SCIENCE CHINA
Information Sciences

• Supplementary File •

Domain-aware behavior cloning for bridging the sim-to-real gap of legged robots

Yong Zhou[1], Jiawei Jiang[1*], Bo Du[1], Hengyi Yang[1] & Qianfan Hu[1]

[1]*School of Computer Science, Wuhan University, Wuhan* 430072*, China*

Appendix A Experiment Details

Appendix A.1 Control of the End Effector

The motor of our Unitree A1 Robot has three modes: velocity control, torque control, and position control. We choose the position control mode because, in position control mode, the action space can be normalized which reduces the difficulty for the reinforcement learning algorithms to search for appropriate action. The action space is normalized by shifting the zero position to the stance pose of the robot. This means that when all the actions are close to zero, which is close to the output of the reinforcement learning policy and the beginning of the training, the algorithm can start at a stance pose rather than falling to the ground. With the desired target position, the electric motor will transfer it to torque via PD control:

$$\tau = K_P(q_{des} - q) + K_D(0 - \dot{q}) \tag{A1}$$

where q is the motor position and $\dot{q}$ is the speed of the motor. We select $K_P = 40$ and $K_D = 0.5$ in our experiments.

Appendix A.2 Reward Design

The reward functions used to train the base policy is demonstrated in Table A1 where v^*_{xy} and v_{xy} denote the commanded and the actual velocity in the x, y direction; p^*_{rpy} and p_{rpy} denote the commanded and the actual robot pose; H^*_z and H_z denote the desired and the actual body height; q, $\dot{q}$, τ denote joint poisition, join velocity and the joint torque.

While training the adaptive policy, tracking joint position and velocity alone may be sufficient. However, legged robots can enter a flying phase where no legs contact the ground, leading to pose-tracking errors. To address this, we augment joint motion tracking with heading and velocity tracking for enhanced performance. The reward functions are defined as:

$$f_{adapt}(x_S, x_T) = Sum \begin{cases} \alpha_1 * exp(-\|h^S_t - h^T_t\|^2/\beta_1), & Heading \quad Tracking \\ \alpha_2 * exp(-\|v^S_t - v^T_t\|^2/\beta_2), & Velocity \quad Tracking \\ \alpha_3 * exp(-\|q^S_t - q^T_t\|^2/\beta_3), & Joint \quad Position \quad Tracking \\ \alpha_4 * exp(-\|\dot{q}^S_t - \dot{q}^T_t\|^2/\beta_4), & Joint \quad Velocity \quad Tracking \end{cases}$$

where v denotes the velocity, S and T specify the source domain and the target domain. The hyperparameter β decides the tracking requirements of the dynamic state. For example, a few steps of exploration in the action space will not affect the velocity of the robots obviously, thus the value of β_2 has to be set to a small number to amplify the minor change in velocity. The hyperparameter α is the weight of each reward component and will be tuned to train an adaptation policy with the best performance.

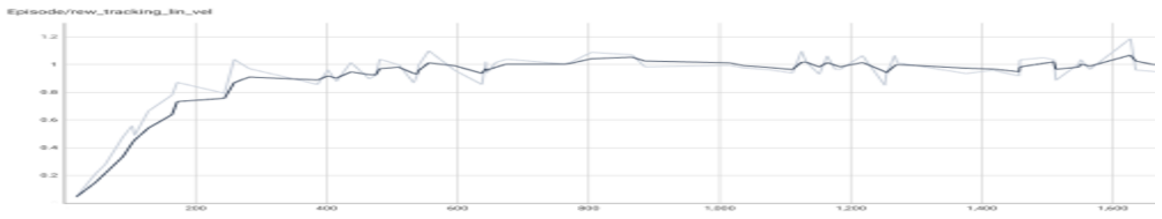
Appendix A.3 Training of the Locomotion Policy

The velocity tracking reward is demonstrated in Figure A1 (a), and the total reward is demonstrated in Figure A1 (b). The training lasts for 70,000 steps while the velocity tracking reward converges at around 1,500 steps. It demonstrates that the robot learns to move quickly but takes a much longer time to adapt to terrains. We use a terrain curriculum to train the robot on easier terrain at the beginning, when the robot is capable of navigating through the terrain, it will be reset at a more difficult terrain which results in higher penalty rewards.

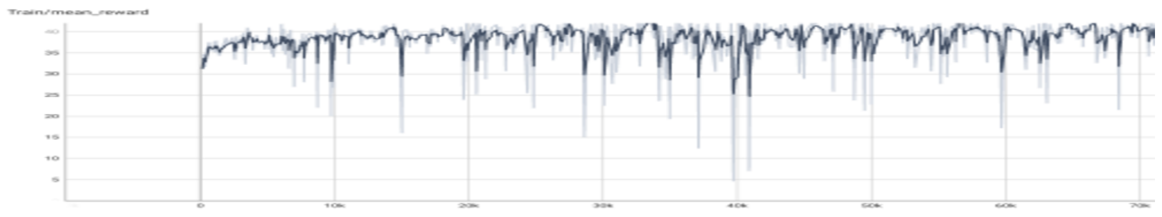
[*] Corresponding author (email: jiawei.jiang@whu.edu.cn)

Table A1 Reward functions. Here, $\Phi(x) := exp(-\frac{||x||^2}{0.25})$, t_{air} denotes the duration between a leg touching the ground twice, and n_c represents the number of collisions.

Function	Equation	Description
$f_0(s_t)$	$\Phi(v^*_{xy} - v_{xy})$	Velocity Tracking
$f_1(s_t)$	$\Phi(p^*_{rpy} - p_{rpy})$	Pose Tracking
$f_2(s_t)$	$\|\|H^*_z - H_z\|\|^2$	Body Height Tracking
$f_3(s_t)$	$-\|\|q\|\|^2 - \|\|\dot{q}\|\|^2$	Joint Motion
$f_4(s_t)$	$-\|\|\dot{q} * \tau\|\|^2$	Energy Consumption
$f_5(s_t)$	$-\|\|a_t - a_{t-1}\|\|^2$	Action Roughness
$f_6(s_t)$	$\sum_{f=0}^{4}(t_{air} - 0.5)$	Feet Drag



(a) The velocity tracking reward during training.



(b) The total reward during training.

Figure A1 The figure demonstrates the training curve of the velocity tracking reward(a) and the total reward(b).

Appendix A.4 Experiment Setup

Environment. We train and validate the policy with our proposed localization modules in simulation. We use the A1 URDF file for the quadrupedal robot and Issac Gym for the physics simulation engine. We deploy the localization modules on a real A1 robot to verify the real-time performance. The policy is designed to run at 50 Hz but owing to the delay real world, it runs at around 43 Hz. We use a prebuilt map with leg odometry technique to generate a local height map in real-time to enable operation on terrains.

Task. The quadrupedal robots are commanded to first walk forward and then backward with a fixed velocity ranging from $0.3 \sim 0.7 m/s$. The robot walks in three environments including free walking, walking under disturbance, and walking on stairs.

The detailed process of the task in the real world can be shown in Figure A2 and Figure A3.

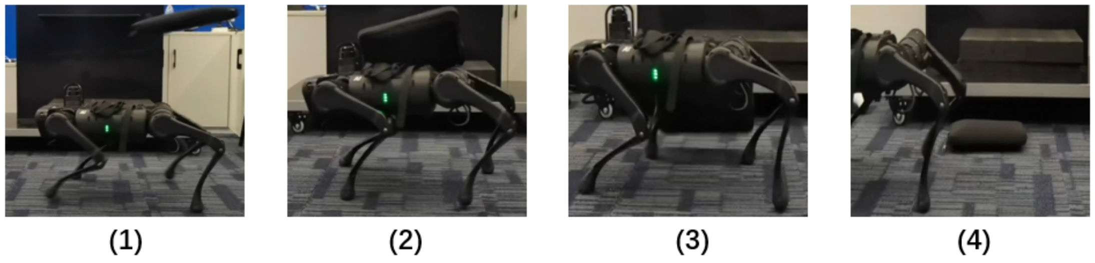


Figure A2 Walking under a sudden disturbance from a dropping 3 kg payload.

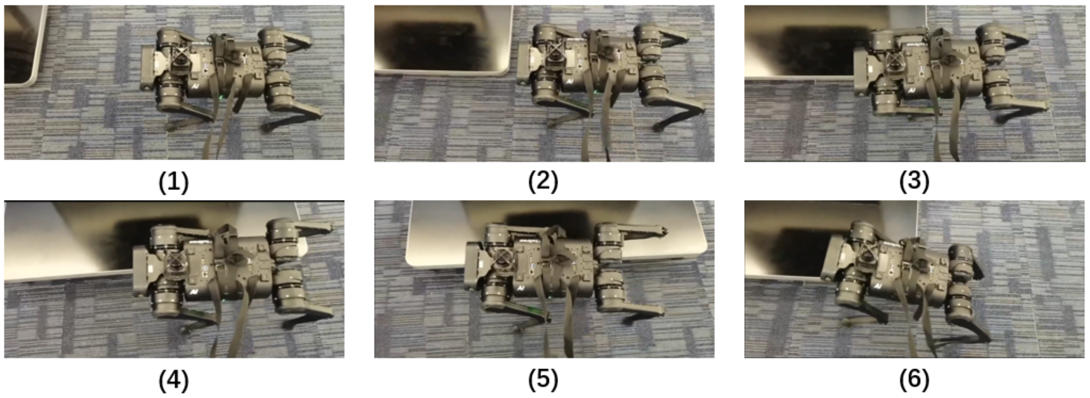


Figure A3 Walking on a stair with 11 cm height.

Appendix A.5 Baselines and Results

Our Experiments are designed to answer three questions:

- *Q1.* Does tracking the next states in the source domain result in better performance in the target domain?
- *Q2.* Can our proposed adaptor provide correct action residual in response to the disturbance?
- *Q3.* Can our proposed framework be deployed on a real robot and maintain the performance metrics in the simulation?

Appendix A.5.1 *Baselines*

We compare our results to the following baselines:

- *Expert Policy.* In simulation, the ground truth domain factors are available for the expert policy. The performance of the expert policy marks the upper bound of RL-based approaches.
- *Domain Randomization (DR)* [4] By expanding the range of randomizing the domain factors in simulation, the locomotion policy learns to adapt to varied environments without knowledge about the domain factors.
- *System Identification (SysID)* [7] The adaptor of SysID predicts the domain factors with a neural network directly. In our experiment, we replace the ground truth domain factors in the expert policy with the predicted domain factors.
- *Domain Adaptation (DA)* [1,2,6] For domain adaptation, a teacher policy is trained using the ground truth embedding of domain factors, while a student policy is trained to approximate this embedding based on state history. Since the ground truth embedding is not available in the real world, we only evaluate and deploy the student policy.
- *DABC (naive)* To evaluate the necessity of including heading tracking and velocity tracking rewards (in addition to joint position and joint velocity tracking rewards defined in Section Appendix A.2), we introduce a naive baseline that trains the adaptive policy using only joint position and joint velocity tracking rewards.

Appendix A.5.2 *Task and Metric*

To evaluate the domain adaptation capability of all methods, we design experiments in both simulation and real-world environments.

Free Velocity Tracking We design a velocity tracking task to evaluate the performance of domain adaptation in simulation. The expert policy operates in the source domain with optimal performance, demonstrating the policy's upper bound. In this setup, the friction coefficient, external mass, deviation of the center of mass, PD value, and actuator effort are set to $1, 0$ kg, $(0, 0, 0), 40, 0.6, 1$, respectively. The baselines and our proposed method are evaluated in the setting when the actuators are different from the training environment, where the domain factors are configured according to Table A2. We deploy the baselines and our method on a real robot. In the real world, the robot follows a smoother velocity curriculum for tracking, progressing from 0.3 m/s to 0.6 m/s over a period of 10 seconds, covering a total distance of 5.5 meters.

Velocity Tracking under Disturbance We design an additional experiment to evaluate the robustness and adaptive capabilities of the robot's walking policies under sudden disturbances in real-world conditions. The experiment will involve subjecting the robot to a sudden disturbance in the form of a payload addition while it is walking. This tests the robot's ability to adapt and maintain stability when its carrying load changes abruptly.

Terrain Traversing Our objective is to evaluate the effectiveness of various domain adaptation policies in enabling the robot to walk on stairs of different heights, both in simulation and real-world settings. The experiment focuses on testing the robot's ability to walk on terrains with varied heights. A high-performance domain adaptation policy should allow the robot to adapt quickly and maintain stability when transitioning from flat surfaces to stairs of different heights.

Metrics We provide a quantitative analysis of the methods through the tasks including the free velocity tracking and the velocity tracking under disturbance. We set the mean velocity tracking reward as the metric. Since in the real world, the velocity of the robot is not directly accessible, we measure the traveled distance of the robot by legged odometry and videos. Then, we compute the velocity according to the measured distance and recorded timestamps. We provide a qualitative analysis via the terrain traversing task. The success rate of terrain traversing is reported for each task. Successfully traversing the terrain is defined as walking on the terrain without a single fall and traveling more than half of the designated distance.

Table A2 Ranges of the Randomization for Domain Factors in the Training Phase and the Selected Setting in the Testing Phase.

Parameters	Training Range	Source Domain	Target Domain
Friction	[0.3, 1.3]	1	0.3
Extra Mass (kg)	[-2, 6]	0	4
Center of Mass (m)	[-0.1, 0.1]	0	0.1
K_P	[35, 45]	40	35
K_D	[0.6, 0.8]	0.6	0.8
Motor Strength	[0.9, 1.1]	1	0.9

Appendix A.5.3 *Simulation Environment*

(Robot Description) We use the open-source URDF (Unified Robot Description Format) file of the A1 quadrupedal robot by Unitree [5] in our simulation environment for training the policies. The A1 robot is equipped with 12 actuators that control the movement of the hip, thigh, and calf for each of its four legs. The robot weighs about 12 kg with 4 legs which has a length of $20cm$. We use the Isaac Gym environment [3] to train the policy with 4096 agents in parallel to speed up training.

(Locomotion Policy) The locomotion policy is a Multi-Layer Perceptron (MLP) with a layer size of [512, 256, 128]. The autoencoder for ground truth domain factors is an MLP with a layer size of [64, 32]. The policy that tracks the velocity on flat ground is trained for 5000 iterations with an Adam optimizer. The locomotion policy that operates on stairs is trained for 30000 iterations with a height map beneath the robot of size $1.6m^2$ included in the input.

(Adaptation Policy) To train the adaptation policies, we use 4096 agents in parallel and collect the state history for 50 steps in a buffer which corresponds to the experience in 1 second. During training, 2048 agents are indexed as the source domain and others are indexed as the target domain. The adaptation policy initially embeds the state history using a Multi-Layer Perceptron (MLP) with layers sized [64, 32]. Subsequently, it employs a three-layer, one-dimensional Convolutional Neural Network (1-D CNN) to convolve the states across the temporal domain, effectively capturing temporal information. The number of input channel, output channel, kernel size, and stride is [32, 32, 8, 4], [32, 32, 5, 1], [32, 32, 5, 1]. Finally, the output of the convolution layer is flattened to predict the target. The output dimension of the adaptation policy of SysID, DA, and our method is 41, 32, and 12.

Appendix A.5.4 *Real Robot*

We deploy all the trained policies on a real A1 robot. The action of the locomotion policy is the residual of the default joint position. The default joint position of the hip, thigh, and calf for the FL, FR, RL, RR leg is [0.1, 0.8, -1.5, 0.1, 1, -1.5, -0.1, 0.8, -1.5, -0.1, 1, -1.5]. The desired joint position of each actuator is converted to torque by a PD controller with K_P, K_D set to 40, 0.5. To ensure stable real-time performance, the command is sent to the actuators via UDP with a frequency of $400Hz$. Our framework is designated to run at 50Hz, but owing to the limitations of onboard computing, the framework finally runs at around 43Hz.

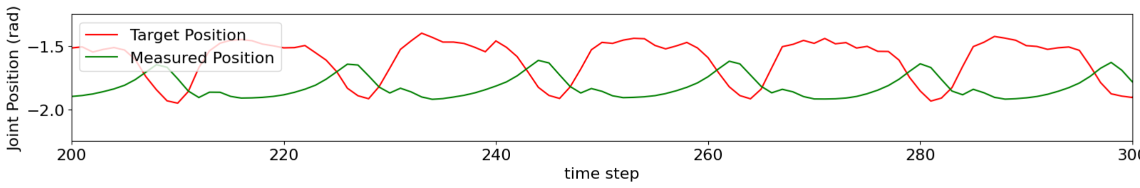
Table A3 We report the mean velocity tracking reward in the simulation and the real world. Since the ground truth domain factors are not available in the real world, we do not deploy the expert policy. The results reported are averaged over 100 trials for each policy in the simulation and 10 trials in the real world.

	Simulation	Real World	
	Free Gait	Free Gait	Disturbance
Expert	0.96 ± 0.01		
DR	0.88 ± 0.02	0.72 ± 0.02	0.65 ± 0.02
SysID	0.80 ± 0.03	0.75 ± 0.02	0.70 ± 0.02
DA	0.85 ± 0.02	0.78 ± 0.01	0.72 ± 0.01
Ours	0.89 ± 0.03	**0.84** ± 0.01	**0.82** ± 0.01

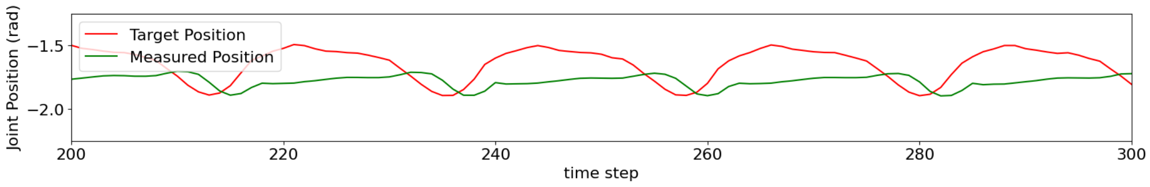
Appendix A.5.5 *Result and Analysis*

Locomotion on flat terrain **(Answer the question Q1 and Q3)**

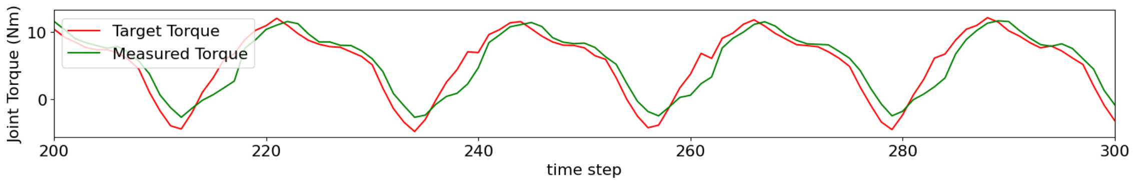
(Simulated Experiment) The result of the locomotion on flat terrains is demonstrated in Table A3. The expert policy acquires nearly all the available rewards; the imperfections mainly stem from the sudden changes in the velocity curriculum. The domain randomization policy outperforms other methods in simulation because it exploits environments that vary only within a preset range of domains. Furthermore, the smaller observation dimension in domain randomization simplifies the reinforcement learning process. However, the effect of domain factors in the real world differs from that in simulation, leading to a significant performance drop. Our proposed method outperforms other baselines because it mimics



(a) Joint Position Tracking in the Simulation



(b) Joint Position Tracking in the Real World



(c) Joint Torque Tracking in the Real World

Figure A4 The data is collected by running our proposed policy which runs in free gait. The measured position is the actual position of the joint while the desired position is the summation of the default joint position and the action. The measured torque on a real robot is collected from the sensor of the motor.

the trajectory of a better expert policy. The expert policy performs best in the domain most frequently visited during the training phase. When the domain randomization technique is used to train the policy, the optimal domain is the mean of the randomization range. Since our policy has been trained to use the embeddings of domain factors from a source domain, we can designate the optimal domain as the source domain and replicate the trajectory. This approach yields a higher upper-bound performance compared to baselines that intuitively target the expert policy with sub-optimal domain factors.

(Real-World Experiment) For real-world experiments, all the policies can be successfully deployed on a real robot while our proposed method suffers less from the domain gap. The results indicate that although the K_P and K_D values of the motors are set identically to those in the simulation, the robot acquires fewer rewards than in the simulated environment. This suggests that the robot walks much slower in the real world than in the simulation.

(Analysis for the Performance Drop) To find out the reason that causes the performance drop, we record the state history and torque history of the actuators as illustrated in Figure A4. The figure demonstrates the motion trajectory of the calf joint of the leg at the front left of the quadrupedal robot. The time horizon demonstrated in the figure is $200 \sim 300$ which corresponds to $4 \sim 5s$ after the beginning of the trial. The motor operates cyclic and stably in the simulation and in the real world, but with different action scale as shown in the position tracking curves. Based on this difference, we analyze the performance drop in two aspects: the joint position and the joint torque.

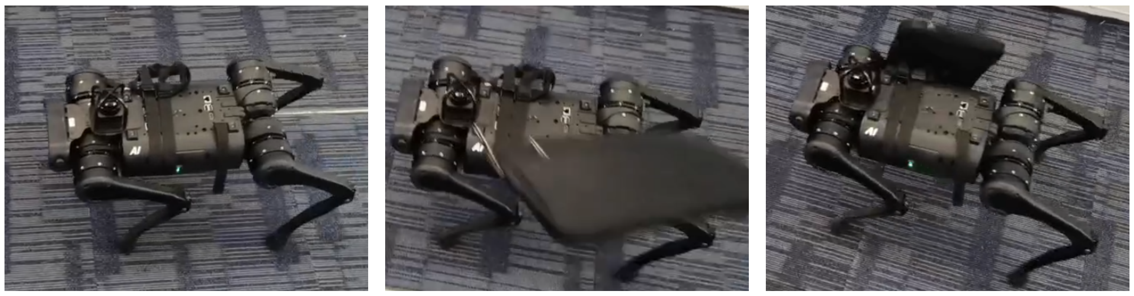
(Joint Position) In the simulation, the joint moves between $-1.6 \sim -1.9$ rads as shown in Figure A4 (a) however in the real world, the joint moves between $-1.7 \sim -1.9$ rads as shown in Figure A4 (b). The curve indicates that the actuator functions on a reduced scale in the real world compared to the simulation environment. Based on the principles of designing a trajectory generator for legged robots, the velocity is determined by the frequency and oscillation amplitude of the legs. In our experiment, the robot moves at a similar frequency to the simulation but with a smaller oscillation amplitude, resulting in a slower speed than the specified requirement. Since the action scale is derived from motor torque commands, this phenomenon may be due to the discrepancy between target torque and actual torque, which occurs only in the real world.

(Joint Torque) To verify this interpretation, we recorded and plotted the torque curves of the desired joint torque and the actual joint torque in the real world, as shown in Figure A4 (c). The plot reveals that the motor's torque output does not exactly follow the desired torque. The actual torque output has a smaller bound than the desired torque output and exhibits a delay in torque tracking. This phenomenon leads to a reduction in action scale in the real world, resulting in a smaller base velocity of the quadrupedal robot compared to the simulation. We can conclude that the adapter's ability to manage the sim-to-real gap of actuator dynamics determines the final tracking performance of the speed curriculum.

Locomotion under disturbance **(Answer the question Q2 and Q3)**

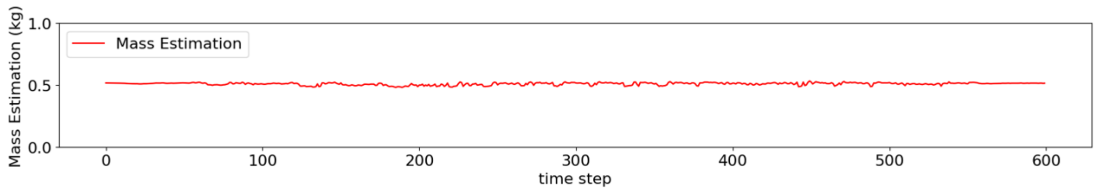
To evaluate the capability of the adaptors in resisting disturbances, we dropped a 3 kg payload during a locomotion task in the real world. The task is demonstrated in Figure A5, and the results are shown in Table A3.

As shown in the result, all the policies are managed to recover from the disturbance but the mechanism of their reaction is different. To explain how the policy reacts to the disturbance, we plot the response of the adaptors in Figure A6.

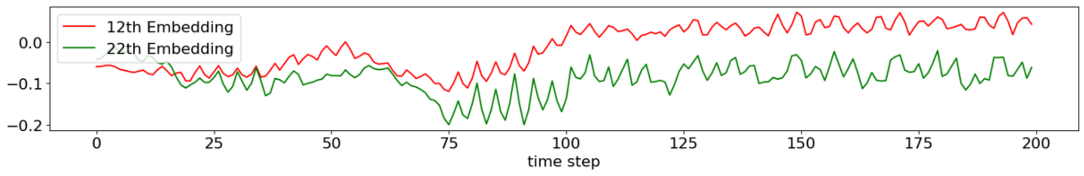


(a) Free Movement (b) Dropping a Payload (c) Impact on the Robot

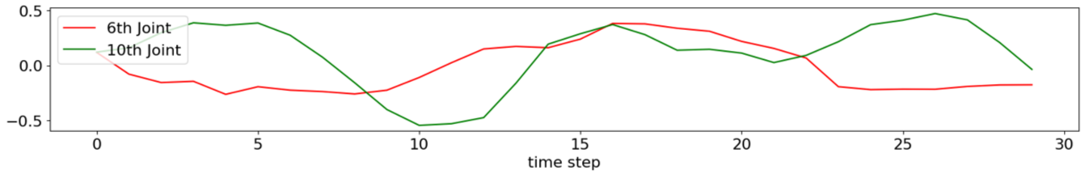
Figure A5 A solid payload that weighs about $3kg$ is dropped from the air on the top of the robot during walking.



(a) Mass Prediction of SysID



(b) The $12th$ and the $22th$ Embedding of DA



(c) Joint Residual Action Given by DABC (Ours)

Figure A6 The response of the adaptors for dropping a payload near the time of impact.

DR Since the DR does not identify the change in payload, the biggest performance drop is reported.

SysID Intuitively, a payload dropped on the trunk of the robot should result in an increased estimation of the external mass. However, as demonstrated in the figure, the mass estimation remains nearly constant throughout the rollout.

(DA) To study the effect of this disturbance on the domain adaptation method, we randomly selected elements from the 32 embeddings of DA, which are the outputs of the encoder, as described in Section Appendix A.5.3. We found that the 12th and 22nd embeddings scaled up in response to the disturbance and scaled down after the disturbance. We plotted the changes in their magnitudes near the timestamp of the disturbance. Note that the payload is thrown onto the robot and drops to the ground immediately, so the impact lasts only a few moments. The plot demonstrates that the DA's adaptor responds to the disturbance, but the correctness of the response cannot be justified due to the ambiguous meaning of the embeddings. The mechanism for reacting to a dropped payload in DA is similar to that of SysID: the disturbance changes the output vectors of the adaptor, thereby affecting the final action to handle the disturbance. This mechanism aligns with the intuition of DA methods, which view the exact values of domain factors as less important for domain adaptation, focusing instead on the final action.

(Our Method) For our method, the response is reflected on the joint action residual given by the adaptor. Intuitively, in response to an extra payload, the motors should raise the action scale to balance the additional weight. The plot shows that joints raise the motion amplitude at $5 \sim 10$ time steps and decrease the amplitude at around 20 time steps. The amplitude of the action residual of the $6th$ joint rises from about 0.25 to about 0.4, and the amplitude of the $12th$ joint raises from 0.4 to 0.55 which means that the adaption for an extra payload is correct.

Locomotion on Terrains **Answer the question Q1 and Q3**

Locomotion on flat ground is a straightforward task for learning-based quadrupedal locomotion, making it suitable for

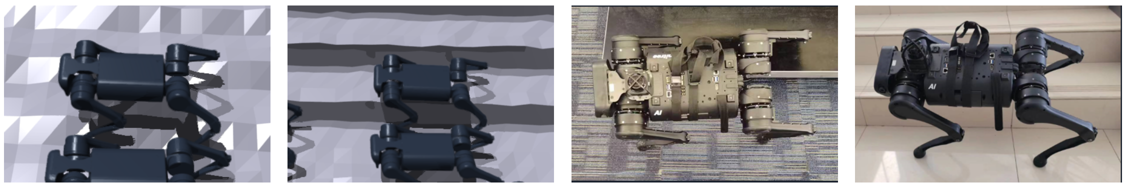

(a) 7*cm* Stairs (Simulation) (b) 17*cm* Stairs (Simulation) (c) 10*cm* Step (Real World) (d) 14*cm* Stairs (Real World)

Figure A7 The Environment of the Terrains with Different Sizes.

Table A4 The Success Rate of Robots Walking on Terrains in 10 Seconds for 100 Trials in the Simulation and 10 Trials in the Real World. We deploy only the policies with acceptable performance on a real quadrupedal robot to avoid hardware damage.

	Simulation				Real World	
	7*cm*	11*cm*	14*cm*	17*cm*	11*cm*	14*cm*
Expert	1	0.99	0.97	0.95	0	0
DR	0.97	0.76	0	0	0	0
SysID	0.93	0.83	0.70	0.52	0	0
DA	0.92	0.90	0.82	0.72	0.8	0
DABC (Naive)	0.87	0.81	0.64	0.51	0	0
DABC (Ours)	0.96	0.91	0.88	0.78	0.9	0.7

providing a quantitative analysis of all baselines. For locomotion on varied terrains, domain adaptors play a critical role in determining whether the trained terrain-aware locomotion policy can be successfully deployed on a real robot. We perform a qualitative analysis across different sizes of stairs in both simulation and the real world, as demonstrated in Figure A7.

(Simulation) The results are shown in Table A4. In easier terrains with heights lower than 11 cm, all methods can traverse the terrain with a success rate over 75%. However, when the terrain height increases to 14 cm, their performance drops drastically. Only the DA and our proposed method achieve a success rate higher than 75% on terrain with a height of 14 cm. In the simulation, most failures result from losing balance on the terrain, which demands strict performance from the domain adaptation module. The trajectories of DR, SysID, and our naive baselines fail to converge due to inaccurate perception of the environmental and motor dynamics. When the terrain height reaches 17 cm, the task becomes challenging for the robot due to its hardware design. For stable movement, the robot, with a leg length of 20 cm, tracks a height of around 26 cm from the ground, requiring it to raise its foot near the bottom of the trunk on 17 cm high stairs. Such movement demands an extremely accurate estimation of all domain factors, which is only feasible for the expert policy. Based on the results, we deployed the policies with a high success rate on challenging terrains, including DA and our method, on a real robot.

(Real World) In real-world experiments, most failures originate from the adaptors' inability to bridge the sim-to-real gap in actuator dynamics. While the output torque precisely follows commanded values in simulation, this correspondence fails under real motors. On stairs with a height of 14 cm, the DA policy attempts to raise the trunk height to traverse the stairs but fails to attain the desired position. Consequently, the DA policy performs cyclic actions: repeatedly trying to raise the height and then dropping back to the initial position. In contrast, our proposed method successfully handles the sim-to-real gap of the actuators, raising the trunk height sufficiently to navigate the stairs. However, our method also encounters issues with domain factors not modeled in the simulation. Specifically, we observed that the robot's elastic foot deforms laterally when attempting to walk on the stairs. Since the simulation models the foot as a rigid body, this deformation leads the robot to encounter states unseen in the simulation. The deformation manifests as unavoidable slippage during walking, causing instability in the locomotion policy. As the height of the stairs increases, the angle between the leg and the ground rises, resulting in greater deformation of the elastic foot.

References

1 Xuxin Cheng, Kexin Shi, Ananye Agarwal, and Deepak Pathak. Extreme parkour with legged robots. *arXiv preprint arXiv:2309.14341*, 2023.
2 Ashish Kumar, Zipeng Fu, Deepak Pathak, and Jitendra Malik. Rma: Rapid motor adaptation for legged robots. *arXiv preprint arXiv:2107.04034*, 2021.
3 Viktor Makoviychuk, Lukasz Wawrzyniak, Yunrong Guo, Michelle Lu, Kier Storey, Miles Macklin, David Hoeller, Nikita Rudin, Arthur Allshire, Ankur Handa, et al. Isaac gym: High performance gpu-based physics simulation for robot learning. *arXiv preprint arXiv:2108.10470*, 2021.
4 Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 23–30. IEEE, 2017.
5 Xingxing Wang. Unitree robotics. *URL https://www. unitree. com*, 2020.
6 Ruihan Yang, Ge Yang, and Xiaolong Wang. Neural volumetric memory for visual locomotion control. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1430–1440, 2023.

7 Wenhao Yu, Jie Tan, C Karen Liu, and Greg Turk. Preparing for the unknown: Learning a universal policy with online system identification. *arXiv preprint arXiv:1702.02453*, 2017.