Physical State Exploration for Reinforcement Learning from Scratch

Allison Pinosky, Thomas A. Berrueta, Olivia Li, and Todd D. Murphey

***Abstract*— Although reinforcement learning (RL) algorithms have demonstrated impressive capabilities in simulation, their transition into the real-world often reveals a performance gap. A key challenge of real-world deployment is ensuring robustness to complex or unmodeled physical phenomena, such as anisotropic friction during locomotion. This discrepancy between simulated and real-world performance underscores the need for hardware benchmarks to rigorously evaluate and improve RL algorithms for physical deployment. In this work, we present NoodleBot—a low-cost, untethered three-link swimmer robot—as a hardware benchmark for RL algorithms. This benchmark is intended to complement embodied learning approaches, where simulations provide confidence that algorithms are able to learn from scratch in challenging environments. We demonstrate three algorithms learning on the platform and compare the results to learning with a simulated swimmer, highlighting the importance of effective state exploration to agent performance. We also show the ability of one algorithm to learn in single-shot hardware deployments. Design, firmware, and software are available open source at `https://github.com/MurpheyLab/NoodleBot`.**

***Index Terms*— Deep Learning in Robotics and Automation, Reinforcement, Machine Learning**

I. Introduction

Reinforcement learning (RL) agents learn to solve tasks by exhaustively sampling data through sequential interactions with their environment. The unsupervised, trial-and-error nature of RL makes it a promising approach to solve complex real-world tasks with minimal human intervention. However, the success of most RL agents has been limited to simulated environments [1], where high-quality data can be sampled across massively parallelized instances of a task in simulation. For this reason, widespread adoption of RL methods in the real-world has been limited, with a recent survey citing sample efficiency, safety, interpretability, and hardware validation of these techniques as key outstanding roadblocks [2]. In the real-world, RL agents must gather experience through embodied interactions with their environment. This introduces additional challenges and constraints that are difficult to reproduce in simulation and can have a negative impact on the learning process. Importantly, since sampling data during real-world deployment takes time, learning some tasks can be costly and time consuming [3]—leading practitioners to consider alternative techniques.

Two common algorithmic approaches to address the challenges of real-world data collection include learning from demonstrations [4]–[7] and sim-to-real transfer [8]–[11].

Authors are with the Department of Mechanical Engineering, Northwestern University, Chicago, IL, USA. Contact: {apinosky, tberrueta, oliviali2026}@u.northwestern.edu, t-murphey@northwestern.edu.

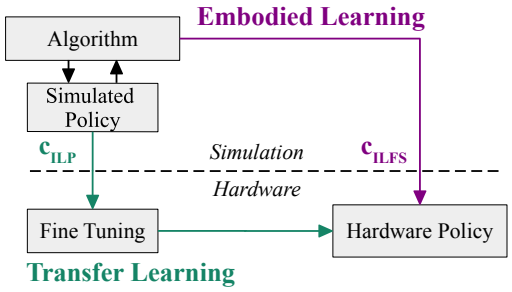

Fig. 1. Simulations Build Confidence. Transfer learning (green) assumes simulations closely mimic the real-world, and *confidence in the learned policy* (c_{ILP}) directly provides confidence to fine-tune in hardware. With embodied learning (purple), simulations stress the algorithm's ability to learn in challenging environments, and *confidence in learning from scratch* (c_{ILFS}) provides confidence to directly learn hardware policies.

However, the success of these methods depends on the quality of the demonstrations and simulators respectively. Moreover, simulators often incur expensive computational costs—especially when the task involves complex physical phenomena, such as granular media [12] and anisotropic frictional interfaces [13]—and even the most sophisticated simulators fail to capture the richness of real-world physics beyond the most trivial environments. As a result, sim-to-real transfer agents still experience a degradation in policy performance referred to as the "reality gap" [14]. Therefore, there is a need for embodied RL algorithms capable of directly adapting to and navigating the complexities of real-world environments. As shown in Fig. 1, simulation still has a role to play in embodied learning. However, the primary role of simulation is to build confidence that embodied algorithms are capable of learning from scratch in challenging environments.

To mitigate the cost of real-world data collection, researchers have sought to develop sample efficient RL algorithms by promoting exploration during learning [15]. Of the many exploration approaches that have shown promise (e.g., [16]–[18]), those based on the principle of maximum entropy (MaxEnt) have attained state-of-the-art performance across simulated robotics benchmarks by maximizing policy entropy as a means of promoting sample efficiency and effective exploration [19], [20]. Other researchers have extended these methods to incorporate neural model ensembles during hardware learning, leveraging access to additional compute to improve sample efficiency [21]. However, recent work in maximum diffusion (MaxDiff) RL has illustrated that policy entropy maximization can be an insufficient proxy for state exploration in some tasks [22]. A limitation of many RL

algorithms—including MaxEnt approaches—is that training is highly sensitive to initialization and algorithms exhibit varied performance across agent experiences. In contrast, MaxDiff has been shown to outperform MaxEnt techniques across several simulated robotics benchmarks by optimizing state transition entropy, highlighting the importance of effective state exploration on performance, sample efficiency, and robustness.

Despite the advances in sample efficient RL approaches, very few methods are ultimately validated in hardware [2]. Part of the reason for this is the lack of standardized, cost-effective, hardware robotics benchmarks similar to the MuJoCo task suite [23]. To this end, there is a pressing need for the development of hardware robotic benchmarks to evaluate and improve RL algorithms under realistic conditions. Real-world deployment of RL has been hindered by the large gap between simulated benchmarks and practical applications, but hardware benchmarks provide a path to close this gap. Prior work on RL hardware benchmarks have largely focused on manipulation [24]–[28] and legged locomotion [28]–[30]. One downside to many of these approaches is that they rely on expensive robot hardware. Even encouraging recent work on the lower-cost end (e.g., [27], [28]) still cost upwards of $3.5k in component costs, so there is still a need for standardized, low-cost, open-source, hardware learning benchmarks in RL. Furthermore, these hardware benchmarks minimize contact area by design to simplify the learning problem, but many real-world tasks we ultimately want to solve require prolonged interaction with the environment.

In this work, we introduce NoodleBot—a 3-link "swimmer" robot (see [31]) designed to serve as a low-cost hardware benchmark for advancing the development of RL algorithms in physical environments (see Fig. 2). In contrast to prior swimmer platforms, the design of NoodleBot is not based on principles like bioinspiration [32]–[34] or modularity [35], [36]. Instead, we aim to present a simple, open-source design that captures many of the complexities inherent to physical systems for approximately $100. Moreover, due to the intrinsic stability of planar swimmer locomotion, NoodleBot presents a compelling platform for tasks requiring long planning horizons.

The contributions of this work can be summarized as follows:

- We introduce the design of a hardware swimmer benchmark for RL.
- We show that state exploration improves task performance both in simulation and in hardware.
- We demonstrate the ability of one algorithm to autonomously learn in single-shot hardware deployments.

The paper is organized as follows. Section II provides background on the tested RL algorithms. Section III presents the robot design. Results from simulations and real-world experiments are presented in Section IV, and finally Section V provides conclusions.

II. Reinforcement Learning Approach

In this work, we benchmark the performance of three deep RL algorithms in hardware. Two of the algorithms belong to the MaxEnt RL family of techniques, which decorrelate an agent's action sequences by adding policy entropy to the rewards in the following objective,

$$\underset{\pi}{\operatorname{argmax}}\ \mathbb{E}_{(s_t,a_t)\sim p,\pi}\left[\sum_{t=1}^{T} r(s_t,a_t)+\alpha\mathcal{H}_\pi[a_t|s_t]\right], \quad (1)$$

where π is the policy, s is the robot state, a is the robot action, $\mathcal{H}_\pi[a_t|s_t]$ denotes the entropy of the policy conditioned on the current state, r is the predicted reward, and α is a temperature parameter. We use Soft Actor-Critic (SAC) [19] to train a model-free MaxEnt policy, and we use Model-Predictive Path Integral Control (NN-MPPI) [20] as our model-based MaxEnt planner.

The third algorithm we evaluate is MaxDiff RL [22]. In prior work, we showed that MaxDiff RL achieves exploration through decorrelating agent experiences (i.e., explored states). Here, we use a model-based MaxDiff RL formulation, which replaces $\mathcal{H}_\pi[a_t|s_t]$ from Eq. 1 with a state-transition based entropy term

$$S_\pi[s_t]=\frac{1}{2}\log\det C_\pi[s_t], \quad (2)$$

where $C_\pi[s_t]$ is the trajectory autocovariance for policy π. In other words, MaxEnt RL explores the controls (indirectly exploring the states), while MaxDiff explores the states directly. Direct state exploration enables MaxDiff to learn from scratch in single-shot scenarios, which are more challenging than the classic episodic RL formulation.

One challenge with real-world deployment of MaxDiff is the tuning of α, which effectively controls the balance between state exploration and task exploitation. Since reward functions can depend on task length-scales and robot parameters, α values can vary from task to task and system to system, which is undesirable. To ensure that our exploration term varies on the same scale as our rewards, in this work we define our temperature parameter as

$$\alpha=\alpha_0\frac{\sigma_r}{\sigma_S}=\mathbb{E}_{(s_t,a_t)\sim p,\pi}\left[\alpha_0\sqrt{\frac{Var(r(s_t,a_t))}{Var(S_\pi[s_t])}}\right], \quad (3)$$

which effectively scales the strength of MaxDiff exploration relative to sampled rewards. By accounting for task and system-dependent scale variations, we are then able to use α_0 as a fixed hyperparameter to tune exploration in a more generalizable manner. In practice, we compute α as

$$\hat{\alpha}(T)=\frac{\alpha_0}{N}\sum_{n=1}^{N}\sqrt{\frac{\sum_{t=1}^{T}|r(s_t^{(n)},a_t^{(n)})-\bar{r}_t|^2}{\sum_{t=1}^{T}|S_\pi[s_t^{(n)}]-\bar{S}_t|^2}}, \quad (4)$$

where N is the number of sample paths, (n) indexes across sample paths, T is the planning horizon, S_π is the trajectory entropy from Eq. 2, r is the predicted reward, and at each point in time their sample averages across sample paths are

given by $\bar{r}_t$ and $\bar{S}_t$, which we calculate empirically. Additionally, by explicitly modeling the temperature parameter as a function of planning horizon T, we are able to adapt our temperature parameter in real-time across rolling windows, effectively introducing an annealing schedule that improves long horizon learning performance.

Implementation: For model-free algorithm (i.e., SAC), the policy is represented by a normal distribution parameterized by a mean function defined as a fully-connected neural network. We use the formulation for SAC specified in [19], including the structure of the soft Q functions, but we modify the batch size and policy representation to match the model-based algorithms. For model-based algorithms (i.e., MaxDiff and NN-MPPI), the state-transition function $f(s_t, a_t)$ and reward function $r(s_t, a_t)$ are both modeled by fully-connected neural networks. The state-transition network is regularized using the negative log-loss of a normal distribution where the variance, $\Sigma_{\text{model}} \in \mathbb{R}^{n \times n}$, is a hyperparameter that is simultaneously learned based on agent experience. The predicted reward utility is improved by the error between the predicted target and target reward equal to $\mathcal{L} = \|r_t + 0.95\, r(s_{t+1}, a_{t+1}) - r(s_t, a_t)\|^2$, similar to the loss functions used in temporal-difference learning [37], [38]. The inclusion of the reward term from the next state and next action helps the algorithm learn in environments with rewards that do not strictly depend on the current state. Table I provides an overview of relevant RL hyperparameters.

	SAC [19]	NN-MPPI [20]	MaxDiff [22]
Target Smoothing Coeffi.	0.01	N/A	N/A
Discount Factor	0.99	0.95	0.95
Reward Scale (S/H)	500 / 10	1 / 1	1 / 1
Planning Horizon (S/H)	N/A	30 / 20	30 / 20
Planning Samples	N/A	1000	1000
Planning Inverse Temp.	N/A	0.1	0.1
Planning Control Noise	N/A	$\mathcal{N}(0, 0.5)$	$\mathcal{N}(0, 0.5)$
Diffusion Dimensions	N/A	N/A	$[x, y, \dot{x}, \dot{y}]$
Diffusion Weights	N/A	N/A	$[1, 1, 0.05, 0.05]$
Diffusion Temp. α_0 (S/H)	N/A	N/A	0.1 / 0.5
MLP Hidden Layers	200×200 (ReLU Activation)		
MLP Final Layer Weights	$\mathcal{U}(-0.03, 0.03)$		
Learning Rate	0.0003 (Adam Optimizer [39])		
Batch Size	128 (5 updates per control step)		

TABLE I. Learning Hyperparmeters. Any differences between simulation (S) and hardware (H) are noted above. Any hyperparameters not listed are the same as those in the original papers.

III. Robot Design

Our hardware swimmer—NoodleBot—is designed to mimic the Gym MuJoCo 3-link swimmer [23]. The goal of the swimming task is to maximize positive x-velocity by applying torque to the robot's rotors in a planar environment. The robot state is

$$s = [x, y, \sin(\theta), \cos(\theta), \phi_1, \phi_2, v_x, v_y, \omega_\theta, \omega_{\phi 1}, \omega_{\phi 2}],$$

where the elements are the 2D robot position, linearized robot angle, joint angles, robot velocity, and joint angular velocities. The robot actions are joint torques $a = [\tau_1, \tau_2]$. The reward function is

$$r(s_t, a_t) = \frac{x_t - x_{t-1}}{dt} - 0.0001\|a_t\|^2, \tag{5}$$

where a are actions provided to the robot, s are robot states, x is the position of the robot in the world frame, $\|\cdot\|$ is the L_2 norm, t is a time index, and dt is the time step.

Although both the simulated and hardware swimmers learn the same reward function, their environments pose substantially different challenges.[1] The simulated swimmer operates a highly viscous media without ground contact forces, while the hardware swimmer may encounter noisy sensors, impassable boundaries, unknown friction coefficients, stick-slip phenomena, degrading actuators, communication dropouts, as well as other real-world conditions that complicate the task. The differences in actuation and drag profiles lead to swimming gaits and policies that differ substantially between simulation and hardware—so much so that zero-shot policy

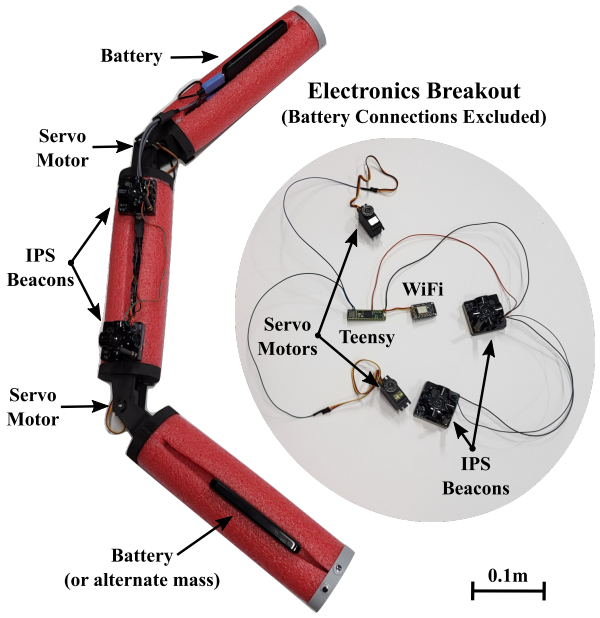

Fig. 2. 3-Link Swimmer Hardware. A top-down view of NoodleBot (left). The inset (right) shows the electronic components housed inside the middle joint. The total cost of the swimmer is approximately $100 excluding the cost of the Indoor Positioning System (IPS). Models of the 3D printed parts, firmware, and assembly instructions are available at `https://github.com/MurpheyLab/NoodleBot`.

Link	Length	Mass	Components
Front	38cm	345g	Battery, End Cap, Bracket, Housing
Middle	43cm	495g	Motors (2x), Brackets (2x), WiFi Module, Teensy, Wiring, IPS Beacons (2x), Housing
Back	38cm	360g	Battery, End Cap, Bracket, Cables, Housing

TABLE II. Hardware Details by Link. NoodleBot is designed to be customizable. Length and mass values are provided as baselines but could easily be adjusted as desired (up to the torque capacity of the motors).

[1] A side-by-side comparison of differences between the original simulated MuJoCo swimmer and our swimmer is included in the GitHub repository.

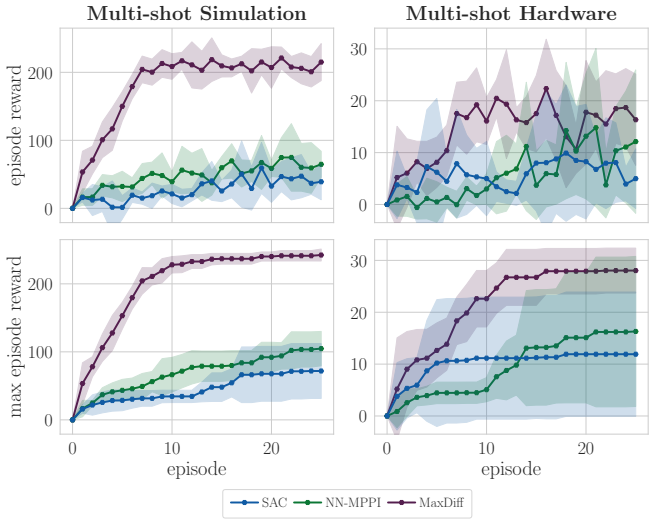

Fig. 3. Multi-shot Learning Rewards. Each column shows results from the same tests–with 5 seeds tested per method. The top row shows the raw episode rewards, and the bottom row shows the cumulative maximum reward achieved by each seed. For all plots, the solid line is the mean across seeds, and the shaded region is the standard deviation. The raw results show that MaxDiff outperforms SAC and NN-MPPI across all episodes in both simulation and hardware. The max reward results show that all MaxDiff seeds learn the task within 25 episodes, while there is large variance in the performance of SAC and NN-MPPI seeds.

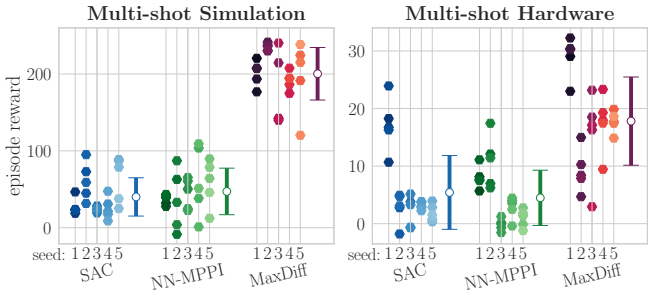

Fig. 4. Multi-shot Evaluation Rewards. Each learned multi-shot model was evaluated on 5 episodes. The hexagons are the raw data for each episode. The error bars show the mean and standard deviation for each method. These plots show that in simulation, all MaxDiff seeds receive a high reward, regardless of the initialization. In hardware, MaxDiff outperforms SAC and NN-MPPI on average. As expected, the results are much noisier than the simulation. The hardware swimmer has to contend with noisy sensors, physical boundaries, and changes in performance over time due to wear on the system.

Experiments	Test Area Dimensions ($x \times y$)	IPS Update Rate
Multi-shot	3.4m × 1.8m	13.7 Hz
Single-shot	7.0m × 4.6m	11.4 Hz

TABLE III. Indoor Positioning Test Parameters. The centralized modem publishes the state information for the mobile beacons, so the larger the test area, the slower the IPS update rate.

transfer is not a viable strategy. However, the simulation experiments provide valuable information about the expected performance of each algorithm in hardware.

A. Hardware

NoodleBot is comprised of three lightweight links with motorized hinge joints as shown in Fig. 2 and described in Table II. The links are constructed using a pool noodle with slots cut out to accommodate batteries and electronics. The links are connected by custom-made hinged motor brackets 3D printed via fused deposition modeling (FDM), and the ends are constrained by 3D printed FDM end caps. The bottom edge of each 3D printed part is made flat (as opposed to circular) to ensure good contact with the floor without rolling, and felt pads are attached to the flat portions to control friction. Standard 180° hobby servo motors (TowerPro SG-5010) provide actuation.

B. Firmware

Low-level robot control is handled on-board by a Teensy microcontroller with a ESP8266 WiFi module attached, where standard commercial battery packs provide USB power (5V, 3A) to the motors, microcontroller, and WiFi module. The Robot Operating System (ROS) is used to communicate states and actions over WiFi between the robot and an external laptop where model updates are performed.

The Marvelmind Indoor Positioning System (IPS) provides robot state information.[2] Test area dimensions and IPS update rates are listed in Table III. Alternate methods of generating state information are possible (e.g., April Tags), but adjustments to the robot firmware may be required.

IV. Experiments

In this work, we benchmark the performance of three deep RL algorithms on both simulated and hardware tasks. We take the embodied learning approach from Fig. 1, where simulations build confidence that algorithms are ready to test in hardware. Thus, the simulated results are primarily included to illustrate the repeatability that we expect from RL algorithms. All RL algorithms are executed on an Intel® i7 CPU @ 2.60GHz x 12 laptop with an Nvidia® GeForce GTX 1650 Max-Q GPU. The laptop has Ubuntu 20.04, ROS Melodic, and Python 3.8 (PyTorch 1.11). Each algorithm was trained with 5 different seeds (random neural network initializations).

Typically, RL agents learn across many episodic deployments of an agent, which we refer to as multi-shot learning. In multi-shot scenarios, agents are allowed to learn for a fixed number of environment interactions before the environment is reset. When the environment is reset, agents are returned to a random initial condition near a fixed start position. Many algorithms rely on this external source of environment randomization to prevent overfitting and escape from local

[2]To use the IPS, four stationary beacons are placed around the test environment with line-of-sight to the robot. The two mobile beacons on the robot provide 2D state and rotation information via radio communication with a centralized modem. For these tests, we use firmware v7.900 for Super-Beacon-2 and Modem HW v5.1-2 with non-inverse architecture.

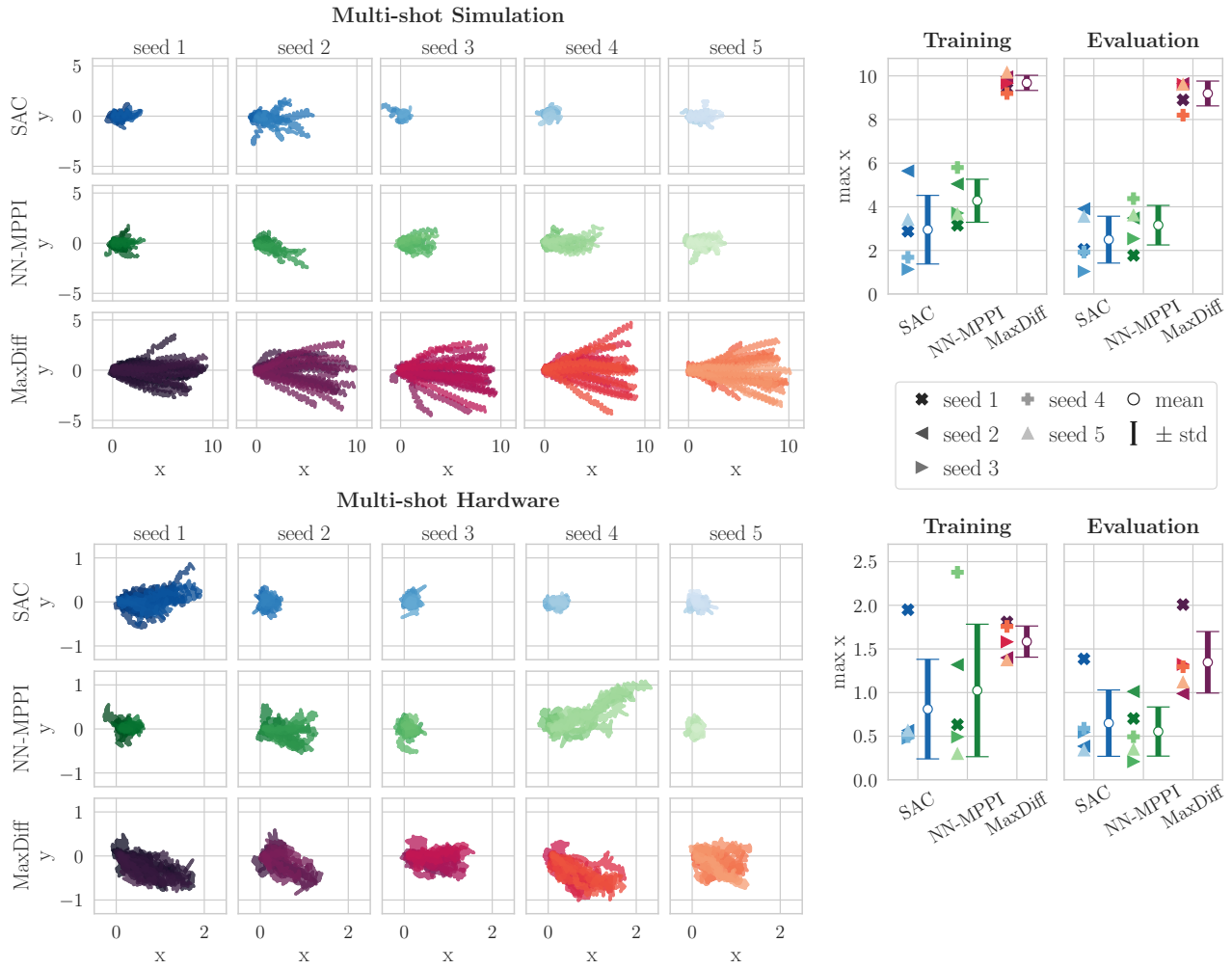

Fig. 5. Multi-shot Paths. The grids on the left show all training paths by seed and training algorithm. The plots on the right summarize the maximum x reached by each seed during training and evaluation. The paths show that all MaxDiff methods spread out more than SAC and NN-MPPI paths in both simulation and hardware. The results show that all MaxDiff seeds reach the same distance, regardless of the initialization, while there is large variance in the maximum distance reached by different SAC and NN-MPPI initializations. See video supplement for demonstrations of learned swimming policies.

minima. However, in hardware systems, repeatedly resetting hardware experiments can be time consuming. Additionally, conditions can drift and evolve over time. Ultimately, we want agents to be able to autonomously learn from scratch and adapt to environmental changes. Therefore, we also test RL agents in single-shot learning scenarios [22]. In single-shot scenarios, agents must learn throughout a single, continuous task attempt (without any environment resets).

A. *Multi-shot*

For the multi-shot tests, each algorithm was trained for 25 episodes of 1000 steps each. At the end of each episode, the model-based policies are reset, and the robot is physically moved to random joint angles at the same initial position in the world frame. Simulated tests take ∼20 minutes, and hardware tests take ∼1 hour. Fig. 3 shows a side-by-side comparison of the episodic reward across training episodes of the simulated and hardware swimmers. The plots show that MaxDiff outperforms NN-MPPI and SAC in simulation. Furthermore, the algorithms perform similarly in both hardware and simulation.

After training, we evaluate each final policy 5 times. The episode rewards are shown in Fig. 4. The simulated results show that all MaxDiff evaluations successfully solved the task. The hardware results show more variance than the simulated results, as expected. MaxDiff still outperforms SAC and NN-MPPI on average, but additional temperature tuning may be required to achieve optimal performance. An important note is that all policy evaluations were performed after the training procedures of each algorithm. As a result, it is likely that the NoodleBot may have experienced actuator degradation between the training and evaluation procedures, which potentially explains the increase in variance. However, since hardware degradation is an unavoidable feature of real-world robot operation, these results show that NoodleBot is a good candidate for a hardware RL benchmarking.

To illustrate how RL policy performance relates to the data collected, we also show agent sample paths for all training episodes on the left side of Fig. 5. These paths indicate that

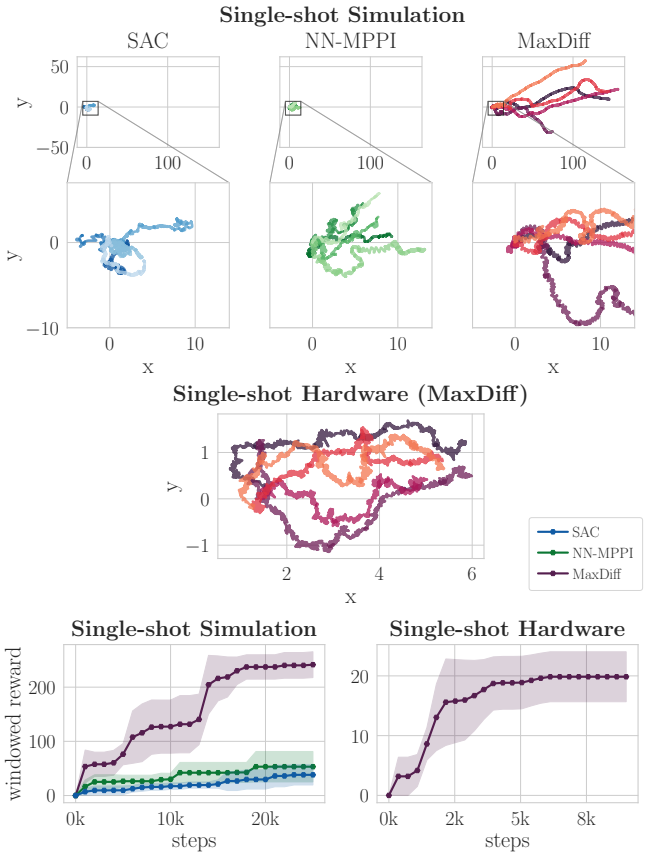

Fig. 6. Single-shot Paths. In the first three rows, each color is a different seed. The top row shows that only MaxDiff is able to learn in single-shot deployments. The middle plot shows that MaxDiff was also able to learn in single-shot hardware deployments. The bottom row show windowed rewards, where each window is 1000 control steps. In the bottom row, the solid line is the mean across seeds, and the shaded region is the standard deviation. The results show that MaxDiff is able to learn the task in single-shot hardware and software deployments. See video supplement for demonstrations of single-shot hardware learning.

when SAC and NN-MPPI collect diverse data, they are able to solve the task, but many seeds fail to explore very far beyond the initial position. MaxDiff, which is designed to directly encourage state exploration, covers a much larger portion of the environment for all seeds in both software and hardware. As a result, MaxDiff attains better, more reliable performance across seeds, underscoring the importance of effective state exploration in RL.

Since the task is to maximize $+x$ velocity, the right section of Fig. 5 summarizes the maximum x position achieved by each seed. These results again illustrate the impact of state exploration on task performance. The best performing SAC seeds (2 in simulation and 1 in hardware) are those which achieve the greatest state coverage during training. Furthermore, the results show that MaxDiff achieves similar max x positions across seeds. Thus, the empirical robustness of MaxDiff across random seeds in both software and hardware tests suggests that MaxDiff presents a promising approach to reliable RL in the real-world.

B. Single-shot

For single-shot tests, the robot is initialized once and is then required to learn continuously. The results are shown in Fig. 6. For simulated tests, we allow the robot to learn for 25,000 control steps. In hardware, we are limited by the size of our test area, so we allow the robot to learn until it reaches the maximum $+x$ position in the test workspace. To enable comparisons to multi-shot tests, we accumulate the rewards over each 1000 control steps into a "windowed reward". We test all three algorithms on the single-shot simulated swimmer, but only MaxDiff was capable of learning a successful locomotion policy. Therefore, we only tested the single-shot hardware swimmer with MaxDiff. The results in Fig. 6 show that MaxDiff is able to learn the task both in simulation and in hardware.

V. Discussion

This work introduces NoodleBot as an open-source, low-cost 3-link swimmer robot platform designed to facilitate hardware experimentation in RL and to encourage the development of RL algorithms that directly grapple with the challenges of real-world operation. Real-world deployment of RL has been hindered by the large gap between simulated benchmarks and practical applications, but hardware benchmarks like NoodleBot provide a path to close this gap and build confidence that embodied RL algorithms can learn from scratch. We demonstrate the performance of three RL algorithms on this benchmark, providing baseline results for future comparisons and extensions. Moreover, our results highlight the importance of effective state exploration and data quality to the performance of RL agents.

The flexibility of our platform also enables future work to explore various aspects of autonomous robot learning, such as comparing the performance of learning algorithms across different body morphologies (e.g., limb masses and lengths), testing in diverse environmental conditions (e.g., different surfaces and interfacial physics), as well as assessing and adapting to robot wear and tear (e.g., actuator degradation and communication dropouts). The platform's potential can be further expanded by incorporating low-cost sensors, such as cameras, to enable research in additional RL domains.

Beyond its research contributions, NoodleBot's affordability and open-source nature position it as a potentially valuable educational tool. Its accessible design and straightforward construction process can empower students and hobbyists to engage directly with the challenges and intricacies of real-world RL, similar to platforms like Duckietown [40]. By providing a tangible platform for implementing and testing RL algorithms, NoodleBot hopes to bridge the gap between theoretical understanding, simulation performance, and practical application, fostering a deeper appreciation for the complexities of embodied learning. As such, our work presents an avenue for researchers and the broader community alike to explore the frontiers of intelligent, adaptive systems in the real-world.

ACKNOWLEDGMENT

This material is supported by Army Research Office Grant W911NF-22-1-0286. T.A.B. is partially supported by the Northwestern University Presidential Fellowship. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the aforementioned institutions.

REFERENCES

[1] V. Mnih, K. Kavukcuoglu, D. Silver, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[2] A. Farooq and K. Iqbal, "A survey of reinforcement learning for optimization in automation," in *IEEE International Conference on Automation Science and Engineering (CASE)*, 2024, pp. 2487–2494.

[3] J. Ibarz, J. Tan, C. Finn, M. Kalakrishnan, P. Pastor, and S. Levine, "How to train your robot with deep reinforcement learning: lessons we have learned," *International Journal of Robotics Research*, vol. 40, no. 4, pp. 698–721, 2021.

[4] J. Langaa and C. Sloth, "Expert initialized reinforcement learning with application to robotic assembly," in *IEEE International Conference on Automation Science and Engineering (CASE)*, 2022, pp. 1405–1410.

[5] X. Sun, J. Li, A. V. Kovalenko, W. Feng, and Y. Ou, "Integrating reinforcement learning and learning from demonstrations to learn nonprehensile manipulation," *IEEE Transactions on Automation Science and Engineering*, vol. 20, no. 3, pp. 1735–1744, 2023.

[6] A. Rajeswaran, V. Kumar, A. Gupta, G. Vezzani, J. Schulman, E. Todorov, and S. Levine, "Learning complex dexterous manipulation with deep reinforcement learning and demonstrations," in *Robotics: Science and Systems*, 2018.

[7] A. Gupta, V. Kumar, C. Lynch, S. Levine, and K. Hausman, "Relay policy learning: Solving long-horizon tasks via imitation and reinforcement learning," in *Conference on Robot Learning (CoRL)*, 2020, pp. 1025–1037.

[8] W. Zhao, J. P. Queralta, and T. Westerlund, "Sim-to-real transfer in deep reinforcement learning for robotics: a survey," in *IEEE Symposium Series on Computational Intelligence*, 2020, pp. 737–744.

[9] L. Smith, J. C. Kew, X. B. Peng, S. Ha, J. Tan, and S. Levine, "Legged robots that keep on learning: Fine-tuning locomotion policies in the real world," in *International Conference on Learning Representations (ICLR)*, 2022, pp. 1593–1599.

[10] H. R. Walke, J. H. Yang, A. Yu, A. Kumar, J. Orbik, A. Singh, and S. Levine, "Don't start from scratch: Leveraging prior data to automate robotic reinforcement learning," in *Conference on Robot Learning (CoRL)*, 2022.

[11] N. Mu, X. Hu, Q.-S. Jia, X. Zhu, and X. He, "Large-scale data center cooling control via sample-efficient reinforcement learning," in *IEEE International Conference on Automation Science and Engineering (CASE)*, 2024, pp. 2780–2785.

[12] C. Li, T. Zhang, and D. I. Goldman, "A terradynamics of legged locomotion on granular media," *Science*, vol. 339, no. 6126, pp. 1408–1412, 2013.

[13] D. Ma and A. Rodriguez, "Friction variability in planar pushing data: Anisotropic friction and data-collection bias," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 3232–3239, 2018.

[14] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-real transfer of robotic control with dynamics randomization," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

[15] S. Amin, M. Gomrokchi, H. Satija, H. van Hoof, and D. Precup, "A survey of exploration methods in reinforcement learning," *arXiv*, 2021.

[16] A. A. Taiga, W. Fedus, M. C. Machado, A. Courville, and M. G. Bellemare, "On bonus based exploration methods in the arcade learning environment," in *International Conference on Learning Representations (ICLR)*, 2020.

[17] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell, "Curiosity-driven exploration by self-supervised prediction," in *International Conference on Machine Learning (ICML)*, 2017, pp. 2778–2787.

[18] H. Tang, R. Houthooft, D. Foote, A. Stooke, O. Xi Chen, Y. Duan, J. Schulman, F. DeTurck, and P. Abbeel, "# exploration: A study of count-based exploration for deep reinforcement learning," in *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[19] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International Conference on Machine Learning (ICML)*, 2018.

[20] G. Williams, N. Wagener, B. Goldfain, P. Drews, J. M. Rehg, B. Boots, and E. A. Theodorou, "Information theoretic MPC for model-based reinforcement learning," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2017.

[21] X. Huang, X. Wang, Y. Zhao, J. Hu, H. Li, and Z. Jiang, "Guided model-based policy search method for fast motor learning of robots with learned dynamics," *IEEE Transactions on Automation Science and Engineering*, vol. 22, pp. 453–465, 2025.

[22] T. A. Berrueta, A. Pinosky, and T. D. Murphey, "Maximum diffusion reinforcement learning," *Nature Machine Intelligence*, vol. 6, no. 5, pp. 504–514, 2024.

[23] E. Todorov, T. Erez, and Y. Tassa, "MuJoCo: A physics engine for model-based control," in *IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2012.

[24] J. Leitner, A. W. Tow, N. Sünderhauf, *et al.*, "The ACRV picking benchmark: A robotic shelf picking benchmark to foster reproducible research," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2017.

[25] A. R. Mahmood, D. Korenkevych, G. Vasan, W. Ma, and J. Bergstra, "Benchmarking reinforcement learning algorithms on real-world robots," in *Conference on Robot Learning (CoRL)*, 2018.

[26] N. Gürtler, S. Blaes, P. Kolev, F. Widmaier, M. Wuthrich, S. Bauer, B. Schölkopf, and G. Martius, "Benchmarking offline reinforcement learning on real-robot hardware," in *International Conference on Learning Representations (ICLR)*, 2023.

[27] B. Yang, D. Jayaraman, J. Zhang, and S. Levine, "REPLAB: A reproducible low-cost arm benchmark for robotic learning," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2019.

[28] M. Ahn, H. Zhu, K. Hartikainen, H. Ponte, A. Gupta, S. Levine, and V. Kumar, "Robel: Robotics benchmarks for learning with low-cost robots," in *Conference on Robot Learning (CoRL)*, 2019.

[29] H. Zhang, S. Yang, and D. Wang, "A real-world quadrupedal locomotion benchmark for offline reinforcement learning," in *International Joint Conference on Neural Networks (IJCNN)*, 2024.

[30] W. Yu, K. Caluwaerts, A. Iscen, *et al.*, "The design of the barkour benchmark for robot agility," in *IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2024, pp. 6818–6825.

[31] E. M. Purcell, "Life at low Reynolds number," *American Journal of Physics*, vol. 45, no. 1, pp. 3–11, 1977.

[32] B. J. Van Stratum, M. P. Austin, K. Shoele, and J. E. Clark, "Comparative model evaluation with a symmetric three-link swimming robot," in *IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2022.

[33] R. Hall, G. Espinosa, S.-S. Chiang, and C. D. Onal, "Design and testing of a multi-module, tetherless, soft robotic eel," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2024.

[34] S. J. A. Raza, A. Dastider, and M. Lin, "Developmentally synthesizing earthworm-like locomotion gaits with Bayesian-augmented deep deterministic policy gradients (DDPG)," in *IEEE International Conference on Automation Science and Engineering (CASE)*, 2020, pp. 1122–1128.

[35] S. Hirose and A. Morishima, "Design and control of a mobile robot with an articulated body," *International Journal of Robotics Research*, vol. 9, no. 2, pp. 99–114, 1990.

[36] C. Wright, A. Johnson, A. Peck, Z. McCord, A. Naaktgeboren, P. Gianfortoni, M. Gonzalez-Rivero, R. Hatton, and H. Choset, "Design of a modular snake robot," in *IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2007.

[37] J. A. Boyan, "Least-squares temporal difference learning," in *International Conference on Machine Learning (ICML)*, 1999.

[38] D. Precup, R. S. Sutton, and S. Dasgupta, "Off-policy temporal-difference learning with function approximation," in *International Conference on Machine Learning (ICML)*, 2001.

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

[40] L. Paull, J. Tani, H. Ahn, *et al.*, "Duckietown: An open, inexpensive and flexible platform for autonomy education and research," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2017.