



上海大学未来技术学院 | 上海大学人工智能研究院
SCHOOL OF FUTURE TECHNOLOGY, SHANGHAI UNIVERSITY INSTITUTE OF ARTIFICIAL INTELLIGENCE, SHANGHAI UNIVERSITY

人工智能导论

—第3课：机器人智能控制

叶林奇

未来技术学院（人工智能研究院）

2024秋季学期





提纲

一、ETH三部曲

二、AMP三部曲

三、Pakour三部曲

四、无人机三部曲

2019

**强化学习首次超越
平地行走、跌倒恢复**

ARTIFICIAL INTELLIGENCE

Learning agile and dynamic motor skills for legged robots

Jemin Hwangbo^{1*}, Joonho Lee¹, Alexey Dosovitskiy², Dario Bellicoso¹, Vassilios Tsounis¹, Vladlen Koltun³, Marco Hutter¹

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

2020

**本体感知盲走
征服野外复杂地形**

ANIMAL ROBOTS

Learning quadrupedal locomotion over challenging terrain

Joonho Lee^{1*}, Jemin Hwangbo^{1,2}, Lorenz Wellhausen¹, Vladlen Koltun³, Marco Hutter¹

Copyright © 2020
The Authors, some
rights reserved;
exclusive licensee
American Association
for the Advancement
of Science. No claim
to original U.S.
Government Works

ANIMAL ROBOTS

Learning robust perceptive locomotion for quadrupedal robots in the wild

Takahiro Miki^{1*}, Joonho Lee¹, Jemin Hwangbo², Lorenz Wellhausen¹, Vladlen Koltun³, Marco Hutter¹

Copyright © 2022
The Authors, some
rights reserved;
exclusive licensee
American Association
for the Advancement
of Science. No claim
to original U.S.
Government Works

2022

**本体感知+视觉
爬山比人快**

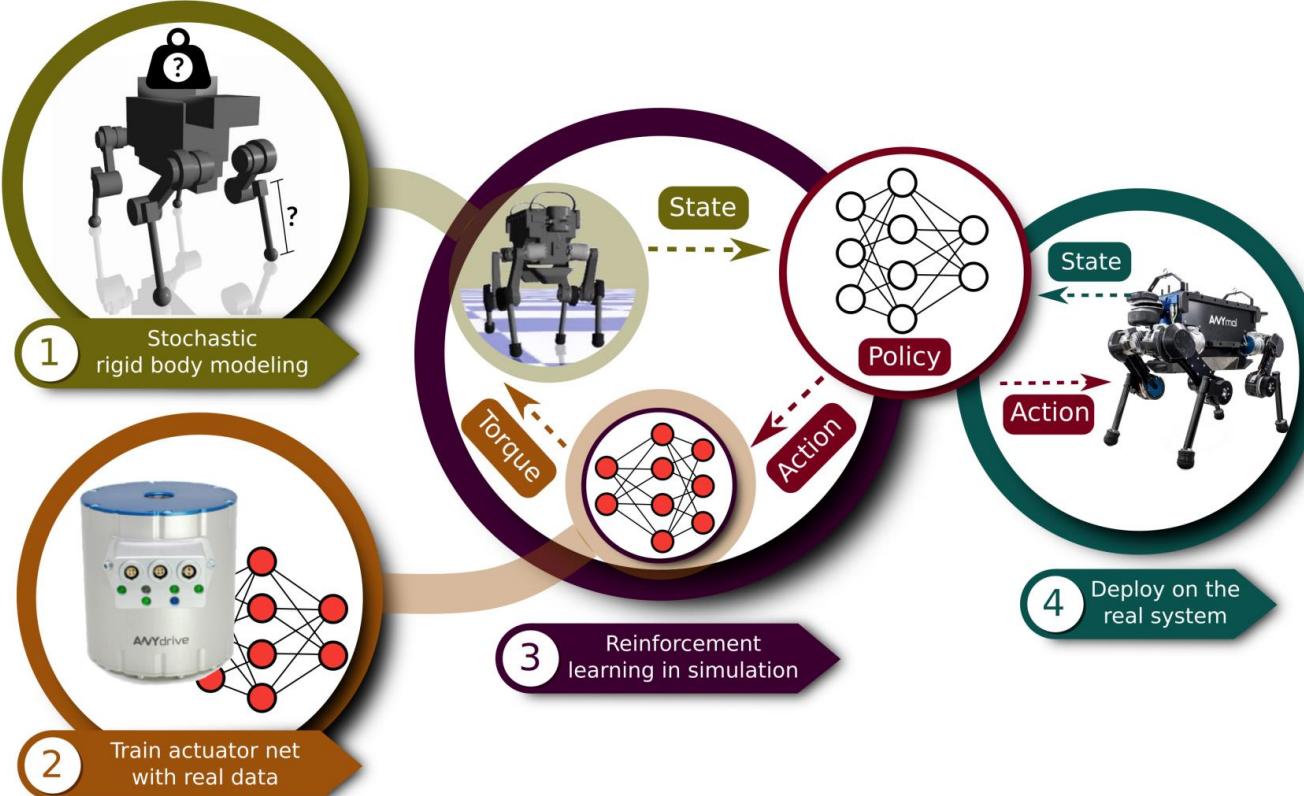


Fig. 1. Creating a control policy. In the first step, we identify the physical parameters of the robot and estimate uncertainties in the identification. In the second step, we train an actuator net that models complex actuator/software dynamics. In the third step, we train a control policy using the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

第一步是确定机器人的物理参数并估计其中的不确定性。

第二步是训练一个致动器网络，建模复杂的致动器/软件动力机制。
(监督学习)

第三步是利用前两步中得到的模型训练一个控制策略。
(强化学习)

第四步是直接在物理系统中部署训练好的策略。

速度跟踪控制，跟得更准，能耗更低，力矩更小

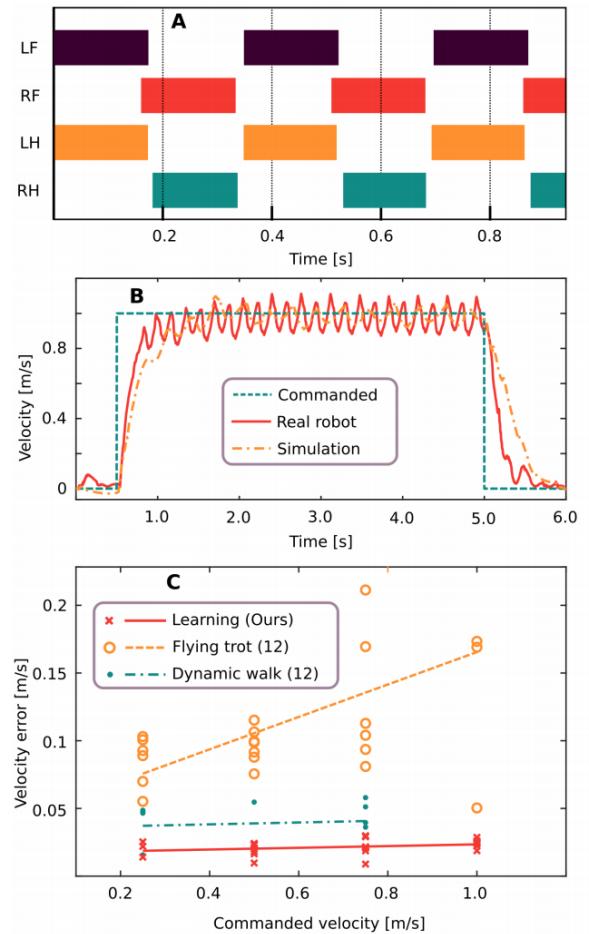


Fig. 2. Quantitative evaluation of the learned locomotion controller. (A) The discovered gait pattern for 1.0 m/s forward velocity command. LF, left front leg; RF, right front leg; LH, left hind leg; RH, right hind leg. (B) The accuracy of the base velocity tracking with our approach. (C-E) Comparison 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.

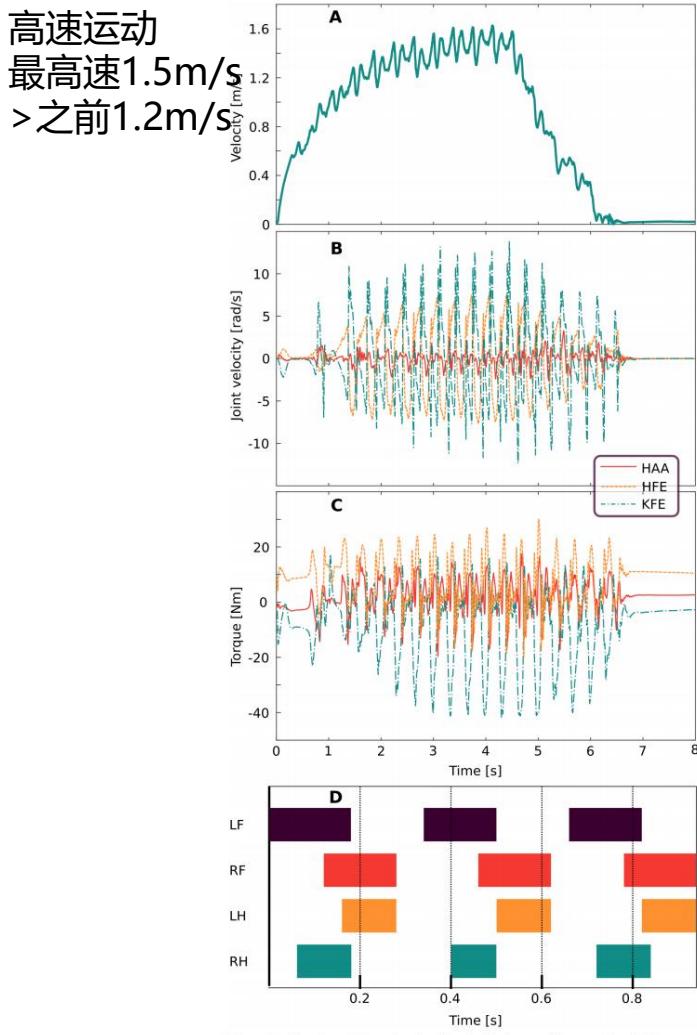


Fig. 3. Evaluation of the trained policy for high-speed locomotion. (A) Forward velocity of ANYmal. (B) Joint velocities. (C) Joint torques. (D) Gait pattern.

跌倒恢复，可以从任意摔倒状态中爬起

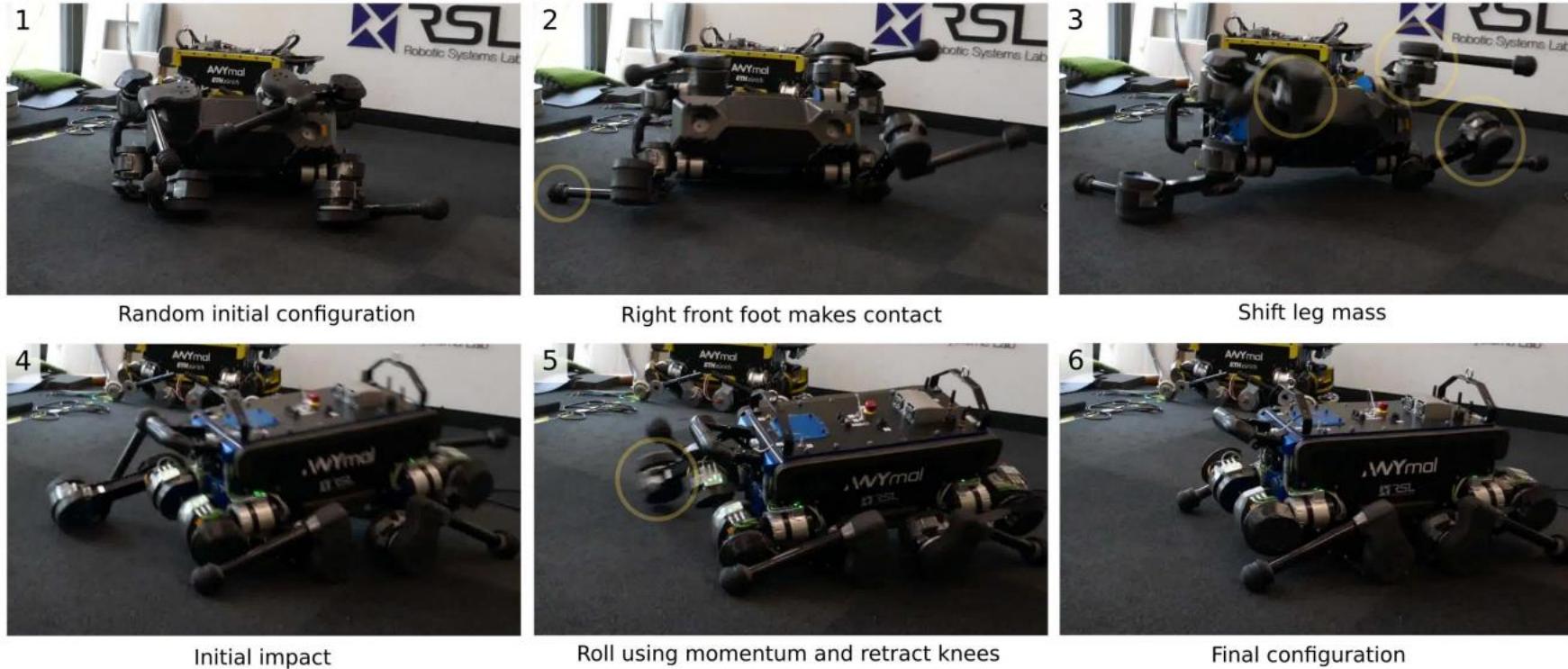


Fig. 4. A learned recovery controller deployed on the real robot. The learned policy successfully recovers from a random initial configuration in less than 3 s.

网络架构

控制器：2隐层MLP
输入状态、指令、历史动作
输出关节角指令
(强化学习)

执行器：3隐层MLP
输入历史角度、速度
输出关节力矩
(监督学习、正弦激励)

历史信息：
 t
 $t-0.01s$
 $t-0.02s$

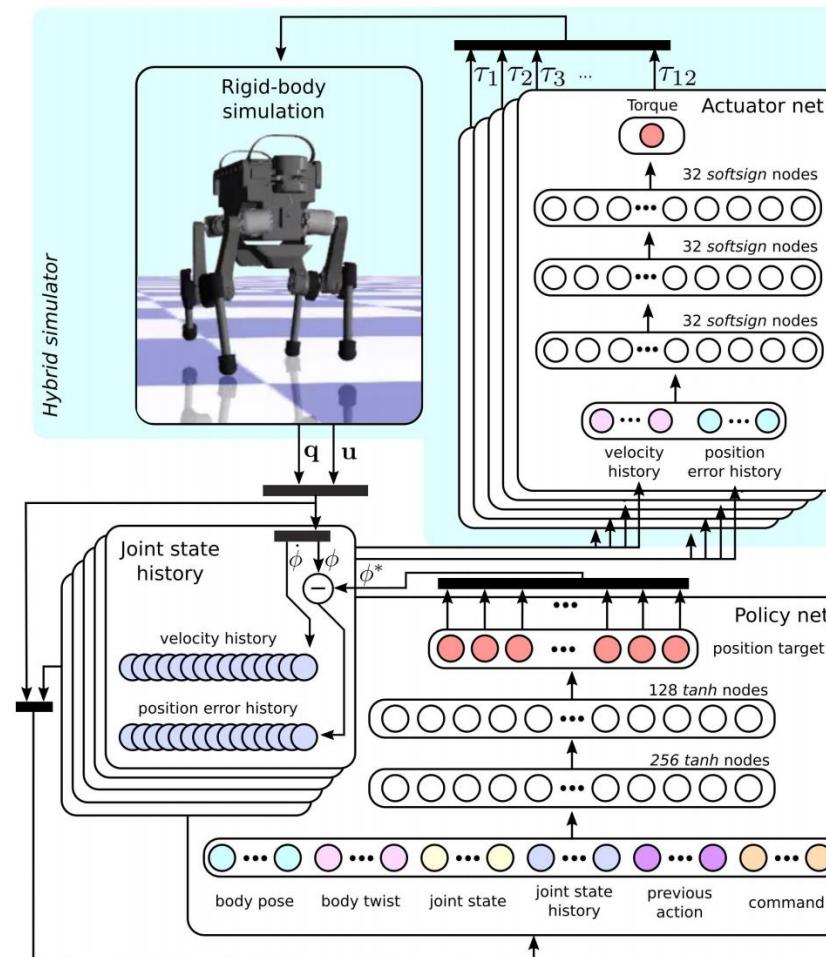


Fig. 5. Training control policies in simulation. The policy network maps the current observation and the joint state history to the joint position targets. The actuator network maps the joint state history to the joint torque, which is used in rigid-body simulation. The state of the robot consists of 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 ϕ subtracted from the joint position target ϕ^* .

奖励设计

课程训练 (kc从0.3 增大至1)

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

We used a logistic kernel to define a bounded cost function $K : \mathbb{R} \rightarrow [-0.25, 0]$ as

$$K(x) = -\frac{1}{e^x + 2 + e^{-x}}. \quad (1)$$

angular velocity of the base cost ($c_w = -6\Delta t$)

$$c_w K(|\omega_{IB}^I - \hat{\omega}_{IB}^I|) \quad (2)$$

linear velocity of the base cost ($c_{v1} = -10\Delta t, c_{v2} = -4\Delta t$)

$$c_{v1} K(|c_{v2} \cdot (v_{IB}^I - \hat{v}_{IB}^I)|) \quad (3)$$

torque cost ($c_\tau = 0.005\Delta t$)

$$k_c c_\tau \|\tau\|^2 \quad (4)$$

joint speed cost ($c_{js} = 0.03\Delta t$)

$$k_c c_{js} \|\dot{\phi}^i\|^2 \quad \forall i \in \{1, 2, \dots, 12\} \quad (5)$$

foot clearance cost ($c_f = 0.1\Delta t, \hat{p}_{f,i,z} = 0.07$ m)

$$k_c c_f (\hat{p}_{f,i,z} - p_{f,i,z})^2 \|v_{ft,i}\|, \quad \forall i, g_i > 0, i \in \{0, 1, 2, 3\}, \quad (6)$$

foot slip cost ($c_{fv} = 2.0\Delta t$)

$$k_c c_{fv} \|v_{ft,i}\|, \quad \forall i, g_i = 0, i \in \{0, 1, 2, 3\} \quad (7)$$

orientation cost ($c_o = 0.4\Delta t$)

$$k_c c_o \| [0, 0, -1]^T - \phi_g \| \quad (8)$$

smoothness cost ($c_s = 0.5\Delta t$)

$$k_c c_s \|\tau_{t-1} - \tau_t\|^2 \quad (9)$$

Section S4. Cost terms for training recovery from a fall

We use $\text{angleDiff} : \mathbb{R} \times \mathbb{R} \rightarrow [0, \pi]$ that computes the minimum angle difference between two angular positions to define a cost function on the joint positions. The symbols used in this section are defined in section S1.

torque cost ($c_\tau = 0.0005\Delta t$)

$$k_c c_\tau \|\tau\|^2 \quad (10)$$

joint speed cost ($c_{js} = 0.2\Delta t, c_{jsmax} = 8$ rad/s)

$$\text{If } |\dot{\phi}^i| > |c_{jsmax}|, \quad k_c c_{js} \|\dot{\phi}^i\|^2 \quad \forall i \in \{1, 2, \dots, 12\} \quad (11)$$

joint acceleration cost ($c_{ja} = 0.0000005\Delta t$)

$$k_c c_{ja} \|\ddot{\phi}^i\|^2 \quad \forall i \in \{1, 2, \dots, 12\} \quad (12)$$

HAA cost ($c_{HAA} = 6.0\Delta t$)

$$\text{If } |\phi_{roll}| < 0.25\pi, \quad k_c c_{HAA} K(\text{angleDiff}(\phi^{HAA}, 0)) \quad (13)$$

HFE cost ($c_{HFE} = 7.0\Delta t, \hat{\phi}^{HFE} = \pm 0.5\pi$ rad (+ for right legs))

$$\text{If } |\phi_{roll}| < 0.25\pi, \quad k_c c_{HFE} K(\text{angleDiff}(\phi^{HFE}, \hat{\phi}^{HFE})) \quad (14)$$

KFE cost ($c_{KFE} = 7.0\Delta t, \hat{\phi}^{KFE} = \mp 2.45$ rad)

$$\text{If } |\phi_{roll}| < 0.25\pi, \quad k_c c_{KFE} K(\text{angleDiff}(\phi^{KFE}, \hat{\phi}^{KFE})) \quad (15)$$

contact slip cost ($c_{cv} = 6.0\Delta t$)

$$k_c c_{cv} \frac{\sum_{n \in I_c} \|v_{c,n}^I\|^2}{|I_c|} \quad (16)$$

body contact impulse cost ($c_{cimp} = 6.0\Delta t$)

$$k_c c_{cimp} \frac{\sum_{n \in I_c \setminus I_{c,f}} \|i_{c,n}^I\|}{|I_c| - |I_{c,f}|} \quad (17)$$

internal contact cost ($c_{cint} = 6.0\Delta t$)

$$k_c c_{cint} |I_{c,i}| \quad (18)$$

orientation cost ($c_o = 6.0\Delta t$)

$$c_o \| [0, 0, -1]^T - \phi_g \| \quad (19)$$

smoothness cost ($c_s = 0.0025\Delta t$)

$$k_c c_s \|\tau_{t-1} - \tau_t\|^2 \quad (20)$$

训练和部署

训练方法：TRPO
2000个并行训练

速度跟踪训练时间4小时（相当于真实时间9天）
摔倒跟踪训练时间11小时（相当于真实时间79天）

关节速度测量值有很大噪声，在仿真中加入随机噪声模拟
身体速度、角速度也加噪声

初始状态随机化

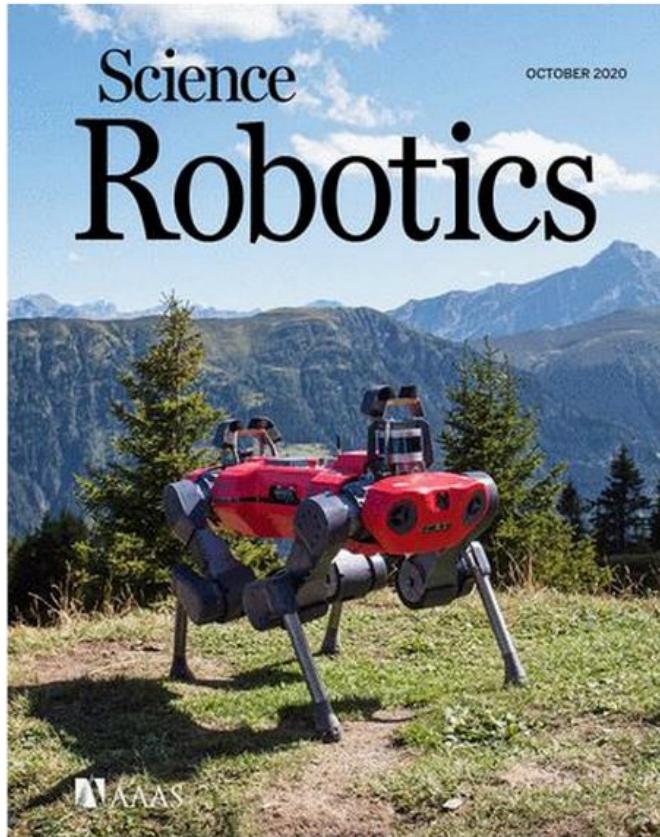
直接部署
只占用0.1%机载CPU算力，远低于MPC

速度跟踪200Hz
跌倒爬起100Hz

Table S3. Initial state distribution for training both the command-conditioned and high-speed locomotion. The initial state is randomized to make the trained policy more robust.

	mean	standard deviation
base position	$[0, 0, 0.55]^T$	1.5 cm
base orientation	$[1, 0, 0, 0]^T$	0.06 rad (about a random axis)
joint position	$[0, 0.4, -0.8, 0, 0.4, -0.8, 0, -0.4, 0.8, 0, -0.4, 0.8]^T$	0.25 rad
base linear velocity	$\mathbf{0}^3$	0.012 m/s
base angular velocity	$\mathbf{0}^3$	0.4 rad/s
joint velocity	$\mathbf{0}^{12}$	2 rad/s

2020 Learning quadrupedal locomotion over challenging terrain 闭眼盲走，轻松越野



瑞士ANYmal机器人登上了2020年《Science Robotics》封面，该机器人通过自学习控制器可以在各种复杂地形保持平衡，并且只依赖本体感受信号（编码器和IMU），不需要摄像头、激光雷达或接触式传感器信息。从其所展示出的高灵活性和强适应性来看，该机器人犹如真正的动物一般自如穿梭于各种环境。



2020 Learning quadrupedal locomotion over challenging terrain

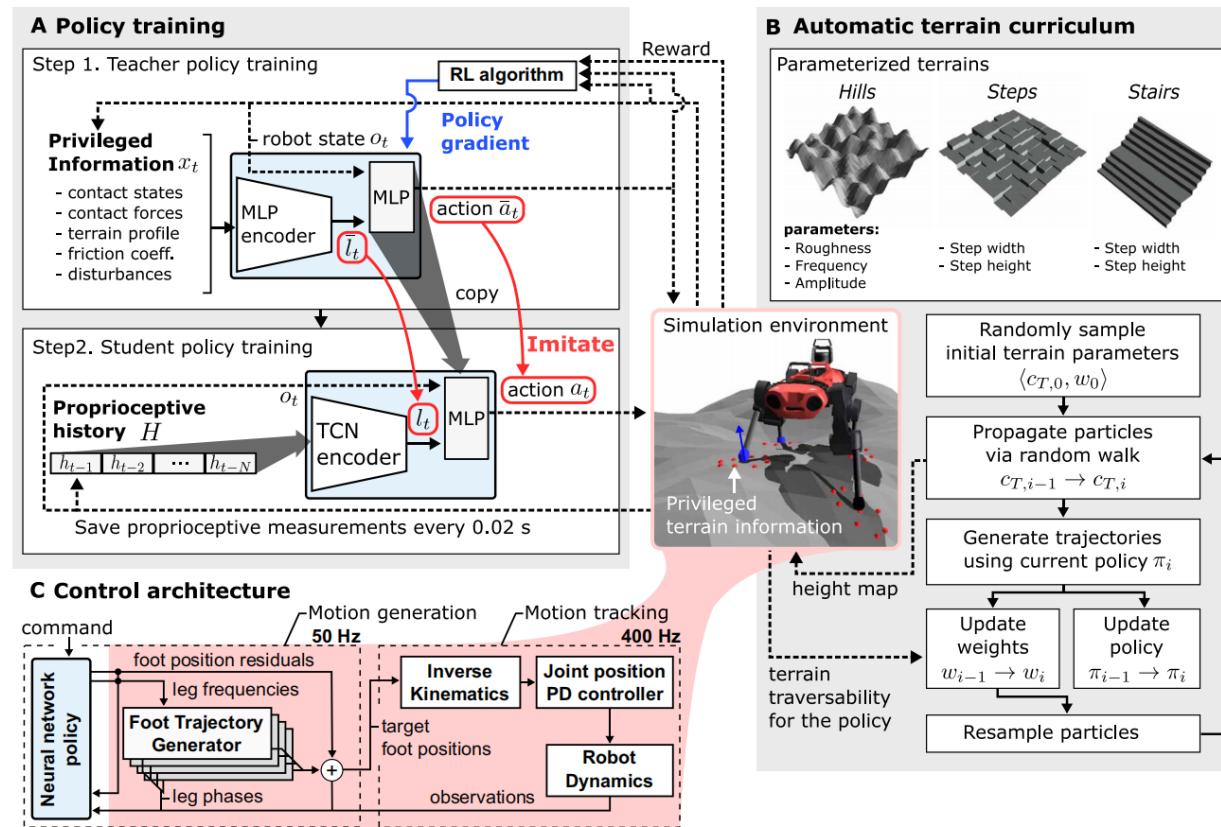
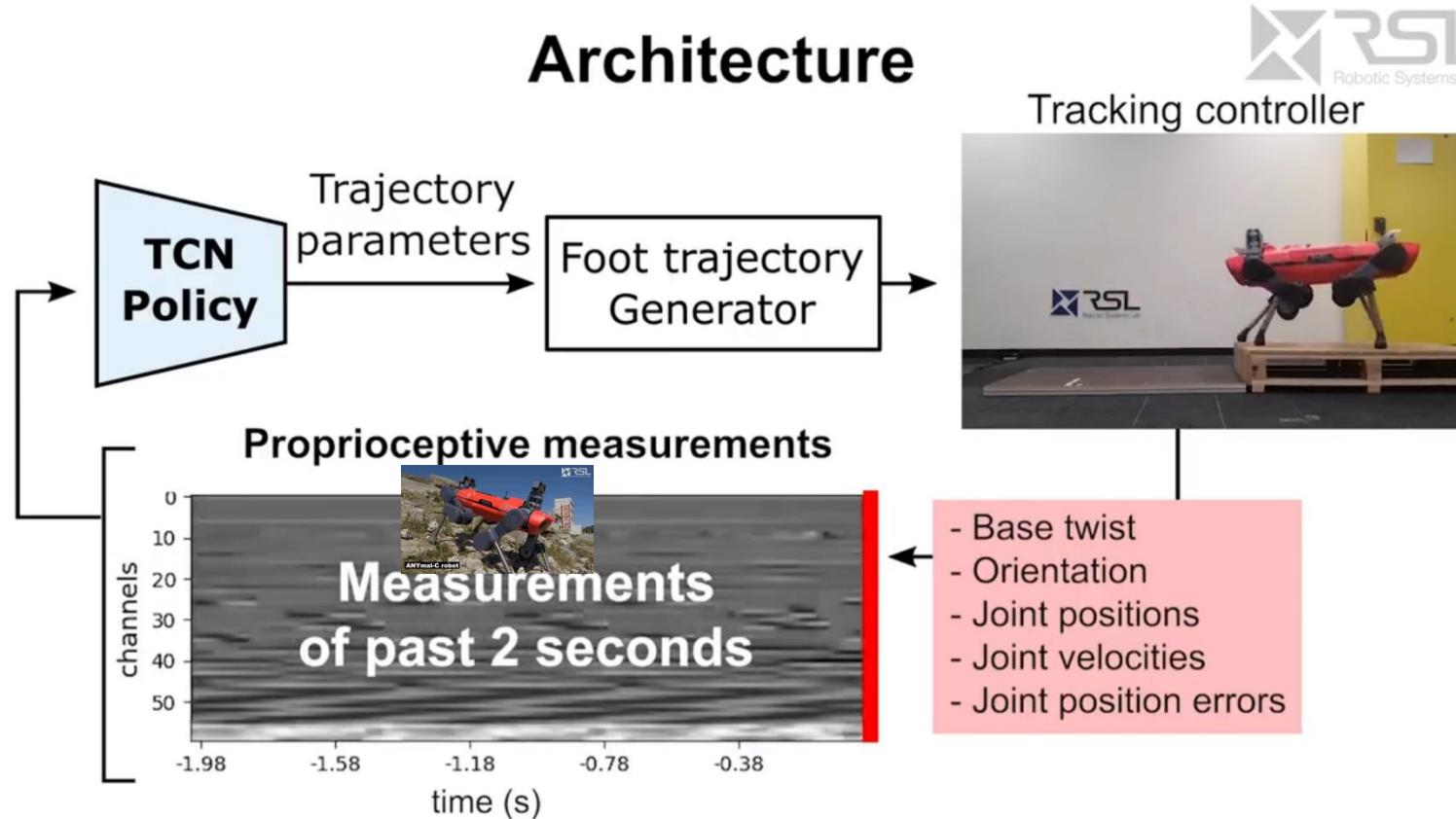


Fig. 4. Creating a proprioceptive locomotion controller. (A) Two-stage training process. First, a teacher policy is trained using RL in simulation. It has access to privileged information that is not available in the real world. Next, a proprioceptive student policy learns by imitating the teacher. The student policy acts on a stream of proprioceptive sensory input and does not use privileged information. (B) An adaptive terrain curriculum synthesizes terrains at an appropriate level of difficulty during the course of training. Particle filtering was used to maintain a distribution of terrain parameters that are challenging but traversable by the policy. (C) Architecture of the locomotion controller. The learned proprioceptive policy modulates motion primitives via kinematic residuals. An empirical model of the joint position PD controller facilitates deployment on physical machines.

首先在模型上，新方法没有使用在机器人当前状态上运行的多层感知器（MLP），而是使用了序列模型，特别是感受状态的时间卷积网络（TCN）。新方法没有使用显式的接触和滑动预估模块，相反的 TCN 会根据需求从本体感受历史中隐式地推理出接触和滑动事件。



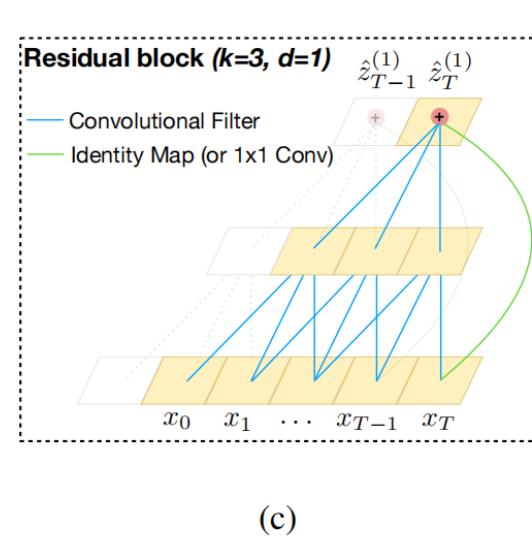
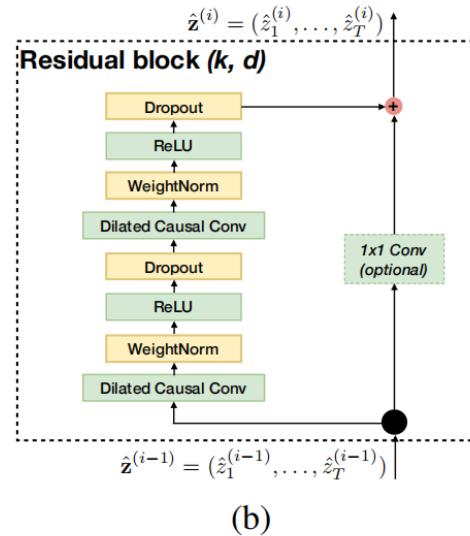
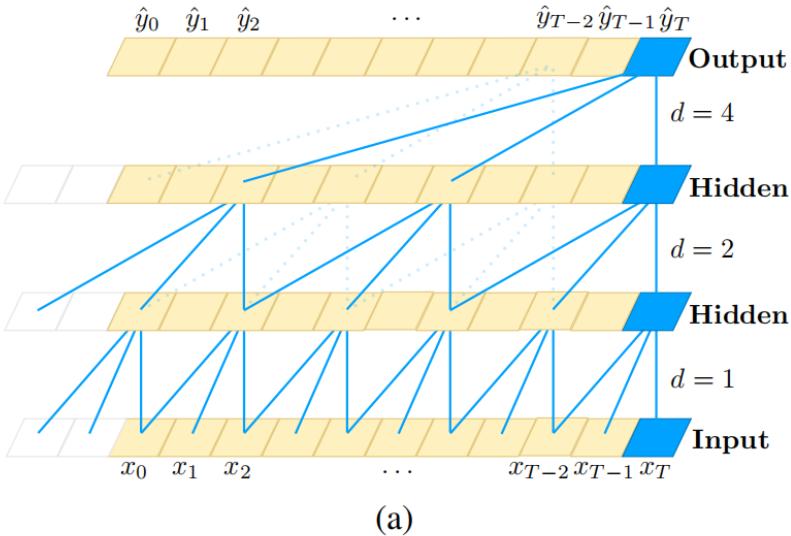


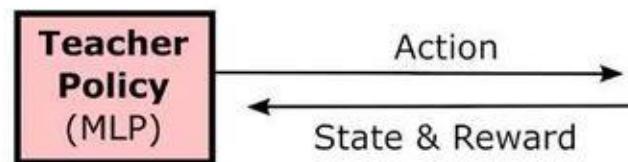
Figure 1. Architectural elements in a TCN. (a) A dilated causal convolution with dilation factors $d = 1, 2, 4$ and filter size $k = 3$. The receptive field is able to cover all values from the input sequence. (b) TCN residual block. An 1×1 convolution is added when residual input and output have different dimensions. (c) An example of residual connection in a TCN. The blue lines are filters in the residual function, and the green lines are identity mappings.

实现优化结果的第二个关键在于特权学习（privileged learning），研究人员发现直接通过强化学习训练出的越野运动策略并不成功：控制信号稀疏，并且所输出的网络无法在合理的时间内学习出正确的运动。新的模型在训练中分为两个阶段，首先训练教师策略，该策略可访问特权信息——真实情况（ground-truth）及机器人接触的情况，随后教师指导纯本体感受的学生控制器学习，后者仅使用机器人本身可用的传感器信息。这种特权学习会在模拟环境中启用，但最终学习到的策略可以在模拟环境，以及真实的物理环境中部署。

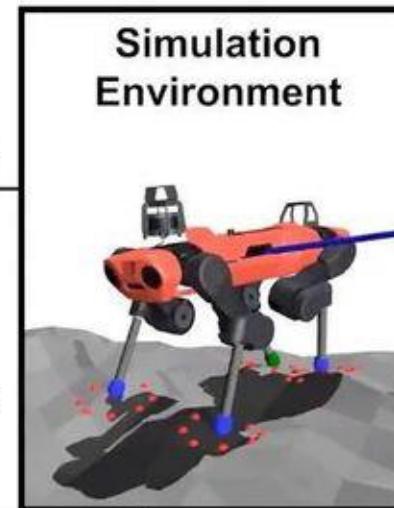
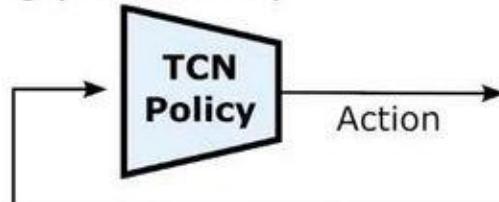
Privileged Training



1. Teacher Training (RL)



2. Student Training (Distillation)

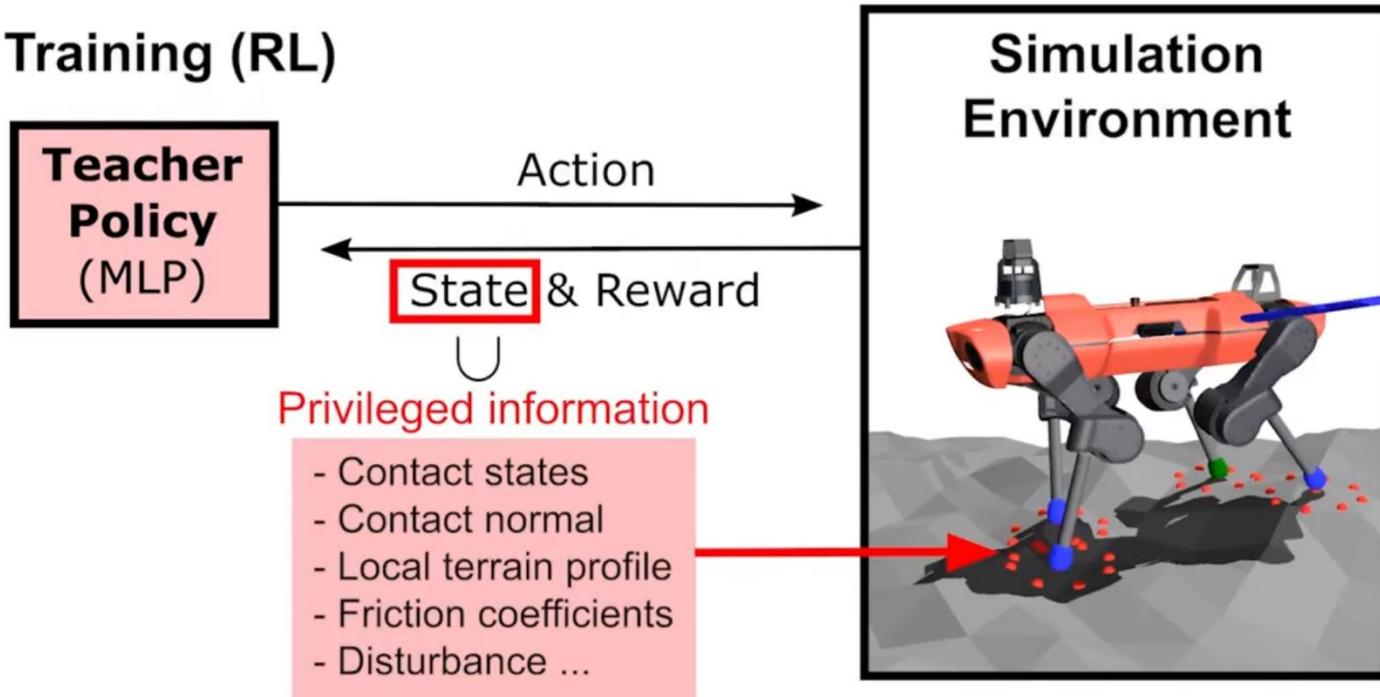


Sequence of Proprioceptive measurements

Training is done in two steps.

Privileged Training

1. Teacher Training (RL)



Privileged Training

2. Student Training (Distillation)

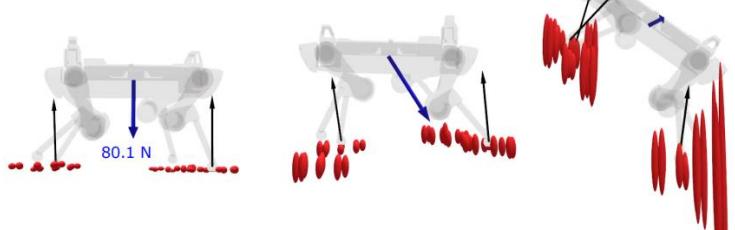
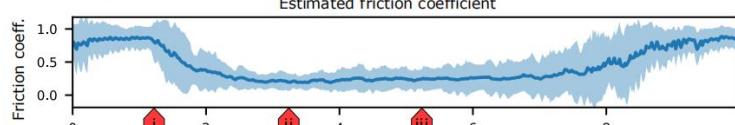
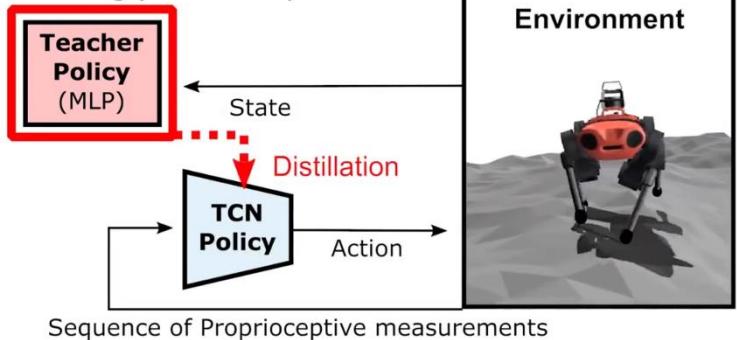
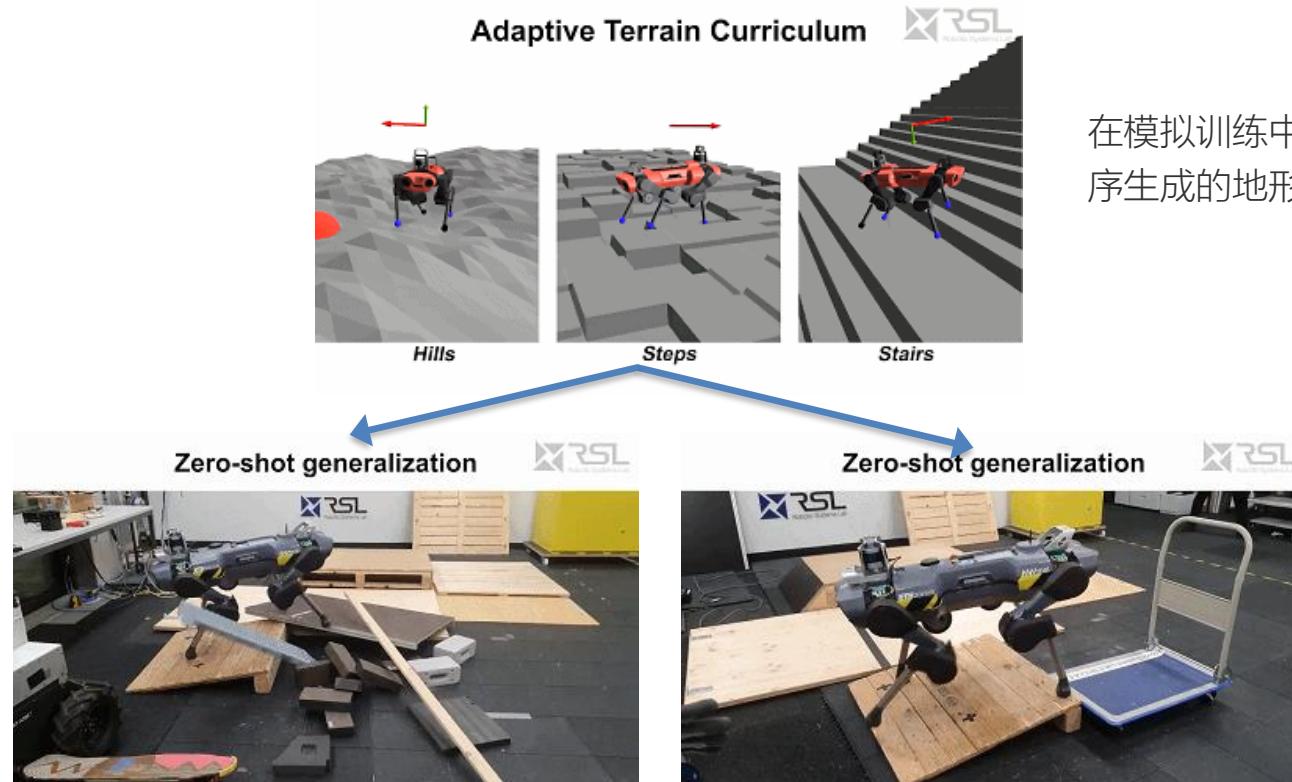


Figure S2: Reconstructed privileged information in different situations. (A) Estimated friction coefficient between the feet and the terrain while traversing a wet whiteboard. The shaded area denotes 95 % confidence interval. (B-D) Reconstruction of the external disturbance and terrain information in different scenarios. Blue arrow: estimated external force applied to the torso. Red ellipsoid: estimated terrain shape around the foot. The center of the ellipsoid refers to the estimated terrain elevation and the vertical length represents uncertainty (1 standard deviation). For each foot, 8 ellipsoids are symmetrically placed along a circle with 10 cm radius. Black arrow: terrain normal at the in-contact foot.

第三个概念是课程学习，对于实现其鲁棒性很重要。该教程根据控制器在训练过程不同阶段的表现，对不同地形进行自适应。本质上，控制器会经历各种合成地形的考验，同时变得更具鲁棒性。研究者评估了参数化地形的可通行性，并使用了粒子滤波来维持中等难度地形参数的分布，以适应神经网络的学习。训练环境的挑战性逐渐增加，促使了这种敏捷性与弹性兼具的全方位控制器的诞生。

研究人员先在模拟环境中训练控制器，随后将训练结果迁移到现实世界中，不需要进行艰苦的建模以及危险且高成本的实地测试，也不受物理世界的极度复杂性限制，该方法或许会引领未来机器人的发展。



在模拟训练中只使用刚性地貌和一组由程序生成的地形剖面，比如山丘和台阶。

当控制器被部署在四足机器人上时，它能够成功应对可变化地形（比如泥土、苔藓、雪地）、动态立足点（比如在杂乱室内环境踩到滚动板、田野中的碎片）和地面障碍物（厚植被、碎石、涌出的水）。

Terrain	grid size	friction coefficient	parameters (c_T)	range
Hills	0.2 m	$\mathcal{N}(0.7, 0.2)$	roughness (m) frequency amplitude (m)	[0.0, 0.05] [0.2, 1.0] [0.2, 3.0]
Slippery Hills	0.2 m	$\mathcal{N}(0.3, 0.1)$	roughness (m) frequency amplitude (m)	[0.0, 0.05] [0.2, 1.0] [0.2, 3.0]
Steps	0.02 m	$\mathcal{N}(0.7, 0.2)$	step width (m) step height (m)	[0.1, 0.5] [0.05, 0.3]
Stairs	0.02 m	$\mathcal{N}(0.7, 0.2)$	step width (m) step height (m)	[0.1, 0.5] [0.02, 0.2]

Table S2: **Parameter spaces \mathcal{C} for simulated terrains.** $\mathcal{N}(m, d)$ denotes that the value is sampled from the Gaussian distribution of mean m and standard deviation d . The friction coefficient is clipped to be above 0.1.

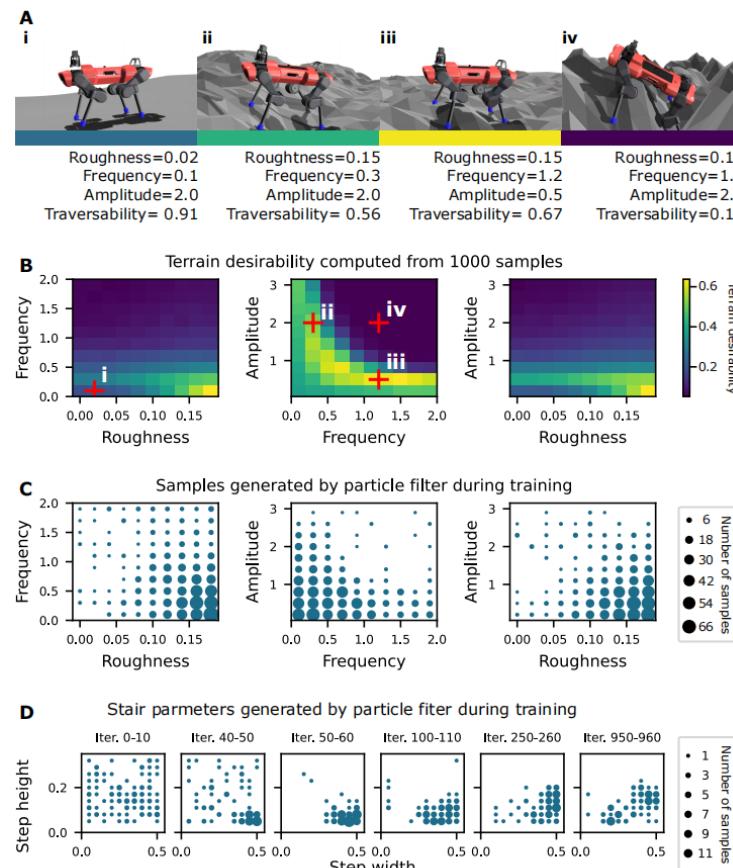


Figure S1: **Illustration of the adaptive curriculum.** **(A)** Examples of *Hills* terrains. The color bar indicates desirability; dark blue represents low desirability. **(B)** Terrain desirability estimated from 1000 trajectories generated by a fully trained teacher policy. The red crosses correspond to the examples presented in A. **(C)** The distribution of terrain profiles sampled by the particle filter during the last 100 iterations of teacher training. **(D)** Evolution of *Stairs* terrain parameters during training.

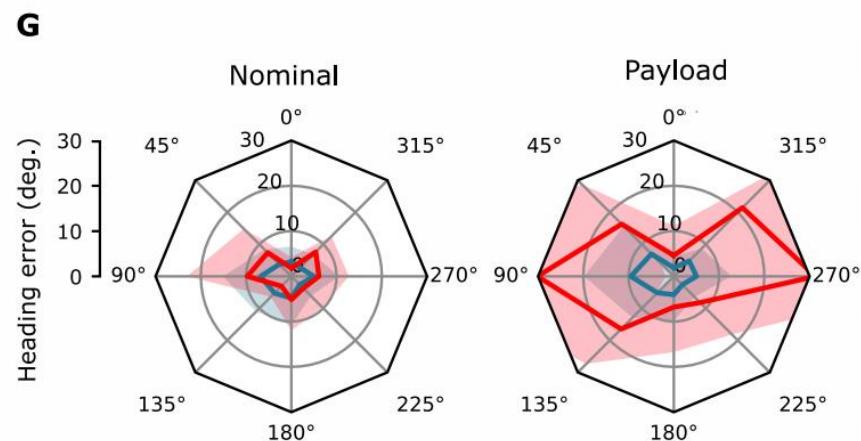
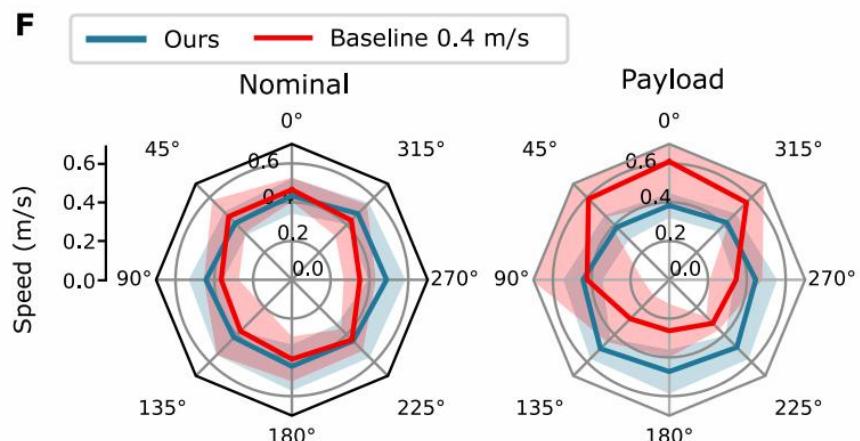
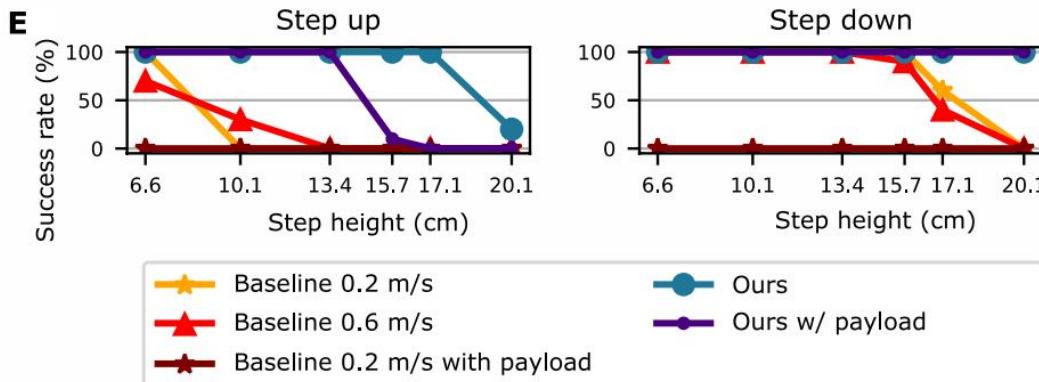
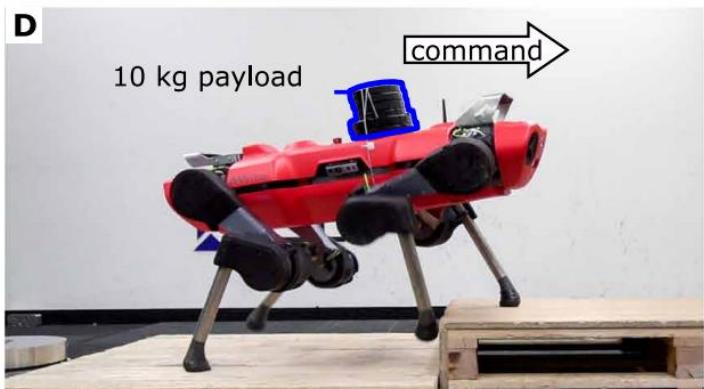


Fig. 3. Evaluation in an indoor environment. **(A)** Locomotion over unstable debris. The robot steps onto loose boards (highlighted in red and blue) that dislodge under the robot's feet. **(B)** The policy exhibits a foot-trapping reflex and overcomes a 16.8-cm step. **(C)** The policy learns to appropriately handle obstructions irrespective of the contact location. Here, it is shown reacting to an obstacle that is encountered mid-shin during the swing phase. **(D)** Controlled experiments with steps and payload. Our controller and a baseline (1, 27) were commanded to walk over a step with and without the 10-kg payload. **(E)** Success rates for different step heights. The success rate was evaluated over 10 trials for each condition. **(F)** Mean linear speeds for different command directions on flat terrain. 0° refers to the front of the robot. Shaded areas denote 95% CIs. **(G)** Mean heading errors for different command directions on flat terrain. Shaded areas denote 95% CIs.

Table 1. Comparison of locomotion performance in natural environments. The mechanical COT is computed using positive mechanical power exerted by the actuators.

Quantity	Controller	Terrain		
		Moss	Mud	Vegetation
Average speed (m/s)	Ours	0.452	0.338	0.248
	Baseline	0.199	0.197	—
Average mechanical COT	Ours	0.423	0.692	1.23
	Baseline	0.625	0.931	—

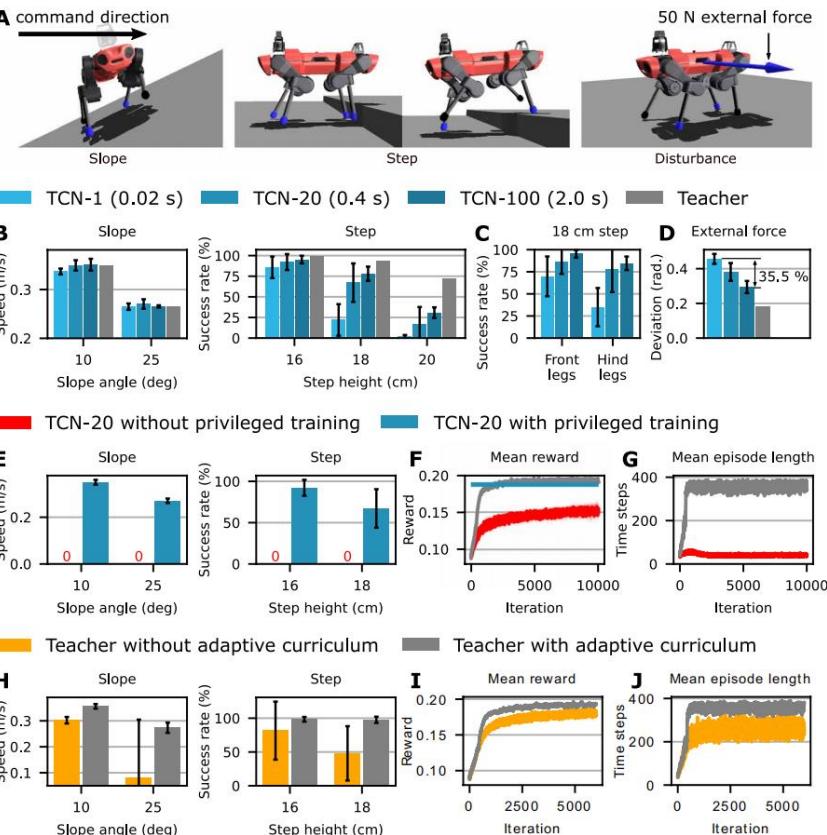


Fig. 5. Ablation studies. We trained each model five times using different random seeds. Error bars denote 95% CIs. (A) Test setups. The robot was commanded to advance for 10 s in the specified direction (black arrow). We conducted 100 trials for each test. On the step test, a trial was considered successful if the robot traversed the step with both front and hind legs. Robots were initialized with random joint configurations. Initial yaw angle was sampled from $U(-\pi/6, \pi/6)$ for the slope test and from $U(-\pi/6, \pi/6)$ for the other tests. The friction coefficients between the feet and the ground were sampled from $U(0.4, 1.0)$. The external force was applied for 5 s in the lateral direction. (B to D) Importance of memory length N in the TCN- N encoder. (E to G) Importance of privileged training. (F) Learning curves for the teacher (gray) and a TCN-20 student trained directly, without privileged training (red). For comparison, the blue line indicates the mean reward of a TCN-20 student trained with privileged training. The reward was computed by running each policy on uniformly sampled terrains. (H to J) Importance of the adaptive curriculum.

状态和特权信息

Data	dimension	x_t	o_t	h_t
Desired direction ($({}^B_{IB}\hat{v}_d)_{xy}$)	2		✓	✓
Desired turning direction ($({}^B_{IB}\hat{\omega}_d)_z$)	1		✓	✓
Gravity vector (e_g)	3		✓	✓
Base angular velocity (${}^B_{IB}\omega$)	3		✓	✓
Base linear velocity (${}^B_{IB}v$)	3		✓	✓
Joint position/velocity ($\theta_i, \dot{\theta}_i$)	24		✓	✓
FTG phases ($\sin(\phi_i), \cos(\phi_i)$)	8		✓	✓
FTG frequencies (ϕ_i)	4		✓	✓
Base frequency (f_o)	1		✓	
Joint position error history	24		✓	
Joint velocity history	24		✓	
Foot target history ($(r_{f,d})_{t-1,t-2}$)	24		✓	
Terrain normal at each foot	12	✓		
Height scan around each foot	36	✓		
Foot contact forces	4	✓		
Foot contact states	4	✓		
Thigh contact states	4	✓		
Shank contact states	4	✓		
Foot-ground friction coefficients	4	✓		
External force applied to the base	3	✓		

Table S4: State representation for proprioceptive controller (top) and the privileged information (bottom).

神经网络架构

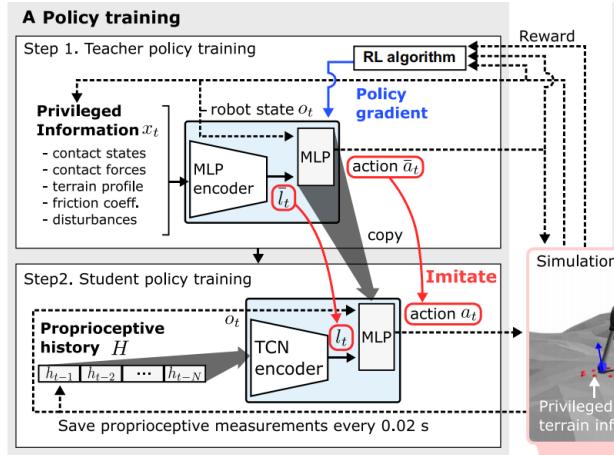
Layer	Teacher		TCN-N Student		GRU Student		Decoder
input	o_t	x_t	o_t	$h (60 \times N)$	o_t	o_t	$\langle o_t, l_t \rangle$
1	id	tanh(72)	id	1D conv dilation 1	id	GRU(68)	relu(196)
2	id	tanh(64)	id	1D conv stride 2	concatenate	Output	
3	concatenate		id	1D conv dilation 2	tanh(256)*		-
4	tanh(256)*		id	1D conv stride 2	tanh(128)*		-
5	tanh(128)*		id	1D conv dilation 4	tanh(64)*		-
6	tanh(64)*		id	1D conv stride 2	Output*		-
7	Output*		id	tanh(64)	-	-	
8	-		concatenate		-	-	
9	-		tanh(256)*		-	-	
10	-		tanh(128)*		-	-	
11	-		tanh(64)*		-	-	
12	-		Output*		-	-	

Table S5: **Neural network architectures.** Unless specified otherwise, the dilation and stride are 1 for convolutional layers. The filter size is fixed to 5. The layers marked with * are copied from the teacher to learners after the teacher training. id refers to the identity map. The TCN-N architecture uses dilated causal convolution (22). Each convolutional layer is followed by a relu activation function.

Model	seq. length	# channels	# param.	SGD time (s)
TCN-1	1	60	161960	9.22e-3 ($\pm 1.78e-3$)
TCN-20	20	44	158300	2.11e-2 ($\pm 1.24e-3$)
TCN-100	100	34	158070	5.07e-2 ($\pm 1.94e-3$)
GRU	100*	-	159640	1.52e-1 ($\pm 1.89e-2$)

Table S6: **Network parameter settings and training time for student policies.** SGD time refers to the computation time required for one stochastic gradient descent update with the batch size given in Table S8. The computation times are presented as empirical means with standard deviations. *The sequence length for the GRU network stands for the sequence length used for Truncated BPTT (51).

训练：Raisim仿真，教师策略TRPO强化学习训练，学生策略监督学习训练（蒸馏）



The student policy is trained via supervised learning. The loss function is defined as

$$\mathcal{L} := (\bar{a}_t(o_t, x_t) - a_t(o_t, H))^2 + (\bar{l}_t(o_t, x_t) - l_t(H))^2 \quad (1)$$

Quantities marked by a bar ($\bar{\cdot}$) denote target values generated by the teacher. We use the dataset aggregation strategy (DAgger) (38). Specifically, training data are generated by rolling out trajectories by the student policy. For each visited state, the teacher policy computes its embedding and action vectors (\cdot). These outputs of the teacher policy are used as supervisory signals associated with the corresponding states. The hyperparameters we used are given in table S8.

部署：域随机化（干扰、摩擦、噪声），速度估计，400Hz i7

During the deployment, the base velocity and orientation are estimated by the state estimator

that relies on inertial measurements and leg kinematics (37).

The neural network policy runs at 400 Hz on an onboard CPU (Intel i7-5600U, 2.6 – 3.2GHz, dual-core 64-bit) integrated into the robot. The Tensorflow C++ API is used for on-board inference.

Name	Time
Teacher policy training	≈ 12 hrs
Student policy training	≈ 4 hrs
Adaptive terrain curriculum	2.9 s

Table S1: **Computation time for training.** The TCN-100 architecture is used for the student policy. The training is conducted on a desktop machine with i7-8700K CPU and a Geforce RTX 2080 GPU.

足端轨迹生成器，使腿能够抬起

The FTG is a function $F(\phi) : [0.0, 2\pi] \rightarrow \mathbb{R}^3$ that outputs foot position targets for each leg. The FTG drives vertical stepping motion when f_i is nonzero. The definition of $F(\phi)$ is given in section S3. The policy

S3. Foot trajectory generator

The foot trajectory is defined as

$$F(\phi_i) = \begin{cases} (h(-2k^3 + 3k^2) - 0.5)^{H_i} z & k \in [0, 1] \\ (h(2k^3 - 9k^2 + 12k - 4) - 0.5)^{H_i} z & k \in [1, 2] \\ -0.5^{H_i} z & \text{otherwise,} \end{cases} \quad (10)$$

where $k = 2(\phi_i - \pi)/\pi$ and h is a parameter for the maximum foot height. Each segment during the swing phase ($k \in [0, 2]$) is a cubic Hermite spline connecting the highest and lowest points with a zero first derivative at the connecting points. Other periodic functions such as $h_i \sin(\phi_i)$ can be used for the FTG. With a set of reasonably tuned f_0 , h and $\phi_{i,0}$, a quadruped can stably step in place. In our setting, $f_0 = 1.25$, $h = 0.2$ m, and $\phi_{i,0}$ are sampled from $U(0, 2\pi)$.

奖励设计

S4. Reward function for teacher policy training

The reward function is defined as $0.05r_{lv} + 0.05r_{av} + 0.04r_b + 0.01r_{fc} + 0.02r_{bc} + 0.025r_s + 2 \cdot 10^{-5}r_\tau$. The individual terms are defined as follows.

- Linear Velocity Reward (r_{lv}): This term maximizes the $v_{pr} = (\overset{B}{IB}v)_{xy} \cdot (\overset{B}{IB}\hat{v}_T)_{xy}$, which is the base linear velocity projected onto the command direction.

$$r_{lv} := \begin{cases} \exp(-2.0(v_{pr} - 0.6)^2) & v_{pr} < 0.6 \\ 1.0 & v_{pr} \geq 0.6 \\ 0.0 & \text{zero command} \end{cases}. \quad (11)$$

The velocity threshold is defined as 0.6 m/s which is the maximum speed reachable on the flat terrain with the existing controller (27).

- Angular Velocity Reward (r_{av}): We motivate the agent to turn as fast as possible along the base z -axis when $(\overset{B}{IB}\hat{\omega}_T)_z$ is nonzero. It is defined as

$$r_{av} := \begin{cases} \exp(-1.5(\omega_{pr} - 0.6)^2) & \omega_{pr} < 0.6 \\ 1.0 & \omega_{pr} \geq 0.6 \end{cases}, \quad (12)$$

where $\omega_{pr} = (\overset{B}{IB}\omega)_z \cdot (\overset{B}{IB}\hat{\omega}_T)_z$.

- Base Motion Reward (r_b): This term penalizes the velocity orthogonal to the target direction and the roll and pitch rates such that the base is stable during the locomotion.

$$r_b := \exp(-1.5v_o^2) + \exp(-1.5\|(\overset{B}{IB}\omega)_{xy}\|^2) \quad (13)$$

where $v_o = \|(\overset{B}{IB}\omega)_{xy} - v_{pr} \cdot (\overset{B}{IB}\hat{v}_T)_{xy}\|$. When the stop command is given, v_o is replaced by $\|\overset{B}{IB}v\|$.

- Foot Clearance Reward (r_{fc}): When a leg is in swing phase, i.e., $\phi_i \in [\pi, 2\pi]$, the robot should lift the corresponding foot higher than the surroundings to avoid collision. We first define the set of such collision-free feet as $\mathcal{F}_{clear} = \{i : r_{f,i} > \max(H_{scan,i}), i \in I_{swing}\}$, where $H_{scan,i}$ is the set of scanned heights around the i -th foot. Then the clearance cost is defined as

$$r_{fc} := \sum_{i \in I_{swing}} (\mathbb{1}_{\mathcal{F}_{clear}}(i)/|I_{swing}|) \in [0.0, 1.0]. \quad (14)$$

- Body Collision Reward (r_{bc}): We want to penalize undesirable contact between the robot's body parts and the terrain to avoid hardware damage.

$$r_{bc} := -|I_{c,body} \setminus I_{c,foot}|. \quad (15)$$

- Target Smoothness Reward (r_s): The magnitude of the second order finite difference derivatives of the target foot positions are penalized such that the generated foot trajectories become smoother.

$$r_s := -||(r_{f,d})_t - 2(r_{f,d})_{t-1} + (r_{f,d})_{t-2}||. \quad (16)$$

- Torque Reward (r_τ): We penalize the joint torques to prevent damaging joint actuators during the deployment and to reduce energy consumption ($\tau \propto$ electric current).

$$r_\tau := -\sum_{i \in joints} |\tau_i|. \quad (17)$$

TCN 100最强, GRU高台阶不行

不采用隐变量 (TCN 100 naive) , 平地性能差不多, 高台阶差 $\mathcal{L} := (\bar{a}_t(o_t, x_t) - a_t(o_t, H))^2$,

S8. Recurrent neural network student policy

We use the TCN architecture for the proprioceptive policy (22). For comparison, we also evaluated a recurrent network with gated recurrent units (GRU) (50). The architectures are specified in Tables S5 and S6. The loss function for training a GRU student policy is defined as

$$\mathcal{L} := (\bar{a}_t(o_t, x_t) - a_t(o_t))^2 + (\bar{l}_t(o_t, x_t) - l_t(o_t))^2. \quad (18)$$

To improve the performance and computational efficiency of the training, we have implemented Truncated Backpropagation Through Time (Truncated BPTT) (51).

Performance on the diagnostic settings presented in Fig. 5A is given in Fig. S3. Overall, the performance of the GRU-based controller is between that of TCN-20 and TCN-100. The performance is comparable to TCN-100 in the slope setting, but the GRU-based controller fails to achieve the performance of TCN-100 in step experiments.

The chief advantage of the TCN is in training efficiency. The training time for the TCN is much faster in comparison to the GRU. The computation times are reported in Table S6.

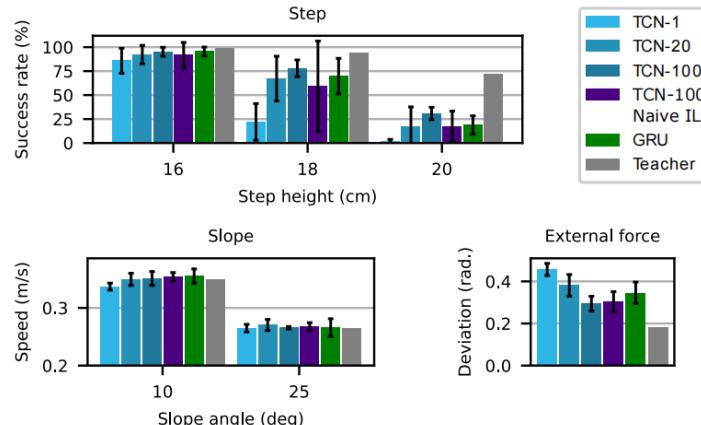


Figure S3: Comparison of neural network architectures for the proprioceptive controller.
We trained each model 5 times using different random seeds. The error bars denote 95% confidence intervals. ‘TCN-100 naive IL’ denotes the TCN-100 network trained using a naive imitation learning method without the latent representation loss (Eq. 19).

视觉加持，如履平地。这篇工作和上一篇工作的主要区别在于使用了感知信息（视觉、雷达）使四足能够获得更加完备的信息。



Attention-based Recurrent Encoder (RNN)

提出了一个基于 attention 机制的 belief state encoder，可以融合感知信息和本体信息，同时可以对未来进行一定的推断。例如，智能体在通过下图的软体材质时，belief 编码器会记录这一信息，并在后面时间步的推断中修改对地形信息的估计。

Privileged Learning

在学习上仍然沿用了上一篇的做法，首先在仿真环境中利用完备的感知信息和本体信息，基于强化学习算法训练一个 teacher 策略。随后 student 使用 belief state encoder 作为状态信息的编码，使用 teacher 策略产生的动作作为监督信息，通过策略蒸馏得到 student 策略。belief state 的优势是能够通过历史信息推断智能体所看不到的部分信息，从而解决部分可观测的问题。在实际训练中，本文方法也使用了其他的技巧，例如 *curriculum Learning, randomization* 等。下图显示了 belief encoder 的具体结构，encoder-decoder 的结构在策略蒸馏损失的基础上，额外的给出了 reconstruction 的损失，可以用于加速学习 belief state。

Fig. 6. Overview of the training methods and deployment. We first train a teacher policy with access to privileged simulation data using RL. This teacher policy is then distilled into a student policy, which is trained to imitate the teacher's actions and to reconstruct the ground-truth environment state from noisy observations. We deploy the student policy zero-shot on real hardware using height samples from a robot-centric elevation map.

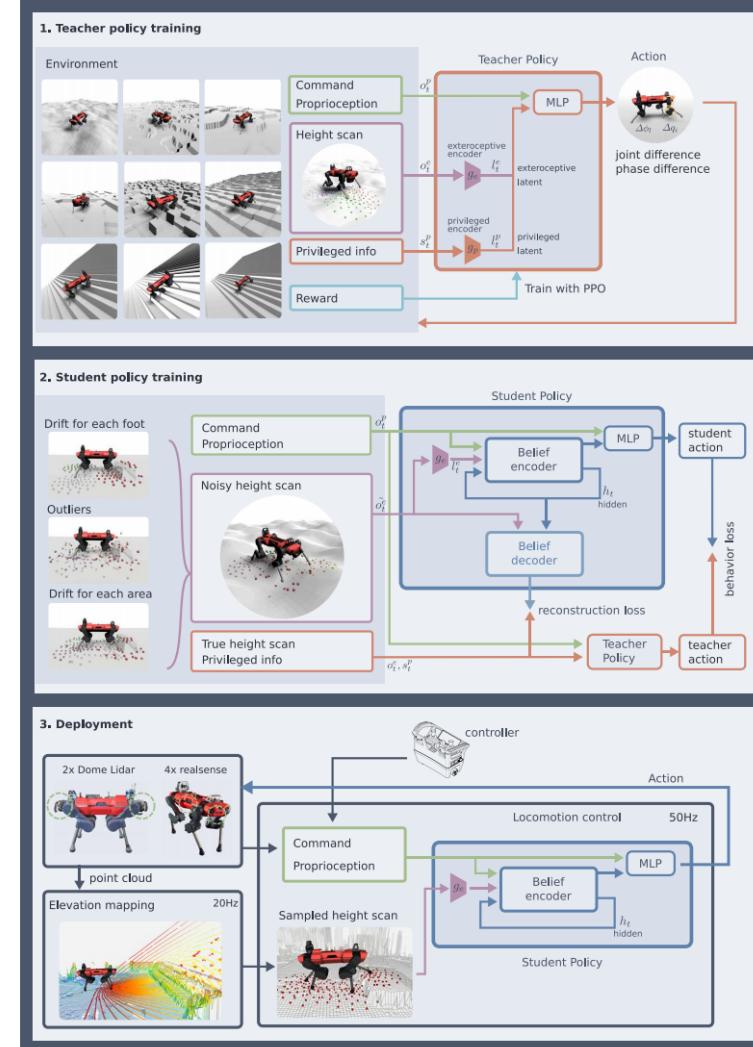
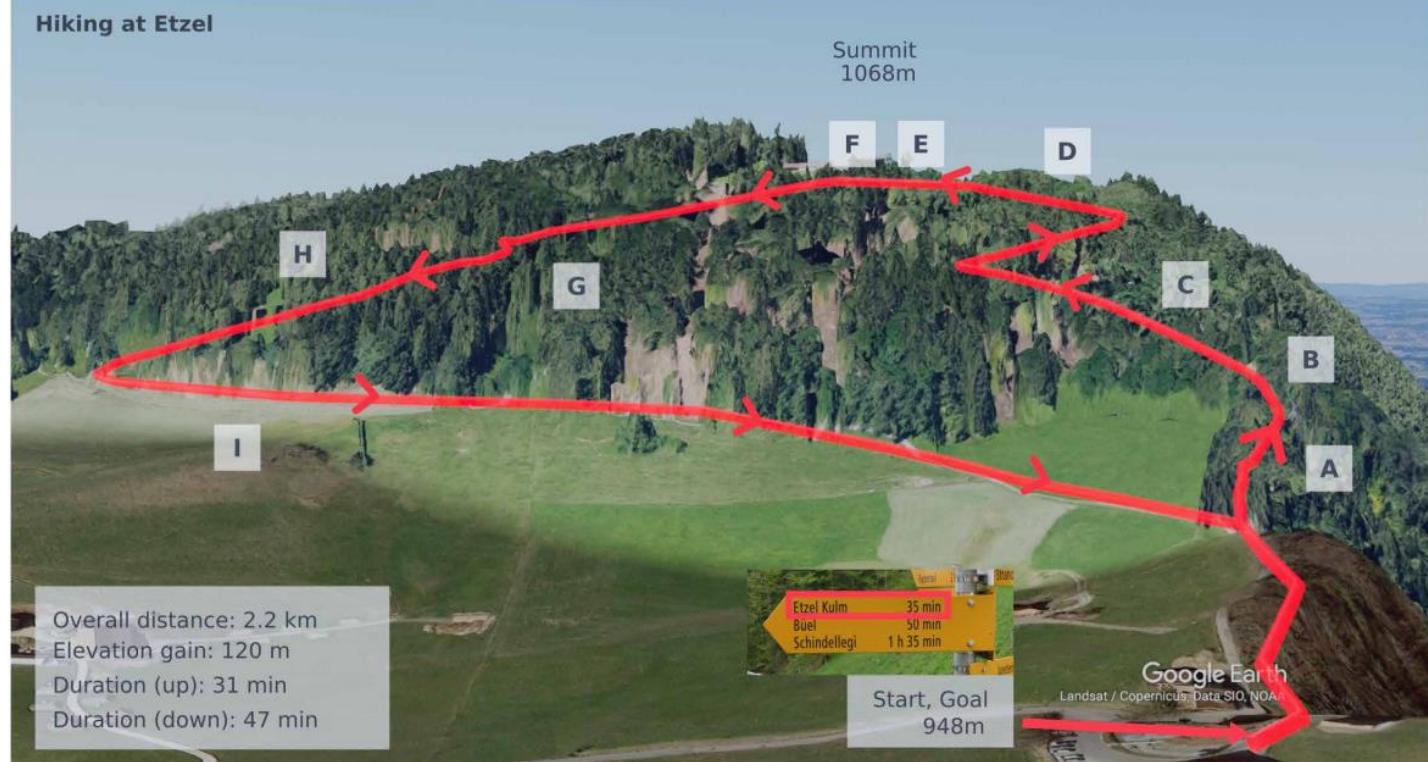


Fig. 2. A hike on the Etzel mountain in Switzerland, completed by ANYmal with our locomotion controller. The 2.2-km route—with 120 m of elevation gain and inclinations up to 38%—encompasses a variety of challenging terrains [(A) to (I)]. ANYmal reached the summit faster than the human time indicated in the official signage and finished the entire route in virtually the same time as given by a hiking guide (47).



视觉表示与挑战

视觉的问题：
不可靠

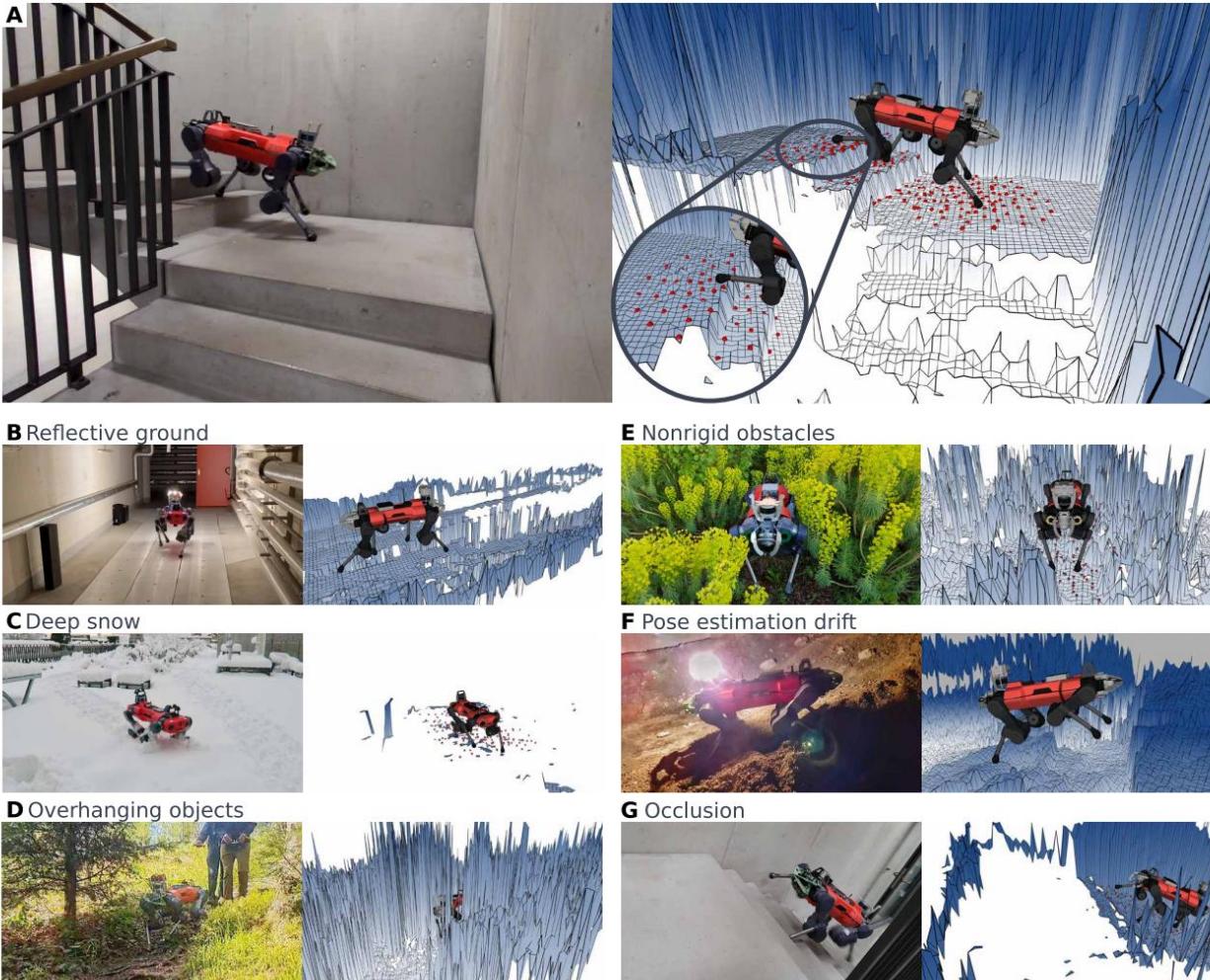


Fig. 3. Exteroceptive representation and challenges. Our locomotion controller perceives the environment through height samples (red dots) from an elevation map (A). The controller is robust to many perception challenges commonly encountered in the field: missing map information due to sensing failure (B, C, and G) and misleading map information due to nonrigid terrain (D and E) and pose estimation drift (F).

性能对比： 有视觉vs无视觉

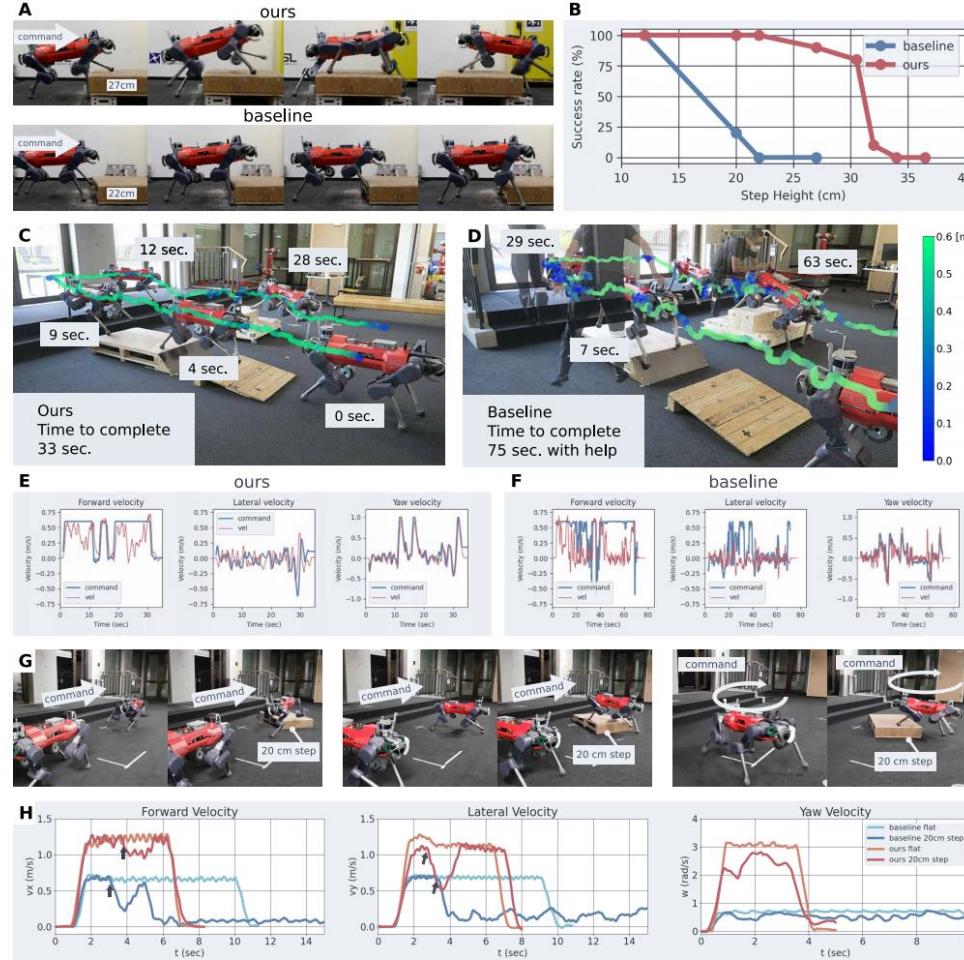


Fig. 4. We compared the presented controller with a proprioceptive baseline. An experiment with steps of varying height shows that our controller can overcome notably higher obstacles than the baseline (**A** and **B**). Our method completes an obstacle course in less than half the time of the baseline and without requiring any human help (**C** and **D**). As seen in the graphs, our controller could follow the command more precisely. Note that the directional command plotted in (**F**) is scaled to 0.6 m/s. (**E** and **F**) Our controller can maintain double the linear velocity of the baseline and achieves a fivefold increase in turning speed. The arrows indicate when the robot reached the step (**G** and **H**).

性能对比：

有视觉
VS
无视觉

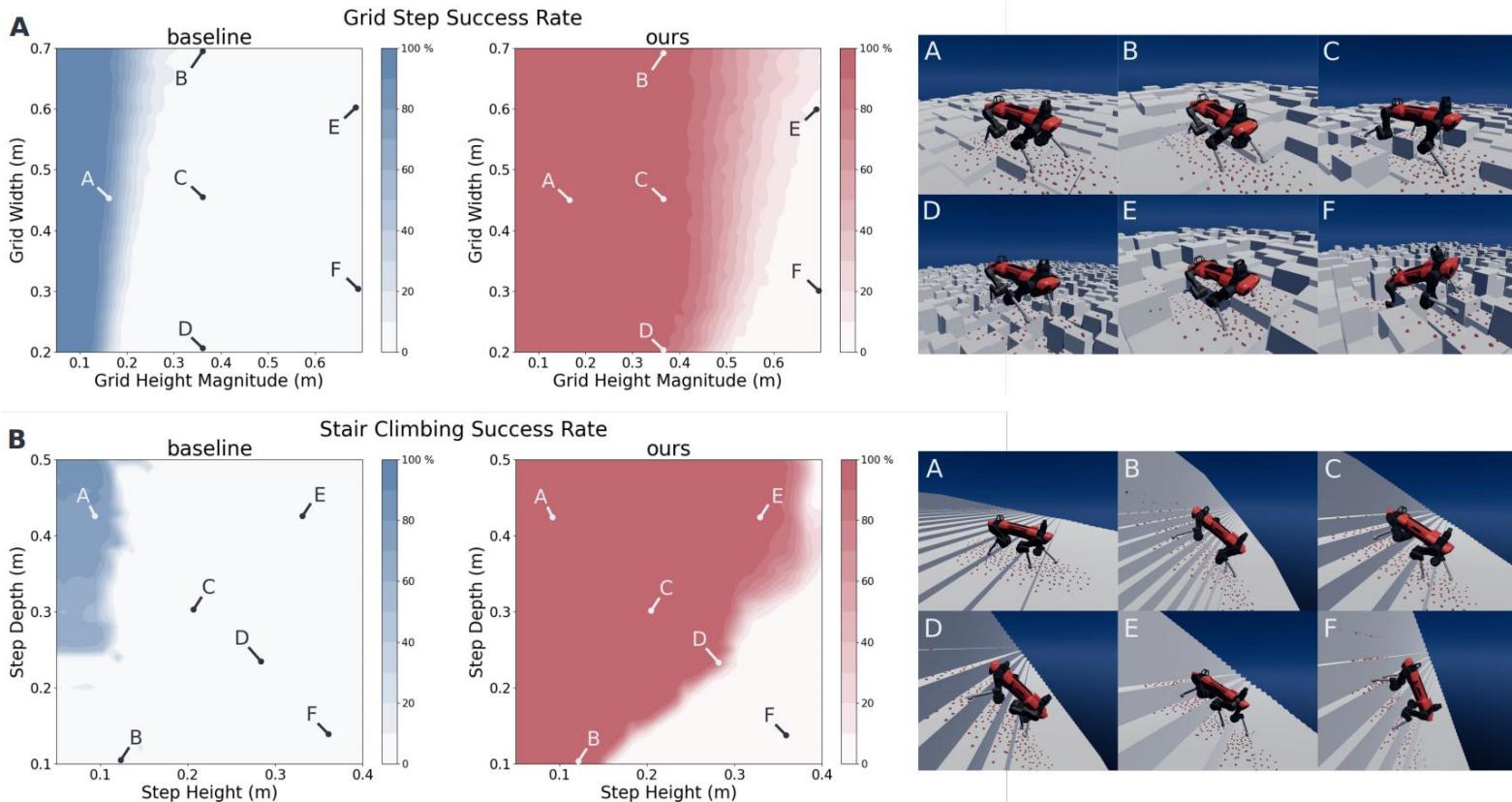


Figure S1: Comparison of the presented controller to a proprioceptive baseline (4) over random terrain. We collected 300 trials with a fixed velocity command over 41×41 different terrain parameter combinations and compared success rates. Our controller was able to traverse a much wider range of terrain profiles on both grid steps (A) and stairs (B).

置信网络： 纠正视觉感知偏差

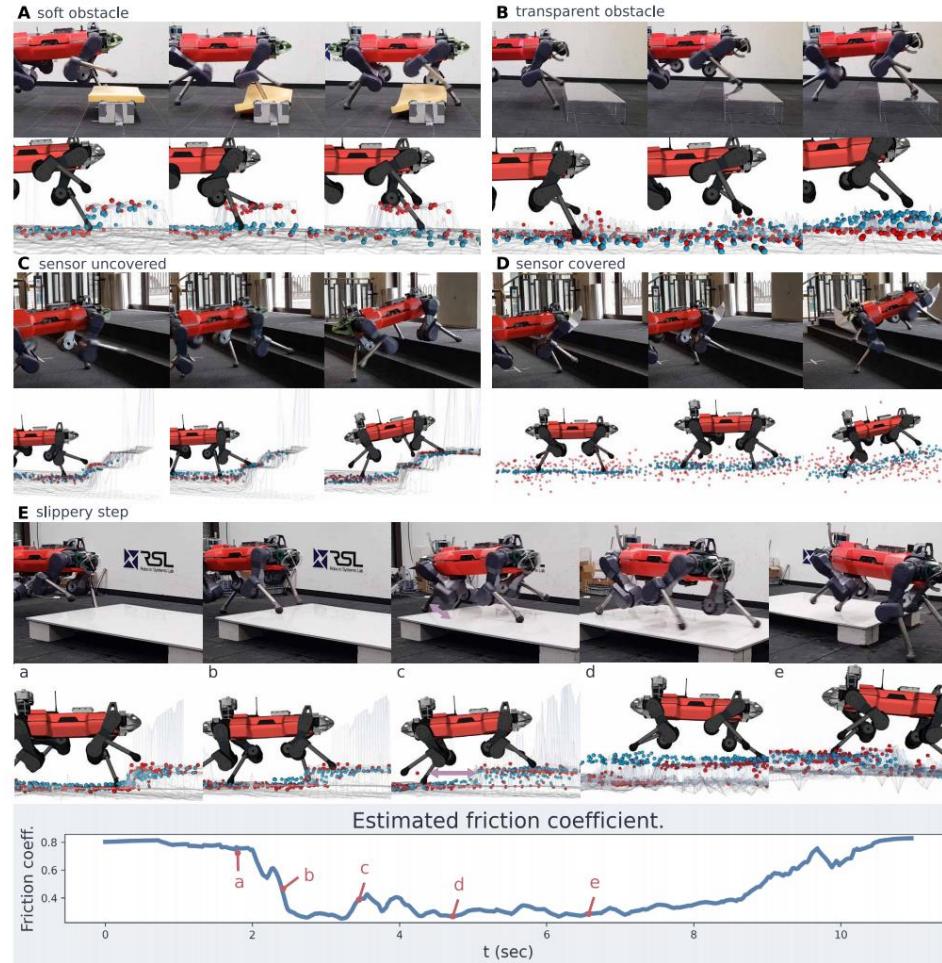


Fig. 5. Internal belief state inspection during perceptive failure using a learned belief decoder. Red dots indicate height samples given as input to the policy. Blue dots show the controller's internal estimate of the terrain profile. (A) After stepping on a soft obstacle that cannot support a foothold, the policy correctly revises its estimate of the terrain profile downward. (B) A transparent obstacle is correctly incorporated into the terrain profile after contact is made. (C) With operational sensors, the robot swiftly and gracefully climbs the stairs, with no spurious contacts. (D) When the robot is blinded by covering the sensors, the policy can no longer anticipate the terrain but remains robust and successfully traverses the stairs. (E) When stepping onto a slippery platform, the policy identifies low friction and compensates for the increase.

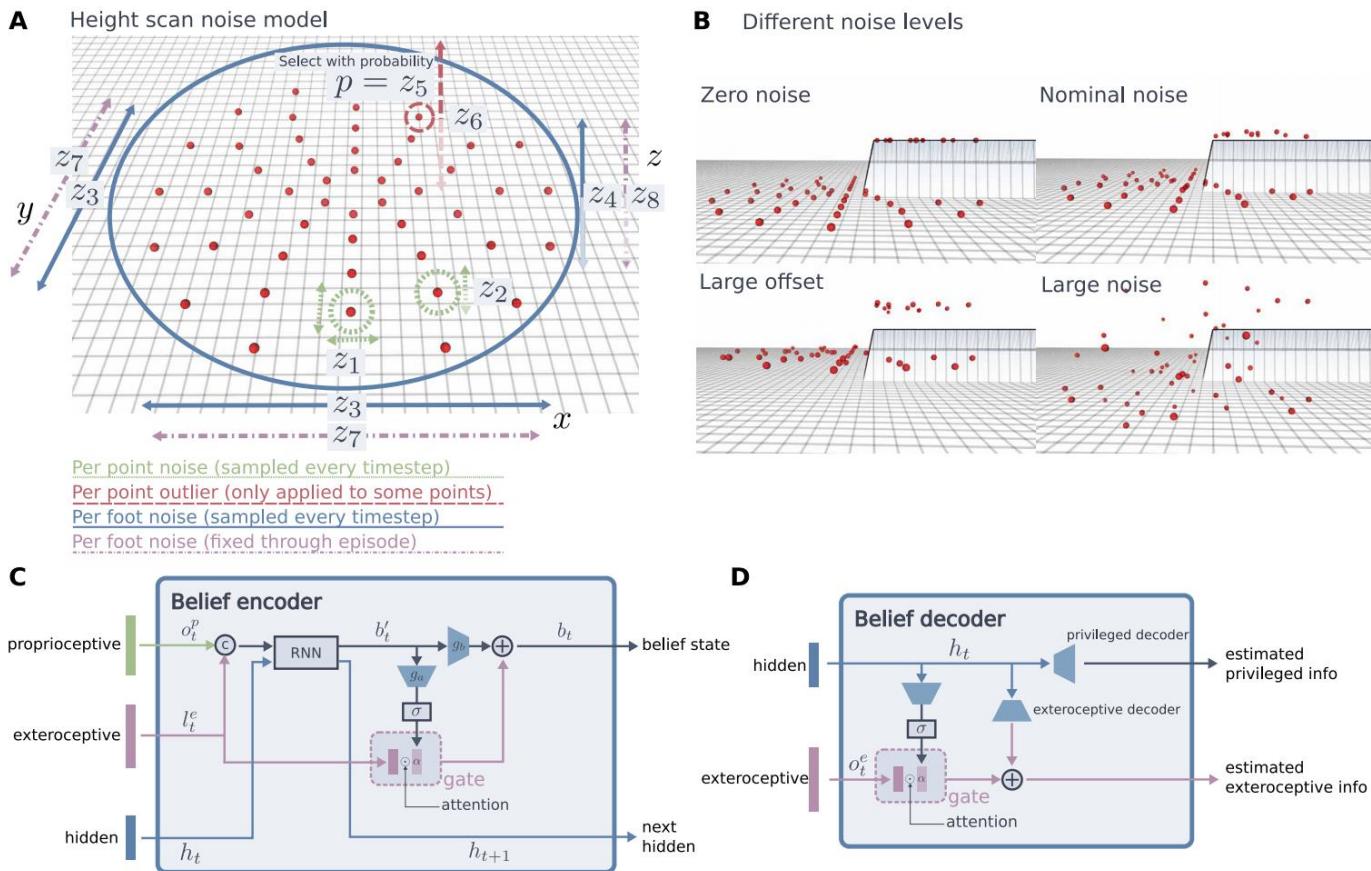


Fig. 7. Details of robust terrain perception components. (A) During student training, random noise is added to the height samples. The noise is sampled from a Gaussian distribution $\mathcal{N}(0, z^l \in \mathbb{R}^8)$, where each z_i^l controls a different noise component i per leg l . (B) We use multiple noise configurations z to simulate different operating conditions. “Zero noise” is applied during teacher training, whereas “nominal noise” represents normal mapping conditions during student training. “Large offset” noise simulates large map offsets due to pose estimation drift or deformable terrain surfaces. “Large noise” simulates a complete lack of terrain information due to occlusion or sensor failure. (C) The student policy belief encoder incorporates a recurrent core and an attentional gate that integrates the proprioceptive and exteroceptive modalities. The gate explicitly controls which aspects of exteroceptive data should pass through. (D) The belief decoder has a gate for reconstructing the exteroceptive data. It is only used during training and for introspection into the belief state.

Teacher policy training

In the first stage of training, we aim to find an optimal reference control policy that has access to perfect, privileged information and enables ANYmal to follow a desired command velocity over randomly generated terrain. The desired command is generated randomly as a vector $\mathbf{v}_{\text{des}} \in \mathbb{R}^3 = (v_x, v_y, w)$, where v_x, v_y represents the longitudinal and lateral velocity, and w represents the yaw velocity, all in the robot's body frame.

We used proximal policy optimization (PPO) (59) to train the teacher policy. The teacher is modeled as a Gaussian policy, $a_t \sim \mathcal{N}(\pi_\theta(o_t = s_t), \sigma I)$, where π_θ is implemented by a multilayer perceptron (MLP) parameterized by θ , and σ represents the variance for each action.

Observation and action

The teacher observation is defined as $o_t^{\text{teacher}} = (o_t^p, o_t^e, s_t^p)$, where o_t^p refers to the proprioceptive observation, o_t^e refers to the exteroceptive observation, and s_t^p refers to the privileged state. o_t^p contains the body velocity, orientation, joint position and velocity history, action history, and each leg's phase. o_t^e is a vector of height samples around each foot with five different radii. The privileged state s_t^p includes contact states, contact forces, contact normals, friction coefficient, thigh and shank contact states, external forces and torques applied to the body, and swing phase duration.

Our action space is inspired by central pattern generators (4). Each leg $l = \{1, 2, 3, 4\}$ keeps a phase variable ϕ_l and defines a nominal trajectory based on the phase. The nominal trajectory is a stepping motion of the foot tip, and we calculate the nominal joint target $q_l(\phi_l)$ for each joint actuator $i = \{1, \dots, 12\}$ using inverse kinematics. The action from the policy is the phase difference $\Delta\phi_l$ and the residual joint position target Δq_i . More details of the observation and action space are in section S5.

RMA: Rapid Motor Adaptation for Legged Robots

Ashish Kumar
UC Berkeley

Zipeng Fu
Carnegie Mellon University

Deepak Pathak
Carnegie Mellon University

Jitendra Malik
UC Berkeley, Facebook

采用A1机器狗

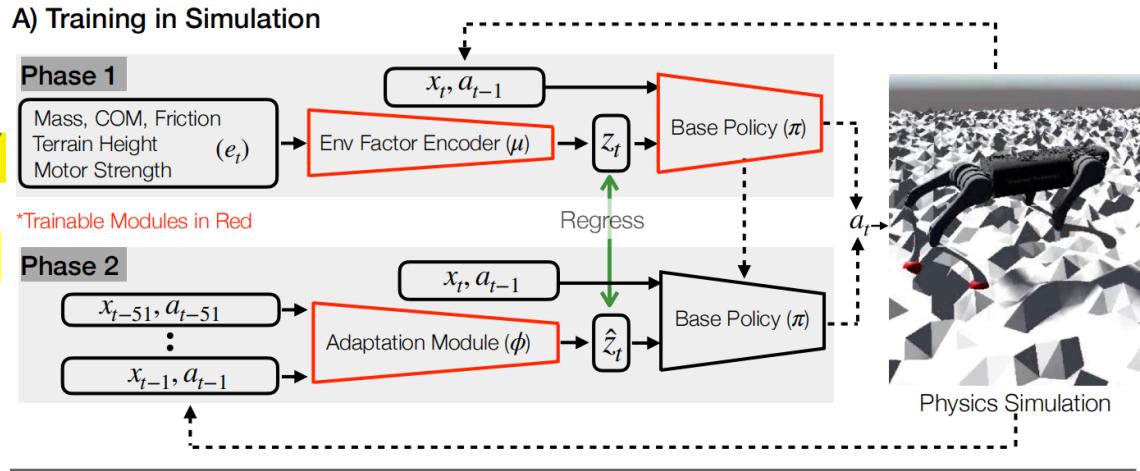


相比ETH工作：

去除了轨迹生成器
不再需要对电机建模
用了不一样的奖励
异步网络 (10Hz和100Hz)
自适应不需要域随机化

$$e_t \in \mathbb{R}^{17}$$

$$z_t \in \mathbb{R}^8$$



B) Deployment

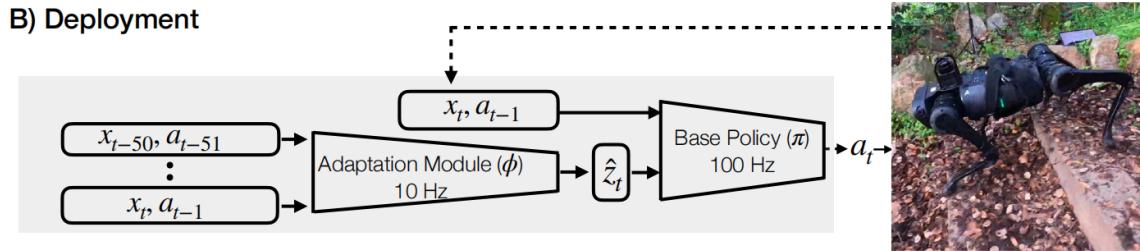


Fig. 2: RMA consists of two subsystems - the base policy π and the adaptation module ϕ . **Top:** RMA is trained in two phases. In the first phase, the base policy π takes as input the current state x_t , previous action a_{t-1} and the privileged environmental factors e_t which is encoded into the latent extrinsics vector z_t using the environmental factor encoder μ . The base policy is trained in simulation using model-free RL. **In the second phase, the adaptation module ϕ is trained to predict the extrinsics \hat{z}_t from the history of state and actions via supervised learning with on-policy data.** **Bottom:** At deployment, the adaptation module ϕ generates the extrinsics \hat{z}_t at 10Hz, and the base policy generates the desired joint positions at 100Hz which are converted to torques using A1's PD controller. Since the adaptation module runs at a lower frequency, the base policy consumes the most recent extrinsics vector \hat{z}_t predicted by the adaptation module to predict a_t . This asynchronous design was critical for seamless deployment on low-cost robots like A1 with limited on-board compute. Videos at: <https://ashish-kmr.github.io/rma-legged-robots/>

RL Rewards: The reward function encourages the agent to move forward with a maximum speed of 0.35 m/s, and penalizes it for jerky and inefficient motions. Let's denote the linear velocity as v , the orientation as θ and the angular velocity as ω , all in the robot's base frame. We additionally define the joint angles as q , joint velocities as \dot{q} , joint torques as τ , ground reaction forces at the feet as f , velocity of the feet as v_f and the binary foot contact indicator vector as g . The reward at time t is defined as the sum of the following quantities:

- 1) Forward: $\min(v_x^t, 0.35)$
- 2) Lateral Movement and Rotation: $-\|v_y^t\|^2 - \|\omega_{yaw}^t\|^2$
- 3) Work: $-\|\tau^T \cdot (q^t - q^{t-1})\|$
- 4) Ground Impact: $-\|f^t - f^{t-1}\|^2$
- 5) Smoothness: $-\|\tau^t - \tau^{t-1}\|^2$
- 6) Action Magnitude: $-\|a^t\|^2$
- 7) Joint Speed: $-\|\dot{q}^t\|^2$
- 8) Orientation: $-\|\theta_{roll, pitch}^t\|^2$
- 9) Z Acceleration: $-\|v_z^t\|^2$
- 10) Foot Slip: $-\|\text{diag}(g^t) \cdot v_f^t\|^2$

课程学习：

惩罚项从很小逐渐变大

扰动范围增大

无地形课程

Training Curriculum: If we naively train our agent with the above reward function, it learns to stay in place because of the penalty terms on the movement of the joints. To prevent this collapse, we follow the strategy described in [23]. We start the training with very small penalty coefficients, and then gradually increase the strength of these coefficients using a fixed curriculum. We also linearly increase the difficulty of other perturbations such as mass, friction and motor strength as the training progresses. We don't have any curriculum on the terrains and start the training with randomly sampling the terrain profiles from the same fixed difficulty.

异步部署：高低频分层网络

不分层的话：学不好

C. Asynchronous Deployment

We train RMA completely in simulation and then deploy it in the real world without any modification or fine-tuning. The two subsystems of RMA run asynchronously and at substantially different frequencies, and hence, can easily run using little on-board compute. The adaptation policy is slow because it operates on the state-action history of 50 time steps, roughly updating the extrinsic vector \hat{z}_t once every 0.1s (10 Hz). The base policy runs at 100 Hz and uses the most recent \hat{z}_t generated by the adaptation module, along with the current state and the previous action, to predict a_t . This asynchronous execution doesn't hurt performance in practice because \hat{z}_t changes relatively infrequently in the real world.

Alternately, we could have trained a base policy which directly takes the state and action history as input without decoupling them into the two modules. We found that this (a) leads to unnatural gaits and poor performance in simulation, (b) can only run at 10Hz on the on-board compute, and (c) lacks the asynchronous design which is critical for a seamless deployment of RMA on the real robot without the need for any synchronization or calibration of the two subsystems. This asynchronous design is fundamentally enabled by the decoupling of the relatively infrequently changing extrinsics vector with the quickly changing robot state.

性能对比：优于自带算法、不加适应 模块成功率率为零

Hardware Details: We use A1 robot from Unitree for all our real-world experiments. A1 is a relatively low cost medium sized robotic quadruped dog. It has 18 degrees of freedom out of which 12 are actuated (3 motors on each leg) and weighs about 12 kg. To measure the current state of the robot, we use the joint position and velocity from the motor encoders, roll and pitch from the IMU sensor and the binarized foot contact indicators from the foot sensors. The deployed policy uses position control for the joints of the robots. The predicted desired joint positions are converted to torque using a PD controller with fixed gains ($K_p = 55$ and $K_d = 0.8$).

Simulation Setup: We use the RaiSim simulator [22] for rigid-body and contact dynamics simulation. We import the A1 URDF file from Unitree [53] and use the inbuilt fractal terrain generator to generate uneven terrain (fractal octaves = 2, fractal lacunarity = 2.0, fractal gain = 0.25, z-scale = 0.27). Each RL episode lasts for a maximum of 1000 steps, with early termination if the height of the robots drops below 0.28m, magnitude of the body roll exceeds 0.4 radians or the pitch exceeds 0.2 radians. The control frequency of the policy is 100 Hz, and the simulation time step is 0.025s.

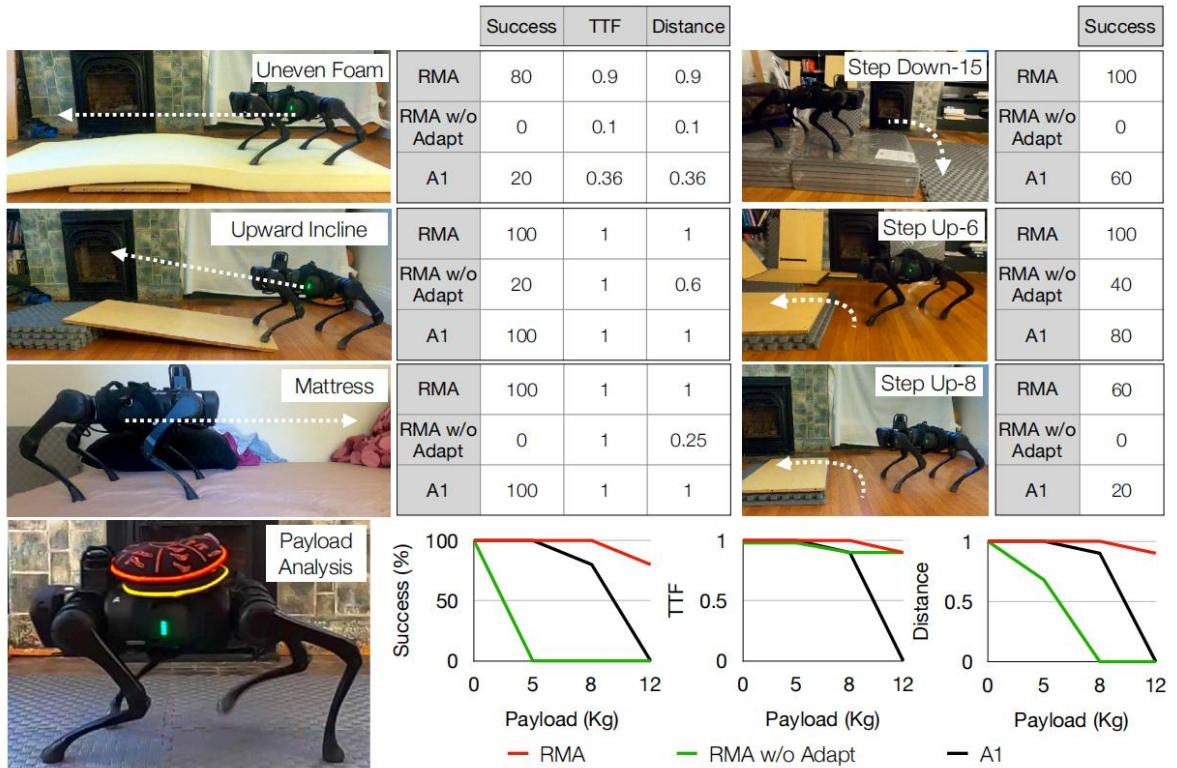


Fig. 3: We evaluate RMA in several out-of-distribution setups in the real world. We compare RMA to A1’s controller and RMA without the adaptation module. We find that RMA steps down a height of 15cm with 80% success rate and walks over unseen deformable surfaces, such as a memory foam mattress and a slightly uneven foam with 100% success rate. It is also able to successfully climb inclines and steps. A1’s controller fails to walk over uneven foam. At the bottom, we also analyze the payload carrying limits of the three methods. We see that the A1 controller’s performance starts degrading at 8Kg payload capacity. RMA w/o adaptation fails to move for payloads more than 8Kg, but rarely falls. For reference, A1 robot weights 12Kg. Overall, the proposed method consistently dominates the baseline methods. The numbers reported are averaged over 5 trials.

策略网络MLP、环境编码器MLP 自适应网络MLP+CNN

Base Policy and Environment Factor Encoder Architecture:

The base policy is a 3-layer multi-layer perceptron (MLP) which takes in the current state $x_t \in \mathbb{R}^{30}$, previous action $a_{t-1} \in \mathbb{R}^{12}$ and the extrinsics vector $z_t \in \mathbb{R}^8$, and outputs 12-dim target joint angles. The dimension of hidden layers is 128. The environment factor encoder is a 3-layer MLP (256, 128 hidden layer sizes) and encodes $e_t \in \mathbb{R}^{17}$ into $z_t \in \mathbb{R}^8$.

Adaptation Module Architecture: The adaptation module first embeds the recent states and actions into 32-dim representations using a 2-layer MLP. Then, a 3-layer 1-D CNN convolves the representations across the time dimension to capture temporal correlations in the input. The input channel number, output channel number, kernel size, and stride of each layer are $[32, 32, 8, 4]$, $[32, 32, 5, 1]$, $[32, 32, 5, 1]$. The flattened CNN output is linearly projected to estimate \hat{z}_t .

策略网络PPO训练12亿步 自适应模块监督学习8千万步

Learning Base Policy and Environmental Factor Encoder Network: We jointly train the base policy and the environment encoder network using PPO [48] for 15,000 iterations each of which uses batch size of 80,000 split into 4 mini-batches. The learning rate is set to $5e-4$. The coefficient of the reward terms are provided in Section III. Training takes roughly 24 hours on an ordinary desktop machine, with 1 GPU for policy training. In this duration, it simulates 1.2 billion steps.

Learning Adaptation Module: We train the adaptation module using supervised learning with on-policy data. We use Adam optimizer [29] to minimize MSE loss. We run the optimization process for 1000 iterations with a learning rate of $5e-4$ each of which uses a batch size of 80,000 split up into 4 mini-batches. It takes 3 hours to train this on an ordinary desktop machine, with 1 GPU for training the policy. In this duration, it simulates 80 million steps.



提纲

一、ETH三部曲

二、AMP三部曲

三、Pakour三部曲

四、无人机三部曲

2021
AMP提出
纯仿真

AMP: Adversarial Motion Priors for Stylized Physics-Based Character Control

XUE BIN PENG*, University of California, Berkeley, USA

ZE MA*, Shanghai Jiao Tong University, China

PIETER ABBEEL, University of California, Berkeley, USA

SERGEY LEVINE, University of California, Berkeley, USA

ANGJOO KANAZAWA, University of California, Berkeley, USA

2022
AMP应用
四轮足直立

Advanced Skills through Multiple Adversarial Motion Priors in Reinforcement Learning

Eric Vollenweider, Marko Bjelonic, Victor Klemm, Nikita Rudin, Joonho Lee and Marco Hutter

IEEE ROBOTICS AND AUTOMATION LETTERS, VOL. 8, NO. 8, AUGUST 2023

2023
AMP应用
四足复杂地形

Learning Robust and Agile Legged Locomotion Using Adversarial Motion Priors

Jinze Wu , Guiyang Xin , Member, IEEE, Chenkun Qi , Member, IEEE, and Yufei Xue 

Peng, Xue Bin, Ze
Ma, Pieter Abbeel,
Sergey Levine, and
Angjoo Kanazawa.
"Amp: Adversarial
motion priors for
stylized physics-
based character
control." *ACM
Transactions on
Graphics (ToG)* 40,
no. 4 (2021): 1-20.

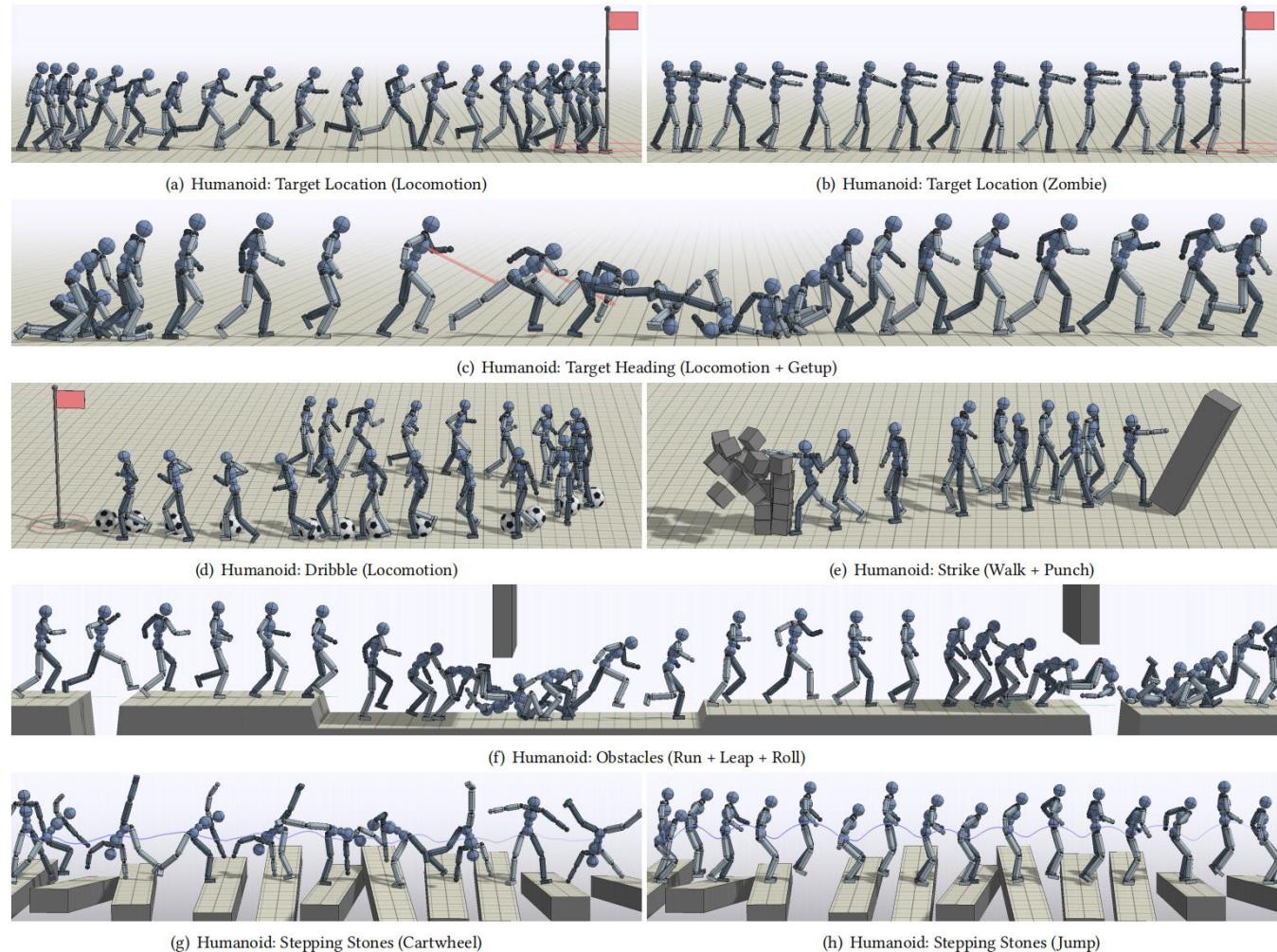


Fig. 3. The motion prior can be trained with large datasets of diverse motions, enabling simulated characters to perform complex tasks by composing a wider range of skills. Each environment is denoted by "Character: Task (Dataset)".

- 生成对抗模仿学习(GAIL)
- 从无标注的视频片段中模仿学习
- 一个网络学会多种动作自主切换
- 无需使用隐变量

based on adversarial imitation learning. High-level task objectives that the character should perform can be specified by relatively simple reward functions, while the low-level style of the character's behaviors can be specified by a dataset of unstructured motion clips, without any explicit clip selection or sequencing. For example, a character traversing an obstacle course might prior also does not require a separate pre-training phase, and instead, can be trained jointly with the policy.

模仿奖励+任务奖励

底层
数据集驱动
动作库先验

高层
简单目标驱动
动作选择

鉴别器与策略同时训练

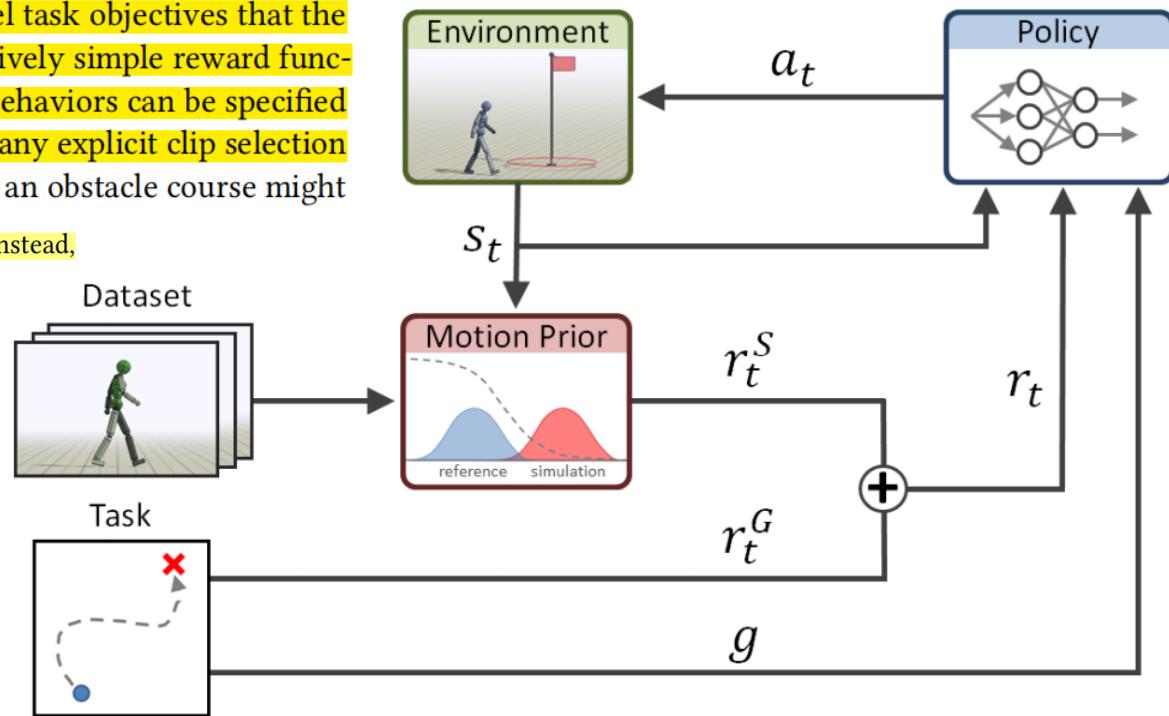


Fig. 2. Schematic overview of the system. Given a motion dataset defining a desired motion style for the character, the system trains a motion prior that specifies style-rewards r_t^S for the policy during training. These style-rewards are combined with task-rewards r_t^G and used to train a policy that enables a simulated character to satisfy task-specific goals g , while also adopting behaviors that resemble the reference motions in the dataset.

In this work, we adopt the loss function proposed for least-squares GAN (LSGAN) [Mao et al. 2017], which has demonstrated more stable training and higher quality results for image synthesis tasks. The following objective is used to train the discriminator,

$$\arg \min_D \mathbb{E}_{d^M(s, s')} [(D(s, s') - 1)^2] + \mathbb{E}_{d^\pi(s, s')} [(D(s, s') + 1)^2]. \quad (6)$$

The discriminator is trained by solving a least-squares regression problem to predict a score of 1 for samples from the dataset and -1 for samples recorded from the policy. The reward function for training the policy is then given by

$$r(s_t, s_{t+1}) = \max [0, 1 - 0.25(D(s_t, s_{t+1}) - 1)^2]. \quad (7)$$

6.1 States and Actions

状态、动作

The state s_t consists of a set of features that describes the configuration of the character's body. The features are similar to those used by Peng et al. [2018a], which include the relative positions of each link with respect to the root, the rotation of each link as represented using the 6D normal-tangent encoding, along with the link's linear and angular velocities. All features are recorded in the character's local coordinate system. Unlike previous systems, which synchronize the policy with a particular reference motion by including additional phase information in the state, such as scalar phase variables [Lee et al. 2019; Peng et al. 2018a] or target poses [Bergamin et al. 2019; Chentanez et al. 2018; Won et al. 2020], our policies are not trained to explicitly imitate any specific motion from the dataset. Therefore, no such synchronization or phase information is necessary.

Each action a_t specifies target positions for PD controllers positioned at each of the character's joints. For spherical joints, each

任务奖励

模仿奖励

Dribbling: To evaluate our system on more complex object manipulation tasks, we train policies for a dribbling task, where the objective is for the character to dribble a soccer ball to a target location. The reward function is given by:

$$r_t^G = 0.1r_t^{cv} + 0.1r_t^{cp} + 0.3r_t^{bv} + 0.5r_t^{bp} \quad (13)$$

$$r_t^{cv} = \exp(-1.5 \max(0, v^* - d_t^{\text{ball}} \cdot \dot{x}_t^{\text{com}})^2) \quad (14)$$

$$r_t^{cp} = \exp(-0.5 \|x_t^{\text{ball}} - x_t^{\text{com}}\|^2) \quad (15)$$

$$r_t^{bv} = \exp(-\max(0, v^* - d_t^* \cdot \dot{x}_t^{\text{ball}})^2) \quad (16)$$

$$r_t^{bp} = \exp(-0.5 \|x_t^* - x_t^{\text{com}}\|^2). \quad (17)$$

Target Heading: In this task, the objective for the character is to move along a target heading direction d^* at a target speed v^* . The goal input for the policy is specified as $g_t = (\dot{d}_t^*, v^*)$, with \dot{d}_t^* being the target direction in the character's local coordinate frame. The task-reward is calculated according to:

$$r_t^G = \exp(-0.25(v^* - d^* \cdot \dot{x}_t^{\text{com}})^2), \quad (11)$$

where \dot{x}_t^{com} is the center-of-mass velocity of the character at time step t , and the target speed is selected randomly between $v^* \in [1, 5]$ m/s. For slower moving styles, such as Zombie and Stealthy, the target speed is fixed at 1 m/s.

Target Location: In this task, the character's objective is to move to a target location x^* . The goal $g_t = \dot{x}_t^*$ records the target location in the character's local coordinate frame. The task-reward is given by:

$$r_t^G = 0.7 \exp(-0.5 \|x^* - x_t^{\text{root}}\|^2) + 0.3 \exp(-(\max(0, v^* - d_t^* \cdot \dot{x}_t^{\text{com}}))^2). \quad (12)$$

Here, $v^* = 1$ m/s specifies a minimum target speed at which the character should move towards the target, and the character will not be penalized for moving faster than this threshold. d_t^* is a unit vector on the horizontal plane that points from the character's root to the target.

$$r_t^G = \begin{cases} 1, & \text{target has been hit} \\ 0.3 r_t^{\text{near}} + 0.3, & \|x^* - x_t^{\text{root}}\| < 1.375m \\ 0.3 r_t^{\text{far}}, & \text{otherwise} \end{cases} \quad (18)$$

If the character is far from the target x^* , r_t^{far} encourages the character to move to the target using a similar reward function as the Target Location task (Equation 12). Once the character is within a given distance of the target, r_t^{near} encourages the character to strike the target with a particular end-effector,

$$r_t^{\text{near}} = 0.2 \exp(-2\|x^* - x_t^{\text{eff}}\|^2) + 0.8 \text{clip}\left(\frac{2}{3} d_t^* \cdot \dot{x}_t^{\text{eff}}, 0, 1\right),$$

策略网络MLP，鉴别器与价值函数也是MLP

6.2 Network Architecture

Each policy π is modeled by a neural network that maps a given state s_t and goal g to a Gaussian distribution over actions $\pi(a_t|s_t, g) = \mathcal{N}(\mu(s_t, g), \Sigma)$, with an input-dependent mean $\mu(s_t, g)$ and a fixed diagonal covariance matrix Σ . The mean is specified by a fully-connected network with two hidden layers, consisting of 1024 and 512 ReLU [Nair and Hinton 2010], followed by a linear output layer. The values of the covariance matrix $\Sigma = \text{diag}(\sigma_1, \sigma_2, \dots)$ are manually-specified and kept fixed over the course of training. The value function $V(s_t, g)$ and discriminator $D(s_t, s_{t+1})$ are modeled by separate networks with a similar architecture as the policy.

6.3 Training

Our policies are trained using a combination of GAIL [Ho and Ermon 2016] and proximal-policy optimization (PPO) [Schulman et al. 2017]. Algorithm 1 provides an overview of the training process. At each time step t , the agent receives a task-reward $r_t^G = r^G(s_t, a_t, s_{t+1}, g)$ from the environment, it then queries the motion prior for a style-reward $r_t^S = r^S(s_t, s_{t+1})$, computed according to Equation 7. The two rewards are combined according to Equation 4 to yield the reward for the particular timestep. Following the approach proposed by Peng et al. [2018a], we incorporate reference state initialization and early termination. Reference state initialization is applied by

ALGORITHM 1: Training with AMP

1: **input** \mathcal{M} : dataset of reference motions
2: $D \leftarrow$ initialize discriminator
3: $\pi \leftarrow$ initialize policy
4: $V \leftarrow$ initialize value function
5: $\mathcal{B} \leftarrow \emptyset$ initialize reply buffer

6: **while** not done **do**
7: **for** trajectory $i = 1, \dots, m$ **do**
8: $\tau^i \leftarrow \{(s_t, a_t, r_t^G)_{t=0}^{T-1}, s_T^G, g\}$ collect trajectory with π
9: **for** time step $t = 0, \dots, T - 1$ **do**
10: $d_t \leftarrow D(\Phi(s_t), \Phi(s_{t+1}))$
11: $r_t^S \leftarrow$ calculate style reward according to Equation 7 using d_t
12: $r_t \leftarrow w^G r_t^G + w^S r_t^S$
13: record r_t in τ^i
14: **end for**
15: store τ^i in \mathcal{B}
16: **end for**

17: **for** update step $= 1, \dots, n$ **do**
18: $b^{\mathcal{M}} \leftarrow$ sample batch of K transitions $\{(s_j, s'_j)\}_{j=1}^K$ from \mathcal{M}
19: $b^{\pi} \leftarrow$ sample batch of K transitions $\{(s_j, s'_j)\}_{j=1}^K$ from \mathcal{B}
20: update D according to Equation 8 using $b^{\mathcal{M}}$ and b^{π}
21: **end for**

22: update V and π using data from trajectories $\{\tau^i\}_{i=1}^m$
23: **end while**

算法流程

任务与平均奖励

Table 1. Performance statistics of combining AMP with additional task objectives. Performance is recorded as the average normalized task return, with 0 being the minimum possible return per episode and 1 being the maximum possible return. The return is averaged across 3 models initialized with different random seeds, with 32 episodes recorded per model. The motion prior can be trained with different datasets to produce policies that adopt distinct stylistic behaviors when performing a particular task.

Character	Task	Dataset	Task Return
Humanoid	Target Heading	Locomotion	0.90 ± 0.01
		Walk	0.46 ± 0.01
		Run	0.63 ± 0.01
		Stealthy	0.89 ± 0.02
		Zombie	0.94 ± 0.00
	Target Location	Locomotion	0.63 ± 0.01
		Zombie	0.50 ± 0.00
	Obstacles	Run + Leap + Roll	0.27 ± 0.10
	Stepping Stones	Cartwheel	0.43 ± 0.03
		Jump	0.56 ± 0.12
	Dribble	Locomotion	0.78 ± 0.05
		Zombie	0.60 ± 0.04
	Strike	Walk + Punch	0.73 ± 0.02
T-Rex	Target Location	Locomotion	0.36 ± 0.03

使用更大的数据集能自动切换步态实现更好地速度跟踪

Table 2. Summary statistics of the different datasets used to train the motion priors. We record the total length of motion clips in each dataset, along with the number of clips, and the number of subjects (e.g. human actors) that the clips were recorded from.

Character	Dataset	Size (s)	Clips	Subjects
Humanoid	Cartwheel	13.6	3	1
	Jump	28.6	10	4
	Locomotion	434.1	56	8
	Run	204.4	47	3
	Run + Leap + Roll	22.1	10	7
	Stealthy	136.5	3	1
	Walk	229.6	9	5
	Walk + Punch	247.8	15	9
	Zombie	18.3	1	1
T-Rex	Locomotion	10.5	5	1

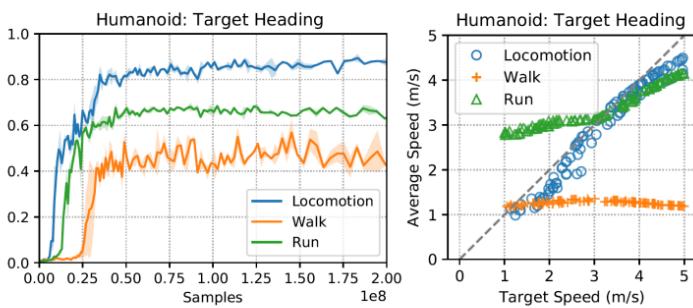


Fig. 4. Performance of Target Heading policies trained with different datasets. **Left:** Learning curves comparing the normalized task returns of policies trained with a large dataset of diverse locomotion clips to policies trained with only walking or running reference motions. Three models are trained using each dataset. **Right:** Comparison of the target speed with the average speed achieved by the different policies. Policies trained using the larger Locomotion dataset is able to more closely follow the various target speeds by imitating different gaits.

Vollenweider, Eric, Marko Bjelonic, Victor Klemm, Nikita Rudin, Joonho Lee, and Marco Hutter. "Advanced skills through multiple adversarial motion priors in reinforcement learning." In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5120-5126. IEEE, 2023.

AMP方法应用于真实四足机器人，实现四足、双足模式切换



Fig. 1. Quadruped-humanoid transformer (<https://youtu.be/kEdr0ARq48A>) with a time-lapse from left to right of a stand-up and sit-down motion (top image), obstacle negotiation (middle image), and indoor navigation (bottom images). The former skill and the humanoid navigation on two legs are achieved through traditional RL training with a task reward formulation. Instead of tuning the sit-down skill, we can reverse the playback of the stand-up motion and use it as a motion prior that helps the robot discover feasible sit-down behaviors avoiding tedious reward function tuning.

本文贡献：提出多AMP，实现多风格切换
三种数据集：
来自已有RL算法的、轨迹优化的、时间倒放的

B. Contribution

This paper introduces the Multi-AMP algorithm and applies it to our real wheeled-legged robot. Like its AMP predecessor [6], this approach automates the imitation objective and motion selection process without heuristics. Furthermore, our extension allows for the intentional switching of multiple different style objectives by changing flags in the command input. The approach can imitate motion priors from three different data sets, i.e., from existing RL controllers, trajectory optimization, and reverse stand-up motions. The latter enables the automatic discovery of feasible sit-down motions on the real robot without tedious reward function tuning. This permits exceptional skills with our wheeled-legged robot in Fig. 1, where the robot can switch between a quadruped and humanoid configuration. To the best of our knowledge, this is the first time such a highly dynamic skill is shown and also the first time that the AMP approach is verified on a real robot.

The training environment of our Multi-AMP pipelines is implemented using the *Isaac Gym simulator* [17], [18], which allows for massively parallel simulation. We spawn 4096 environments in parallel to learn all three tasks simultaneously in a single neural network. The number of environments per task is weighted according to their approximate difficulty, e.g., [1, 1, 5] in the case of the tasks described above (While important, the training is not very sensitive to this weighting). The state-transitions collected during the roll-outs of these environments are mapped using a function $\phi(s)$ such that it extracts the linear and angular base velocity, gravity direction in base frame, the base's height above ground, joint position and velocity, and finally the position of the wheels relative to the robot's base-frame, i.e., $\phi(s) = (\dot{x}_{base}, x_z, e_{base}, q, \dot{q}, x_{ee,base}) \in \mathbb{R}^{50}$. The task reward definitions for the three tasks are in Table I and II.

TABLE II

REWARDS FOR AOW STANDING UP, SITTING DOWN, AND NAVIGATING
WHILE STANDING

TABLE I

TASK-REWARDS.

	formula	weight
All tasks		
r_τ	$\ \tau\ ^2$	-0.0001
$r_{\dot{q}}$	$\ \dot{q}\ ^2$	-0.0001
$r_{\ddot{q}}$	$\ \ddot{q}\ ^2$	-0.0001
4-legged locomotion		
$r_{lin\ vel}$	$e^{\ \dot{x}_{target, xy} - \dot{x}\ ^2 / 0.25}$	1.5
$r_{ang\ vel}$	$e^{\ \omega_{target, z} - \omega\ ^2 / 0.25}$	1.5
Ducking		
r_{duck}	$e^{0.8 * x_{goal} - x }$	2
Stand-up	see Tab. II	

symbols	description	
$q^{robot} \in \mathbb{H}$	Robot base-frame rotation	
$p^{robot} \in \mathbb{R}^3$	Robot base-frame position	
q	Joint DOF positions (excl. wheels)	
q_{hl}	Hind-Leg DOF position	
α	$\angle(\text{robot-x axis, world z axis})$	
f	Feet on ground (binary)	
s	Standing robots (binary)	
stand-up	formula	weight
r_α	$\frac{\pi/2 - \alpha}{\pi/2}$	2
r_{height}	p_z^{robot}	3
r_{feet}	f	-2
r_{wheels}	$\sum \dot{q}_{front\ wheels}^2 * (1 - f)$	-0.003
$r_{shoulder}$	$\ q_{shoulder}\ ^2$	-1
$r_{stand\ pose}$	$\exp(-0.1 * \ q_{hl} - q_0, hl\ ^2)$	1
sit-down	formula	weight
$r_{un-stand}$	$\max(\frac{\pi/2 - \alpha}{\pi/2} * 3, 0)$	-3
$r_{sit-down}$	$\frac{\min(\alpha, \pi/2)}{\pi/2}$	2.65
$r_{dof\ vel}$	$\ \dot{q}\ ^2$	-0.015
$r_{dof\ pos}$	$\exp(-0.5 * \ q_0 - q\ ^2) * \frac{\alpha}{\pi/2}$	3
navigation	formula	weight
$r_{track\ lin}$	$\exp(-4 * \ \dot{x}_{des} + p_{local,z}^{robot}\ ^2) * s$	2
$r_{track\ ang}$	$\exp(-4 * \ \omega_{des} - \omega_{local,x}^{robot}\ ^2) * s$	2

We use an actuator model for the leg joints to bridge the sim-to-real gap [23] while an actuator model is not needed for the velocity controlled wheels. Moreover, we apply strategies to increase the policy’s robustness, such as rough terrain training (see rough terrain robustness in Fig. 1), random disturbances, and game inspired curriculum training [18]. The highly dynamic stand-up task is especially prone to these robustness measures, which we solve by introducing timed pushes and joint-velocity-based trajectory termination. The former identifies the most critical movement-phase and pushes the policy in the worst way. This increases the number of disturbances the policy experiences during these critical phases, rendering it more robust, which also helps with sim-to-real efforts. Furthermore, by terminating the trajectory if the joint velocity of any DOF exceeds the actuator’s limits, the policy learns to keep a safety tolerance to these limits.

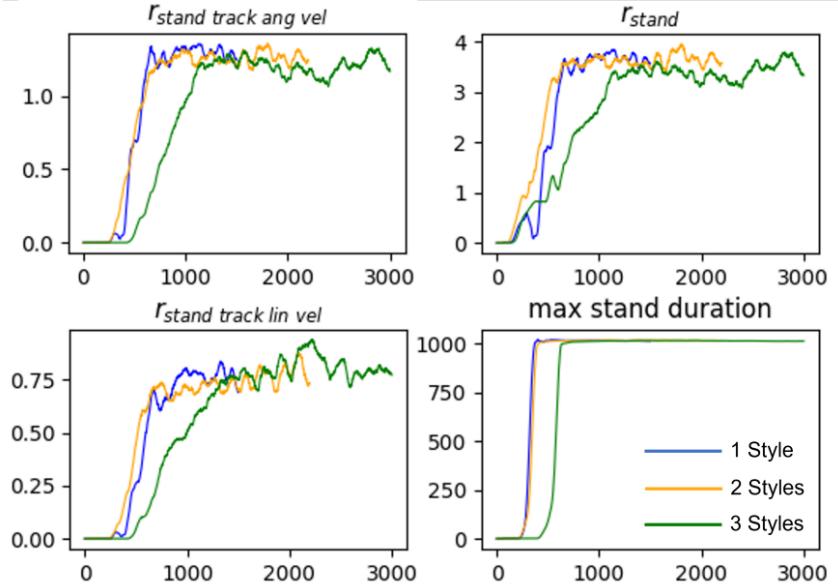


Fig. 6. Multi-AMP learning capability of the stand-up task. The horizontal axis denotes the number of epochs, and the vertical axis represents the rewards after post-processing for comparability. Furthermore, the maximum stand duration is plotted over the number of epochs.

Wu, Jinze, Guiyang Xin, Chenkun Qi, and Yufei Xue. "Learning robust and agile legged locomotion using adversarial motion priors." *IEEE Robotics and Automation Letters* (2023).

AMP与teacher-student方法结合，实现复杂地形运动



Fig. 1. Our robot can achieve robust locomotion over challenging terrains and agile locomotion over natural terrains. The robot can successfully traverse curbs, stairs, rocky and vegetation while running and spinning at high speed over natural terrains. Whether on even or complex terrains, at high or low speeds, the emergent behaviors learned by our robot exhibit a natural gait and smooth motion.

鉴别器训练数据集来自单刚体轨迹优化 (TOWR) , 只包含平地运动数据
学生策略采用LSTM代替ETH的TCN

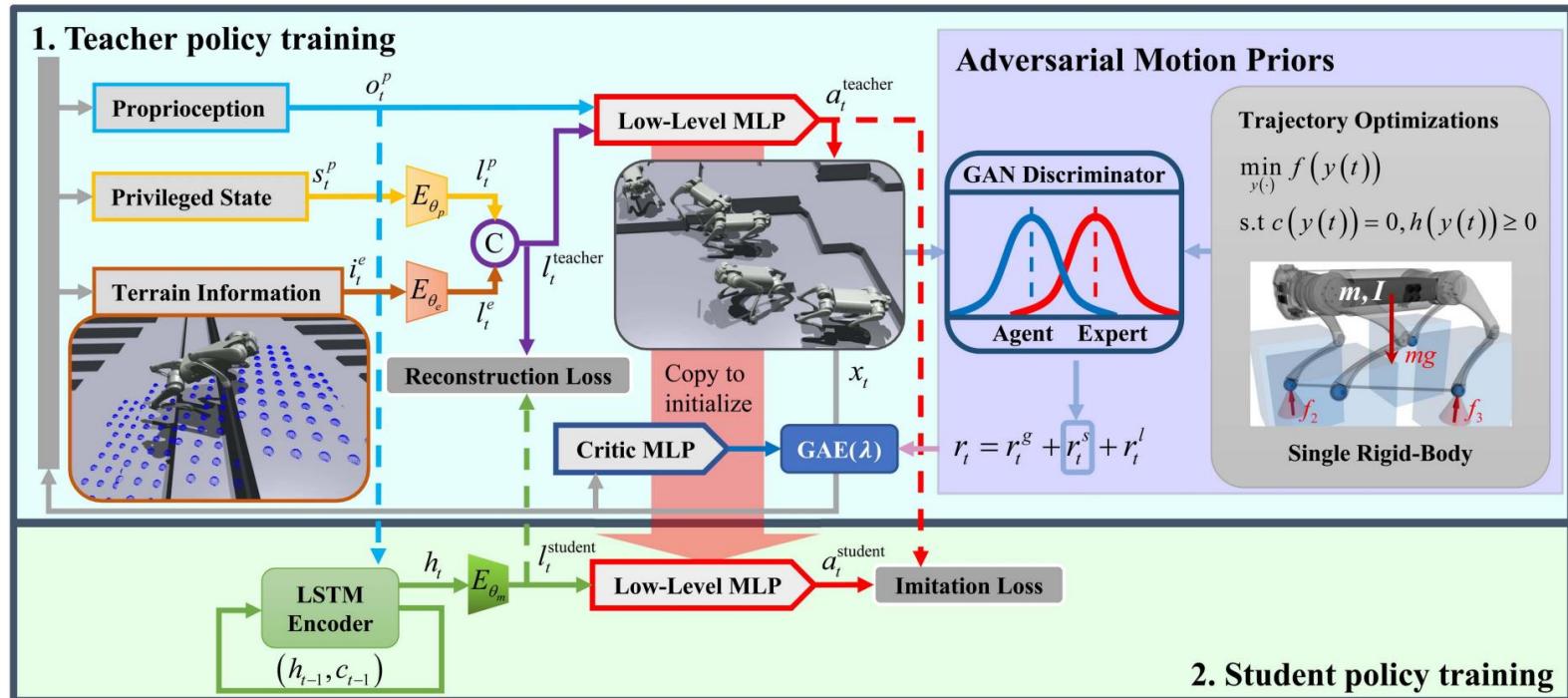


Fig. 2. Overview of the training methods. We first train a teacher policy with access to proprioceptive observation o_t^p , privileged state s_t^p and terrain information i_t^e using RL. The total reward r_t obtained by the teacher consists of a task reward r_t^g , a style reward r_t^s based on AMP, and a regularization reward r_t^l which imposes constraints on the motion. A student policy is then trained by a supervised fashion, where the student imitates the teacher's action a_t^{teacher} and **reconstructs the latent representation l_t^{teacher}** using only proprioceptive observation o_t^p . We reuse the learned weights of the teacher's low-level net to initialize the student's low-level net to speed up training. The motion priors are generated from the TO, using a single rigid-body model.

Simulation: We trained 4096 parallel agents on different types of terrains using the IsaacGym simulator [9]. The teacher policy and the student policy were trained with 400 and 200 million simulated time steps, respectively. The overall training time for the two stages was seven hours of wall-clock time. Each RL episode lasts for a maximum of 1000 steps, equivalent to 20 s, and terminates early if it reaches the termination criteria. The control frequency of the policy is 50 Hz in the simulation. All trainings were performed on a single NVIDIA RTX 3090Ti GPU.

Termination: We terminate an episode and start the next one when the robot reaches the termination criteria, which include trunk collisions with the ground, large body inclinations, and being trapped for a long period of time.

Dynamics Randomization: To improve the robustness of our policy and facilitate transfer from simulation to the real world, we randomize the mass of the trunk and legs, the mass and position of payload applied to the body of the robot, the ground friction and restitution coefficients, the motor strength, the joint-level PD gains, and the initial joint positions in each episode. Some of these dynamic parameters are considered as privileged state s_t^p to aid the teacher policy training. In addition, the same observation noise as in [9] is added during the training phase in the simulation. The randomization ranges for each parameter are detailed in Table II.

Due to the instability of RL in the early stage, it is difficult to train robots directly on very complex terrains. We adopt and improve the automated terrain curriculum in [9]. We create a height-field map with 100 terrains arranged in a 20×10 grid. Each row has the same type of terrain arranged in increasing difficulty, while each terrain has a length and width of 8 m. The rough flats are constructed by adding noise increased from ± 1 cm to ± 8 cm. The inclination of the slopes increases from 0 deg to 30 deg. The waves are constructed by three sine waves across the terrain length. The amplitude of these waves increases from 20 cm to 50 cm. The stairs have a fixed width of 30 cm and a step height increased from 5 cm to 23 cm. The discrete obstacles only have two height levels increased from ± 5 cm to ± 15 cm. At the beginning of training, all robots are equally assigned to all terrain types with the lowest difficulty. The robot is only moved to a more difficult terrain once it has adapted to its current terrain difficulty. This adaptation is obtained when the robot can step out of the current terrains with more than 85% of the average linear velocity tracking reward. In contrast, they are reset to easier terrains if they fail to travel at least half of the distance required by their command linear velocity at the end of an episode. To avoid skills forgetting, robots solving the hardest terrains are looped back to a randomly selected difficulty of current terrain type.



提纲

- 一、ETH三部曲**
- 二、AMP三部曲**
- 三、Pakour三部曲**
- 四、无人机三部曲**

2023

ANYmal Parkour: Learning Agile Navigation for Quadrupedal Robots

David Hoeller

ETH Zurich

NVIDIA

dhoeller@ethz.ch

Equal contribution

Nikita Rudin

ETH Zurich

NVIDIA

rudinn@ethz.ch

Equal contribution

Dhionis Sako

ETH Zurich

dsako@ethz.ch

Marco Hutter

ETH Zurich

mahutter@ethz.ch

Robot Parkour Learning

Ziwen Zhuang^{*13} Zipeng Fu^{*2} Jianren Wang⁴ Christopher Atkeson⁴ Sören Schwertfeger³
Chelsea Finn² Hang Zhao¹⁵

¹Shanghai Qi Zhi, ²Stanford, ³ShanghaiTech, ⁴CMU, ⁵Tsinghua, ^{*}project co-leads

Extreme Parkour with Legged Robots

Xuxin Cheng*

Kexin Shi*

Ananye Agarwal

Deepak Pathak

Carnegie Mellon University

Hoeller, David, Nikita Rudin, Dhionis Sako, and Marco Hutter. "ANYmal Parkour: Learning Agile Navigation for Quadrupedal Robots." *arXiv preprint arXiv:2306.14874* (2023).



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

三个模块，8个网络：

1. 感知模块生成隐向量和地图

2. 运动模块包含多个底层动作

3. 导航模块规划路径选择动作

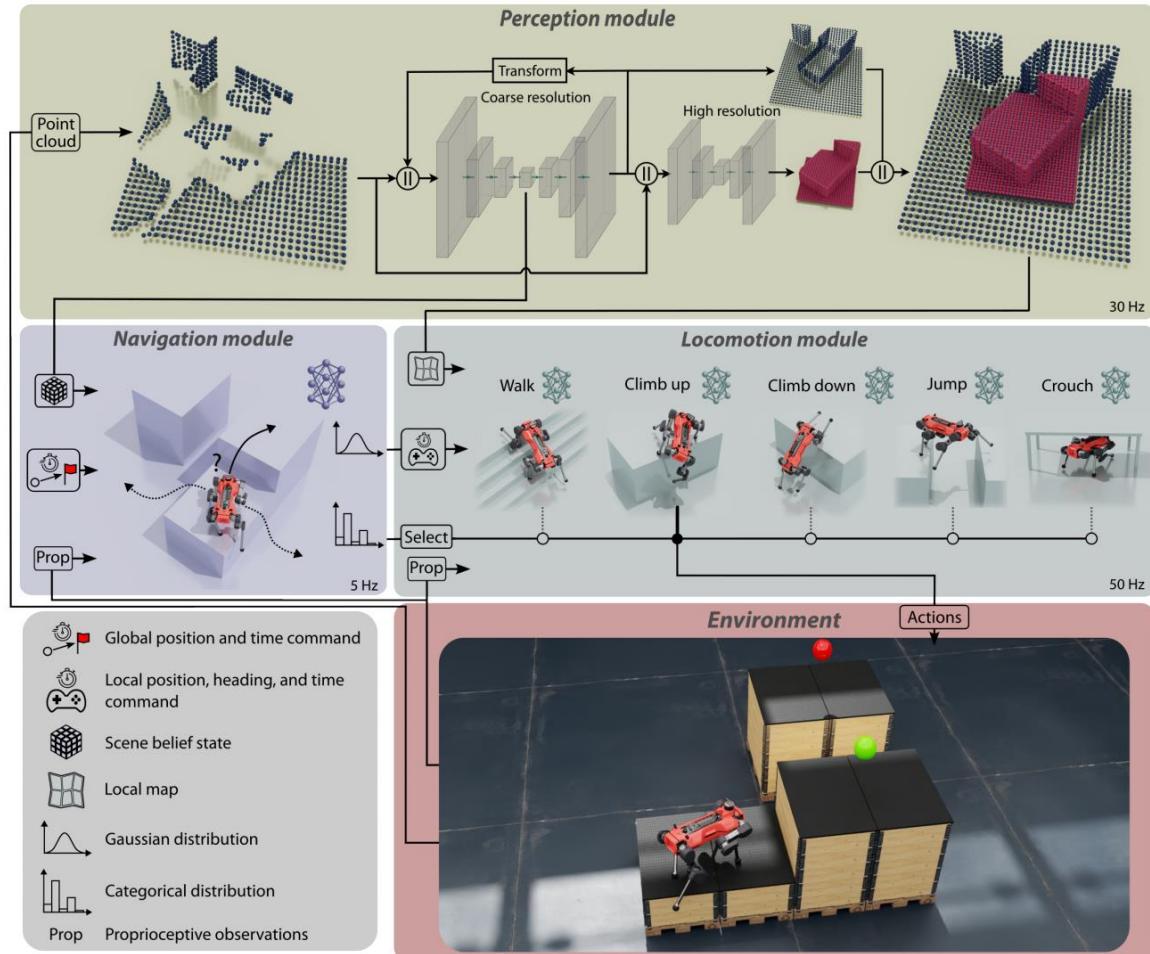


Fig. 2: Description of our approach. We decompose the problem into three components: The perception module receives the point cloud measurements to estimate the scene's layout and produces a latent tensor and a map. The locomotion module contains several low-level skills that can overcome specific scenarios.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

自主越障，随机摆放的障碍物 自主选择步态

We deploy the pipeline on the quadrupedal robot ANYmal D. It weighs around 55 kg and has 12 series elastic actuators capable of producing a torque of 85 N m each. To

perceive the environment, it is equipped with a total of six Intel Realsense depth cameras (two in the front, two in the back, one left, one right), and a Velodyne Puck LiDAR. The whole system is implemented in several ROS nodes across different onboard computers. The locomotion and navigation modules operate synchronously in a single node on the onboard computer. The perception module is implemented on an NVIDIA Jetson Orin and operates asynchronously with the rest of the system, i.e., the navigation and locomotion policies take the last received message from the perception module to infer their respective networks. The supplementary video summarizes the proposed approach and shows indoor and outdoor experiments on the real robot.

The three learning-based modules operate together without expert demonstration, offline computation, or a priori knowledge of the environment and enable the robot to reliably reach a target across different arrangements of randomized obstacles.

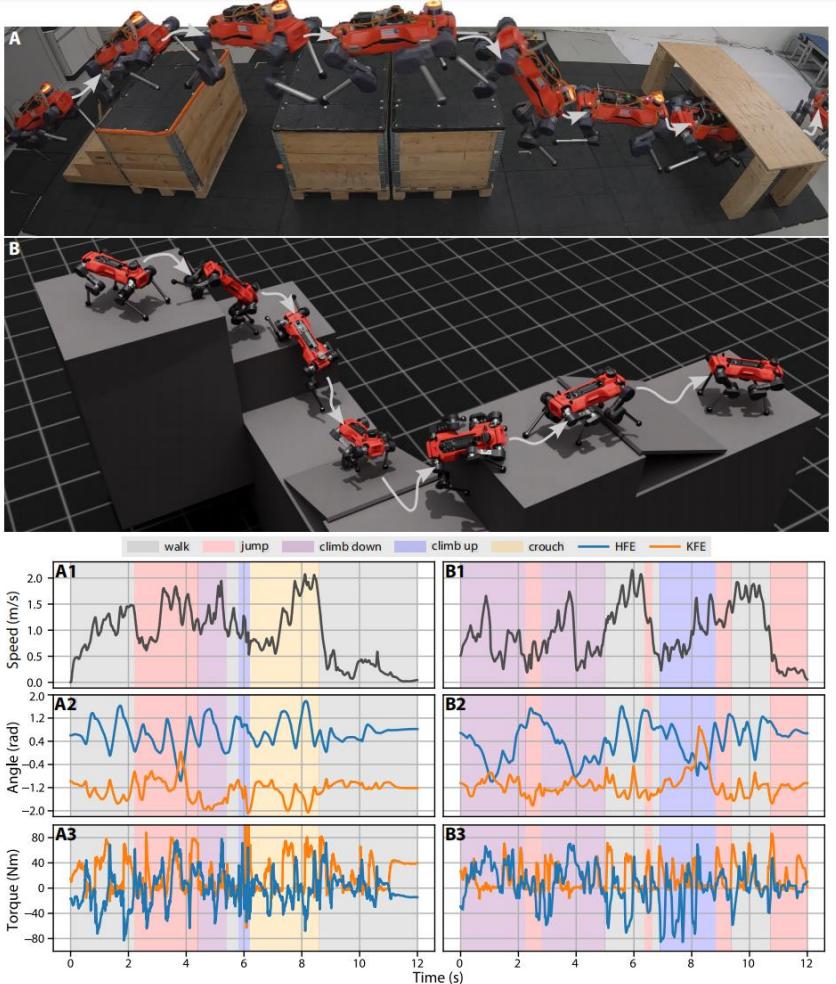


Fig. 3: Deployment of the pipeline on the robot ANYmal D. (A) Trajectory on the real robot. (B) Trajectory in simulation. (A1)-(A3) and (B1)-(B3) depict the profiles of the robot's speed, the selected skills, and two joint angles and torques corresponding to (A) and (B), respectively. The system leverages the motor's full torque capabilities and uses large deflections of the joints to reach high speeds and overcome challenging obstacles.

运动模块

五种基本动作

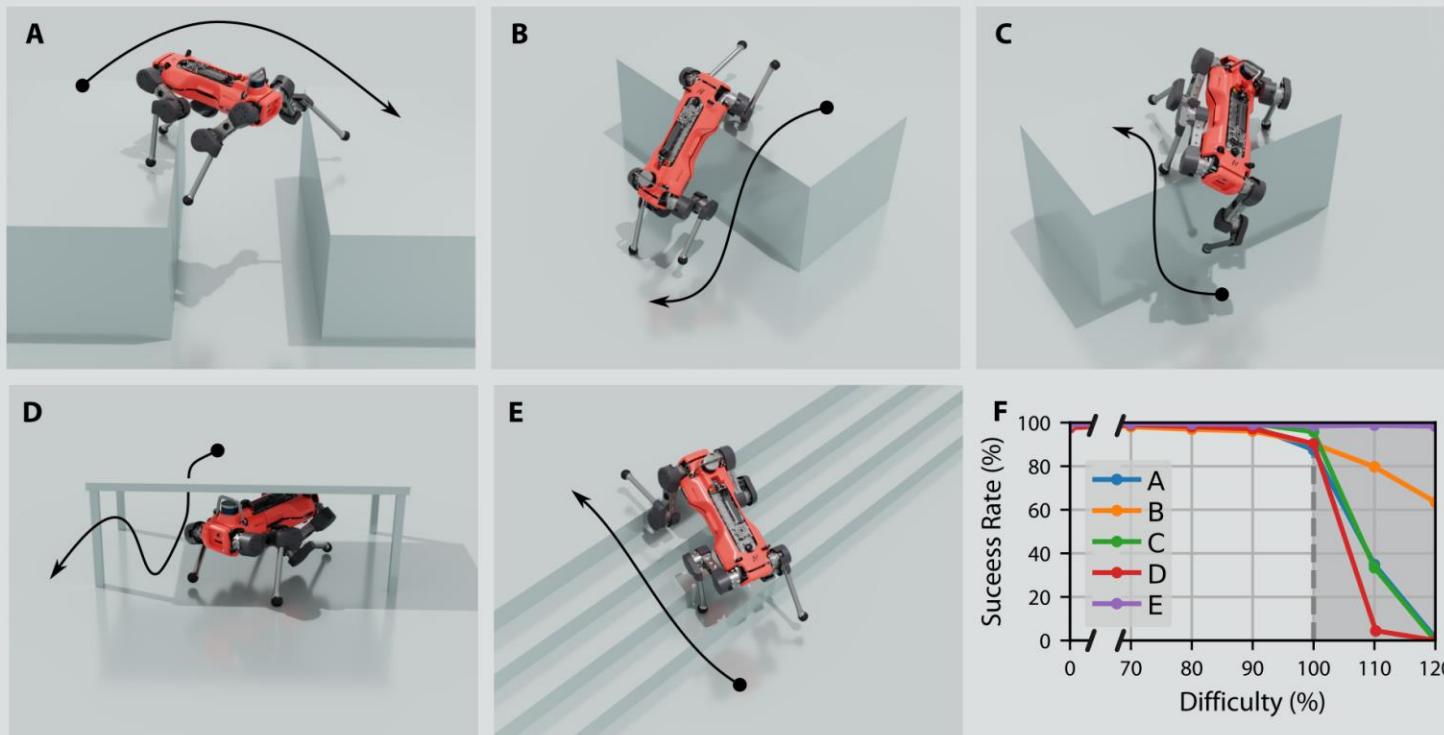


Fig. 4: Training scenarios of the locomotion skills with the resulting behaviors. (A) Jumping. (B) Climbing down. (C) Climbing up. (D) Crouching. (E) Walking. (F) Success rate of each skill for obstacles of varying difficulty. (G) Ranges of parameters used during training (0% to 100% in F).

自主路径选择

改变箱子高度

两次走不同路径

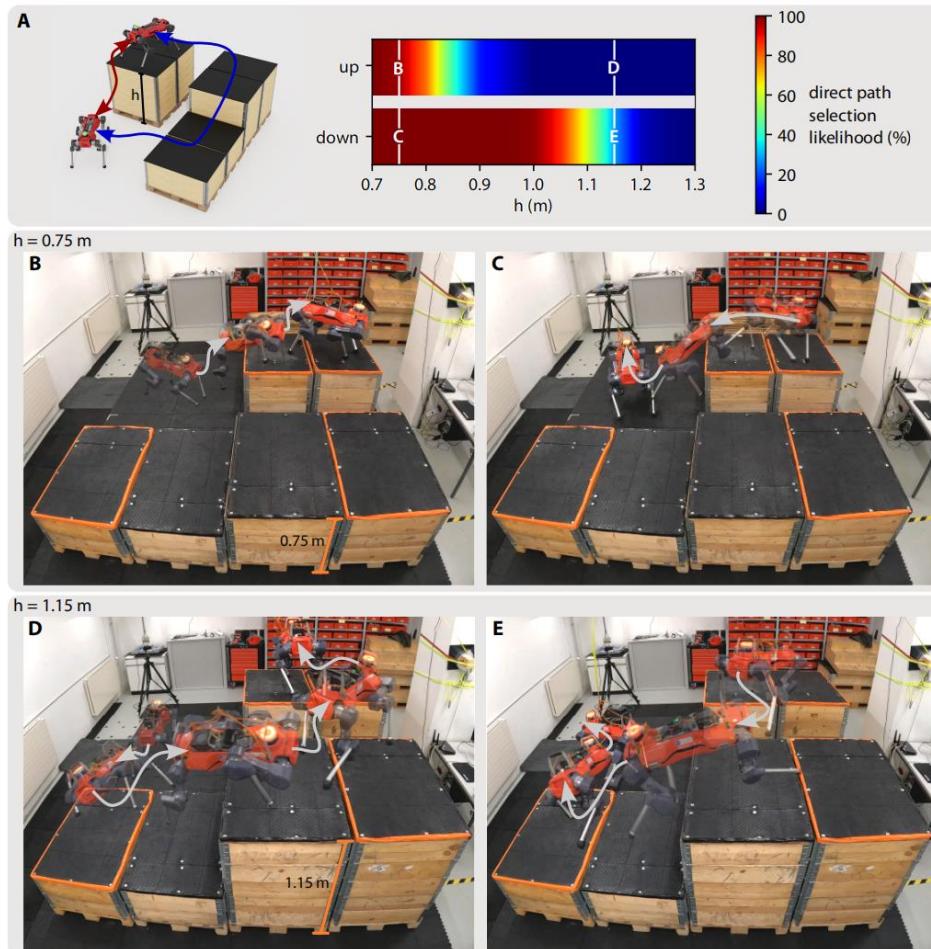


Fig. 5: Adaptive path selection. The robot starts on the ground and is given a target on top of the box in the back, and then commanded back to the initial position. (A) Likelihood of going up and down along the direct path (red line) as a function of the height of the box. (B) and (C) Deployment on the robot for $h = 0.75\text{ m}$. (D) and (E) Deployment on the robot for $h = 1.15\text{ m}$. For the same targets and box placement, the navigation policy chooses a different path depending on the height of the boxes to reach the goal.

训练场景

尺寸、布局随机化

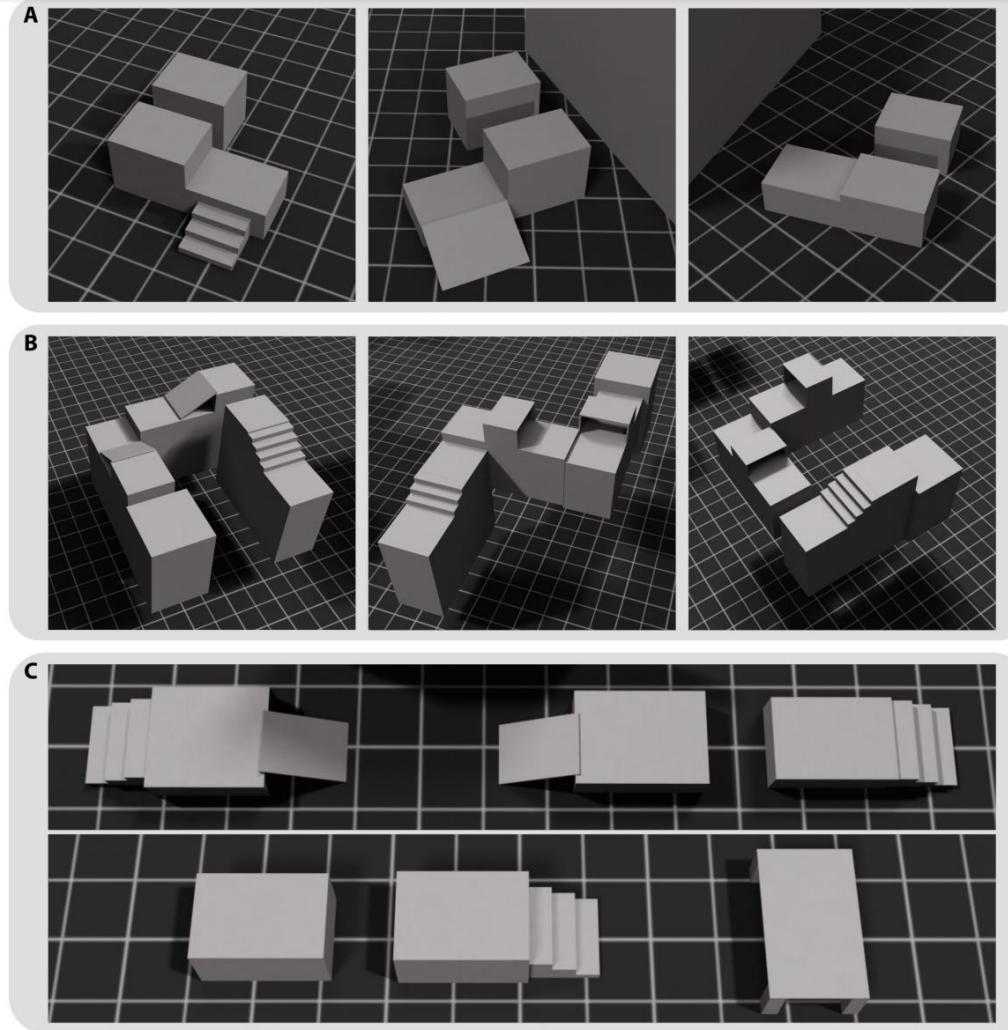


Fig. 7: Types of environments used for training. The dimensions of the individual obstacles and the arrangements are randomized.

Zhuang, Ziwen, Zipeng Fu, Jianren Wang, Christopher Atkeson, Soeren Schwertfeger, Chelsea Finn, and Hang Zhao. "Robot parkour learning." *arXiv preprint arXiv:2309.05665* (2023).

单个网络

两阶段训练法



Figure 1: We present a framework for learning parkour skills on low-cost robots. Our end-to-end vision-based parkour learning system enable the robot to climb high obstacles, leap over large gaps, crawl beneath low barriers, squeeze through thin slits and run. Videos are on the [project website](#).

- **a two-stage RL method** for overcoming difficult exploration problems, involving a pre-training stage with soft dynamics constraints and a fine-tuning stage with hard dynamics constraints;

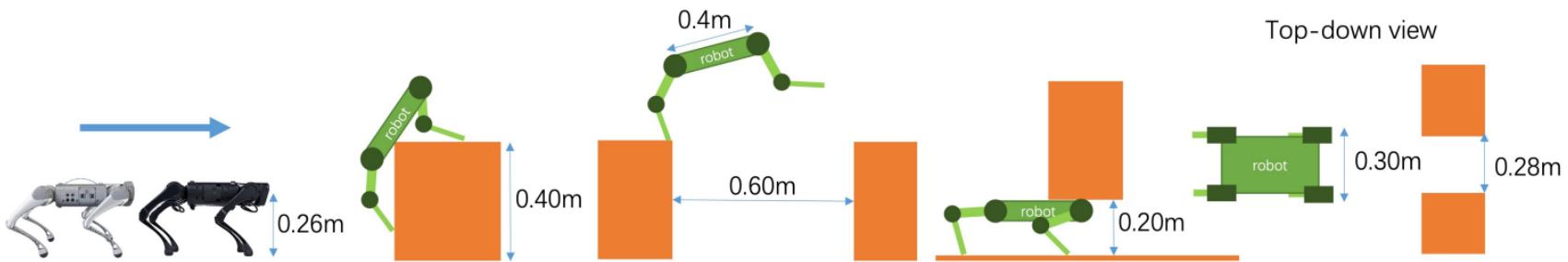


Figure 2: We illustrate the challenging obstacles that our system can solve, including climbing high obstacles of 0.40m (1.53x robot height), leap over large gaps of 0.60m (1.5x robot length), crawling beneath low barriers of 0.20m (0.76x robot height), squeezing through thin slits of 0.28m by tilting (less than the robot width).

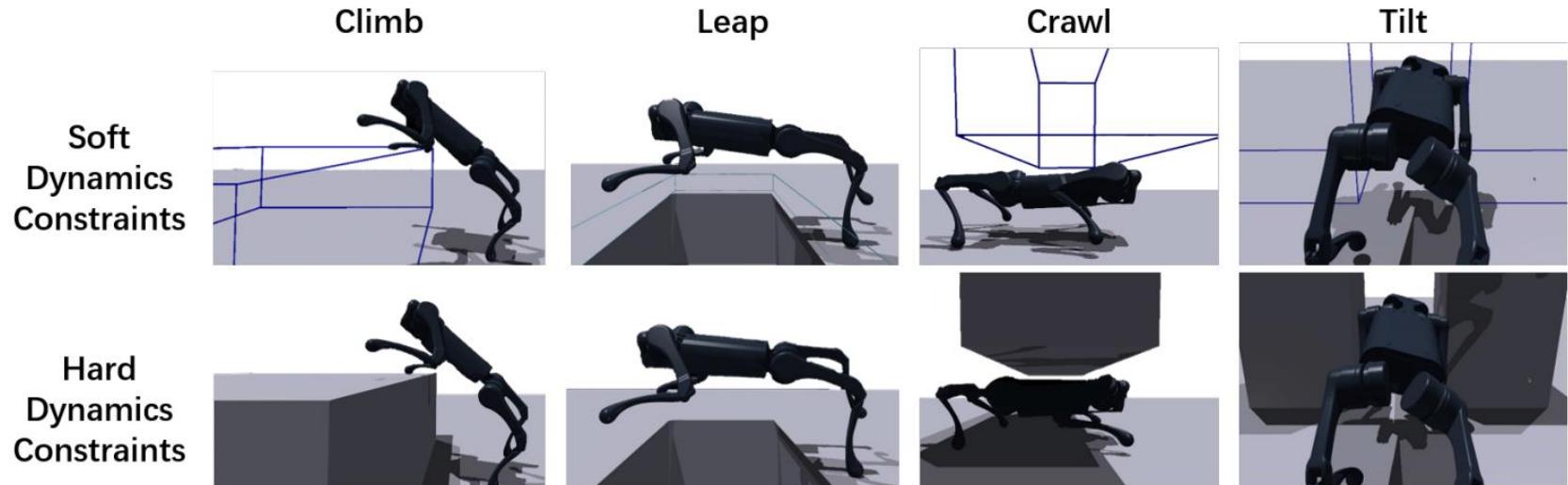


Figure 3: Soft dynamics constraints and hard dynamics constraints for each skill. Given soft dynamics constraints, the obstacles are penetrable.

3.1 Parkour Skills Learning via Two-Stage RL

Since depth images are costly to render, and directly training RL on visual data is not always stable, we use privileged visual information about the environments to help RL to generate specialized parkour skills in simulation. The privileged visual information includes the distance from the robot's current position to the obstacle in front of the robot, the height of the obstacle, the width of the obstacle, and a 4-dimensional one-hot category representing the four types of obstacles. We formulate each specialized

skill policy as a [gated recurrent neural network \(GRU \[114\]\)](#). The inputs to a policy other than the recurrent latent state are proprioception $s_t^{\text{proprio}} \in \mathbb{R}^{29}$ (row, pitch, base angular velocities, positions and velocities of joints), last action $a_{t-1} \in \mathbb{R}^{12}$, the privileged visual information e_t^{vis} , and the privileged physics information e_t^{phy} . We use a similar approach to prior work [8, 10] to sample physics properties like terrain friction, center of mass of the robot base, motor strength and etc to enable domain adaptation from simulation to the real world. The policy outputs the target joint positions $a_t \in \mathbb{R}^{12}$.

We train all the specialized skill policies $\pi_{\text{climb}}, \pi_{\text{leap}}, \pi_{\text{crawl}}, \pi_{\text{tilt}}, \pi_{\text{run}}$ separately on corresponding terrains shown in Figure 3 using the same reward structure. We use the formulation of minimizing mechanical energy in [35] to derive a general skill reward r_{skill} suitable for generating all skills with natural motions, which only consists of three parts, a forward reward r_{forward} , an energy reward r_{energy} and an alive bonus r_{alive} :

$$r_{\text{skill}} = r_{\text{forward}} + r_{\text{energy}} + r_{\text{alive}},$$

where

$$r_{\text{forward}} = -\alpha_1 * |v_x - v_x^{\text{target}}| - \alpha_2 * |v_y|^2 + \alpha_3 * e^{-|\omega_{\text{yaw}}|},$$

$$r_{\text{energy}} = -\alpha_4 * \sum_{j \in \text{joints}} |\tau_j \dot{q}_j|^2, \quad r_{\text{alive}} = 2.$$

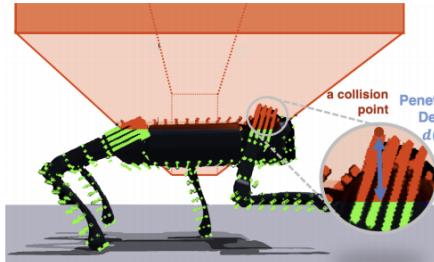


Figure 4: We show collisions points on the robot. Collision points that penetrate obstacles are in red.

	Success Rate (%) ↑					Average Distance (m) ↑				
	Climb	Leap	Crawl	Tilt	Run	Climb	Leap	Crawl	Tilt	Run
Blind	0	0	13	0	100	1.53	1.86	2.01	1.62	3.6
MLP	0	1	63	43	100	1.59	1.74	3.27	2.31	3.6
No Distill	0	0	73	0	100	1.57	1.75	2.76	1.86	3.6
RMA [8]	-	-	-	74	-	-	-	-	2.7	-
Ours (parkour policy)	86	80	100	73	100	2.37	3.05	3.6	2.68	3.6
Oracles w/o Soft Dyn	0	0	93	86	100	1.54	1.73	3.58	1.73	3.6
Oracles	95	82	100	100	100	3.60	3.59	3.6	2.78	3.6

Table 2: We test our method against several baselines and ablations in the simulation with a max distance of 3.6m. We measure the success rates and average distances of every skill averaged across 100 trials and 3 random seeds. Our parkour policy shows the best performance using only sensors that are available in the real world. We evaluate on the test environments with obstacles proprieties that are more difficult than the ones of training environments shown in Table 1.

Measured at every time step, v_x is the forward base linear velocity, v_x^{target} is the target speed, v_y is the lateral base linear velocity, ω_{yaw} is the base angular yaw velocity, τ_j is the torque at joint j , ω_{yaw} is the joint velocity at joint j , and α are hyperparameters. We set the target speed for all skills to around 1 m/s. We use the second power of motor power at each joint to reduce both the average and the variance of motor power across all joints. See the supplementary for all hyperparameters.

预训练阶段：软约束，障碍物可穿透，使用穿透奖励训练

Denote a collision point on the collision bodies as p , an indicator function of whether p violates the soft dynamics constraints as $\mathbb{1}[p]$, and the distance of p to the penetrated obstacle surface as $d(p)$. The volume of penetration can be approximated by the sum of $\mathbb{1}[p]$ over all the collision points, and the average depth of penetration can be approximated by the sum of $d(p)$. In Figure 4, the collisions points violating the soft dynamics constraints ($\mathbb{1}[p] = 1$) are in red, and those with $\mathbb{1}[p] = 0$ are in green. Concretely, the penetration reward is

$$r_{\text{penetrate}} = - \sum_p (\alpha_5 * \mathbb{1}[p] + \alpha_6 * d(p)) * v_x,$$

微调阶段：硬约束，障碍物不可穿透

RL Fine-tuning with Hard Dynamics Constraints. After the pre-training stage of RL is near convergence, we fine-tune every specialized parkour skill policy on the realistic hard dynamics constraints (shown in Figure 3); hence, no penetrations between the robots and obstacles are possible at the second stage of RL. We use PPO to fine-tune the specialized skills using only the general skill reward r_{skill} . We randomly sample obstacle properties from the ranges listed in Table 1 during fine-tuning. Since the running skill is trained on terrains without obstacles, we directly train the running skill with hard dynamics constraints and skip the RL pre-training stage with soft dynamics constraints.

蒸馏：消除特权信息依赖，五个独立策略合为一个

3.2 Learning a Single Parkour Policy by Distillation

The learned specialized parkour skills are five policies that use both the privileged visual information e_t^{vis} , and the privileged physics information e_t^{phy} . However, the ground-truth privilege information is only available in the simulation but not in the real world. Furthermore, each specialized policy can only execute one skill and cannot autonomously execute and switch between different parkour skills based on visual perception of the environments. We propose to use DAgger [44, 45] to distill a single vision-based parkour policy π_{parkour} using only onboard sensing from the five specialized skill policies $\pi_{\text{climb}}, \pi_{\text{leap}}, \pi_{\text{crawl}}, \pi_{\text{tilt}}, \pi_{\text{run}}$. We randomly sample obstacles types and properties from Table 1 to form a simulation terrain consisting of 40 tracks and 20 obstacles on each track. Since we have full knowledge of the type of obstacle related to every state s_t , we can assign the corresponding specialized skill policy $\pi_{s_t}^{\text{specialized}}$ to teach the parkour policy how to act at a state. For example, we assign the climb policy π_{climb} to supervise the parkour policy given a high obstacle. We parameterize the policy as a GRU. The inputs except the recurrent latent state are the proprioception s_t^{proprio} , the previous action a_{t-1} and a latent embedding of the depth image I_t^{depth} processed by a small CNN. The distillation objective is

$$\arg \min_{\theta_{\text{parkour}}} \mathbb{E}_{s_t, a_t \sim \pi_{\text{parkour}}, \text{sim}} \left[D \left(\pi_{\text{parkour}} \left(s_t^{\text{proprio}}, a_{t-1}, I_t^{\text{depth}} \right), \pi_{s_t}^{\text{specialized}} \left(s_t^{\text{proprio}}, a_{t-1}, e_t^{\text{vis}}, e_t^{\text{phy}} \right) \right) \right],$$

where θ_{parkour} are the network parameters of the parkour policy, sim is the simulator with hard dynamics constraints, and D is the divergence function which is binary cross entropy loss for policy networks with tanh as the last layer. Both policies π_{parkour} and $\pi_{s_t}^{\text{specialized}}$ are stateful. More details of the parkour policy network are in the supplementary.

迁移:

视觉gap

3.3 Sim-to-Real and Deployment

Although the distillation training in Section 3.2 can bridge the sim-to-real gap in physical dynamics properties such as terrain friction and mass properties of the robot [8, 10], we still need to address the **sim-to-real gap in visual appearance** between the rendered depth image in simulation and the onboard depth image taken by a depth camera in the real world. Shown in Figure 5, we apply pre-processing techniques to both the raw rendered depth image and the raw real-world depth image. We apply depth clipping, pixel-level Gaussian noise, and random artifacts to the raw rendered depth image, and apply depth clipping, hole filing, spatial smoothing and temporal smoothing to the raw real-world depth image.

The depth images in both simulation and real-world have a resolution of 48 * 64. Due to the limited onboard computation power, the refresh rate of onboard depth image is 10Hz. Our parkour policy operates at 50Hz in both simulation and the real world to enable agile locomotion skills, and asynchronously fetches the latest latent embedding of the depth image processed by a small CNN. The output actions of the policy are target joint positions which are converted to torques on the order of 1000Hz through a PD controller of $K_p = 50$ and $K_d = 1$. To ensure safe deployment, we apply a torque limits of 25Nm by clipping target joint positions: $\text{clip}(q^{\text{target}}, (K_d * \dot{q} - 25)/K_p + q, (K_d * \dot{q} + 25)/K_p + q)$.

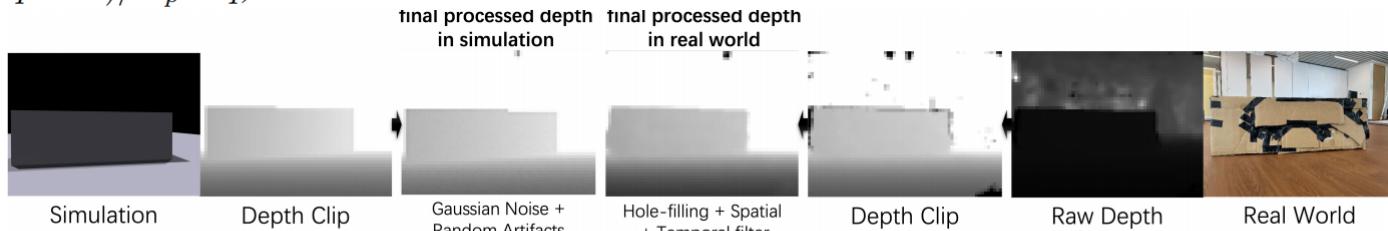


Figure 5: We bridge the visual gap between simulation and real world by applying pre-processing techniques. We use depth clipping, Gaussian noise and random artifacts in simulation, and depth clipping and hole-filling, spatial and temporal filters in the real world.

4 Experimental Results

Robot and Simulation Setup. We use IsaacGym [119] as the simulator to train all the policies. To train the specialized parkour skills, we construct large simulated environments consisting of 40 tracks and 20 obstacles on each track. The obstacles in each track have linearly increasing difficulties based on the obstacle property ranges in Table 1. We use a Unitree A1 and a Unitree Go1 that are equipped with Nvidia Jetson NX for onboard computation and Intel RealSense D435 for onboard visual sensing. More details are in the supplementary.

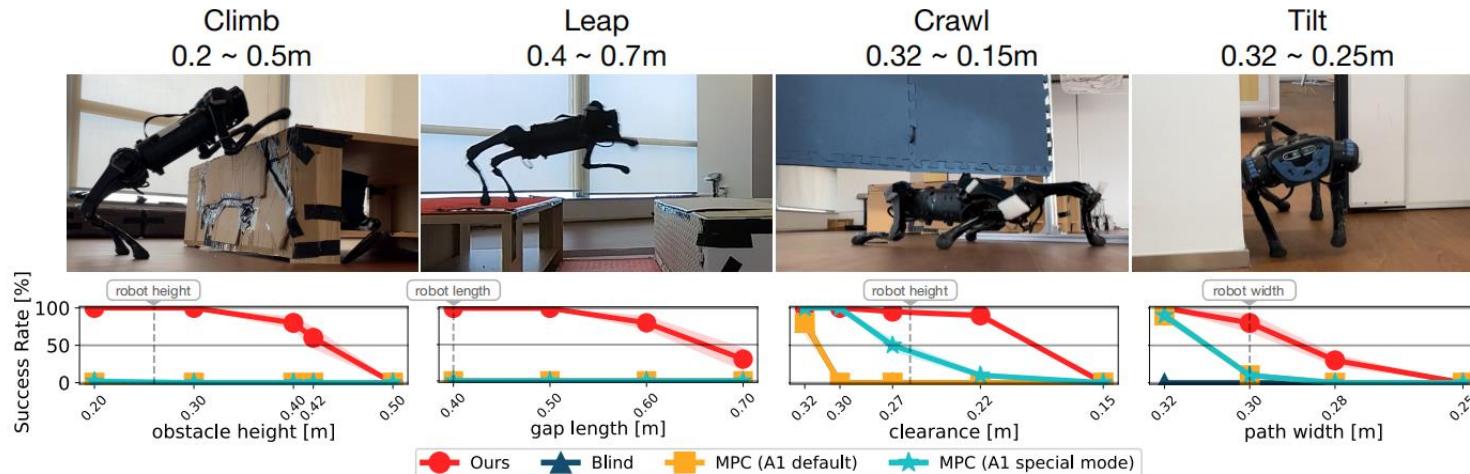


Figure 6: Real-world indoor quantitative experiments. Our parkour policy can achieve the best performance, compared with a blind policy and built-in MPC controllers. We control the MPC in A1 special mode by teleoperating the robot lower down or tilt the body during crawling and tilt respectively.

Vision is crucial for learning parkour. We compare the Blind baseline with our approach. Shown in Table 2, without depth sensing and relying only on proprioception, the distilled blind policy cannot complete any climbing, leaping or tilting trials and can only achieve a 13% success rate on crawling. This is expected, as vision enables sensing of the obstacle properties and prepares the robot for execute agile skills while approaching the obstacles.

RL pre-training with soft dynamics constraints enables

parkour skills’ learning. We compare the RND, Oracles w/o Soft Dyn and ours (Oracles w/ Soft Dyn), all trained using privileged information without the distillation stage. We aim to verify that our method of RL pre-training with soft dynamics constraints can perform efficient exploration.

In Figure 7, we measure the average success rates of each method averaged over 100 trials across all the parkour skills that require exploration including climbing, leaping, crawling and tilting. We trained using three random seeds for each method to measure the standard deviations. Our method using RL pre-training with soft dynamics constraints can achieve much faster learning progress and a better final success rate around 95%. We notice that RND struggles to learn meaningful behaviors with scenarios that require fine-grained maneuvres such as crawling through a thin slit, due to its tendency to reach states where future states are difficult to predict. Both RND and Oracles w/o Soft Dyn cannot make any learning progress on climbing and leaping, the two most difficult parkour skills. More plots showing the success rates for each skill separately are in the supplementary.

Recurrent networks enable parkour skills requiring memories.

We compare the MLP baseline with ours using a GRU to parameterize the vision-based parkour policy. Shown in Table 2, the MLP baseline cannot learn the climbing and leaping skills and achieve much lower performance on crawling and tilting. Both climbing and leaping requires the robot to hold a short-term memory of the past visual perceptions. For example, during climbing when the robot has its front legs on the obstacles, it still needs memory about the spatial dimensions of the obstacle captured in past depth images to control the rear legs to complete the climbing.

Distillation is effective for Sim2Real.

We compare the RMA baseline and the No Distill baseline with ours. Although RMA can achieve similar performance on one skill that it is trained on, i.e. tilting, RMA fixes the network parameters of the MLP which processes the latent embeddings of the backbone GRU, and directly copies them from the specialized skill to the distilled policy. Consequently, it cannot distill multiple specialized skill policies, which have different MLP parameters, into one parkour policy. No Distill cannot learn climbing, leaping and tilting due to the complexity of training directly from visual observations without privileged information.

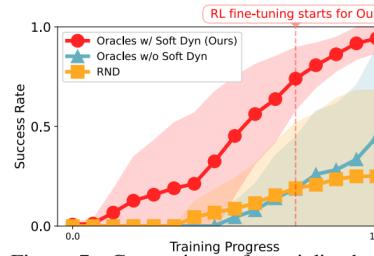


Figure 7: Comparison of specialized oracles trained with soft dynamics constraints with baselines averaged across every skill and three trials.

Cheng, Xuxin, Kexin Shi, Ananye Agarwal, and Deepak Pathak. "Extreme parkour with legged robots." *arXiv preprint arXiv:2309.14341* (2023).

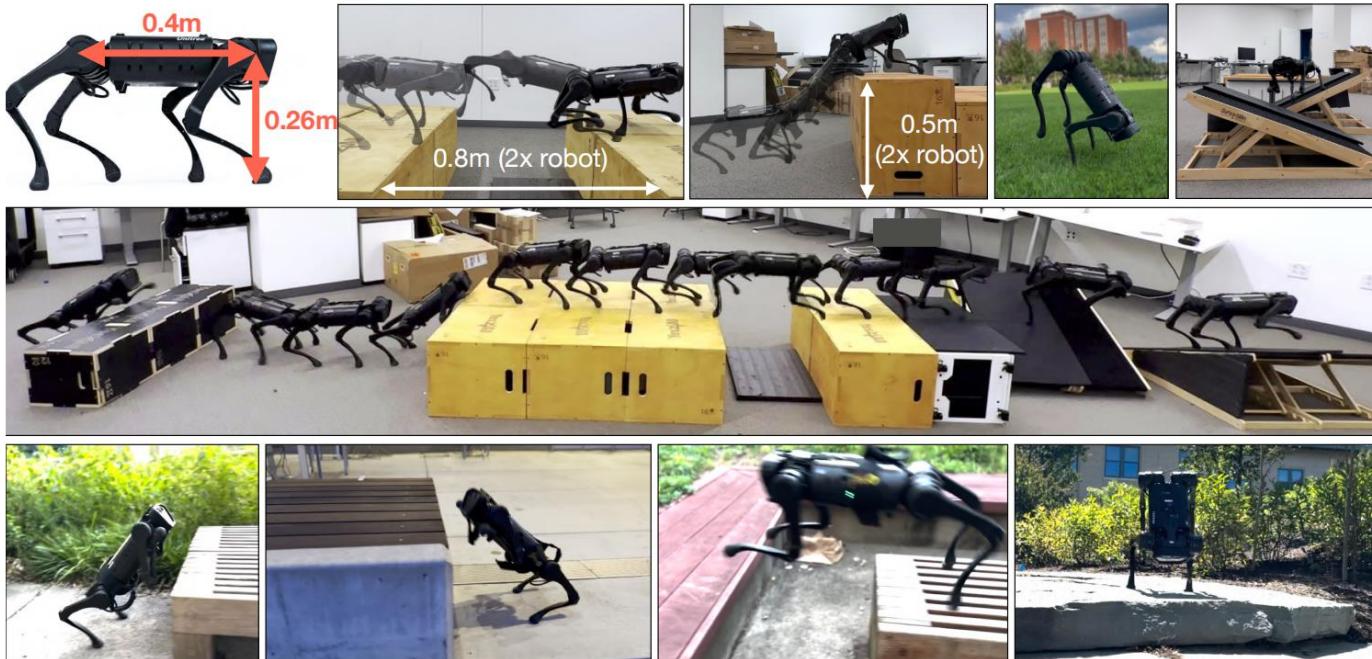


Figure 1: Extreme Parkour: Low-cost robot with imprecise actuation can perform precise athletic behaviors directly from a high-dimensional image without any explicit mapping and planning. The robot is able to long jump across gaps $2\times$ of its own length, high jump over obstacles $2\times$ its own height, run over tilted ramps, and walk on just front two legs (handstand) – all with *a single neural network operating directly on depth from a single, front-facing camera*. Parkour videos at <https://extreme-parkour.github.io/>.

of these papers, we propose a conceptually simple framework that results in more extreme parkour behaviors. The simplicity comes from three ideas: (i) instead of privileged abstractions, we use scandots as privileged information that generalizes across terrain geometries, (ii) allowing the policy to decide its own heading at deployment depending on the obstacle. This allows us to demonstrate the capability of jumping across tilted ramps. And (iii) a unified general-purpose reward principle. Furthermore, we are able to cross gaps that are upto $2\times$ the length of the robot and jump obstacles that are $2\times$ its height, whereas concurrent work using the A1 robot jumps at most $1.5\times$ its height and $1.5\times$ its length (Table 1).

Method	Robot	Climb	Gap	Ramp	Handstand
Rudin et. al [31]	AnymalC	1.1	0.75	×	×
Hoeller et. al [12]*	AnymalC	2	1.5	×	×
Zhuang et. al [47]*	Unitree-A1	1.6	1.5	×	×
Extreme Parkour (ours)	Unitree-A1	2	2	37°	✓

Table 1: Comparison of parkour setups. Starred papers in 2nd and 3rd row are concurrent works (recently released). The numbers in Climb and Gap denote the relative size of obstacles with respect to quadruped’s height and length respectively. Notably, our method is able to push the low-cost A1 robot to extreme scenarios which are *twice* the height and length of robot. Anymal is an industry-standard high-quality robot and therefore much more expensive.

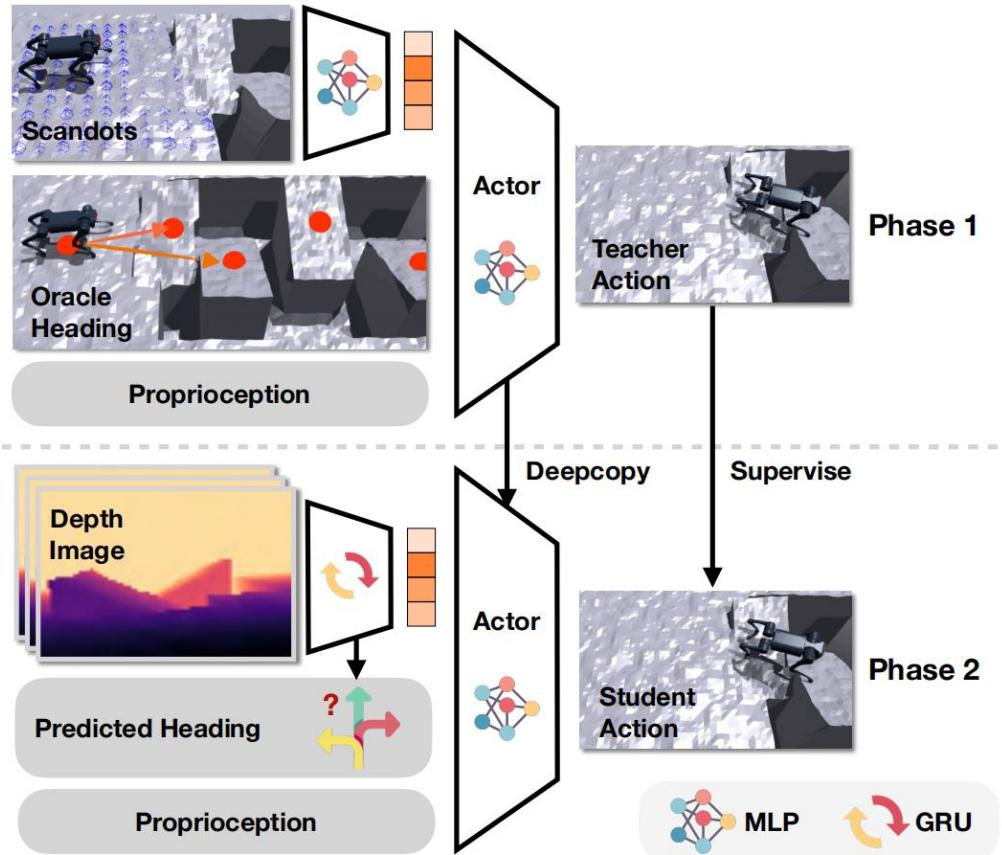


Figure 2: Training overview. In phase 1, we use RL to learn a locomotion policy with access to privileged information like environment parameters and scandots [2] in addition to heading direction from waypoints. We use **Regularized Online Adaptation (ROA)**[9] to train an estimator to recover environmental information from the history of observations. In phase 2, we distill from scandots into a policy that operates from onboard depth and *automatically decides its heading (yaw) direction* conditioned on the obstacle.

3.1 Unified Reward for Extreme Parkour

The rewards used in [2] do not transfer directly to the parkour case. The robot cannot follow arbitrary direction commands and instead must have the freedom to choose the optimal direction. Instead of randomly sampling directions, we compute direction using waypoints placed on the terrain (Fig. 3) as

$$\hat{\mathbf{d}}_w = \frac{\mathbf{p} - \mathbf{x}}{\|\mathbf{p} - \mathbf{x}\|} \quad (1)$$

where \mathbf{p} is the next waypoint location and \mathbf{x} is robot location in the world frame. The velocity tracking reward is then computed as

$$r_{tracking} = \min(\langle \mathbf{v}, \hat{\mathbf{d}}_w \rangle, v_{cmd}) \quad (2)$$

where $\mathbf{v} \in \mathbb{R}^2$ is the robot's current velocity in world frame and $v_{cmd} \in \mathbb{R}$ is the desired speed. Note that [2] tracks velocity in the base frame but world frame is used. This is done to prevent the robot from exploiting the reward and learning the unintended behavior of turning around the obstacle.

While the above reward is sufficient for diverse parkour behavior, for challenging obstacles the robot tends to step close to the edge to minimize energy usage. This behavior is risky and does not transfer well to real settings. We therefore add a term to penalize foot contacts near terrain edges.

$$r_{clearance} = - \sum_{i=0}^4 c_i \cdot M[p_i] \quad (3)$$

c_i is 1 if i th foot touches the ground. M is a boolean function which is 1 iff the point p_i lies within 5cm of an edge. p_i is the foot position for each leg.

The rewards defined above typically lead to a gait that uses all four legs. However, a defining feature of parkour is walking in different styles that are aesthetically pleasing but may not be biomechanically optimal. To explore this diversity, we introduce a term to track a desired forward vector using the same inner product design principle, which can be controlled by the operator at test time.

$$r_{stylized} = W \cdot [0.5 \cdot \langle \hat{\mathbf{v}}_{fwd}, \hat{\mathbf{c}} \rangle + 0.5]^2 \quad (4)$$

where $\hat{\mathbf{v}}_{fwd}$ is the unit vector pointing forward along the robot's body, $\hat{\mathbf{c}}$ is also a unit vector indicating the desired direction and W is a binary number to switch the reward on/off. In our case, we train the robot to do a handstand and choose $\hat{\mathbf{c}} = [0, 0, -1]^T$. W is sampled randomly in $\{0, 1\}$ at training time and controlled via remote at deployment time.



提纲

一、ETH三部曲

二、AMP三部曲

三、Pakour三部曲

四、无人机三部曲

2021 «Science Robotics»
University of Zurich

SCIENCE ROBOTICS | RESEARCH ARTICLE

NAVIGATION

Learning high-speed flight in the wild

Antonio Loquercio^{1,*†}, Elia Kaufmann^{1†}, René Ranftl², Matthias Müller²,
Vladlen Koltun³, Davide Scaramuzza¹

Copyright © 2021
The Authors, some
rights reserved;
exclusive licensee
American Association
for the Advancement
of Science. No claim

2022 «Science Robotics»
California Institute of
Technology

SCIENCE ROBOTICS | RESEARCH ARTICLE

AUTONOMOUS VEHICLES

Neural-Fly enables rapid learning for agile flight in strong winds

Michael O'Connell†, Guanya Shi†, Xichen Shi, Kamyar Azizzadenesheli,
Anima Anandkumar, Yisong Yue, Soon-Jo Chung*

Copyright © 2022
The Authors, some
rights reserved;
exclusive licensee
American Association
for the Advancement
of Science. No claim
to original U.S.
Government Works

2023 «Nature»
University of Zurich

Article

Champion-level drone racing using deep reinforcement learning

<https://doi.org/10.1038/s41586-023-06419-4>

Received: 5 January 2023

Elia Kaufmann^{1,✉}, Leonard Bauersfeld¹, Antonio Loquercio¹, Matthias Müller², Vladlen Koltun³
& Davide Scaramuzza¹

Loquercio, Antonio, Elia Kaufmann, René Ranftl, Matthias Müller, Vladlen Koltun, and Davide Scaramuzza. "Learning high-speed flight in the wild." *Science Robotics* 6, no. 59 (2021): eabg5810.

Quadrotors are agile. Unlike most other machines, they can traverse extremely complex environments at high speeds. To date, only expert human pilots have been able to fully exploit their capabilities. Autonomous operation with onboard sensing and computation has been limited to low speeds. State-of-the-art methods generally separate the navigation problem into subtasks: sensing, mapping, and planning. Although this approach has proven successful at low speeds, the separation it builds upon can be problematic for high-speed navigation in cluttered environments. The subtasks are executed sequentially, leading to increased processing latency and a compounding of errors through the pipeline. Here, we propose an end-to-end approach that can autonomously fly quadrotors through complex natural and human-made environments at high speeds with purely onboard sensing and computation. The key principle is to directly map noisy sensory observations to collision-free trajectories in a receding-horizon fashion. This direct mapping drastically reduces processing latency and increases robustness to noisy and incomplete perception. The sensorimotor mapping is performed by a convolutional network that is trained exclusively in simulation via privileged learning: imitating an expert with access to privileged information. By simulating realistic sensor noise, our approach achieves zero-shot transfer from simulation to challenging real-world environments that were never experienced during training: dense forests, snow-covered terrain, derailed trains, and collapsed buildings. Our work demonstrates that end-to-end policies trained in simulation enable high-speed autonomous flight through challenging environments, outperforming traditional obstacle avoidance pipelines.

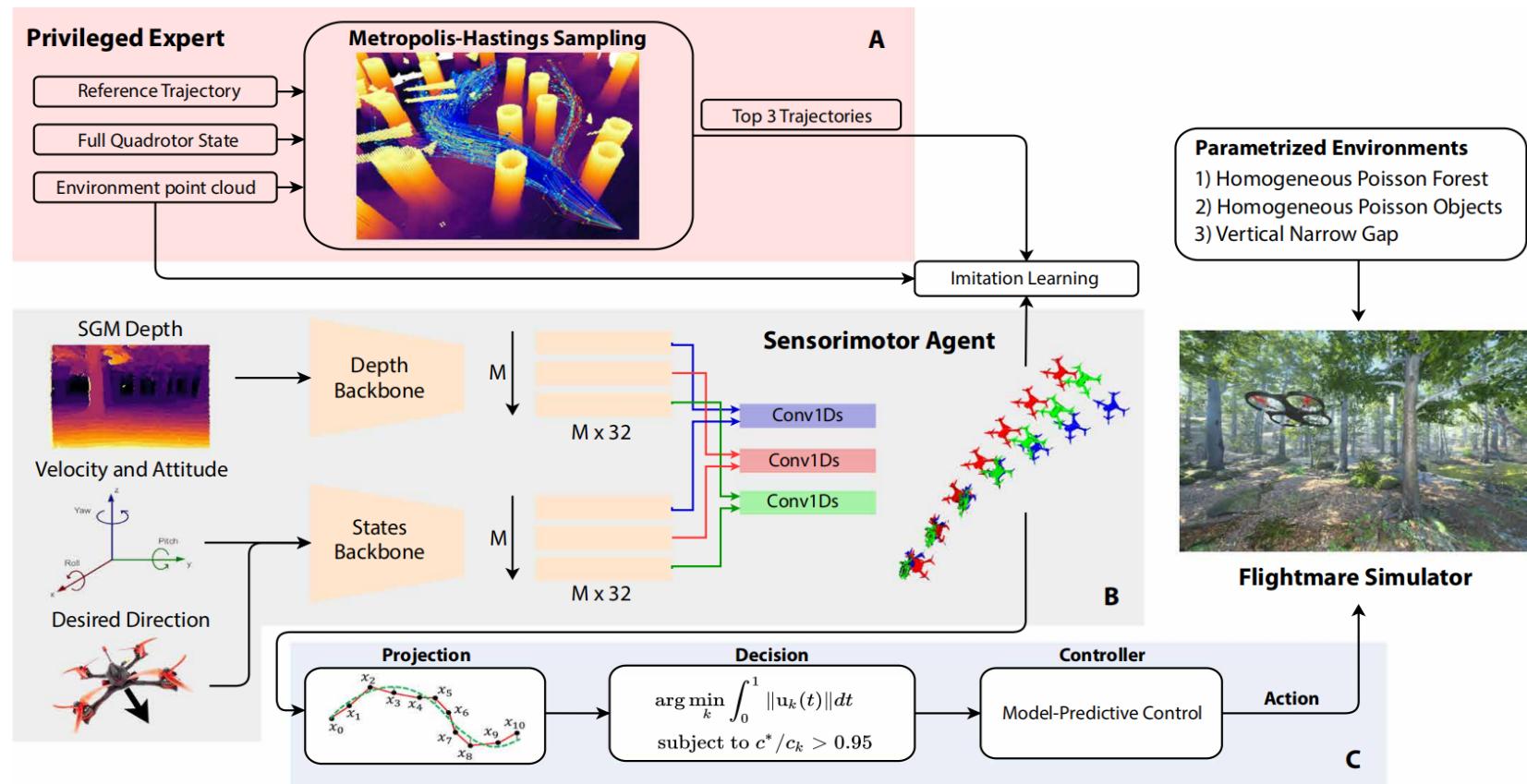


Fig. 6. Method overview. (A) Our offline planning algorithm computes a distribution of collision-free trajectories to follow a reference trajectory. The trajectories are computed with M-H sampling and are conditioned on complete 3D knowledge of the environment, which is represented by a point cloud. (B) A sensorimotor agent is trained with imitation learning to predict the best three trajectories from the estimated depth, the drone's velocity and attitude, and the desired direction that encodes the goal. (C) The predictions are projected on the space of polynomial trajectories and ranked according to their predicted collision cost c_k . The trajectory with the lowest predicted cost c_k is then tracked with a model-predictive controller. If multiple trajectories have similar predicted cost (within a 5% range of the minimum $c^* = \min c_k$), the one with the smallest actuation cost is used.

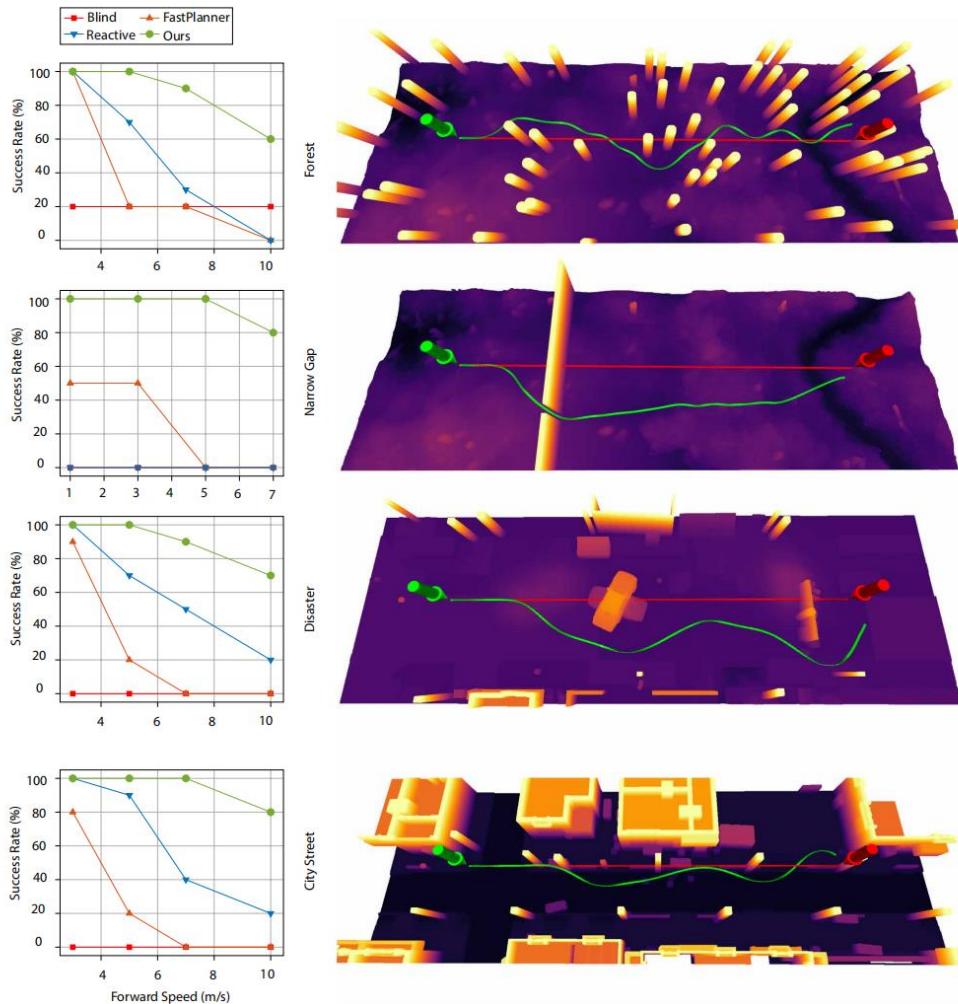


Fig. 4. Experiments in different simulated environments. **Left column:** Success rates at various speeds. **Right column:** Point cloud of the environment together with paths taken by different policies from start (green arrow) to end (red arrow). The paths taken by the blind and our policy are illustrated by red and green lines, respectively. Our approach consistently outperforms the baselines in all environments and at all speeds. Sample observations from these environments are shown in fig. S4.

O'Connell, Michael, Guanya Shi, Xichen Shi, Kamyar Azizzadenesheli, Anima Anandkumar, Yisong Yue, and Soon-Jo Chung. "Neural-fly enables rapid learning for agile flight in strong winds." *Science Robotics* 7, no. 66 (2022): eabm6597.

Executing safe and precise flight maneuvers in dynamic high-speed winds is important for the ongoing commoditization of uninhabited aerial vehicles (UAVs). However, because the relationship between various wind conditions and its effect on aircraft maneuverability is not well understood, it is challenging to design effective robot controllers using traditional control design methods. We present Neural-Fly, a learning-based approach that allows rapid online adaptation by incorporating pretrained representations through deep learning. Neural-Fly builds on two key observations that aerodynamics in different wind conditions share a common representation and that the wind-specific part lies in a low-dimensional space. To that end, Neural-Fly uses a proposed learning algorithm, domain adversarially invariant meta-learning (DAIML), to learn the shared representation, only using 12 minutes of flight data. With the learned representation as a basis, Neural-Fly then uses a composite adaptation law to update a set of linear coefficients for mixing the basis elements. When evaluated under challenging wind conditions generated with the Caltech Real Weather Wind Tunnel, with wind speeds up to 43.6 kilometers/hour (12.1 meters/second), Neural-Fly achieves precise flight control with substantially smaller tracking error than state-of-the-art nonlinear and adaptive controllers. In addition to strong empirical performance, the exponential stability of Neural-Fly results in robustness guarantees. Last, our control design extrapolates to unseen wind conditions, is shown to be effective for outdoor flights with only onboard sensors, and can transfer across drones with minimal performance degradation.

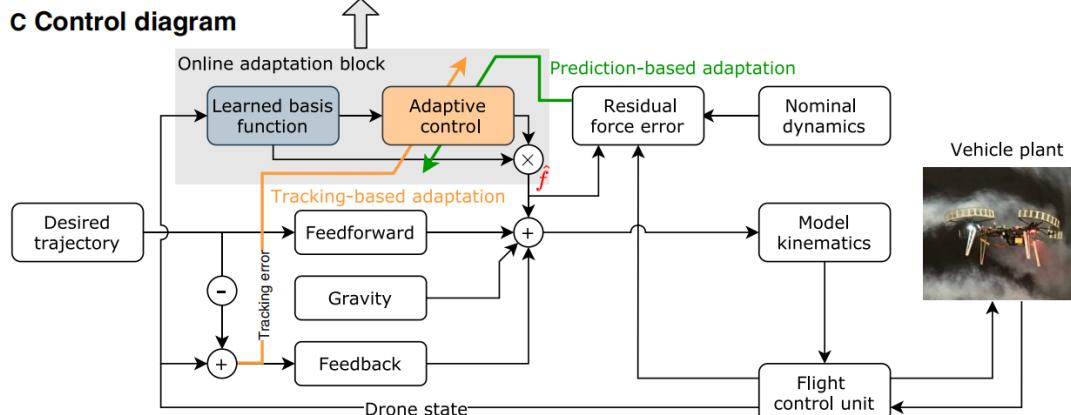
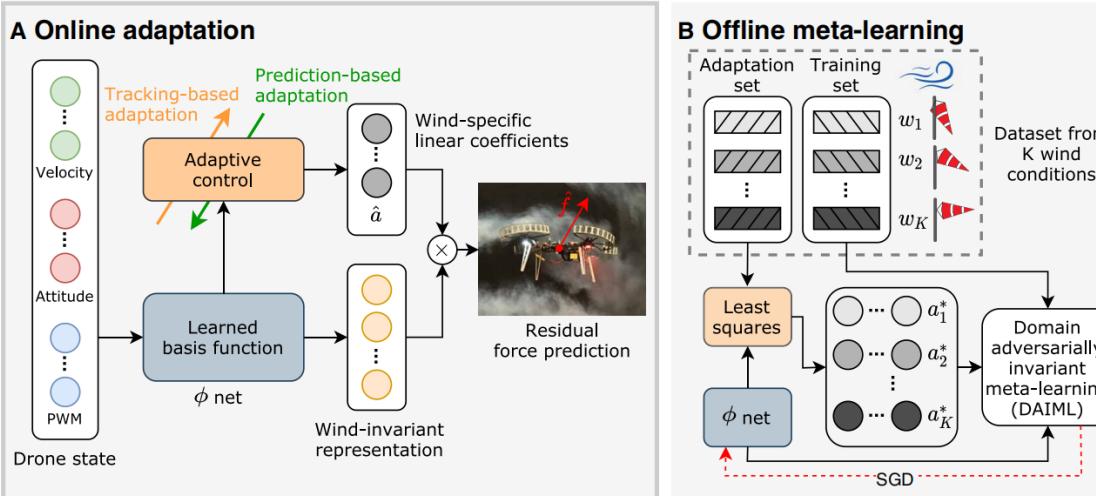


Fig. 2. Offline meta-learning and online adaptive control design. (A) The online adaptation block in our adaptive controller. Our controller leverages the meta-trained basis function ϕ , which is a wind-invariant representation of the aerodynamic effects, and uses composite adaptation (that is, including tracking error-based and prediction error-based adaptation) to update wind-specific linear weights \hat{a} . The output of this block is the wind-effect force estimate, $\hat{f} = \phi\hat{a}$. (B) The illustration of our meta-learning algorithm DAIML. We collected data from wind conditions $\{w_1, \dots, w_K\}$ and applied Algorithm 1 to train the ϕ net. (C) The diagram of our control method, where the gray part corresponds to (A). Interpreting the learned block as an aerodynamic force allows it to be incorporated into the feedback control easily.

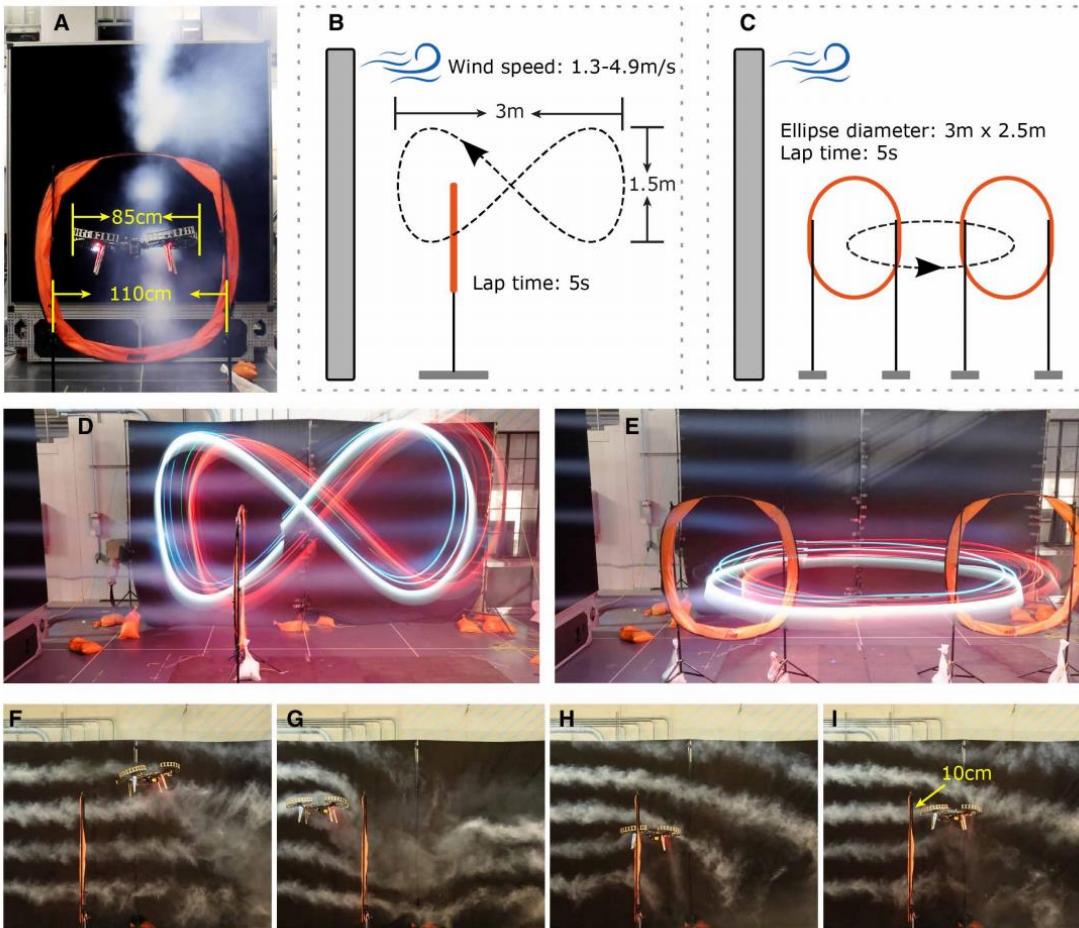


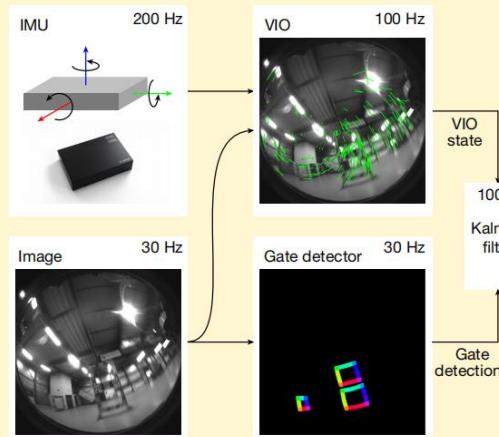
Fig. 1. Agile flight through narrow gates. (A) Caltech Real Weather Wind Tunnel system, the quadrotor UAV, and the gate. In our flight tests, the UAV follows an agile trajectory through narrow gates, which are slightly wider than the UAV itself, under challenging wind conditions. (B and C) Trajectories used for the gate tests. In (B), the UAV follows a figure-8 through one gate, with a wind speed of 3.1 m/s or time-varying wind condition. In (C), the UAV follows an ellipse in the horizontal plane through two gates, with a wind speed of 3.1 m/s. (D and E) Long-exposure photos (with an exposure time of 5 s) showing one lap in two tasks. (F to I) High-speed photos (with a shutter speed of 1/200 s) showing the moment the UAV passed through the gate and the interaction between the UAV and the wind.

Kaufmann, Elia, Leonard Bauersfeld, Antonio Loquercio, Matthias Müller, Vladlen Koltun, and Davide Scaramuzza. "Champion-level drone racing using deep reinforcement learning." *Nature* 620, no. 7976 (2023): 982-987.

First-person view (FPV) drone racing is a televised sport in which professional competitors pilot high-speed aircraft through a 3D circuit. Each pilot sees the environment from the perspective of their drone by means of video streamed from an onboard camera. Reaching the level of professional pilots with an autonomous drone is challenging because the robot needs to fly at its physical limits while estimating its speed and location in the circuit exclusively from onboard sensors¹. Here we introduce Swift, an autonomous system that can race physical vehicles at the level of the human world champions. The system combines deep reinforcement learning (RL) in simulation with data collected in the physical world. Swift competed against three human champions, including the world champions of two international leagues, in real-world head-to-head races. Swift won several races against each of the human champions and demonstrated the fastest recorded race time. This work represents a milestone for mobile robotics and machine intelligence², which may inspire the deployment of hybrid learning-based solutions in other physical systems.

a Real-world operation

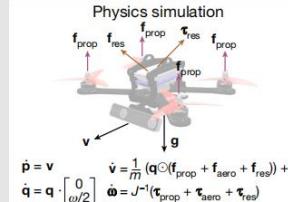
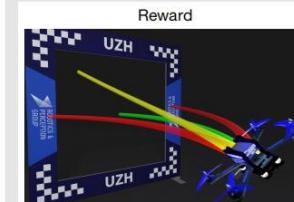
Perception system



Real-world deployment

**b** RL training loop

Simulation environment



Ground-truth state

Perception residual

Observed state

Residual force and torque

Real-world experience

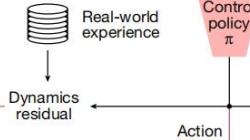
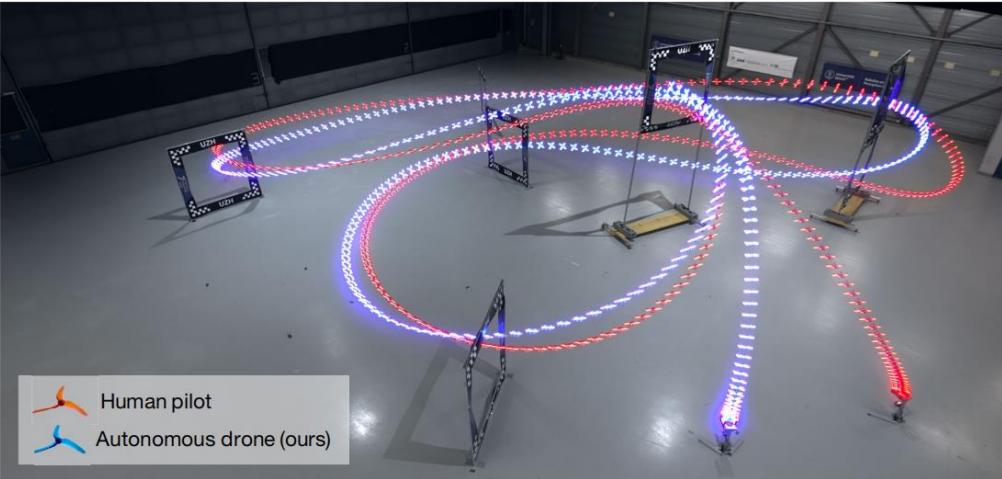


Fig. 2 | The Swift system. Swift consists of two key modules: a perception system that translates visual and inertial information into a low-dimensional state observation and a control policy that maps this state observation to control commands. Control commands specify desired collective thrust and body rates, the same control modality that the human pilots use. **a**, The perception system consists of a VIO module that computes a metric estimate of the drone state from camera images and high-frequency measurements obtained by an inertial measurement unit (IMU). The VIO estimate is coupled with a neural network that detects the corners of racing gates in the image

stream. The corner detections are mapped to a 3D pose and fused with the VIO estimate using a Kalman filter. **b**, We use model-free on-policy deep RL to train the control policy in simulation. During training, the policy maximizes a reward that combines progress towards the centre of the next racing gate with a perception objective to keep the next gate in the field of view of the camera. To transfer the racing policy from simulation to the physical world, we augment the simulation with data-driven residual models of the vehicle's perception and dynamics. These residual models are identified from real-world experience collected on the race track. MLP, multilayer perceptron.

a Drone racing: human versus autonomous



b Head-to-head competition



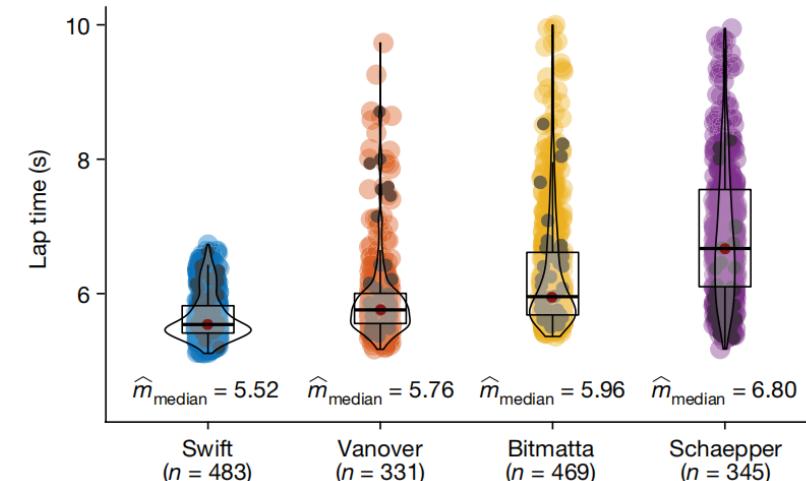
c Human champions



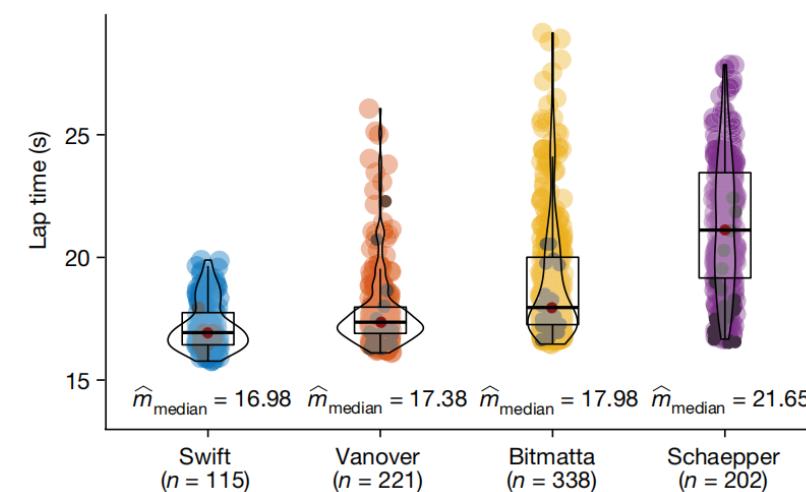
Fig. 1 | Drone racing. **a**, Swift (blue) races head-to-head against Alex Vanover, the 2019 Drone Racing League world champion (red). The track comprises seven square gates that must be passed in order in each lap. To win a race, a competitor has to complete three consecutive laps before its opponent.

b, A close-up view of Swift, illuminated with blue LEDs, and a human-piloted drone, illuminated with red LEDs. The autonomous drones used in this work rely

only on onboard sensory measurements, with no support from external infrastructure, such as motion-capture systems. **c**, From left to right: Thomas Bitmatta, Marvin Schaepper and Alex Vanover racing their drones through the track. Each pilot wears a headset that shows a video stream transmitted in real time from a camera aboard their aircraft. The headsets provide an immersive ‘first-person-view’ experience. **c**, Photo by Regina Sablotny.

a Single lap time comparison

Three laps comparison

**b** Head-to-head racing results

	Number of races	Best time-to-finish	Wins	Losses	Win ratio
A. Vanover versus Swift	9	17.956 s	4	5	0.44
T. Bitmatta versus Swift	7	18.746 s	3	4	0.43
M. Schaepper versus Swift	9	21.160 s	3	6	0.33
Swift versus human pilots	25	17.465 s	15	10	0.60

Fig. 3 | Results. **a**, Lap-time results. We compare Swift against the human pilots in time-trial races. Lap times indicate best single lap times and best average times achieved in a heat of three consecutive laps. The reported statistics are computed over a dataset recorded during one week on the race track, which corresponds to 483 (115) data points for Swift, 331 (221) for A. Vanover, 469 (338) for T. Bitmatta and 345 (202) for M. Schaepper. The first number is the number of single laps and the second is the number of three consecutive laps. The dark points in each distribution correspond to laps flown in race conditions. **b**, Head-to-head results. We report the number of head-to-head races flown by each pilot, the number of wins and losses, as well as the win ratio.

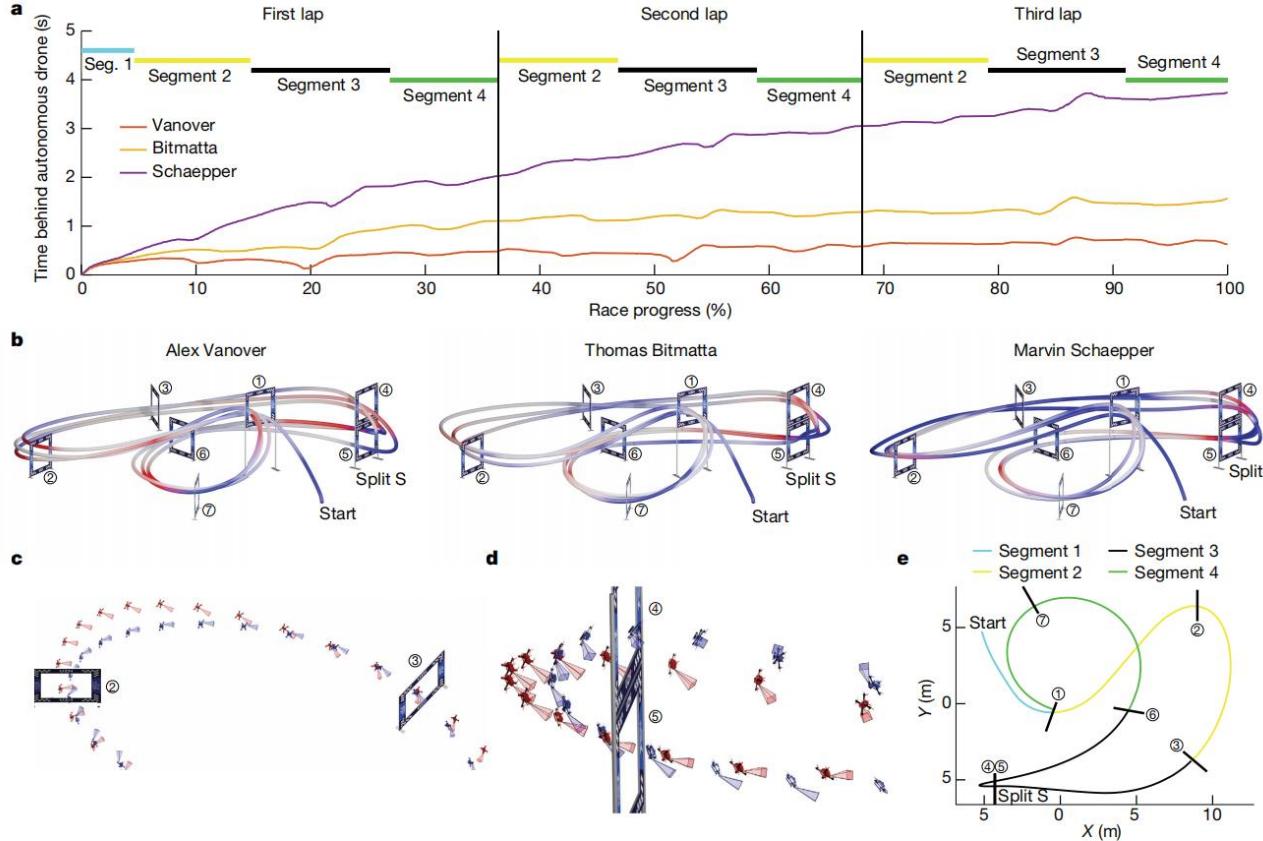


Fig. 4 | Analysis. **a**, Comparison of the fastest race of each pilot, illustrated by the time behind Swift. The time difference from the autonomous drone is computed as the time since it passed the same position on the track. Although Swift is globally faster than all human pilots, it is not necessarily faster on all individual segments of the track. **b**, Visualization of where the human pilots are faster (red) and slower (blue) compared with the autonomous drone. **Swift is consistently faster at the start and in tight turns, such as the split S.** **c**, Analysis of the manoeuvre after gate 2. Swift in blue, Vanover in red. **Swift gains time against human pilots in this segment as it executes a tighter turn while maintaining comparable speed.** **d**, Analysis of the split S manoeuvre. Swift in blue, Vanover in red. **The split S is the most challenging segment in the race track, requiring a carefully coordinated roll and pitch motion that yields a descending half-loop through the two gates. Swift gains time against human pilots on this segment as it executes a tighter turn with less overshoot.**

e, Illustration of track segments used for analysis. Segment 1 is traversed once at the start, whereas segments 2–4 are traversed in each lap (three times over the course of a race).



谢谢大家