Unsupervised Skill Discovery as Exploration for Learning Agile Locomotion

Seungeun Rho*, **Kartik Garg***, **Morgan Byrd, Sehoon Ha**
Georgia Institute of Technology
{srho31, kgarg65, abyrd45, sehoonha}@gatech.edu

Abstract: Exploration is crucial for enabling legged robots to learn agile locomotion behaviors that can overcome diverse obstacles. However, such exploration is inherently challenging, and we often rely on extensive reward engineering, expert demonstrations, or curriculum learning—all of which limit generalizability. In this work, we propose Skill Discovery As eXploration (**SDAX**), a novel learning framework that significantly reduces human engineering effort. SDAX leverages unsupervised skill discovery to autonomously acquire a diverse repertoire of skills for overcoming obstacles. To dynamically regulate the level of exploration during training, SDAX employs a bi-level optimization process that autonomously adjusts the degree of exploration. We demonstrate that SDAX enables quadrupedal robots to acquire highly agile behaviors including crawling, climbing, leaping, and executing complex maneuvers such as jumping off vertical walls. Finally, we deploy the learned policy on real hardware, validating its successful transfer to the real world.

Keywords: Unsupervised Skill Discovery, Exploration, Legged Locomotion

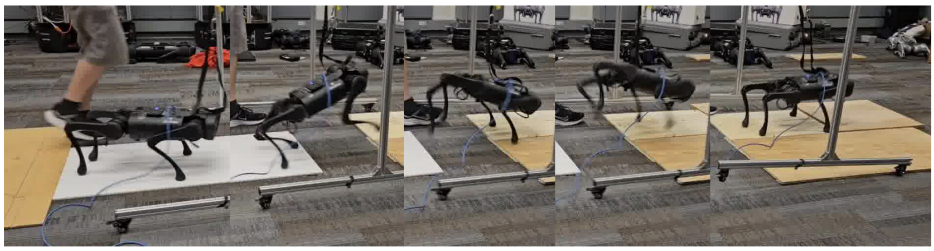


Figure 1: We deployed our policy on a real robot. The robot successfully leaps over the gap.

1 Introduction

In recent years, combining legged robot locomotion with deep reinforcement learning (deep RL) has led to remarkable advances in agility [1, 2, 3, 4, 5, 6, 7, 8]. However, existing methods often depend on additional techniques to master challenging skills, including: (1) reward engineering informed by domain expertise [9, 10, 11], (2) demonstration datasets [12, 13, 14, 15], and (3) carefully crafted curriculum learning [16]. In this work, we introduce **S**kill **D**iscovery **A**s e**X**ploration **(SDAX)**, a framework capable of solving highly challenging tasks—such as the *wall-jump* shown in Figure 9—by autonomously exploring a diverse set of strategies. SDAX leverages unsupervised reinforcement learning to achieve this, eliminating the need for manually designed curricula or demonstration data.

*Co-first authors. Project page - https://seungeunrho.github.io/projects/SDAX/

9th Conference on Robot Learning (CoRL 2025), Seoul, Korea.

Unsupervised RL offers a framework for learning diverse *skills*, where each skill corresponds to a highly correlated sequence of behaviors and is represented by a vector $z \in \mathbb{R}^n$. The intrinsic reward in unsupervised RL encourages the policy to exhibit different behaviors when conditioned on different skill vectors z, a process commonly referred to as *unsupervised skill discovery*. We harness this learning process to explore a wide range of high-level action strategies, ultimately enabling the emergence of agile behaviors necessary to solve challenging tasks.

In detail, SDAX combines two objectives: **solving the given task** and **finding diverse solutions**. Solving the task is represented by maximizing the task reward. The task reward is kept simple, such as following forward velocity commands to move toward task completion. On the other hand, exploring diverse behaviors is achieved by maximizing a diversity reward, which is derived from skill discovery methods. This encourages the agent to try various approaches to find the desired height, orientation, velocity, or angular velocity needed to solve the task. However, balancing two distinct objectives is not straightfoward and one may overpower the other. If the task reward dominates, agents may not sufficiently explore diverse behaviors. Conversely, if the diversity reward dominates, agents may spend too much time exploring, failing to solve the task. This is analogous to the exploration-exploitation trade-off in RL [17]. To address this problem, we introduce a learnable parameter λ to balance the two objectives. We train λ to automatically adjust the weight of the diversity reward to maximize the task reward.

In summary, SDAX aims to adopt skill discovery methods for high-level explorations to optimize the task-specific reward. The primary contributions of this work are as follows: (1) We propose a novel framework that combines RL and unsupervised skill discovery algorithms to automatically learn agile locomotion skills. (2) We provide a thorough derivation of a bi-level optimization framework for training the balancing parameter λ. We also demonstrate that SDAX of adapting λ robustly finds the optimal value for a given task. (3) We evaluate SDAX against manual exploration strategies on four challenging locomotion tasks: jumping, leaping, crawling, and a wall-jump.

2 Related Work

Unsupervised Skill Discovery. The goal of unsupervised skill discovery is to establish an association between a latent skill vector z and the resulting skill-conditioned policy $\pi(a|s, z)$. Two main families of approaches have been proposed for achieving this goal. The first family maximizes the mutual information between skills and states, $I(z; s)$. Examples include DIAYN [18] and VIC [19]. The second family maximizes the Wasserstein Dependency Measure, I_{WDM}, as seen in methods such as LSD [20], METRA [21], and LGSD [22]. Since SDAX is agnostic to the choice of skill discovery approach, we evaluate it using representative algorithms from both families: DIAYN and METRA.

Learning Agile Locomotion. Recently, learning-based methods have demonstrated highly agile locomotion capabilities such as high-speed running [23, 24], jumping [25, 26], and climbing [27, 28]. Our work aims to cover not only jumping, running, and leaping, but also *wall-jumping*, which involves a parkour-style motion combining flipping and jumping using walls.

The work most related to ours is that of Zhuang et al. [9], which used a manually designed reward that penalizes the overlap between the robot and imaginary obstacles. They trained agents to minimize these overlaps, resulting in the learning of agile behaviors. In contrast, we aim to train a similar set of tasks without the need for such reward designs. Instead, we allow an unsupervised RL method to discover the skills required to solve these tasks.

Skill Discovery for Locomotion. Recently, several approaches have incorporated skill discovery into locomotion training pipelines. Cheng et al. [29] used skill discovery methods to obtain diverse skills for solving tasks. A core difference is that they assume access to a near-optimal policy and value function from the start, and introduce skill discovery to diversify skills based on the given policy. In contrast, SDAX uses skill discovery in-the-loop to obtain an optimal policy for solving

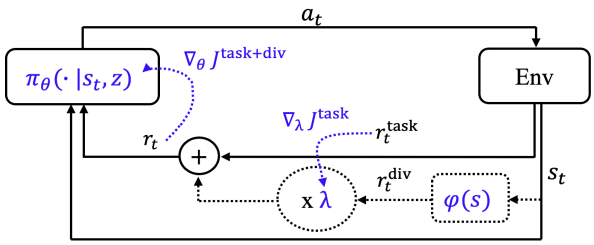


Figure 2: A figure of bi-level optimization for π_θ and λ. The task reward gives the gradient signal for training λ, and the sum of both sources of rewards provides the gradient signal for optimizing π_θ.

the task. Atanassov et al. [30] proposed replacing the objective of LSD with a "norm-matching" objective to obtain more diverse and controllable skills. Our aim is not to replace or outperform existing skill discovery methods, but rather to leverage them as a high-level exploration strategy to acquire agile behaviors.

3 Unsupervised Skill Discovery as Exploration

3.1 Problem Formulation

We regard the problem of training a control module for a legged robot as a Markov Decision Process (MDP) defined as $\mathcal{M} \equiv \{\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P}, \gamma\}$, where $\mathcal{S}$ is a state space, $\mathcal{A}$ is an action space composed of joint position targets for a PD controller of the robot, $\mathcal{R}$ is a reward function, $\mathcal{P}$ is a transition probability, and γ is a discount factor. The objective of RL is to obtain an optimal policy π which maximizes the expected sum of the discounted reward $J = \mathbb{E}_\pi\left[\sum_{t=0}^{\infty} \gamma^t r_t\right]$. π can be parameterized with the neural network θ, so here we denote policy as π_θ. However, instead of training a standard policy $\pi_\theta(a|s)$, we train a skill-conditioned policy $\pi_\theta(a|s, z)$, where z is randomly sampled from a fixed prior distribution, $z \sim p(z)$, for each episode and remains fixed throughout the episode.

3.2 Algorithm

Our objective is to find the policy parameter θ that optimizes the expected sum of both the task reward r^{task} and the diversity reward r^{div}.

$$\theta = \arg\max_\theta J^{\text{task+div}} = \arg\max_\theta \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{\infty} \gamma^t (r_t^{\text{task}} + \lambda r_t^{\text{div}})\right]$$

A learnable parameter λ determines the weight of r^{div}, and we refer to it as the balancing parameter. The task reward r_t^{task} specifies the goal of the task. It can be defined for each task and should be kept simple, such as a forward velocity tracking reward. Regardless of the value of λ, the policy π is always conditioned on a particular z. Conditioning the policy on different values of z results in different behaviors, so training a skill-conditioned policy with $\lambda = 0$ effectively means we are training a group of different policies, all of which converge into a single behavior. When λ becomes large, the diversity reward dominates, and each policy learns a distinct skill, but none of them are capable of solving the task. Thus, determining the appropriate value of λ is crucial. In the following paragraphs, we will explain how the balancing parameter λ is trained and how r^{div} is defined.

Train Balancing Parameter As depicted in the Figure 2, we utilize a bi-level optimization framework to train both policy π and a learnable balancing parameter λ, which is similar to LIRPG [31]. While θ is trained to maximize $J^{\text{task+div}}$, λ is trained to maximize only $J^{\text{task}} = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{\infty} \gamma^t r_t^{\text{task}}\right]$. It is worth noting that our ultimate goal is to solve the given task. So, the intuitive meaning of training λ solely depending on the task reward is that we determine the degree of diversity reward only

to maximize the task performance. Ideally, when the diversity reward helps solve the task, λ will be increased, and if it rather deters training, λ will be decreased.

More concretely,

$$\lambda = \arg\max_{\lambda} J^{\text{task}}. \tag{1}$$

The challenge here is that we cannot directly compute the gradient of J^{task} with respect to λ. To address this, we apply the chain rule to decompose the gradient as follows:

$$\nabla_\lambda J^{\text{task}} = \nabla_\theta J^{\text{task}} \nabla_\lambda \theta. \tag{2}$$

To make this expression tractable, it can be further expanded into the following final form:

$$\nabla_\lambda J^{\text{task}} \approx \alpha A^{\text{task}} \nabla_{\theta'} \log \pi_{\theta'}(a|s,z) \cdot A^{\text{div}} \nabla_\theta \log \pi_\theta(a|s,z). \tag{3}$$

Here, α is the learning rate, θ' denotes the parameter θ after a single update, and A^{task} and A^{div} denote the advantage values computed using r^{task} and r^{div}, respectively. The detailed derivation of Equation (3) is provided in Appendix A. Now we can directly compute this term using a sample-based approximation.

The intuitive meaning of this formula is that if the gradient vectors from the task reward and the diversity reward point in a similar direction, λ should be increased; otherwise, it should be decreased. The key difference between SDAX and Zheng et al. [31] is that instead of training the intrinsic reward function itself, we fix the intrinsic reward as the diversity reward, and we only train the balancing parameter λ to determine the degree of it.

Diversity Reward For the diversity reward r^{div}, we follow the formulation of METRA [21]. They train skills to maximize the Wasserstein Dependency Measure [32] $I_{\text{WDM}} = I_W(S;Z)$. Maximization of I_{WDM} can be translated into the following objective:

$$\sup_{\pi,\phi} \mathbb{E}_{P(\tau,z)} \left[\sum_{t=0}^{T-1} (\phi(s_{t+1}) - \phi(s_t))^T z \right] \text{ s.t. } \|\phi(s) - \phi(s')\|_2 \leq 1, \forall (s,s') \in \mathcal{S}_{\text{adj}},$$

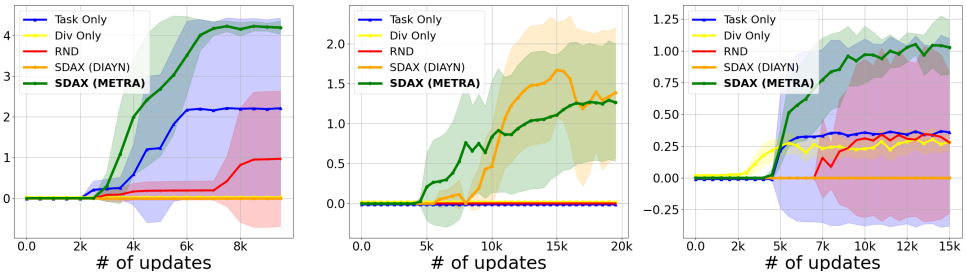
Here, $\phi : S \to Z$ is a learnable representation function that maps the state into a latent skill space. Optimization of this term can be achieved by simply using an off-the-shelf RL algorithm to maximize the reward $r^{\text{div}} = (\phi(s_{t+1}) - \phi(s_t))^T z$. To ensure that ϕ satisfies the constraint, we use dual gradient descent with a Lagrange multiplier κ with a small margin $\epsilon > 0$. Please refer to Park et al. [21] for more details.

Skill Selection A typical unsupervised skill discovery method requires careful selection of the optimal skill vector z during the testing phase. However, we observed that as training progresses, an increasing proportion of the learned skills exhibit successful behaviors, a phenomenon we refer to as "positive collapse"(Section 4.3). Therefore, in this work, we simply select a random skill z for reporting performance, rather than selectively choosing it or training a high-level controller.

Implementation Details We introduced two separate value networks, $v^{\text{task}}_{\psi_1}$ and $v^{\text{div}}_{\psi_2}$, due to the presence of two distinct reward sources: r^{task} and r^{div}. Using a single value network to model the value of $r^{\text{task}} + \lambda r^{\text{div}}$ led to unstable training, as the scale of the rewards varied with changes in λ. Pseudo-code for our algorithm is provided in the appendix D.1.

4 Experimental Results

In this section, we evaluate the proposed framework by training policies on a set of agile locomotion tasks. First, we examine three robot parkour learning tasks from Zhuang et al. [9], including leaping, climbing, and crawling, which require distinctive control strategies to overcome obstacles. On these tasks, we experiment with how skill discovery methods can aid in learning agile behaviors and evaluate SDAX against baselines.



(a) Leap (b) Climb (c) Crawl

Figure 3: Training curves, which denote the number of objects passed over the number of updates. Our method with METRA can solve all the tasks and exhibits better sample efficiency.

We use Isaac Gym [33] as the simulation engine, and our codebase builds on the work of Rudin et al. [34]. All experiments are conducted using the Unitree A1 robot. The gap for the leap task is 48cm, the platform height for the climb task is 25cm, and the gap height for the crawl task is 29cm. We adopt Proximal Policy Optimization (PPO) [35] as our main reinforcement learning algorithm, and details of the observation space are provided in Appendix D.3. Depending on the task, policies typically converge within 10k–20k iterations, requiring approximately 8–16 hours of training on an NVIDIA A40 GPU.

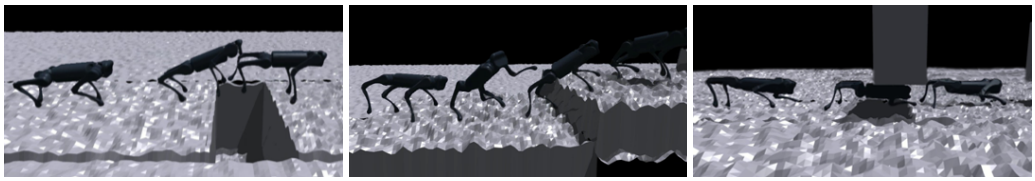
4.1 Learning Agile Locomotion Skills

We compared SDAX against the following baseline algorithms:

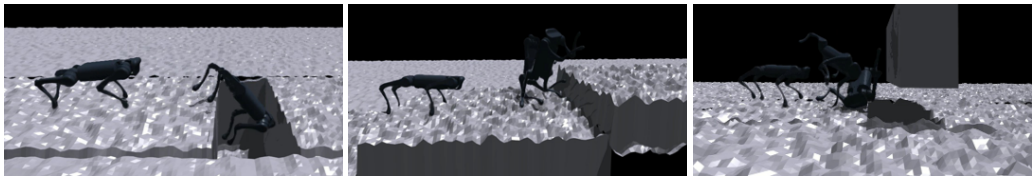
- *Task-only*: An RL baseline trained only with task specific rewards r^{task}.
- *Div-only*: An RL baseline trained using diversity reward r^{div} only.
- *RND*: It combines r^{task} with an exploration reward instead of a diversity reward.
- *SDAX with DIAYN*: r^{div} is computed with DIAYN reward.
- *SDAX with METRA*: r^{div} is computed with METRA reward.

We designed the same task reward across all baseline methods and tasks, with the primary goal of incentivizing agents to move forward. Details of the task rewards are provided in Appendix D.2. Since SDAX can be regarded as an exploration mechanism, we included RND [36], one of the most widely adopted exploration algorithms, as a baseline. For both the diversity reward and exploration bonus in RND, we manually specify sub-dimensions of the state space, ensuring that the learning process focuses on exploration within the specified sub-dimensions. Specifically, we selected base heights for climbing and crawling tasks and forward velocity for leaping. Additionally, to expedite the learning of the *Div-only* agent, we provided the robot's base x position as an additional input to the skill discovery algorithm. This facilitated the exploration of diverse x positions, ultimately helping the agent move forward.

SDAX enables learning the skills needed to solve each task. We present the training curves of SDAX and all baseline algorithms in Figure 3 over five different seeds. We measured the number of obstacles passed in each task, where each task contains three consecutive obstacles of the same configuration. SDAX successfully learned the necessary motor skills for all tasks. Compared to the *Task-only* baseline, we observed that incorporating diversity rewards helps in learning agile locomotion skills. However, relying solely on diversity rewards (*Div-only*) fails to achieve meaningful skills, highlighting that a balanced interplay between task and diversity rewards is critical for success. Additionally, a comparison with *RND* shows that diversity-based approaches outperform exploration-based rewards. We believe this is because naive exploration-based methods focus

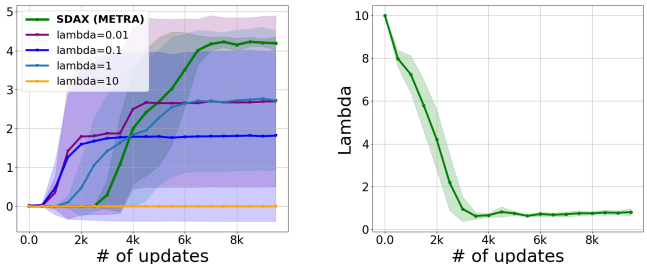


(a) Leap - success (b) Climb - success (c) Crawl - success



(d) Leap - failure (e) Climb - failure (f) Crawl - failure

Figure 4: Visualization of the diverse skills explored by the robot during training.



(a) Training curve (b) Corresponding curve of λ

Figure 5: SDAX outperforms all the baseline rewards with fixed value of lambda.

on state-level exploration, incentivizing agents to visit nearby unvisited states, making skill-level exploration challenging. In contrast, skill discovery methods inherently facilitate skill-level exploration, as they encourage skills to explore distinct sets of states, allowing agents to transition to entirely new regions. Lastly, it is worth noting that METRA outperforms DIAYN as a skill discovery module. This is because the diversity reward r^{div} from DIAYN can be maximized even with small differences between states, as long as the discriminator network can distinguish them, whereas METRA seeks diversity without saturation.

Skill discovery enables high level exploration. We also provide qualitative evidence demonstrating how skill discovery methods enhance exploration. Figure 4 illustrates example behaviors of SDAX using two different skills for each task based on an actual model checkpoint from training. To observe the behaviors of different skills, we kept the model fixed and fed different skill vectors to the policy. As a result, both successful and unsuccessful episodes were generated from the same policy, using different skill vectors. In the leaping task, some skills enabled the agent to powerfully kick off the ground, gaining enough height to clear the gap, while others resulted in weak jumps that led to failure. In the crawling task, certain skills lowered the robot's body posture to pass under the obstacle effectively, whereas others caused the robot to jump and lose balance. These examples illustrate that the agent explores a diverse range of heights—some of which solve the task while others do not. When a particular skill starts solving the task, the task reward increases, leading to successful task completion. In this sense, skill discovery functions as a high-level exploration module.

4.2 Learning Balancing Parameter λ

Selecting the appropriate value for λ is crucial, as the scale of both the task reward and diversity reward is difficult to determine a priori. If either the task reward or the diversity reward dominates,

Leap			**Climb**			**Crawl**		
3k	5k	10k	7k	10k	20k	5k	10k	15k
43.1±4.3	90.6±2.6	97.1±2.3	31.0±3.1	58.1±5.3	65.1±4.2	20.1±4.1	54.7±4.7	59.9±5.8

Table 1: Ratio of successful skill vectors z for each checkpoint (%). We randomly sampled 100 skills to measure success rate, and repeated ten times to determine the standard deviation.

the agent's learning process can be significantly hindered. In this section, we demonstrate how our algorithm effectively adjusts λ during training. We compare our adaptive approach to fixed values of λ, using four different settings: $0.01, 0.1, 1, 10$. These experiments were conducted on the leaping tasks from the previous section, with each method trained using three different random seeds. We measured performance based on the number of obstacles passed.

Our method outperforms fixed λ values. Figure 5(a) shows that our adaptive method outperforms all fixed-value experiments. SDAX demonstrated both superior sample efficiency and final performance compared to rest of the λ values. Figure 5(b) illustrates how the learned λ values evolve during training. The value starts at 10.0 and gradually decreases, suggesting that our algorithm learned that decreasing λ helps maximize task rewards over time.

It is also important to note that our method does not correspond to a single fixed λ value throughout training. In other words, there may not exist a single value of λ that could yield an identical training curve. SDAX adjusts λ dynamically, resulting in different values at different stages of training, which allows the agent to achieve an appropriate balance of diversity and task reward throughout the learning process.

4.3 Positive Collapse: Convergence of Skills into a Solution Space

One potential challenge of incorporating a skill discovery module into the learning process is the difficulty of selecting the exact skill that solves the task after training, especially if only a small portion of the skill space is effective. However, we observed that as training progresses, a growing number of skill vectors $z \sim \mathcal{N}(0, I)$ become capable of solving the task. To demonstrate this, we selected model checkpoints at various stages of training and measured the success rate out of random skill vectors. The results are presented in Table 1. For the leap task, initially, around 43% of the skills were successful, but this number eventually approached nearly 97%. Similarly, for the climb and crawl tasks, the proportion of successful skills increased steadily. This helps evaluation of the policy with right z, as presented in appendix B.

This suggests that once a viable solution is discovered, different skill vectors converge into similar behaviors with the solution. This contrasts with a typical skill discovery scenario where only a small subset of skills solves the task. We observe that this phenomenon of later convergence is facilitated by task rewards: when a skill finds a successful solution, the corresponding trajectory receives higher rewards, which results in the increased probability of the corresponding actions taken. Because all skills share the same policy network, this learning propagates to other skill-conditioned behaviors, leading to what we term a "positive collapse" of skills. This is beneficial because it mitigates the issue of selecting the right skill.

4.4 Wall-jump : Learning Super Agile Tasks

Lastly, we pushed our method to its limits by introducing a new task named *wall-jump*. It requires the robot to perform a sequence of highly agile motions, including running, jumping, flipping, and landing in a specific order. To make this feasible, we devised a guideline-based reward that is widely adopted in robotics[37, 38]. The reward encourages the agent to follow the guideline specified by a user. We used this reward as r^{task}. More details about the reward design can be found in Appendix E.1. The exact guideline is shown in Figure 9(a) in Appendix. Note that the guideline only provides the target trajectory for the root position while not offering any information about orientation.

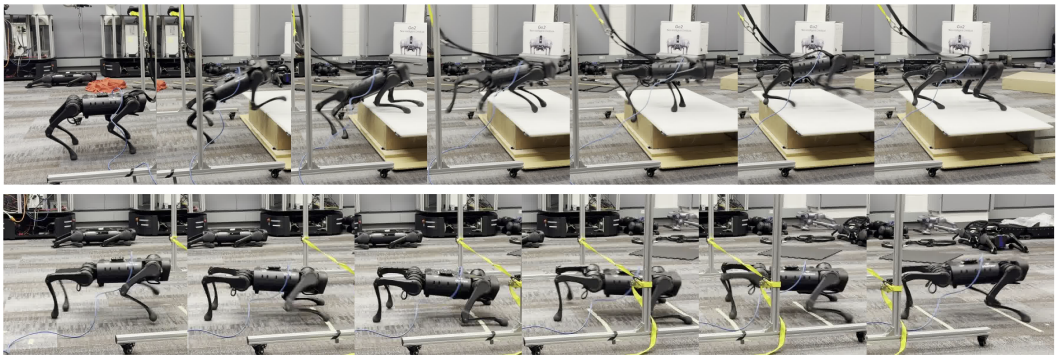


Figure 7: Both the climbing (top) and crawling (bottom) policies' skills were tested on the real robot.

However, providing the guideline alone was not sufficient for the agent to successfully perform the wall-jump. Figure 9(b) shows the resulting behavior of the agent trained solely with r^{task}. The robot was able to follow the guideline up until it reached the perpendicular wall, but then crashed its back against the wall. The cumulative reward for this episode was about 5.0, as shown by the blue curve in Figure 9(e). We observe that the robot needs to acquire a specific orientation to kick off the wall and land safely.

Therefore, we provided the robot's base's `roll`, `pitch`, and `yaw` as input to the skill discovery algorithm, allowing our method to explore and learn diverse orientations of the robot when needed. Figures 9(c) and (d) show the resulting behavior. Our method was able to acquire the specific orientation needed to kick off the wall. As a result, SDAX achieved a successful wall-jump (Figure 6) with a much higher task return of 9.5 as indicated by the green curve in Figure 9(e).

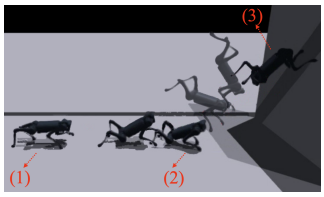


Figure 6: Wall-jump.

4.5 Hardware Experiments

After successfully training a policy that solves the task in simulation, we fine-tune it for real-world deployment by introducing observation noise and domain randomization [39]. Specifically, we continue training for an additional 5,000 steps with these modifications to improve robustness. Full details of the noise and randomization parameters are provided in Appendix D.4 and Appendix D.5. To evaluate whether the discovered skills transfer to the real world, we deploy the policy on a Unitree A1 robot. As shown in Figure 1 and Figure 7, the robot successfully performs agile maneuvers such as leaping over a 46 cm gap, climbing a 25 cm platform, and crawling under a 27 cm-high obstacle. Furthermore, as illustrated in Figure 8 in Appendix, the crawling policy remains robust even under varying terrain conditions. Please refer to the supplemental video for more details.

5 Conclusion

In this work, we introduced a novel learning framework that integrates unsupervised skill discovery with reinforcement learning to enable legged robots to acquire highly agile locomotion behaviors without relying on demonstration data or curriculum design. By balancing skill-level exploration and task rewards through a bi-level optimization process, our method allows robots to discover diverse behaviors such as crawling, climbing, leaping, and executing agile maneuvers like wall-jumping. We further demonstrate that the policy learned through our framework can be successfully deployed on real-world hardware.

6 Limitations

While our work proposes a novel training framework, it comes with certain limitations. First, effective training requires manual specification of sub-dimensions of the state space to guide exploration. For instance, to induce crawling behavior, we assume that exploring different body heights is essential, and therefore explicitly use height as an input to the diversity objective.

Another empirical observation is that it was better to add observation noise after learning a successful policy using our main algorithm. We observed that applying excessive observation noise during skill discovery makes training unstable. This is because the diversity reward relies on distinguishing newly visited states, and noise can obscure meaningful differences, confusing the reward signal. To address this, we first train the skill discovery module under low-noise conditions, and once effective skills are acquired, we fine-tune the policy in a second phase with higher observation noise. We believe that improving the robustness of the skill discovery module—particularly its ability to operate under noisy observations or heavy domain randomization—would further enhance the applicability and reliability of SDAX.

Acknowledgments

This research has been funded by the Industrial Technology Innovation Program (P0028404, development of a product level humanoid mobile robot for medical assistance equipped with bidirectional customizable human-robot interaction, autonomous semantic navigation, and dual-arm complex manipulation capabilities using large-scale artificial intelligence models) of the Ministry of Industry, Trade and Energy of Korea.

References

[1] J. Tan, T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez, and V. Vanhoucke. Sim-to-real: Learning agile locomotion for quadruped robots. *arXiv preprint arXiv:1804.10332*, 2018.

[2] J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter. Learning quadrupedal locomotion over challenging terrain. *Science robotics*, 5(47):eabc5986, 2020.

[3] J. Hwangbo, J. Lee, A. Dosovitskiy, D. Bellicoso, V. Tsounis, V. Koltun, and M. Hutter. Learning agile and dynamic motor skills for legged robots. *Science Robotics*, 4(26):eaau5872, 2019.

[4] Z. Xie, X. Da, M. Van de Panne, B. Babich, and A. Garg. Dynamics randomization revisited: A case study for quadrupedal locomotion. pages 4955–4961, 2021.

[5] X. Song, Y. Yang, K. Choromanski, K. Caluwaerts, W. Gao, C. Finn, and J. Tan. Rapidly adaptable legged robots via evolutionary meta-learning. pages 3769–3776, 2020.

[6] T. Haarnoja, S. Ha, A. Zhou, J. Tan, G. Tucker, and S. Levine. Learning to walk via deep reinforcement learning. *arXiv preprint arXiv:1812.11103*, 2018.

[7] L. Smith, J. C. Kew, T. Li, L. Luu, X. B. Peng, S. Ha, J. Tan, and S. Levine. Learning and adapting agile locomotion skills by transferring experience. *arXiv preprint arXiv:2304.09834*, 2023.

[8] S. Luo, S. Li, R. Yu, Z. Wang, J. Wu, and Q. Zhu. Pie: Parkour with implicit-explicit learning framework for legged robots. *IEEE Robotics and Automation Letters*, 2024.

[9] Z. Zhuang, Z. Fu, J. Wang, C. Atkeson, S. Schwertfeger, C. Finn, and H. Zhao. Robot parkour learning. In *Conference on Robot Learning (CoRL)*, 2023.

[10] X. Cheng, K. Shi, A. Agarwal, and D. Pathak. Extreme parkour with legged robots. *arXiv preprint arXiv:2309.14341*, 2023.

[11] Y. Yang, G. Shi, X. Meng, W. Yu, T. Zhang, J. Tan, and B. Boots. Cajun: Continuous adaptive jumping using a learned centroidal controller. *arXiv preprint arXiv:2306.09557*, 2023.

[12] M. Bogdanovic, M. Khadiv, and L. Righetti. Model-free reinforcement learning for robust locomotion using demonstrations from trajectory optimization. *Frontiers in Robotics and AI*, 9:854212, 2022.

[13] O. Kilinc and G. Montana. Reinforcement learning for robotic manipulation using simulated locomotion demonstrations. *Machine Learning*, pages 1–22, 2022.

[14] C. Li, M. Vlastelica, S. Blaes, J. Frey, F. Grimminger, and G. Martius. Learning agile skills via adversarial imitation of rough partial demonstrations. In *Conference on Robot Learning*, pages 342–352. PMLR, 2023.

[15] Z. He, K. Lei, Y. Ze, K. Sreenath, Z. Li, and H. Xu. Learning visual quadrupedal loco-manipulation from demonstrations. *arXiv preprint arXiv:2403.20328*, 2024.

[16] A. Kumar, Z. Fu, D. Pathak, and J. Malik. Rma: Rapid motor adaptation for legged robots. *arXiv preprint arXiv:2107.04034*, 2021.

[17] R. S. Sutton. Reinforcement learning: An introduction. *A Bradford Book*, 2018.

[18] B. Eysenbach, A. Gupta, J. Ibarz, and S. Levine. Diversity is all you need: Learning skills without a reward function. *arXiv preprint arXiv:1802.06070*, 2018.

[19] K. Gregor, D. J. Rezende, and D. Wierstra. Variational intrinsic control. *arXiv preprint arXiv:1611.07507*, 2016.

[20] S. Park, J. Choi, J. Kim, H. Lee, and G. Kim. Lipschitz-constrained unsupervised skill discovery. *arXiv preprint arXiv:2202.00914*, 2022.

[21] S. Park, O. Rybkin, and S. Levine. Metra: Scalable unsupervised rl with metric-aware abstraction. *arXiv preprint arXiv:2310.08887*, 2023.

[22] S. Rho, L. Smith, T. Li, S. Levine, X. B. Peng, and S. Ha. Language guided skill discovery. *arXiv preprint arXiv:2406.06615*, 2024.

[23] G. B. Margolis, G. Yang, K. Paigwar, T. Chen, and P. Agrawal. Rapid locomotion via reinforcement learning. *arXiv preprint arXiv:2205.02824*, 2022.

[24] Z. Fu, A. Kumar, J. Malik, and D. Pathak. Minimizing energy consumption leads to the emergence of gaits in legged robots. *arXiv preprint arXiv:2111.01674*, 2021.

[25] Z. Li, X. B. Peng, P. Abbeel, S. Levine, G. Berseth, and K. Sreenath. Robust and versatile bipedal jumping control through multi-task reinforcement learning. *arXiv preprint arXiv:2302.09450*, 2023.

[26] Y. Yang, X. Meng, W. Yu, T. Zhang, J. Tan, and B. Boots. Continuous versatile jumping using learned action residuals. In *Learning for Dynamics and Control Conference*, pages 770–782. PMLR, 2023.

[27] N. Rudin, D. Hoeller, M. Bjelonic, and M. Hutter. Advanced skills by learning locomotion and local navigation end-to-end. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2497–2503. IEEE, 2022.

[28] J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter. Learning quadrupedal locomotion over challenging terrain. *Science robotics*, 5(47):eabc5986, 2020.

[29] J. Cheng, M. Vlastelica, P. Kolev, C. Li, and G. Martius. Learning diverse skills for local navigation under multi-constraint optimality. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5083–5089. IEEE, 2024.

[30] V. Atanassov, W. Yu, A. L. Mitchell, M. N. Finean, and I. Havoutis. Constrained skill discovery: Quadruped locomotion with unsupervised reinforcement learning. *arXiv preprint arXiv:2410.07877*, 2024.

[31] Z. Zheng, J. Oh, and S. Singh. On learning intrinsic rewards for policy gradient methods. *Advances in Neural Information Processing Systems*, 31, 2018.

[32] S. Ozair, C. Lynch, Y. Bengio, A. Van den Oord, S. Levine, and P. Sermanet. Wasserstein dependency measure for representation learning. *Advances in Neural Information Processing Systems*, 32, 2019.

[33] V. Makoviychuk, L. Wawrzyniak, Y. Guo, M. Lu, K. Storey, M. Macklin, D. Hoeller, N. Rudin, A. Allshire, A. Handa, and G. State. Isaac gym: High performance gpu-based physics simulation for robot learning, 2021.

[34] N. Rudin, D. Hoeller, P. Reist, and M. Hutter. Learning to walk in minutes using massively parallel deep reinforcement learning. In *Conference on Robot Learning*, pages 91–100. PMLR, 2022.

[35] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[36] Y. Burda, H. Edwards, A. Storkey, and O. Klimov. Exploration by random network distillation. *arXiv preprint arXiv:1810.12894*, 2018.

[37] Z. Tang, D. Kim, and S. Ha. Learning agile motor skills on quadrupedal robots using curriculum learning. In *International Conference on Robot Intelligence Technology and Applications*, volume 3, 2021.

[38] J. Gu, S. Kirmani, P. Wohlhart, Y. Lu, M. G. Arenas, K. Rao, W. Yu, C. Fu, K. Gopalakrishnan, Z. Xu, et al. Rt-trajectory: Robotic task generalization via hindsight trajectory sketches. *arXiv preprint arXiv:2311.01977*, 2023.

[39] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. 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.

[40] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12, 1999.

[41] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[42] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

[43] D.-A. Clevert, T. Unterthiner, and S. Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

A Proof of Equation 3

We begin with the Equation (1), which is the decomposition of $\nabla_\lambda J^{\text{task}}$ using the chain rule,

$$\nabla_\lambda J^{\text{task}} = \nabla_{\theta'} J^{\text{task}} \nabla_\lambda \theta'.$$

Here, we can compute the first term $\nabla_{\theta'} J^{\text{task}}$ using the policy gradient theorem [40]

$$\nabla_{\theta'} J^{\text{task}} \approx A^{\text{task}} \nabla_{\theta'} \log \pi_{\theta'}(a|s,z) \tag{4}$$

To compute the second term $\nabla_\lambda \theta'$, we first derive θ'

$$\begin{aligned} \theta' &= \theta + \alpha \nabla_\theta J^{\text{task+div}}(\theta) \\ &= \theta + \alpha A^{\text{task+div}} \nabla_\theta \log \pi_\theta(a|s,z) \end{aligned} \tag{5}$$

Here, $\alpha \in \mathbb{R}$ is a learning rate. Then we can plug in this result to compute $\nabla_\lambda \theta'$:

$$\begin{aligned} \nabla_\lambda \theta' &= \nabla_\lambda (\theta + \alpha A^{\text{task+div}} \nabla_\theta \log \pi_\theta(a|s,z)) \\ &= \nabla_\lambda (\alpha A^{\text{task+div}} \nabla_\theta \log \pi_\theta(a|s,z)) \\ &= \nabla_\lambda (\alpha A^{\text{task}} + \alpha\lambda A^{\text{div}}) \nabla_\theta \log \pi_\theta(a|s,z) \\ &= \alpha A^{\text{div}} \nabla_\theta \log \pi_\theta(a|s,z) \end{aligned} \tag{6}$$

Finally, we can compute the value of $\nabla_\lambda J^{\text{task}}$ by plugging in the Eq. (4) and Eq. (6):

$$\nabla_\lambda J^{\text{task}} \approx A^{\text{task}} \nabla_{\theta'} \log \pi_{\theta'}(a|s,z) * \alpha A^{\text{div}} \nabla_\theta \log \pi_\theta(a|s,z) \tag{7}$$

This concludes the derivation of the equation 3.

B Evaluation results after training

Table 2: Number of obstacles passed (our of three)

Methods	Leap	Crawl	Climb
Task Only	0.00 ± 0.0	1.00 ± 1.4	0.00 ± 0.0
Robot Parkour Learning	2.23 ± 0.8	2.12 ± 1.3	2.82 ± 0.4
SDAX (random z)	2.34 ± 1.1	1.84 ± 1.5	1.72 ± 1.5
SDAX (fixed z)	$\mathbf{2.85 \pm 0.5}$	$\mathbf{3.00 \pm 0.0}$	$\mathbf{2.93 \pm 0.3}$

Thanks to *positive collapse*, increasingly larger portions of z can solve the task as training progresses. This allows us to reliably find a working z by randomly sampling, for example, 100 vectors, evaluating their outcomes, and selecting the best-performing one. In Table 2, we present the evaluation results after training is completed.

For "SDAX (fixed z)", we first selected the best-performing z from 100 randomly sampled vectors, then re-evaluated this z. For "SDAX (random z)," we sampled z at the beginning of each episode. We also included Robot Parkour Learning (RPL) [9] as a baseline. For RPL, we kept the total number of updates the same as in our method, with the first 70% of training on soft dynamics and the remaining 30% fine-tuning on hard dynamics. The table reports the average number of obstacles passed over 100 episodes after training is completed.

C Crawl on Diverse Terrain

To evaluate the robustness of the crawling policy trained with our skill discovery approach, we deployed it on two different terrains: wood and a rubber mat. As shown in Figure 8, the robot successfully crawled under the obstacle in both settings, demonstrating its ability to generalize across varying surface conditions.

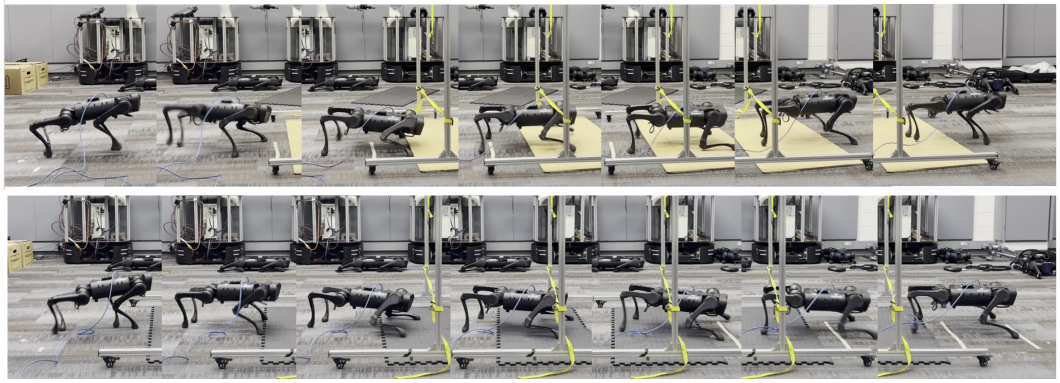


Figure 8: The crawling policy demonstrates robustness by successfully navigating under obstacles on different terrains: wood (top) and rubber mat (bottom).

D Implementation Detail

D.1 Algorithm

Algorithm 1

Initialize skill-conditioned policy $\pi_\theta(a|s,z)$, value functions $v^{\text{task}}_{\psi_1}$ and $v^{\text{div}}_{\psi_2}$, representation function $\phi(s)$, Lagrange multiplier κ, Balancing parameter λ, data buffer $\mathcal{D}$
for $i \leftarrow 1$ to # of epochs **do**
  for $j \leftarrow 1$ to # of episodes per epoch **do**
    Sample skill $z \sim \mathcal{N}(0, I)$
    while episode not terminates **do**
      Sample action $a \sim \pi(a|s,z)$
      Execute a and receive s' and r^{task}
      Compute $r^{\text{div}} = (\phi(s') - \phi(s))^T z$
      Add $\{s, a, r^{\text{task}}, r^{\text{div}}, s'\}$ to data buffer $\mathcal{D}$
    end while
  end for
  for $\{s, a, r^{\text{task}}, r^{\text{div}}, s'\}$ in $\mathcal{D}$ **do**
    Update $\phi(s)$ to maximize $\mathbb{E}_{(s,z,s')\sim\mathcal{D}}\left[(\phi(s') - \phi(s))^T z + \kappa \cdot \min(\epsilon, 1 - \|\phi(s) - \phi(s')\|_2^2)\right]$
    Update κ to minimize $\mathbb{E}_{(s,z,s')\sim\mathcal{D}}\left[\kappa \cdot \min(\epsilon, 1 - \|\phi(s) - \phi(s')\|_2^2)\right]$
    Update θ using PPO with reward $r = r^{\text{task}} + \lambda * r^{\text{div}}$
    Update ψ_1 and ψ_2 using r^{task} and r^{div} respectively
    Update λ using Eq. 3
  end for
end for

D.2 Task reward detail

The first three terms about tracking commands specify the goal of the task, while the other three terms regularize unrealistic, infeasible motions.

Table 3: Task rewards

Name	Mathematical Expression	Coefficients value
Tracking angular velocity	$e^{-\lvert w_{yaw}\rvert}$	0.05
Tracking linear velocity	$\lvert v_x - v_x^{target}\rvert$	-1
Alive	-	2
Torque squared	$\sum_{j \in \text{joints}} \lvert\tau_j \dot{q}_j\rvert^2$	-1e-6
Exceed dof pos limits	$\sum_{j \in \text{joints}} \max(\lvert\text{dof}_j\rvert - \text{dof}_{\text{lim}}, 0)$	-0.1
Exceed torque limits	$\sum_{j \in \text{joints}} \max(\lvert\tau_j\rvert - \tau_{\text{lim}}, 0)$	-0.2

D.3 Observation space

Table 4: A1 Robot Observations

Name	Description	Dimension
Base position	x,y,z position of the robot's base	3
Base rotation	Yaw, Pitch, Roll of robot's base	3
Base velocity	Velocity of robot's base in x,y,z direction	3
Base ang vel	Angular velocity of robot's base	3
Gravity projection	Vector indicates direction of the gravity	3
Velocity command	Velocity command given by users	3
DOF position	Current angle of each DOF	12
DOF velocity	Angular velocity of each DOF	12
Previous action	Action executed in previous step	12
Distance to obstacle	Distance to obstacle	1
Obstacle properties	Difficulty and length of the obstacle	2
Obstacle info	One hot encoding to identify the obstacle type	5
Sidewall distance	Distance to side wall	2
Sampled skill	Sampled skill for current episode	1
Sum		65

In the hardware experiments, we used a motion capture system to obtain global measurements such as the robot's base position and velocity. We also experimented with including certain robot configuration parameters—such as mass and motor strength—as part of the observation. Since these parameters are used in domain randomization, providing them could help the policy to be aware of the current configuration. However, given that training was successful both with and without these configuration inputs in both simulation and real-world settings, we conclude that this information is not critical for policy performance.

D.4 Observation noise

Table 5: Observation noise coefficients per task

Obs	Crawl	Leap	Jump
Base position	$\pm$0.1	$\pm$0.1	$\pm$0.1
Base rotation	$\pm$0.1	$\pm$0.1	$\pm$0.1
Base lin vel	$\pm$0.1	$\pm$0.1	$\pm$0.1
Base ang vel	$\pm$0.2	$\pm$0.2	$\pm$0.2
DOF position	$\pm$0.01	$\pm$0.01	$\pm$0.01
DOF velocity	$\pm$1.5	$\pm$1.5	$\pm$1.5
Gravity projection	$\pm$0.05	$\pm$0.05	$\pm$0.05
Distance to obstacle	$\pm$0.0	$\pm$0.05	$\pm$0.05

D.5 Domain randomization parameters

Table 6: Domain Randomization Parameters

Parameter	Range	Form
Mass	[-1.0, 3.0]	Additive
Friction	[0.0, 2.0]	Multiplicative
Motor strength	[0.9, 1.1]	Multiplicative
Init base x position	[0.2, 0.6]	Additive
Init base y position	[-0.25, 0.25]	Additive
Init DOF position range	[0.5, 1.5]	Multiplicative

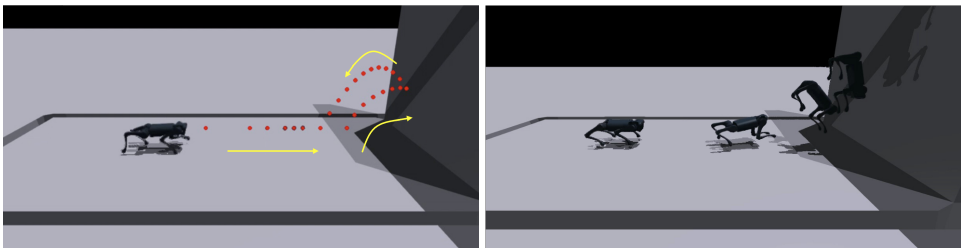
Domain randomization parameters were sampled from a uniform distribution with the range above.

D.6 Hyperparameters

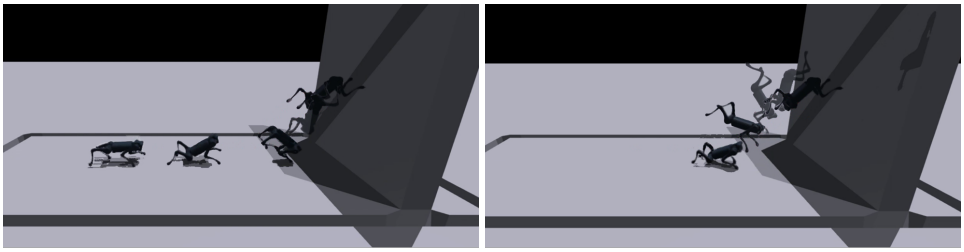
Table 7: Hyperparameters of our method

Name	Value
Learning rate	0.0005
Optimizer	Adam[41]
PPO clip threshold	0.2
PPO number of epochs	5
GAE λ [42]	0.95
Discount factor γ	0.99
Horizon length	24
Entropy coefficient	0.001
Policy network π	MLP with [512, 256, 128],
Activation of π	ELU[43]
Value network υ	MLP with [512, 256, 128]
Activation of υ	ELU[43]
Representation function ϕ from METRA	MLP with [256, 256, 256]
Activation of ϕ	ReLU
Initial Lagrange coefficient κ from METRA	30

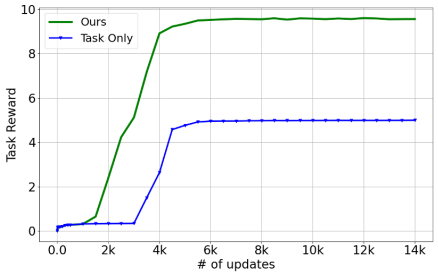
E Details of the Wall-jump experiments



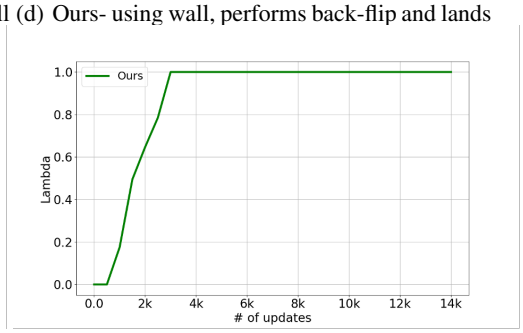
(a) Red dots - Human drawn guidelines (b) *Task-only* makes robots crash



(c) Ours- A robot runs and front-flips to kick wall (d) Ours- using wall, performs back-flip and lands



(e) Training curves



(f) λ curve

Figure 9: Our method enables robots to solve the wall-jump task.

E.1 Details of the guideline following reward

For the wall-jump task, we defined a special task reward, r^{task}, based on a guideline provided by a human. The guideline consists of a sequence of n points:

$$g_{i=0,1,\ldots,n-1} \in \mathbb{R}^3$$

Let the robot's base position in global 3D space be denoted as $\boldsymbol{x} \in \mathbb{R}^3$. At each time step, the robot has a target point g_i, starting with g_0. When the robot reaches the current target, it moves on to the next target, g_{i+1}. A target is considered reached when the distance between $\boldsymbol{x}$ and g_i falls below a threshold $h \in \mathbb{R}$, i.e., $||\boldsymbol{x} - g_i||_2 < h$.

Then, the reward can be defined as follows:

$$r_t = e^{-||\boldsymbol{x}-g_i||_2}$$

This term has the desirable property of being bounded between 0 and 1. It approaches 0 when the robot is infinitely far from the current target and becomes 1 when the robot exactly reaches the target. This property contributes to stability during the learning process. We optimized this reward using reinforcement learning (RL) to train the agent to follow the given guideline.