination is essential to get a transferable motion. Without it, the robot learns to jump down from the top, which is possible in simulation but leads to potential damage on the real robot.

Climbing up The robot starts on the ground and must climb on top of a box with a height of up to $1\,\mathrm{m}$. We use a curriculum on the height of the box. We allow the robot to make contact with the base and knees by reducing the weights of the corresponding penalties. This leads to the natural progression where the policy learns to climb using its knees and then uses its feet instead when possible.

Crouching The crouching policy has the specificity of dealing with overhanging obstacles. The robot must reach a target located on the other side of a narrow horizontal passage with a minimum height of $0.4\,\mathrm{m}$. We use a curriculum on the height of the passage. We provide the same 2.5D map as the other policies. As such, it sees the obstacle from the top and cannot differentiate a table from a box. This does not pose a problem since it is only trained in such scenarios and will always try to go under the obstacle.

The walking policy is trained on a mix of terrains (60% stairs, 20% slopes, and 20% randomized obstacles). The other specialized skills are all trained with 80% of their corresponding obstacle and 20% of random rough terrain. This leads to more natural gaits and better performance upon deployment.

Implementation details

On the real robot, the pipeline is implemented using the ROS ecosystem. The locomotion and navigation modules are implemented in a single C++ node on the robot's main onboard PC. The perception module is implemented in a Python node and leverages the Jetson Orin's GPU to handle the exteroceptive measurements. It is worth mentioning that the point cloud messages of the six Realsense cameras reach the perception node with a delay of up to $250\,\mathrm{ms}$, which is prohibitively long for fast maneuvers such as climbing. Therefore, we disable point cloud publishing for these cameras and directly subscribe to the depth image messages instead, reducing the delay to $25\,\mathrm{ms}$. The perception node waits for new data from all the sensors, projects the depth images to point clouds, transforms the point clouds (with the LiDAR as well) into a common frame, and converts to data to the voxel grid representation. These operations are parallelized using custom CUDA kernels using Warp (*41*).

We train all the policies and collect the data for the perception module using the Isaac Gym simulator (*42*), where we deploy 4096 agents in parallel. To generate and prepare the perceptive

data for the perception network, we develop custom CUDA kernels with Warp. At every time step, these kernels perform raycasting for the six depth cameras and the LiDAR for each robot ($\approx$ 140 million rays total) and directly convert them to the voxel grid inputs without copying memory. To speed up training, the resolution of the depth cameras is reduced to 64x64. The computation times of the components in the perceptive pipeline are described in Tab. S6. It can be seen that the preparation of the input for the perception module (ray-casting and voxelization) is two orders of magnitude more expensive than the height-scanning for the skills. This is the main reason the low-level skills do not use the depth cameras or the perception module's latent as input during training (as a rough estimate, the training time for the individual skills would increase from 1.5 to 7 hours, and for the navigation policy from 4.5 to 38 hours).

Table S6: Perception-related simulation performance during training with 4096 robots.

	Ray-casting & voxelization	Perception module inference	Skill height sampling
Time (s)	0.32	0.017	0.003

Symmetric data augmentation for locomotion training

The position-tracking formulation of the locomotion training does not constrain the robot's trajectory between the initial and target positions. This allows the robot to learn complex behaviors but also leads to asymmetric motions. For example, the climbing policy only learns to climb forwards and prefers to turn the robot around if facing an obstacle backward, leading to unfavorable situations when the robot must cross multiple obstacles with different skills. We solve these issues by exploiting the symmetric nature of the robot. Based on the duplication method of (*43*), we augment each environment transition with all symmetric variants by transforming the observations and actions accordingly. Specifically, we use front-back and left-right symmetries of the ANYmal D robot. The authors in (*43*), however, mention that their duplication method suffers from poor convergence due to the off-policy nature of the mirrored states. In-

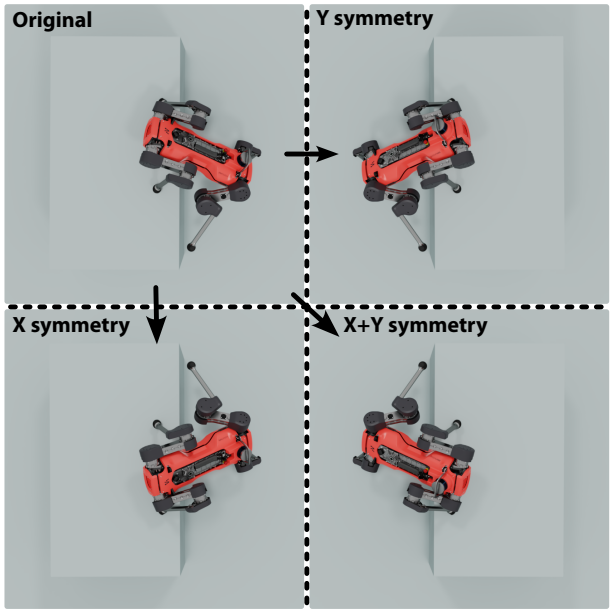

Figure S1: Representation of the symmetric state augmentation. The original state (top-left) is augmented into four symmetric states using the X and Y symmetries of the robot.

deed, these augmentations result in low probabilities for the transformed actions for not fully trained policies. We resolve this issue by setting the probability of the original actions to all symmetric variants. Intuitively, we bootstrap the learning process of a randomly initialized policy since we know that at convergence symmetric states will lead to symmetric actions with equal probability.

Measurement blind spots

Measurement blind spots during a box climbing maneuver can be seen in Fig. S2. Due to the height of the box, the robot cannot perceive the top surface at the beginning. During the climb, large occluded regions occur because of limb obstructions. It can also be seen that the camera arrangement is not particularly favorable for locomotion since there are blind spots immediately below the robot.

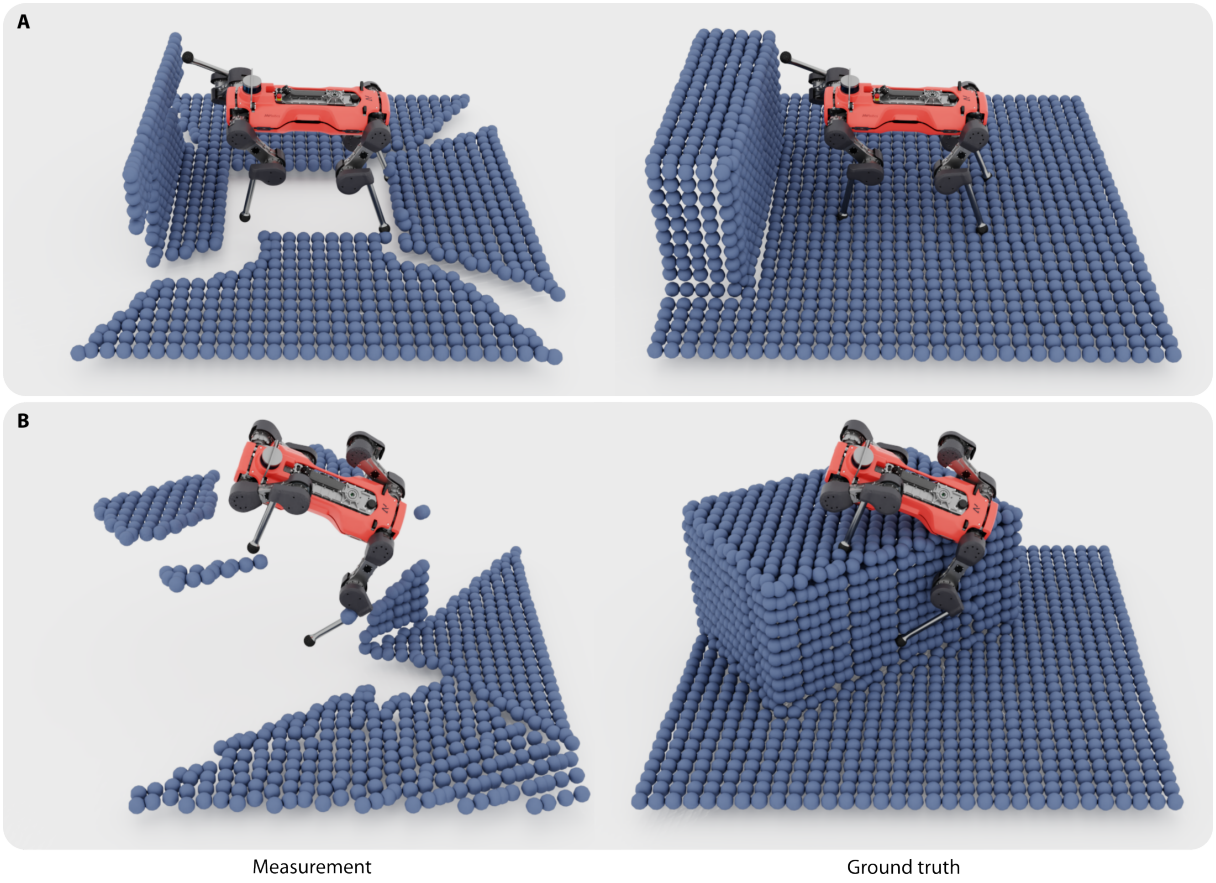

Figure S2: Measurement blind spots occurring during a box climbing maneuver. (**A**) The sensors do not perceive the top surface at the beginning. The perception module has to use the points on the edge of the front surface to estimate the height of the box and correctly reconstruct the top. (**B**) During the climb, the limbs obstruct the cameras resulting in large occluded areas.

Supplementary Results

Navigation across long ranges

Fig. S3 depicts a scenario with a distant goal in a U-shaped terrain. The planner understands that it cannot cross the wide gap by jumping, and it must first distance itself from the target to solve the task.

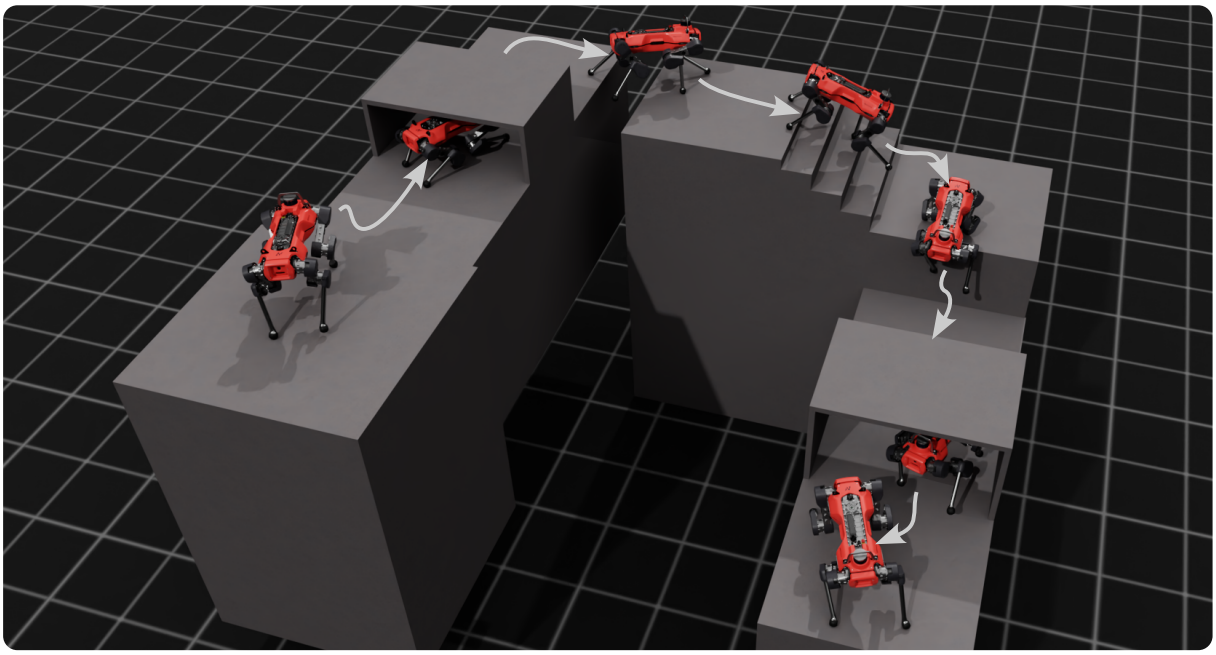

Figure S3: The planner can reach remote targets, even if it has to distance itself from the goal first.

Ablation of the navigation module's action and input spaces

In Table S7, we analyze the importance of the different components of the navigation policy's action space. We remove the timer output and set a fixed time for the low-level policies (No T); we remove the heading output and set it to be in the direction of the next target (No H); we remove both (No H, No T). The study is performed under the same conditions as the comparison with the manually coded trajectory (Table 1). The results show that using the heading and time commands for the low-level skills increases the performance of the system. Specifically, we can see that adding the heading command increases the success rate on terrains where the robot must quickly turn multiple times on the spot (Terrain Fig. 5 (A)), whereas the time command leads to better performance in longer terrains (Terrain Fig. 5 (B) & (C)), where the high-level policy must use fast motions on simple obstacles, but slow down in risky parts.

To motivate the choice of inputs for the navigation policy, we perform an ablation study.

Table S7: Comparison of the navigation policy's performance against different formulations. (**No T**): no time output. (**No H**): no heading output. (**No H, No T**): no heading or time output.

	Ours	No T	No H	No H, No T
Terrain: Fig. 5 - A	98.2%	94.7%	89.5%	81.1%
Terrain: Fig. 5 - B	96.3%	89.4%	88.1%	71.9%
Terrain: Fig. 5 - C	97.6%	91.6%	94.6%	94.0%

We compare the convergence of our navigation policy against one without velocity input, one without orientation, and one without both in Fig. S4. Using the robot's velocity might seem of limited use for a policy that runs at 5 Hz, but the results show that it speeds up convergence and improves final performance. A possible explanation is that the velocity allows the policy to get a better overview of what the robot is currently doing. For example, it would allow the navigation module to distinguish between cases where the robot is in the middle of a climb or falling back down from a failed climb.

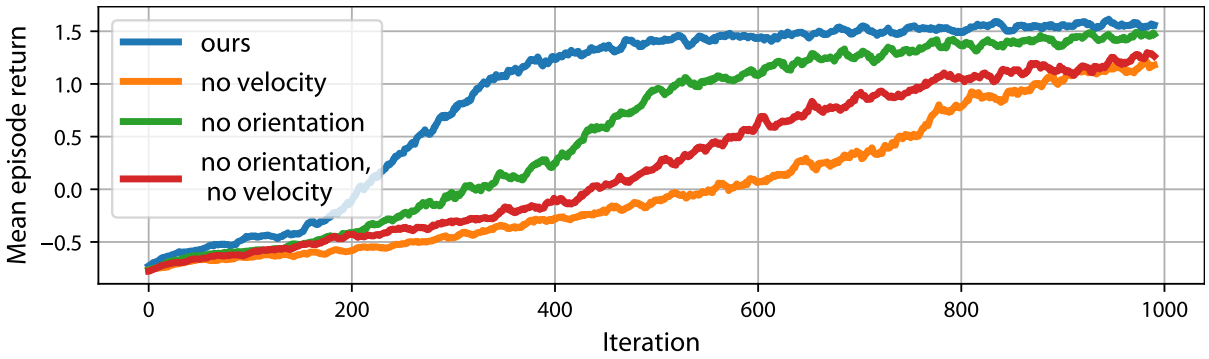

Figure S4: Influence of the navigation policy's input space on convergence.

Ablation of the measurement delay compensation

In this section, we highlight the importance of delay compensation for skill performance. The skill's perceptive inputs are a set of height scans that are queried from the map output of the perception module. This mechanism is convenient since it allows to compensate for measurement and processing delays. Indeed, when the locomotion module receives a map, it is registered in

the world frame at the time of the measurements. In between map updates, the set of height measurements for the skill is computed at the correct location of the robot. Fig. S5 depicts the performance of the skills in simulation if we could not compensate for delays for the most difficult terrain setting. This can be emulated by offsetting the map by the specific delay. The climb and climb down policies can cope with the delays since they use the shanks of the robot to guide the trajectory. Such robustness is achieved due to map shift randomization during training. On the other hand, delays have a strong impact on the gap policy's performance, since the robot does not use the shanks for that skill and a shifted map cannot be compensated for.

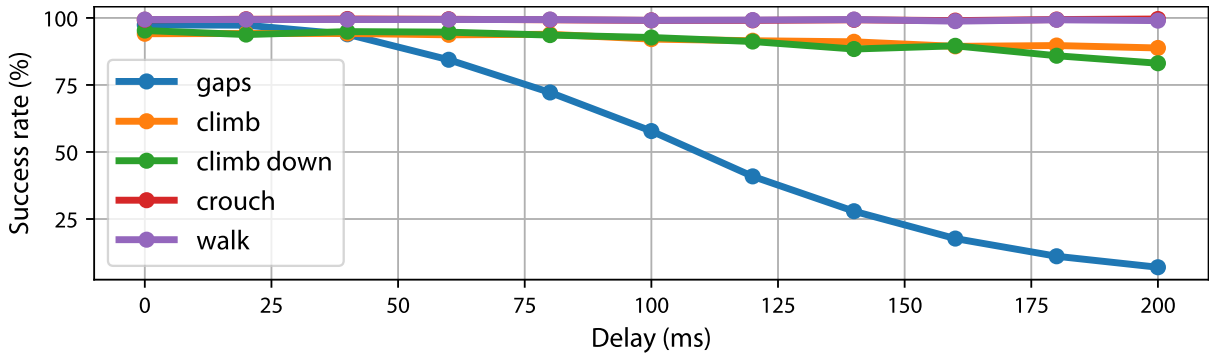

Figure S5: Influence of measurement delays on skill performance if they could not be compensated for.

Ablation of the perception refinement

To highlight the importance of the perception module's refinement strategy, we deploy the jumping skill on top of a narrow box and only give it access to the coarse map (pre-refinement map with $12.5\,\mathrm{cm}$ voxel size). In Fig. S6 (A), the robot is turning on the spot, and in Fig. S6 (B), the robot is commanded to cross the gap. In both cases, one of the robot's feet misses the box's top surface, resulting in dangerous maneuvers. This is due to the slightly inflated coarse map (middle column), causing the skill to misinterpret the region as being suitable to step on. On the other hand, the refined map (right column) matches the scene more accurately, and using

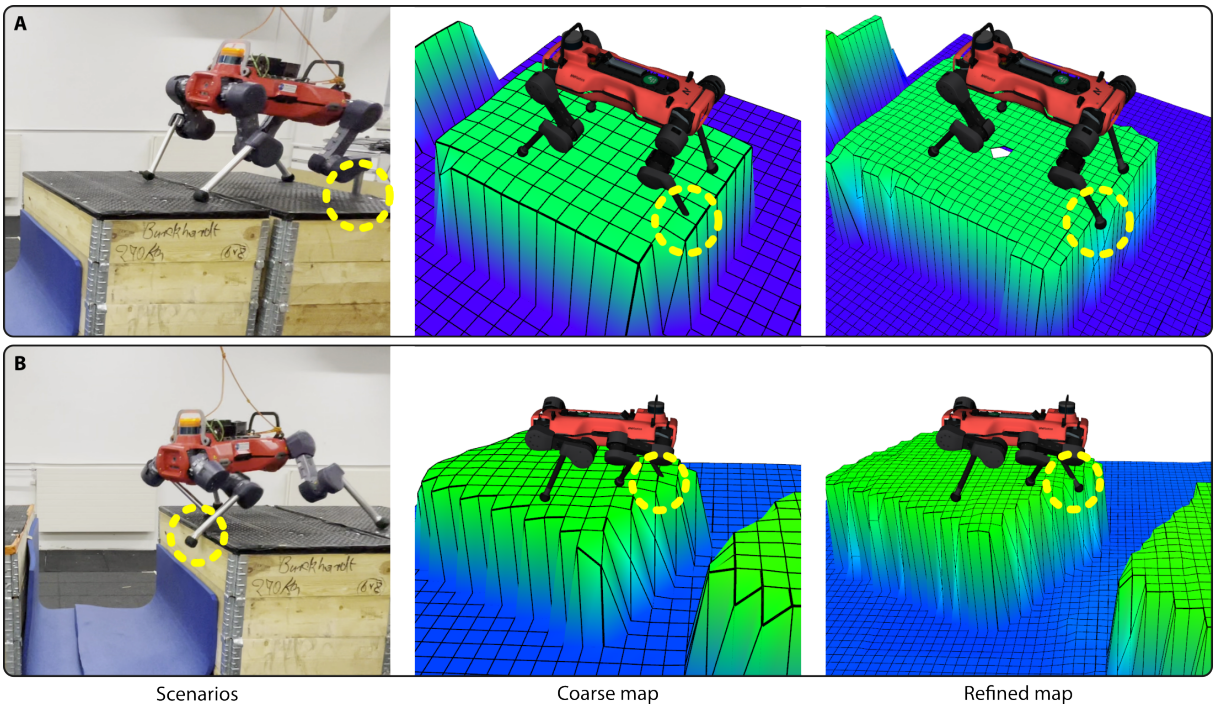

Figure S6: Deployment of a skill on a set of boxes without the refined map. The robot is more prone to misplace its feet due to the coarser map, resulting in a severe crash in (B). The refinement step reconstructs the scene more accurately.

it would probably have prevented the missteps. This difference is mostly due to the different resolutions these networks operate at. Indeed, on the border objects, misclassifying the edge by one cell in the voxel grid has a more severe impact at coarser scales.

Incorrect terrain reconstructions

There are situations where the network produces wrong outputs (Fig. S7). When climbing on the first box (left), the network tends to hallucinate a stair behind it in the measurement blind spot, probably due to a data-set imbalance. This does not impede the performance of the navigation and locomotion modules since the network quickly corrects this erroneous output once it has a better view of the situation. Also, the table is sometimes inflated when the robot crawls underneath (right). This comes from a combination of measurement sparsity on the top

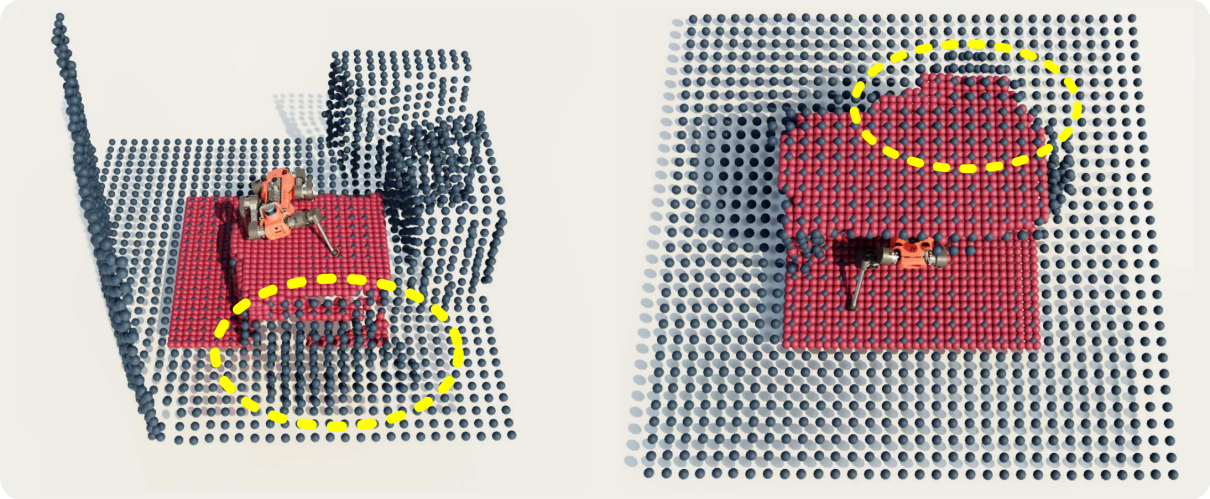

Figure S7: Incorrect reconstructions with our method, highlighted in yellow. On the left, the network hallucinates a stair. On the right, it inflates the shape of the table.

surface and state estimation drift, which is more pronounced for crouching maneuvers. Again, this does not pose a problem to complete the task, since the robot would stay crouched for longer in the worst case.