BricksRL: A Platform for Democratizing Robotics and Reinforcement Learning Research and Education with LEGO

Sebastian Dittert
Universitat Pompeu Fabra
sebastian.dittert@upf.edu

Vincent Moens
PyTorch Team, Meta
vincentmoens@gmail.com

Gianni De Fabritiis
ICREA, Universitat Pompeu Fabra
g.defabritiis@gmail.com

Abstract

We present BricksRL, a platform designed to democratize access to robotics for reinforcement learning research and education. BricksRL facilitates the creation, design, and training of custom LEGO robots in the real world by interfacing them with the TorchRL library for reinforcement learning agents. The integration of TorchRL with the LEGO hubs, via Bluetooth bidirectional communication, enables state-of-the-art reinforcement learning training on GPUs for a wide variety of LEGO builds. This offers a flexible and cost-efficient approach for scaling and also provides a robust infrastructure for robot-environment-algorithm communication. We present various experiments across tasks and robot configurations, providing built plans and training results. Furthermore, we demonstrate that inexpensive LEGO robots can be trained end-to-end in the real world to achieve simple tasks, with training times typically under 120 minutes on a normal laptop. Moreover, we show how users can extend the capabilities, exemplified by the successful integration of non-LEGO sensors. By enhancing accessibility to both robotics and reinforcement learning, BricksRL establishes a strong foundation for democratized robotic learning in research and educational settings.

1 Introduction

As the field of artificial intelligence continues to evolve, robotics emerges as a fascinating area for deploying and evaluating machine learning algorithms in dynamic, real-life settings [14, 39, 46]. These applications allow embodied agents to interact within complex environments, similar to humans and animals, they must navigate a variety of challenging constraints during their learning process.

Reinforcement learning (RL), in particular, has emerged as a promising approach to learning complex behavior with robots [20, 29]. Despite the rich potential for innovation, the learning process of algorithms under real-world conditions is a challenge [1, 38, 46]. The complexity of setting up a robotics lab, combined with the high cost of equipment and the steep learning curve in RL, often limits the ability of researchers, educators, and hobbyists to contribute to and benefit from cutting-edge developments. To address these challenges, we introduce BricksRL, a comprehensive open-source framework designed to democratize access to robotics and RL. BricksRL builds upon Pybricks [44], a versatile Python package for controlling modular LEGO robotics hub, motors and sensors, actively maintained and supported by a vibrant community, and TorchRL [6], a modern framework for training

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

https://doi.org/10.52202/079017-2394

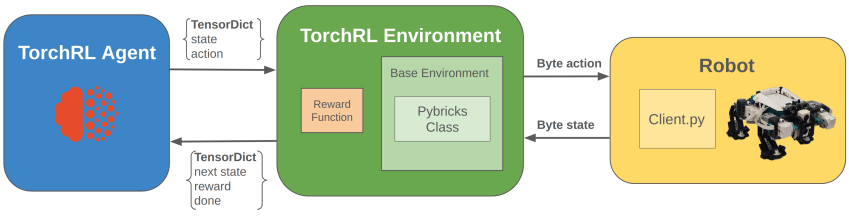


Figure 1: Communication overview of the agent, environment and robot.

RL agents. This synergy provides an integrated solution that simplifies the creation, modularity, design, and training of custom robots in the real world.

The use of LEGO parts as the basic building blocks for constructing the robots for BricksRL has the advantage of being cheap and widely available, which facilitates entry, but also makes it easier to replace parts in the event of repairs and keeps costs down. In addition, the building blocks allow full reusability of all parts, which is not the case with other robots, and no special tools are required for construction or maintenance, which also keeps costs very low. By abstracting the complexities of robot programming given a gym-like interface and RL algorithm implementation, BricksRL opens the door for a broader audience to engage with robotics research and education, making it more accessible to researchers, educators, and hobbyists alike. The low cost, wide availability, and ease of deployment also allow the introduction and use of BricksRL as a new artificial intelligence benchmark to test and evaluate RL algorithms in robotics for a variety of tasks and robots.

The contributions of our work with BricksRL are threefold. First, we provide a unified platform integrating Pybricks and TorchRL within BricksRL, which facilitates the physical control of cost-effective robotic systems and the design of gym-like training environments for RL algorithms. This setup enables scalable training across a diverse array of robots, tasks, and algorithms. Second, we demonstrate how to extend the capabilities of BricksRL beyond the standard sensor set provided by LEGO. By integrating a camera sensor, we expand the platform's capabilities, allowing for the creation of more diverse robots and tasks and thereby broadening its potential applications. Third, we present preliminary results that underscore the framework's robustness and adaptability for real-world robotics applications. Furthermore, we provide explicit examples of sim2real transfer and the application of datasets with offline RL, demonstrating the practical utility of BricksRL in robotics and RL research. We make the source code available at: `https://github.com/BricksRL/bricksrl`. The building plans, and evaluation videos of the experiments are publicly available at `https://bricksrl.github.io/ProjectPage/`.

2 Related Work

High acquisition costs, ranging from 10,000$ to over 200,000$, pose significant barriers to robotics research. This pricing affects various robotics types, including robotic arms (e.g., Franka [31], Kuka [23] and advanced robotic hands [29, 37, 41], similar to costly quadruped or humanoid robots designed for locomotion [5, 18, 42].

The popularization and increased consumer accessibility of 3D printing and DIY projects have heightened interest in low-cost robotics, thereby broadening access and facilitating the entry into robotics [1–3, 7, 9–11, 22, 32], lowering the initial costs for simple quadrupeds starting at 300$ robotic arms and hands for 20,000$. However, there is a requisite need for access to a 3D printer, a workshop, and equipment for the construction, maintenance, and repair of these robots. Additionally, projects and companies have been established to cater to the niche of low-cost robotics with pre-built robots for educational purposes that fall within a similar price range [4, 33–35]. Nevertheless, similar to off-the-shelf industrial robots, these and DIY robots are static, and it is not assured that printed parts or other components can be repurposed for different robots or adapted for new tasks. This limitation often confines experiments to a single robot and setting, which can be considered restrictive.

LEGO parts provide standardized and robust components that facilitate simple reconstruction, modularity of designs, and reproducibility. This modularity enables the construction and prototyping of various robots and the adaptation to different tasks, thereby simplifying the testing and benchmarking of RL algorithms across diverse robotic configurations. The initial cost for a starter kit to construct

robots starts at approximately 400$ [40], and can be augmented with additional sets, specific elements, or sensors as required. [36] demonstrates the application of LEGO for constructing robots for under 300$. However, using aluminum extrusions and 3D printed components, coupled with control via a Raspberry Pi rather than LEGO's internal PrimeHub, diminishes the system's flexibility.

In contrast, the robots in BricksRL use only LEGO elements for construction and control. The simple integration of additional sensors is demonstrated but not necessary. Further, users interact with the robots via a gym-like environment as is common in RL, simplifying the interaction. In comparison, the industry standard for managing robots and sensors is the Robotics Operating System (ROS) [33]. It offers numerous tools, however, its steep learning curve can be a barrier for researchers, students, hobbyists, and beginners starting with RL and robotics.

The use of LEGO for education in robotics has a rich history through sets such as MINDSTORMS [26] or education sets [19]. These are used not only in official educational institutions [25, 26, 43] but also in annual competitions around the world [17, 45] that attract a substantial number of children and students. To the best of our knowledge, these competitions do not currently incorporate machine learning techniques such as RL. Being tailored to these groups, BricksRL could bridge that gap and provide easy access to state-of-the-art algorithms.

3 BricksRL

The underlying architecture of BricksRL has three main components: the agent, the environment, and the robot 1. TorchRL is utilized to develop the agent and the environment, while Pybricks is employed for programming on the robot side. In the following sections, we will examine each component individually and discuss the communication mechanisms between them.

3.1 Agents

BricksRL utilizes TorchRL's modularity to create RL agents. Modules such as replay buffers, objective functions, networks, and exploration modules from TorchRL are used as building blocks to create agents that enable a uniform and clear structure. For our experiments and to showcase the integration of RL algorithms within BricksRL, we have selected three state-of-the-art algorithms for continuous control: TD3, SAC, and DroQ [12, 15, 16]. We primarily chose these off-policy algorithms for their simplicity and their proven ability of sample-efficient training, which is essential as we mostly train our robots online. However, due to the flexible and modular structure of TorchRL, BricksRL can be easily adapted to include any algorithm developed within or compatible with the TorchRL framework, allowing us to emphasize the general applicability of our system rather than the specific strategies. For example, methods commonly used in robotics, such as imitation learning [47] and the use of foundation models [28], are available with TorchRL and can be seamlessly integrated into BricksRL.

3.2 Environment

BricksRL incorporates TorchRL's environment design principles among other components, to standardize the structure and organization of its environments.

PybricksHub Class. Developed by BricksRL, the PybricksHub class plays an important role in facilitating communication with the LEGO Hub, which controls the robots. It achieves this through Bluetooth Low Energy (BLE) technology, which enables efficient, low-power communication. This class is designed to manage a two-way data exchange protocol, critical for meeting the real-time requirements of interactive applications. Importantly, the PybricksHub class seamlessly bridges asynchronous communication with the traditionally synchronous structure of RL environments.

EnvBase. In BricksRL, environments are designed as classes that inherit from EnvBase provided by TorchRL, which is a foundational class for building RL environments. This structure gives access to the TorchRL ecosystem and simplifies the creation of environments. Users can create custom environments or extend existing environments for new tasks. All that needs to be done is to adapt the observation and action specifications and, if necessary, define a new reward function and adapt the step and reset function of the environment.

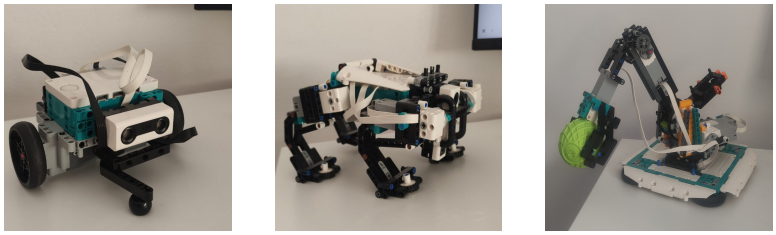
 https://doi.org/10.52202/079017-2394

A key advantage of using TorchRL's EnvBase in BricksRL is the ability to apply environment transforms, a fundamental feature of TorchRL. These transforms enable simple manipulation of the data exchanged between the environment and the agent. TorchRL provides a wide range of useful transforms, such as frame stacking, observation normalization, and image transformations, which are particularly valuable for real-world robotics. Additionally, the integration of foundation models like VIP [28] through transforms expands the experimentation capabilities within BricksRL. Detailed descriptions of the environments we implemented for our experiments, along with a template for creating custom environments, can be found in A.2 and A.2.8, respectively.

Agent-Environment Communication. For communication and data exchange between agent and environment, BricksRL makes use of TensorDict [6] as a data carrier. TensorDict enables a standardized and adaptable transmission of information between agent and environment. TensorDict can handle a wide range of data as observation by accurately describing the observation specs in the environment, without modifying the agent's or environment's essential structure. It enables users to shift between vector and picture observations or a combination of the two. This is a crucial building component for being flexible to train different algorithms on robots performing a variety of tasks with and without sensors of the LEGO ecosystem.

3.3 LEGO Robots

In our experiments demonstrating the capabilities of BricksRL, we selected three distinct robot types to serve as an introductory platform for RL in robotics. These robots vary in complexity and their capacity for environmental interaction, reflecting a progressive approach to robotic research. Additionally, we incorporated various sensors and embraced a range of robot classifications, showcasing a broad spectrum of applications and use cases.



(a) (b) (c)

Figure 2: Three robots that we used in the experiments: (a) 2Wheeler, (b) Walker, (c) RoboArm.

2Wheeler. The 2Wheeler robot 2a is built by us to represent an elementary robotic platform designed for introductory purposes, incorporating the LEGO's Hub, a pair of direct current (DC) motors equipped with rotational sensors, and an ultrasonic sensor for determining the proximity to objects. The independent control capability of the DC motors endows the robot with high maneuverability.

Walker. The Walker 2b, a quadrupedal robot as built in a standard LEGO robotics kit, is equipped with four motors, each integrated with rotational sensors and an additional ultrasonic sensor. In comparison to the 2Wheeler robot 3.3, the Walker variant exhibits more degrees of freedom and a higher level of complexity due to its increased motor count and the fact that it uses legs instead of wheels. In terms of structural design, this robot bears similarity to prevalent quadruped robots typically employed in the domain of locomotion control [39].

RoboArm. The RoboArm 2c is built by us and is similar to the Walker 4 motors with rotation sensors equipped, however, it has a higher range of motion. Further, it is the only static robot and includes another branch of robot types used for tasks like grasping and reaching or manipulating objects [20, 21, 27].

In general, Pybricks allows wide access to different motors and sensors as well as IMU (Inertial Measurement Unit) data in the LEGO's hub, which permits a variety of possible modular robot architectures and applications. For a detailed overview, please refer to the official Pybricks documentation

[30]. Furthermore, we highlight the large community that has already created various robots, which can be rebuilt or used as inspiration. Any robot can be used with BricksRL as long as it is available in Pybricks's interface. We welcome collaborations and encourage community members who want to experiment with BricksRL to contact us for support.

Robot-Environment Communication. BricksRL employs a bidirectional data flow between the robot and the environment, facilitated by MicroPython in Pybricks. The agent's actions are transmitted as byte streams via standard input (stdin), where they are parsed and applied to the robot's motors. At the same time, the robot sensor data is sent back to the environment through standard output (stdout) for state evaluation and action selection. Each robot uses a dedicated client script to manage its motors, sensors, and control flow for specific tasks. For exact details and an example of a typical client script, see the provided template in A.3.

Communication Speed. In robotics and motion control, the rate of communication is crucial, necessitating high frequencies to ensure rapid responsiveness and precision in response to environmental changes or disturbances. Position or torque-based control systems in quadrupedal robots, for instance, operate within a query frequency range of 20 to 200 Hz [8, 24]. This frequency range enables these robots to swiftly adjust to variations in terrain during locomotion.

Likewise, the hub's control loop is capable of exceeding frequencies of 1000 Hz, making it suitable for managing complex robotic systems. Yet, when integrating with the BricksRL communication framework, a reduction in the system-wide frequency, including agent-environment and environment-robot communications, to 11 Hz was observed. This decrease is primarily due to the overhead introduced by utilizing stdin and stdout for communication. The process of reading from and writing to these streams, which requires system calls, is inherently slower compared to direct memory operations. Additionally, the necessity to serialize and deserialize data through 'ustruct.unpack' and 'ustruct.pack' adds to this overhead, as it requires converting data between binary formats used in communication and the Python object representation, which is time-consuming.

Despite the overhead, BricksRL's communication speed, while on the lower spectrum, remains within a reasonable range for robotic system applications. For instance,[13, 39] have demonstrated that effective motion control in quadrupedal robots can be achieved at much lower frequencies, such as 20 or even 8 Hz, indicating that robust and dynamic locomotion can be maintained even at reduced communication frequencies. Moreover, we show in our experiments that communication frequency is task and robot depending, for specific tasks optimal behaviors can be learnt faster with lower frequencies 7.

3.4 Modularity and Reusability

The use of interlocking LEGO parts, and various sensors allows for endless possibilities in designing and building robots and robot systems. Additionally, precise construction plans and the reusability of components create additional opportunities. Unlike classic robot systems, users are not limited to a single design and functionality. Instead, robots can be customized to specific requirements or tasks, and can even be reassembled for new challenges or ideas once the initial task is completed successfully. Building on this variable foundation with an infinite number of robotic applications, BricksRL enables easy interaction by abstracting the complexities of the underlying communication processes. To train RL algorithms, users can interact with the robots in gym-like environments, which provide a natural and intuitive interface. This enables researchers and hobbyists to train any RL algorithm, such as on-policy or off-policy, model-based or model-free.

To further illustrate modularity and scalability of BricksRL we extended the set of sensors and show how easy it is to integrate sensors outside of the LEGO ecosystem. Namely, we integrate a USB webcam camera into an environment (A.2.7) showcased in the experiments, demonstrating that additional sensors can further augment the scope of applications to train robots with RL and BricksRL.

4 Experiments

In our experiments, we aim to address several critical questions regarding the feasibility and efficiency of training LEGO robots using RL algorithms in the real world with BricksRL. Thereby taking into

 https://doi.org/10.52202/079017-2394

account the practical challenges of training LEGO robots such as the lack of millimeter-precise robot constructions, the presence of backlash, and noisy sensors.

Robot	Environments
2Wheeler	RunAway-v0
	Spinning-v0
Walker	Walker-v0
	WalkerSim-v0†
Roboarm	RoboArm-v0
	RoboArmSim-v0†
	RoboArm-mixed-v0*

Table 1: Overview of BricksRL Robot and Environment Settings. Environments marked with an asterisk (*) utilize LEGO sensors and image inputs as observations for the agent. Environments indicated by a dagger (†) denote simulations of the real robot and do not use the real robot for training.

Therefore we developed various task-specific environments to demonstrate the adaptability and ease of use of BricksRL, highlighting the scalability of training across different algorithms and robots with diverse sensor arrays. Tasks ranging from driving and controlling the 2Wheeler to learning to walk with the Walker and reaching tasks for the RoboArm demonstrate the applicability of BricksRL. Table 1 shows a complete overview of all environments used in our experiments.

In our experiments, we primarily focus on online learning, where the robot directly interacts with the real world, encompassing the challenges inherent to this approach. However, we have also developed simulation environments for certain tasks. Training in these simulations is significantly faster compared to real-world training, as confirmed by our comparative experiments. Additionally, we use these simulation environments to demonstrate the sim2real capabilities of LEGO robots with BricksRL.

A complete overview and description of the environments implemented including action and observation specifications as well as the definition of the reward function can be found in the appendix A.2. We also provide an environment template 1 that demonstrates the straightforward process of creating custom environments using BricksRL.

In all of our experiments, we initiated the training process with 10 episodes of random actions to populate the replay buffer. The results are obtained by over 5 seeds for each algorithm and compared against a random policy. Evaluation scores of the trained policies are displayed in 2. We further provide videos of trained policies for each robot and task A.1. Hyperparameter optimization was not conducted for any of the algorithms, and we adhered to default settings. Comprehensive information on the hyperparameter is provided in the appendix 12. Although the option to utilize environment transformations, such as frame stacking and action repetition, was available, we opted not to use these features to maintain the simplicity of our setup. Further details are available in the appendix A.2.

	Environment						
Algorithm	RunAway-v0	Spinning-v0	Walker-v0	WalkerSim-v0	RoboArm-v0	RoboArmSim-v0	RoboArm-mixed-v0
TD3	7.64 ± 2.31	$\mathbf{7558.21 \pm 28.61}$	-62.94 ± 16.76	-78.49 ± 9.75	$\mathbf{-20.29 \pm 31.35}$	-12.78 ± 21.09	-60.24 ± 16.06
SAC	8.72 ± 2.82	7407.20 ± 109.53	$\mathbf{-52.04 \pm 8.79}$	$\mathbf{-55.03 \pm 4.95}$	-27.77 ± 37.13	$\mathbf{-3.45 \pm 2.66}$	$\mathbf{-18.21 \pm 6.98}$
DroQ	$\mathbf{8.96 \pm 0.87}$	7456.85 ± 18.02	-57.63 ± 10.44	-56.62 ± 2.81	-55.02 ± 62.45	-14.04 ± 26.05	-19.39 ± 11.07
Random	-0.51 ± 1.84	71.97 ± 501.79	-191.99 ± 18.19	-191.99 ± 18.19	-149.26 ± 88.19	-149.26 ± 88.19	-57.23 ± 10.01

Table 2: The table displays the mean and standard deviation of evaluation rewards for the trained TD3, SAC, DroQ algorithms, and a random policy, based on experiments conducted across 5 evaluation episodes and 5 different seeds.

4.1 2Wheeler

In the `RunAway-v0` task for the 2Wheeler robot, we trained RL algorithms over 40 episodes. Training sessions were completed in approximately 15 minutes per run for each agent. All algorithms successfully mastered the task, as shown in Figure 3. Notably, despite the simplicity of the task, algorithms adopted unique strategies. TD3 maximized its actions, achieving the highest distance from the wall but causing rapid acceleration, tilting, and noisy measurements, leading to occasional

abrupt episode termination. In contrast, the DroQ agent used smaller actions, resulting in more stable but shorter distances and avoiding premature episode endings A.5.

For the `Spinning-v0` task, we trained the agents over 15 episodes. The training was completed in about 10 minutes, with all agents effectively learning to solve the task, as illustrated in Figure 3. Table 2 includes the evaluation scores for both tasks of the 2Wheeler.

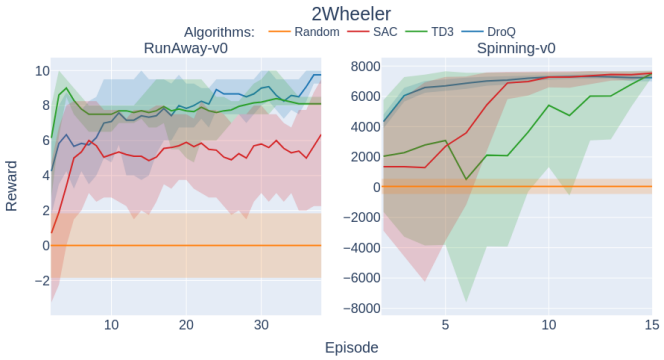


Figure 3: Training results for 2Wheeler robot for the `RunAway-v0` and the `Spinning-v0` environment.

4.2 Walker

In the `Walker-v0` environment, we trained the Walker robot over 75 episodes. The entire training process took approximately 70 minutes. All agents successfully developed a forward-moving gait. Remarkably, the DroQ algorithm achieved this in significantly fewer episodes (5-10), requiring only about 15 minutes of training, as illustrated in Figure 4. We trained the agents with a communication frequency of 2 Hz, instead of the maximum frequency of 11 Hz, as we observed that training at the lower frequency was faster and more stable. Results of a direct comparison can be found in the appendix 7.

Figure 4 also presents results from training in the `WalkerSim-v0` environment. To be comparable with the `WalkerSim-v0`, we similarly trained the agents for 75 episodes but noted marked differences in training duration: TD3 and SAC completed within 1-3 minutes, whereas DroQ required about 20 minutes due to a higher updating ratio. Simulation results revealed higher training performance with smoother and more stable learning curves compared to real-world training 4.

The final evaluation scores for policies trained in both real-world and simulation environments, tested on the actual robot, are summarized in Table 2. Although the simulation-trained policies slightly underperformed, they demonstrated effective sim2real transfer, highlighting their efficiency with considerably less training time and reduced supervision.

	Success Rate (%)	
Algorithm	RoboArm-v0	RoboArm-mixed-v0
TD3	88	8
SAC	72	68
DroQ	64	68
Random	32	40
TD3*	88	-
SAC*	100	-
DroQ*	88	-

Table 3: Comparison of success rates for different agents in the `RoboArm-v0` and `RoboArm-mixed-v0` environments. Success is defined as the agent reaching the goal or goal position within a specified threshold. Agents marked with an asterisk (*) were initially trained in the `RoboArmSim-v0` environment. Each algorithm was evaluated for 5 epochs with 5 different seeds, totaling 25 experiments per agent and task.

https://doi.org/10.52202/079017-2394

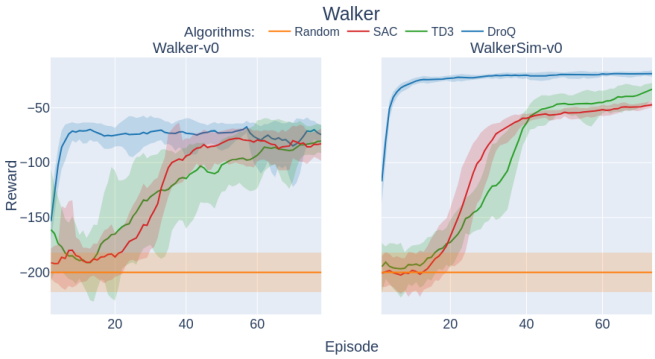


Figure 4: Training performance for Walker robot for the `Walker-v0` and the `WalkerSim-v0` environment.

4.3 RoboArm

In the `RoboArm-v0` task, agents were trained for 250 episodes. Training durations varied, with TD3 and SAC completing in about 1 hour on the real robot, whereas DroQ required close to 2 hours. By contrast, training in the `RoboArmSim-v0` environment proved much quicker: TD3 and SAC finished within 1-2 minutes, and DroQ in approximately 25 minutes. The outcomes, depicted in Figure 5, confirm that all agents successfully learned effective policies.

To enhance the interpretation of the training outcomes, we also plotted the final error—defined as the deviation from the target angles at the last step of each episode—and the total number of steps taken per episode. The data reveals a consistent decrease in both the final error and the number of steps throughout the training period. This indicates not only improved accuracy but also increased efficiency, as episodes terminated sooner when goals were successfully met.

The evaluation results are detailed in Table 2. Additionally, we compiled success rates that illustrate how often each agent reached the goal position within the predefined threshold. These success rates, derived from evaluation runs across 5 seeds with each seed running 5 episodes, are presented in Table 3. Notably, the policies trained in the `RoboArmSim-v0` environment achieved superior evaluation scores and also higher success rates upon testing. This demonstrates a successful sim2real transfer, achieving a significantly reduced training time.

Lastly, we present the training results for the `RoboArm_mixed-v0` environment, where the algorithms underwent training over 250 episodes. Training durations varied significantly due to the complexity added by integrating additional image observations: SAC was completed in 40 minutes, TD3 in 60 minutes, and DroQ took three hours. The inclusion of image data likely introduced considerable noise in the training results, as illustrated in Figure 6, which displays the rewards achieved by the agents and the corresponding episode steps. Interestingly, while SAC and DroQ successfully learned effective policies, TD3 struggled to adapt, failing to develop a viable strategy. The success of SAC and DroQ is evident in the chart of episode steps, showing a decrease in steps over the training period, which indicates a more efficient achievement of the goal position.

The evaluation results, detailed in Table 2, confirm the performances. Notably, out of 25 evaluation trials, both SAC and DroQ successfully reached the goal position 17 times, as recorded in Table 3. This demonstrates the robustness of the SAC and DroQ algorithms in handling the complexities introduced in the `RoboArm_mixed-v0` environment.

4.4 Offline Training

Offline RL uses pre-collected datasets to train algorithms efficiently, avoiding real-world interactions and complex simulations. To further highlight the capabilities of BricksRL for education and research in robotics and RL we collected offline datasets for the LEGO robots. With those datasets, BricksRL allows training of the LEGO robots via offline RL or imitation learning, both of which are state-of-the-art methods for RL in robotics.

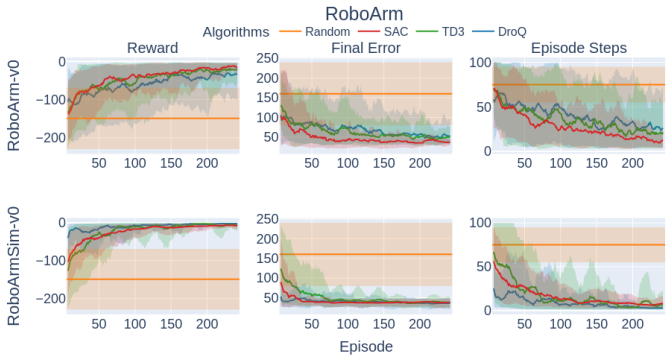


Figure 5: Training outcomes for the RoboArm robot in both the `RoboArm-v0` and `RoboArmSim-v0` environments. The plot also includes the final error at the epoch's last step and the total number of episode steps.

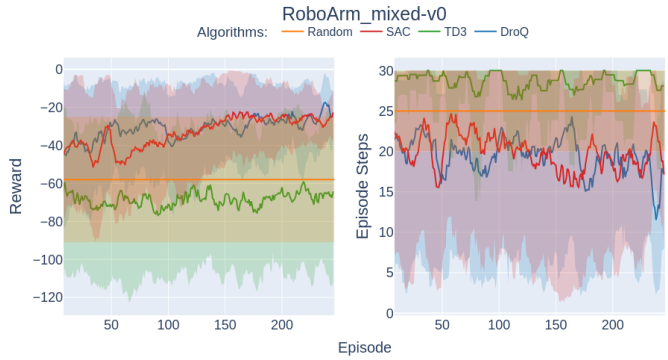


Figure 6: Training performance of the RoboArm robot in the `RoboArm_mixed-v0` environment, showing both the reward and the number of episode steps required to reach the target location.

For BricksRL, we curated datasets for three robot configurations: 2Wheeler, Walker, and RoboArm. These datasets include both expert and random data for four tasks in our experiments (Walker-v0, RoboArm-v0, RunAway-v0, Spinning-v0). Details about the datasets and dataset generation can be found in the appendix A.7. Using these datasets, we demonstrated that offline RL with BricksRL is feasible, successfully training both online and offline RL algorithms and applying them to a real robot. The evaluation performance, shown in Table 4, highlights the superior performance of offline RL algorithms, particularly with expert data, while online algorithms struggle, suggesting overfitting or poor generalization. For further details on training parameters, please refer to the appendix A.8.

Agent	Walker-v0		RoboArm-v0		RunAway-v0		Spinning-v0	
	Random	Expert	Random	Expert	Random	Expert	Random	Expert
TD3	−79.71	−153.45	−124.22	−201.94	**19.86**	9.06	**6160.25**	6168.28
SAC	− **66.91**	−255.41	−54.67	−218.76	14.74	10.80	5416.11	**9349.52**
BC	−202.46	−85.65	−117.72	−7.34	−0.27	18.13	35.55	9150.02
IQL	−136.13	**−74.80**	−76.89	**−3.10**	14.07	18.80	4544.28	9096.60
CQL	−72.75	−77.93	**−46.91**	−17.41	19.60	**19.74**	4509.31	9099.03

Table 4: Evaluation Results: Online (TD3, SAC) and Offline (BC, IQL, CQL) RL Algorithms. Scores represent the mean reward averaged over 5 episodes and 5 random seeds.

https://doi.org/10.52202/079017-2394

5 Conclusion

In this paper, we introduce BricksRL and detail its benefits for robotics, RL, and educational applications, emphasizing its cost-effectiveness, reusability, and accessibility. In addition, we showcased its practical utility by deploying three distinct robots, performing various tasks with a range of sensors, across more than 100 experiments. Our results underscore the viability of integrating state-of-the-art RL methodologies through BricksRL within research and educational contexts. By providing comprehensive building plans and facilitating access to BricksRL, we aim to establish this investigation as a foundational proof of concept for utilizing LEGO-based robots to train RL algorithms. Moving forward, avenues for further research include creating more complex robots and tasks, exploring applications in multi-agent settings, and leveraging large datasets to enhance RL training through transformer-based imitation learning. Ultimately, BricksRL sets the stage for a future where accessible, reusable robotic systems support and expand RL research, collaborative learning, and interactive education.

Acknowledgements

We thank A. De Fabritiis Campos, M. De Fabritiis Campos and P. Vallecillos Cusco for providing their LEGOs to this project.

References

[1] Michael Ahn, Henry Zhu, Kristian Hartikainen, Hugo Ponte, Abhishek Gupta, Sergey Levine, and Vikash Kumar. "ROBEL: Robotics Benchmarks for Learning with Low-Cost Robots". en. In: *Proceedings of the Conference on Robot Learning*. ISSN: 2640-3498. PMLR, May 2020, pp. 1300–1313. URL: https://proceedings.mlr.press/v100/ahn20a.html (visited on 04/09/2024).

[2] Jorge Aldaco, Travis Armstrong, Robert Baruch, Jeff Bingham, Sanky Chan, Debidatta Dwibedi, Chelsea Finn, Pete Florence, Spencer Goodrich, Wayne Gramlich, Alexander Herzog, Jonathan Hoech, Thinh Nguyen, Ian Storz, Baruch Tabanpour, Jonathan Tompson, Ayzaan Wahid, Ted Wahrburg, Sichun Xu, Sergey Yaroshenko, and Tony Z Zhao. "ALOHA 2: An Enhanced Low-Cost Hardware for Bimanual Teleoperation". en. In: ().

[3] Rafia Bindu, Sazid Alam, and Asif Neloy. "A Cost-Efficient Multipurpose Service Robot using Raspberry Pi and 6 DOF Robotic Arm". In: Mar. 2019, pp. 16–22. DOI: 10.1145/3325693.3325701.

[4] Rinu Boney, Jussi Sainio, Mikko Kaivola, Arno Solin, and Juho Kannala. *RealAnt: An Open-Source Low-Cost Quadruped for Education and Research in Real-World Reinforcement Learning*. arXiv:2011.03085 [cs]. June 2022. DOI: 10.48550/arXiv.2011.03085. URL: http://arxiv.org/abs/2011.03085 (visited on 04/09/2024).

[5] *Boston Dynamics' Spot Robot Dog Now Available for $74,500 - IEEE Spectrum*. en. URL: https://spectrum.ieee.org/boston-dynamics-spot-robot-dog-now-available (visited on 05/01/2024).

[6] Albert Bou, Matteo Bettini, Sebastian Dittert, Vikash Kumar, Shagun Sodhani, Xiaomeng Yang, Gianni De Fabritiis, and Vincent Moens. *TorchRL: A data-driven decision-making library for PyTorch*. arXiv:2306.00577 [cs]. Nov. 2023. DOI: 10.48550/arXiv.2306.00577. URL: http://arxiv.org/abs/2306.00577 (visited on 05/01/2024).

[7] Ken Caluwaerts, Atil Iscen, J. Chase Kew, Wenhao Yu, Tingnan Zhang, Daniel Freeman, Kuang-Huei Lee, Lisa Lee, Stefano Saliceti, Vincent Zhuang, Nathan Batchelor, Steven Bohez, Federico Casarini, Jose Enrique Chen, Omar Cortes, Erwin Coumans, Adil Dostmohamed, Gabriel Dulac-Arnold, Alejandro Escontrela, Erik Frey, Roland Hafner, Deepali Jain, Bauyrjan Jyenis, Yuheng Kuang, Edward Lee, Linda Luu, Ofir Nachum, Ken Oslund, Jason Powell, Diego Reyes, Francesco Romano, Feresteh Sadeghi, Ron Sloat, Baruch Tabanpour, Daniel Zheng, Michael Neunert, Raia Hadsell, Nicolas Heess, Francesco Nori, Jeff Seto, Carolina Parada, Vikas Sindhwani, Vincent Vanhoucke, and Jie Tan. *Barkour: Benchmarking Animal-level Agility with Quadruped Robots*. en. arXiv:2305.14654 [cs]. May 2023. URL: http://arxiv.org/abs/2305.14654 (visited on 05/13/2024).

[8] Shuxiao Chen, Bike Zhang, Mark W. Mueller, Akshara Rai, and Koushil Sreenath. *Learning Torque Control for Quadrupedal Locomotion*. arXiv:2203.05194 [cs, eess]. Mar. 2023. URL: http://arxiv.org/abs/2203.05194 (visited on 03/06/2024).

[9] Marc Peter Deisenroth and Carl Edward Rasmussen. "PILCO: A Model-Based and Data-Efficient Approach to Policy Search". en. In: (), p. 8.

[10] Hongjie Fang, Hao-Shu Fang, Yiming Wang, Jieji Ren, Jingjing Chen, Ruo Zhang, Weiming Wang, and Cewu Lu. *Low-Cost Exoskeletons for Learning Whole-Arm Manipulation in the Wild*. arXiv:2309.14975 [cs]. Sept. 2023. URL: http://arxiv.org/abs/2309.14975 (visited on 04/22/2024).

[11] Zipeng Fu, Tony Z Zhao, and Chelsea Finn. "Learning Bimanual Mobile Manipulation with Low-Cost Whole-Body Teleoperation". en. In: ().

[12] Scott Fujimoto, Herke van Hoof, and David Meger. *Addressing Function Approximation Error in Actor-Critic Methods*. Number: arXiv:1802.09477 arXiv:1802.09477 [cs, stat]. Oct. 2018. URL: http://arxiv.org/abs/1802.09477 (visited on 06/20/2022).

[13] Siddhant Gangapurwala, Luigi Campanaro, and Ioannis Havoutis. *Learning Low-Frequency Motion Control for Robust and Dynamic Robot Locomotion*. arXiv:2209.14887 [cs]. Feb. 2023. DOI: 10.48550/arXiv.2209.14887. URL: http://arxiv.org/abs/2209.14887 (visited on 03/06/2024).

[14] Tuomas Haarnoja, Ben Moran, Guy Lever, Sandy H. Huang, Dhruva Tirumala, Markus Wulfmeier, Jan Humplik, Saran Tunyasuvunakool, Noah Y. Siegel, Roland Hafner, Michael Bloesch, Kristian Hartikainen, Arunkumar Byravan, Leonard Hasenclever, Yuval Tassa, Fereshteh Sadeghi, Nathan Batchelor, Federico Casarini, Stefano Saliceti, Charles Game, Neil Sreendra, Kushal Patel, Marlon Gwira, Andrea Huber, Nicole Hurley, Francesco Nori, Raia Hadsell, and Nicolas Heess. *Learning Agile Soccer Skills for a Bipedal Robot with Deep Reinforcement Learning*. arXiv:2304.13653 [cs]. Apr. 2023. URL: http://arxiv.org/abs/2304.13653 (visited on 03/04/2024).

[15] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. *Soft Actor-Critic Algorithms and Applications*. arXiv:1812.05905 [cs, stat]. Jan. 2019. URL: http://arxiv.org/abs/1812.05905 (visited on 03/06/2024).

[16] Takuya Hiraoka, Takahisa Imagawa, Taisei Hashimoto, Takashi Onishi, and Yoshimasa Tsuruoka. *Dropout Q-Functions for Doubly Efficient Reinforcement Learning*. arXiv:2110.02034 [cs]. Mar. 2022. URL: http://arxiv.org/abs/2110.02034 (visited on 03/16/2024).

[17] *Home Page | FIRST LEGO League*. en. URL: https://www.firstlegoleague.org/ (visited on 04/30/2024).

[18] Marco Hutter, Christian Gehring, Dominic Jud, Andreas Lauber, C. Dario Bellicoso, Vassilios Tsounis, Jemin Hwangbo, Karen Bodie, Peter Fankhauser, Michael Bloesch, Remo Diethelm, Samuel Bachmann, Amir Melzer, and Mark Hoepflinger. "ANYmal - a highly mobile and dynamic quadrupedal robot". In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. ISSN: 2153-0866. Oct. 2016, pp. 38–44. DOI: 10.1109/IROS.2016.7758092. URL: https://ieeexplore.ieee.org/document/7758092 (visited on 05/01/2024).

[19] *Juguetes de LEGO® Education | Oficial LEGO® Shop ES*. es. URL: https://www.lego.com/es-es/themes/lego-education (visited on 05/13/2024).

[20] Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, and Sergey Levine. *QT-Opt: Scalable Deep Reinforcement Learning for Vision-Based Robotic Manipulation*. arXiv:1806.10293 [cs, stat]. Nov. 2018. DOI: 10.48550/arXiv.1806.10293. URL: http://arxiv.org/abs/1806.10293 (visited on 03/04/2024).

[21] Dmitry Kalashnikov, Jacob Varley, Yevgen Chebotar, Benjamin Swanson, Rico Jonschkowski, Chelsea Finn, Sergey Levine, and Karol Hausman. *MT-Opt: Continuous Multi-Task Robotic Reinforcement Learning at Scale*. arXiv:2104.08212 [cs]. Apr. 2021. DOI: 10.48550/arXiv.2104.08212. URL: http://arxiv.org/abs/2104.08212 (visited on 03/04/2024).

[22] Nathan Kau. *Stanford Pupper: A Low-Cost Agile Quadruped Robot for Benchmarking and Education*. en. arXiv:2110.00736 [cs]. Feb. 2022. URL: http://arxiv.org/abs/2110.00736 (visited on 04/30/2024).

 https://doi.org/10.52202/079017-2394

[23] *LBR iiwa*. en-DE. URL: https://www.kuka.com/en-de/products/robot-systems/industrial-robots/lbr-iiwa (visited on 05/01/2024).

[24] Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. "Learning Quadrupedal Locomotion over Challenging Terrain". In: *Science Robotics* 5.47 (Oct. 2020). arXiv:2010.11251 [cs, eess], eabc5986. ISSN: 2470-9476. DOI: 10.1126/scirobotics.abc5986. URL: http://arxiv.org/abs/2010.11251 (visited on 03/06/2024).

[25] *Lego Robotics - Wagner College Lifelong Learning*. en-US. URL: https://wagner.edu/lifelong-learning/robotics/ (visited on 05/13/2024).

[26] *Lego Robotics Competition - Fort Hays State University (FHSU)*. en. URL: https://www.fhsu.edu/smei/lego-robotics/ (visited on 05/13/2024).

[27] Sergey Levine, Peter Pastor, Alex Krizhevsky, and Deirdre Quillen. *Learning Hand-Eye Coordination for Robotic Grasping with Deep Learning and Large-Scale Data Collection*. arXiv:1603.02199 [cs]. Aug. 2016. DOI: 10.48550/arXiv.1603.02199. URL: http://arxiv.org/abs/1603.02199 (visited on 03/04/2024).

[28] Jason Yecheng Ma, Shagun Sodhani, Dinesh Jayaraman, Osbert Bastani, Vikash Kumar, and Amy Zhang. "VIP: TOWARDS UNIVERSAL VISUAL REWARD AND REPRESENTATION VIA VALUE-IMPLICIT PRE- TRAINING". en. In: ().

[29] OpenAI, Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, Jonas Schneider, Nikolas Tezak, Jerry Tworek, Peter Welinder, Lilian Weng, Qiming Yuan, Wojciech Zaremba, and Lei Zhang. *Solving Rubik's Cube with a Robot Hand*. arXiv:1910.07113 [cs, stat]. Oct. 2019. DOI: 10.48550/arXiv.1910.07113. URL: http://arxiv.org/abs/1910.07113 (visited on 03/04/2024).

[30] *Pybricks Documentation — pybricks v3.5.0 documentation*. URL: https://docs.pybricks.com/en/stable/ (visited on 04/30/2024).

[31] *Research*. en. URL: https://franka.de/research (visited on 05/01/2024).

[32] Paulo Rezeck, Hector Azpurua, Mauricio FS Correa, and Luiz Chaimowicz. *HeRo 2.0: A Low-Cost Robot for Swarm Robotics Research*. arXiv:2202.12391 [cs]. Mar. 2022. DOI: 10.48550/arXiv.2202.12391. URL: http://arxiv.org/abs/2202.12391 (visited on 03/04/2024).

[33] *Robot Kits & STEM Toys for K-12 Schools and Home Education|Makeblock*. en. URL: https://www.makeblock.com/ (visited on 05/13/2024).

[34] *Robotik in Schulen - die Robotik für Bildung 4.0*. de-DE. URL: https://variobotic.de/produkt-kategorie/robotik-in-schulen/ (visited on 05/13/2024).

[35] *Robotis | The study of humanexperiences*. en. URL: https://en.robotis.com (visited on 05/13/2024).

[36] Logan Saar, Haotong Liang, Alex Wang, Austin McDannald, Efrain Rodriguez, and A Gilad Kusne. "A Low-Cost Robot Science Kit for Education with Symbolic Regression for Hypothesis Discovery and Validation". en. In: ().

[37] *Shadow Dexterous Hand Series - Research and Development Tool*. en. Running Time: 66. Sept. 2023. URL: https://www.shadowrobot.com/dexterous-hand-series/ (visited on 05/01/2024).

[38] Laura Smith, Yunhao Cao, and Sergey Levine. *Grow Your Limits: Continuous Improvement with Real-World RL for Robotic Locomotion*. arXiv:2310.17634 [cs]. Oct. 2023. DOI: 10.48550/arXiv.2310.17634. URL: http://arxiv.org/abs/2310.17634 (visited on 03/04/2024).

[39] Laura Smith, Ilya Kostrikov, and Sergey Levine. *A Walk in the Park: Learning to Walk in 20 Minutes With Model-Free Reinforcement Learning*. arXiv:2208.07860 [cs]. Aug. 2022. DOI: 10.48550/arXiv.2208.07860. URL: http://arxiv.org/abs/2208.07860 (visited on 03/04/2024).

[40] *STEM & STEAM Solutions for the Classroom*. en-us. URL: https://education.lego.com/en-us/ (visited on 05/13/2024).

[41] *The Dexterous Hands - INSPIRE*. en-US. URL: https://en.inspire-robots.com/product-category/the-dexterous-hands (visited on 05/01/2024).

[42] *UNITREE Robotics® SHOP | Official Unitree Robot Dogs A1 Go1*. en-US. URL: https://unitreerobotics.net/ (visited on 05/01/2024).

[43] Carnegie Mellon University. *LEGO Curriculum - Carnegie Mellon Robotics Academy - Carnegie Mellon University*. en. URL: https://www.cmu.edu/roboticsacademy/roboticscurriculum/Lego%20Curriculum/index.html (visited on 05/13/2024).

[44] Laurens Valk. *Pybricks*. en. URL: https://pybricks.com/ (visited on 04/30/2024).

[45] *WRO RoboMission - LEGO Roboter lösen Aufgaben auf WRO Parcours*. URL: https://www.worldrobotolympiad.de/world-robot-olympiad/robomission (visited on 05/09/2024).

[46] Philipp Wu, Alejandro Escontrela, Danijar Hafner, Ken Goldberg, and Pieter Abbeel. *DayDreamer: World Models for Physical Robot Learning*. en. arXiv:2206.14176 [cs]. June 2022. URL: http://arxiv.org/abs/2206.14176 (visited on 04/26/2024).

[47] Maryam Zare, Parham M. Kebria, Abbas Khosravi, and Saeid Nahavandi. *A Survey of Imitation Learning: Algorithms, Recent Developments, and Challenges*. arXiv:2309.02473 [cs, stat]. Sept. 2023. DOI: 10.48550/arXiv.2309.02473. URL: http://arxiv.org/abs/2309.02473 (visited on 05/22/2024).

A Appendix

A.1 Repository and Website

BricksRL repository and our project website with evaluation videos and building instructions can be found at the following locations:

- GitHub: BricksRL
- Project Website

A.2 Environments

A.2.1 RunAway-v0

The `RunAway-v0` environment presents a straightforward task designed for the 2Wheeler robot. The objective is to maximize the distance measured by the robot's ultrasonic sensor. To accomplish this, the agent controls the motor angles, deciding how far to move forward or backward. This environment operates within a continuous action space, and each episode spans a maximum of 20 steps. Episodes will terminate early if the robot reaches the maximum distance of 2000 mm.

Action and Observation Specifications. TorchRL's `BoundedTensorSpecs` are employed to define the action and observation specifications, which have 1 and 5 dimensions, respectively. Table 5 outlines the specific ranges. Actions are initially defined in the range of $[-1, 1]$ but are linearly mapped to the range of $[-100, 100]$ before being applied to both the left and right wheel motors.

Type	Num	Specification	Min	Max
Action Spec	0	motor	-1	1
Observation Spec	0	left motor angle	0.0	360.0
	1	right motor angle	0.0	360.0
	2	pitch angle	-90	90
	3	roll angle	-90	90
	4	distance	0.0	2000.0

Table 5: Combined action and observation specifications for the `RunAway-v0` environment.

Reward Function. The reward function for the RunAway-v0 environment is defined as:

$$R_t = \begin{cases} +1 & \text{if distance}_t > \text{distance}_{t-1} \\ -1 & \text{if distance}_t < \text{distance}_{t-1} \\ 0 & \text{else} \end{cases} \tag{1}$$

 https://doi.org/10.52202/079017-2394

A.2.2 Spinning-v0

The `Spinning-v0` environment is another setup designed for the 2Wheeler robot. Unlike `RunAway-v0`, this environment does not use the ultrasonic sensor to measure the robot's distance to objects in front of it. Instead, at each reset, a value is randomly selected from the discrete set $0, 1$, which explicitly dictates the rotational direction in which the robot should spin. The robot's IMU (Inertial Measurement Unit) enables the tracking of various parameters, including angular velocity. This angular velocity is part of the agent's observation in the `Spinning-v0` environment, providing it with information on its rotational direction to complete the task. The action space is also continuous, and each episode has a length of 50 steps.

Action and Observation Specifications. The `BoundedTensorSpec` for the observation in the `Spinning-v0` environment comprises six floating-point values: the left and right motor angles, the pitch and roll angles, the angular velocity ω_z, and the rotational direction. Table 6 details these specifications. The action specification is defined as two floating-point values representing the rotation angles applied to the left and right motors. Actions are initially defined in the range of $[-1, 1]$ but will be transformed to the range of $[-100, 100]$ before being applied to the motors, as detailed in Table 6.

Type	Num	Specification	Min	Max
Action Spec	0	left motor	-1	1
	1	right motor	-1	1
Observation Spec	0	left motor angle	0.0	360.0
	1	right motor angle	0.0	360.0
	2	pitch angle	-90	90
	3	roll angle	-90	90
	4	angular velocity ω_z	-100	100
	5	direction	0	1

Table 6: Combined action and observation specifications for the `Spinning-v0` environment.

Reward Function. The reward function for the Spinning-v0 environment is defined in 1. The angular velocity ω_z is directly used as a reward signal and encourages adherence to a predefined rotational orientation.

$$R_t = \begin{cases} \omega_{z,t} & \text{if rotational direction} = 0 \text{ (spinning left)}, \\ -\omega_{z,t} & \text{otherwise (spinning right)}. \end{cases} \tag{2}$$

A.2.3 Walker-v0

In the `Walker-v0` environment for the Walker robot, the objective is to master forward movement using its four legs. To achieve this, the robot is provided with data on the current angles of each leg's motors, along with IMU readings that include both pitch and roll angles, ensuring operational safety. Additionally, an ultrasonic sensor is used to momentarily halt the agent's actions when the detected distance falls below a predefined threshold, preventing collisions with obstacles. Each episode consists of 100 steps. To achieve reduced communication speed, a waiting time is added after the actions are applied to the motors.

Action and Observation Specifications. In the `Walker-v0` environment, the observation specification consists of seven floating-point values: four motor angles (one for each leg), the pitch and roll angles, and the distance measurements from the ultrasonic sensor. The action specification includes four floating-point values corresponding to the four leg motors. These action values are initially defined in the range of $[-1, 1]$ but are linearly mapped to the range of $[-100, 0]$ before being applied, as detailed in Table 7.

Reward Function. The total reward is a sum of penalties for the actions and differences in angles, encouraging synchronized movement and appropriate angular differences between the legs of the

Type	Num	Specification	Min	Max
Action Spec	0	left front motor	-1	1
	1	right front motor	-1	1
	2	left back motor	-1	1
	3	right back motor	-1	1
Observation Spec	0	left front motor angle	0.0	360.0
	1	right front motor angle	0.0	360.0
	2	left back motor angle	0.0	360.0
	3	right back motor angle	0.0	360.0
	4	pitch angle	-90	90
	5	roll angle	-90	90
	6	distance	0.0	2000.0

Table 7: Combined action and observation specifications for the `Walker-v0` environment.

walker. The reward components are defined as follows: - $R_{\text{action},t}$: Penalty for the magnitude of actions taken. - $R_{\text{lf-rb},t}$: Penalty for the angular difference between the left front (lf) and right back (rb) motor angles. - $R_{\text{rf-lb},t}$: Penalty for the angular difference between the right front (rf) and left back (lb) motor angles. - $R_{\text{lf-rf},t}$: Penalty for the deviation from 180 degrees between the left front (lf) and right front (rf) motor angles. - $R_{\text{lb-rb},t}$: Penalty for the deviation from 180 degrees between the left back (lb) and right back (rb) motor angles.

$$\mathrm{R}_t = R_{\text{action},t} + R_{\text{lf-rb},t} + R_{\text{rf-lb},t} + R_{\text{lf-rf},t} + R_{\text{lb-rb},t} \tag{3}$$

$$R_{\text{action},t} = -\frac{\sum \text{actions}_t}{40} \tag{4}$$

$$R_{\text{lf-rb},t} = -\frac{\text{angular\_difference}(\theta_{\text{lf},t}, \theta_{\text{rb},t})}{180} \tag{5}$$

$$R_{\text{rf-lb},t} = -\frac{\text{angular\_difference}(\theta_{\text{rf},t}, \theta_{\text{lb},t})}{180} \tag{6}$$

$$R_{\text{lf-rf},t} = -\frac{180 - \text{angular\_difference}(\theta_{\text{lf},t}, \theta_{\text{rf},t})}{180} \tag{7}$$

$$R_{\text{lb-rb},t} = -\frac{180 - \text{angular\_difference}(\theta_{\text{lb},t}, \theta_{\text{rb},t})}{180} \tag{8}$$

with the angular difference defined as:

$$\text{angular\_difference}(\theta_{1,t}, \theta_{2,t}) = |((\theta_{2,t} - \theta_{1,t} + 180) \mod 360) - 180| \tag{9}$$

A.2.4 WalkerSim-v0

Additionally, we have developed a simulated version of the `Walker-v0` environment, called `WalkerSim-v0`. This simulation mirrors the real-world setup without requiring communication with the actual robot or using PyBricks. In the simulation, both IMU measurements and ultrasonic sensor inputs, which are used in the `Walker-v0` environment, are set to zero. This simplification is made because modeling or simulating these sensors can be challenging due to their complexity and the nuances involved in accurately replicating their readings. However, since these sensors are primarily used for safety in the real world, their absence is not a concern in the simulated environment.

In `WalkerSim-v0`, the next motor states are calculated by simulating the transition dynamics. This involves transforming the action output into the angle range and then applying the corresponding actions by adding to the current motor state. To model real-world inaccuracies, we add noise to the motor states, sampled from a Gaussian distribution with a mean of 0 and a standard deviation of 0.1. Similar to the real-world environment the `WalkerSim-v0` episodes consist of 100 interactions.

 https://doi.org/10.52202/079017-2394

Action and Observation Specifications. Action and observation specifications are the same as in the `Walker-v0` 7.

Reward Function. The reward function for the `WalkerSim-v0` is the same as in the `Walker-v0` environment.

A.2.5 RoboArm-v0

The `RoboArm-v0` is a pose-reaching task. At every reset, random goal angles for the four motors are sampled, defining a target pose. The objective is to adjust the robot's articulation to reach the specified goal pose within 100 steps. To achieve this the robot is provided with the current motor angles of all joints. The design of this environment permits the user to choose whether the robot tackles the task with dense or sparse rewards, effectively adjusting the difficulty level.

Action and Observation Specifications. The observation specification for the `RoboArm-v0` environment consists of eight floating-point values: four current motor angles and four goal motor angles, as detailed in Table 8. The action specification is defined by four floating-point values for the four motors, initially in the range of $[-1, 1]$. Before applying the specific actions to each motor, these values are transformed as follows: the rotation motor actions are linearly mapped to the range $[-100, 100]$, the low motor actions to $[-30, 30]$, the high motor actions to $[-60, 60]$, and the grab motor actions to $[-25, 25]$.

Type	Num	Specification	Min	Max
Action Spec	0	rotation motor	-1	1
	1	low motor	-1	1
	2	high motor	-1	1
	3	grab motor	-1	1
Observation Spec	0	rotation motor angle	0.0	360.0
	1	low motor angle	10	70
	2	high motor angle	-150	10
	3	grab motor angle	-148	-45
	4	goal rotation motor angle	0	360
	5	goal low motor angle	10	70
	6	goal high motor angle	-150	10
	7	goal grab motor angle	-148	-45

Table 8: Combined action and observation specifications for the `RoboArm-v0` environment.

Reward Function. The reward function for the `RoboArm-v0` environment can be chosen to be either dense or sparse. In the sparse case, the reward is 1 if the distance between the current motor angles and the goal motor angles is below a defined threshold; otherwise, it is 0. In our experiments, however, we used the dense reward function, which is calculated as follows:

$$R_t = -\frac{\|\Delta\vec{\theta}_{\text{deg},t}\|_1}{100} \tag{10}$$

where $\Delta\vec{\theta}_{\text{deg},t}$ is the vector of shortest angular distances at time step t between the goal motor angles $\vec{\theta}_{\text{goal},t}$ and the current motor angles $\vec{\theta}_{\text{current},t}$, defined as:

$$\begin{aligned}\Delta\vec{\theta}_{\text{deg},t} = \text{degrees}\Big(\arctan 2 \Big(&\sin(\text{radians}(\vec{\theta}_{\text{goal},t}) - \text{radians}(\vec{\theta}_{\text{current},t})), \\ &\cos(\text{radians}(\vec{\theta}_{\text{goal},t}) - \text{radians}(\vec{\theta}_{\text{current},t}))\Big)\Big)\end{aligned} \tag{11}$$

A.2.6 RoboArmSim-v0

`RoboArmSim-v0` mirrors the real-world `RoboArm-v0` environment but is entirely simulated, removing the need for physical interaction with an actual robot or the use of PyBricks. In this virtual setup, the RoboArm's task remains a pose-reaching challenge, where it must align its articulation to match randomly sampled goal angles for its four motors, setting a target pose at every reset. The robot in the simulation is provided with the current motor angles of all joints and the goal angles to accomplish this task within 100 steps. Crucially, the simulation is straightforward since it does not require modeling or simulating complex sensor measurements for state transitions, but only the motor angles, simplifying the simulation of state transitions significantly. Similar to the `WalkerSim-v0` we add Gaussian noise ($\mathcal{N}(0, 0.05)$) to the actions before the linear mapping and addition with the current motor states. Like its real-world counterpart, this environment allows users to choose between dense and sparse reward structures, facilitating the adjustment of the task's difficulty level.

Action and Observation Specifications. Action and observation specifications are the same as in the `RoboArm-v0` 8.

Reward Function. In the `RoboArmSim-v0` environment we use the same dense reward function as defined for the `RoboArm-v0` environment.

A.2.7 RoboArm-mixed-v0

In the `RoboArm_mixed-v0` environment, an additional sensor input is available to the robot through a webcam. The RoboArm holds a red ball in its hand and must move it to a target position. Each episode has a maximum of 30 steps. The target position is randomly selected and displayed as a green circle in the image. The image serves as additional information for the algorithm and is used to determine if the conditions to solve the task are met.

Action and Observation Specifications. The full observation specifications for the `RoboArm_mixed-v0` environment consist of three floating-point values representing the three motor angles (rotation motor, low motor, high motor) and image observation specifications with a shape of (64, 64), as detailed in Table 9.

The action specifications are also three floating-point values in the range of $[-1, 1]$, which will be transformed before being applied to the specific motor. The rotation motor angles are transformed to the range of $[-90, 90]$, the low motor angles to the range of $[-30, 30]$, and the high motor angles to the range of $[-60, 60]$.

Type	Num	Specification	Min	Max
Action Spec	0	rotation motor	-1	1
	1	low motor	-1	1
	2	high motor	-1	1
Observation Spec	0	rotation motor angle	0.0	360.0
	1	low motor angle	10	70
	2	high motor angle	-150	10
Image Observation Spec	0	image observation Size: (64, 64)	0	255

Table 9: Combined action and observation specifications for the `RoboArm_mixed-v0` environment.

Reward Function. To calculate the reward for the mixed observation environment `RoboArm_mixed-v0`, we utilize the Python package `OpenCV` to detect the red ball and measure the distance to the target location depicted in the image. First, we convert the image from BGR to HSV and define a color range to identify the contours. For each detected contour, we calculate the distance to the center of the green target circle and take the mean distance as the reward:

https://doi.org/10.52202/079017-2394

$$R_t = -\frac{\sum \text{distances}_t}{n_t \cdot 100} \tag{12}$$

where n_t is the number of distances detected at the current time step. If no contours are detected, we use the previous reward as the current reward.

A.2.8 Task Environment Template

To illustrate the simplicity of using the TorchBricksRL BaseEnv, which manages communication between the environment and the robot, we provide an example in Listing 1. This code can serve as a template for creating new custom environments.

```python
import torch

from environments.base.base_env import BaseEnv
from tensordict import TensorDict, TensorDictBase
from torchrl.data.tensor_specs import BoundedTensorSpec, CompositeSpec


class TaskEnvironment(BaseEnv):
    # Define your action and state dimension, needs to be adapted depending on your task and robot!
    action_dim = 1  # One action to control the wheel motors together
    state_dim = 5  # 5 sensors readings (left motor angle, right motor angle, pitch, roll, distance)

    # Define observation space ranges.
    motor_angles = [0, 360]
    roll_angles = [-90, 90]
    pitch_angles = [-90, 90]
    distance = [0, 2000]

    observation_key = "vec_observation"

    def __init__(
        self,
    ):
        self._batch_size = torch.Size([1])

        # Define Action Spec.
        self.action_spec = BoundedTensorSpec(low=-1, high=1, shape=(1, self.action_dim))
        # Define Observation Spec.
        bounds = torch.tensor(
            [
                self.motor_angles,
                self.motor_angles,
                self.roll_angles,
                self.pitch_angles,
                self.distance,
            ]
        )
        observation_spec = BoundedTensorSpec(
            low=bounds[:, 0],
            high=bounds[:, 1],
        )
        self.observation_spec = CompositeSpec({self.observation_key: observation_spec}, shape=(1,))

        super().__init__(
            action_dim=self.action_dim, state_dim=self.state_dim,
        )


```

```
    def _reset(self, tensordict: TensorDictBase, **kwargs) ->
TensorDictBase:
        # Get initial state from hub.
        observation = self.read_from_hub()
        # Could also add external sensors here and return them as well
.
        # img = self.camera.read()
        return TensorDict(
            {
                self.observation_key: norm_observation.float(),
                # image_obs: img.float(),
            },
            batch_size=[1],
        )

    def reward(self, state, action, next_state) -> Tuple[float, bool]:
        # Define your reward function.
        # ...
        reward, done = 0, False
        return reward, done

    def _step(self, tensordict: TensorDictBase) -> TensorDictBase:
        # Send action to hub to receive next state.
        action = tensordict.get("action").cpu().numpy().squeeze(0)
        self.send_to_hub(action)
        # Read next state from hub.
        next_observation = self.read_from_hub()

        # Compute the reward.
        state = tensordict.get(self.original_vec_observation_key)
        next_state = next_tensordict.get(self.
original_vec_observation_key)
        reward, done = self.reward(
            state=state,
            action=action,
            next_state=next_state,
        )
        # Create output TensorDict.
        next_tensordict = TensorDict(
            {
                self.observation_key: self.normalize_state(
next_observation).float(),
                "reward": torch.tensor([reward]).float(),
                "done": torch.tensor([done]).bool(),
            },
            batch_size=[1],
            device=tensordict.device,
        )
        return next_tensordict
```

Listing 1: Task environment template

A.3 Client Script

For each task and robot, a custom client script is required to facilitate interaction between the robot and the environment. The client.py script defines the configuration of motors, sensors, and the workflow for processing and exchanging data. This script must be uploaded to the Pybricks Hub and updated whenever the robot's configuration changes, such as when motors or sensors are added or removed. Listing 2 provides a simple example of a client script tailored for the `RunAway-v0` task. In this example, a single float value representing the action is used to control the motors, while sensor data is collected and transmitted back to the environment.

```
import ustruct
from micropython import kbd_intr
```


https://doi.org/10.52202/079017-2394

```python
from pybricks.hubs import InventorHub
from pybricks.parameters import Direction, Port
from pybricks.pupdevices import Motor, UltrasonicSensor
from pybricks.robotics import DriveBase
from pybricks.tools import wait
from uselect import poll
from usys import stdin, stdout


# Initialize the Inventor Hub.
hub = InventorHub()

# Initialize the drive base.
left_motor = Motor(Port.E, Direction.COUNTERCLOCKWISE)
right_motor = Motor(Port.A)
drive_base = DriveBase(left_motor, right_motor)
# Initialize the distance sensor.
sensor = UltrasonicSensor(Port.C)

keyboard = poll()
keyboard.register(stdin)

while True:

    # Optional: Check available input.
    while not keyboard.poll(0):
        wait(1)

    # Read action values for the motors.
    action = ustruct.unpack("!f", stdin.buffer.read(4))[0]
    # Apply the action to the motors
    drive_base.straight(action, wait=True)

    # Read sensors to get current state of the robot.
    (left_m_angle, right_m_angle) = (left_motor.angle(), right_motor.angle())
    (pitch, roll) = hub.imu.tilt()
    dist = sensor.distance()

    # Send the current state back to the environment.
    out_msg = ustruct.pack(
        "!fffff", left_m_angle, right_m_angle, pitch, roll, dist
    )
    stdout.buffer.write(out_msg)
```

Listing 2: Client script example.

A.4 Communication Frequency

Figure 7 offers a direct performance comparison of the DroQ agent on the `Walker-v0` task at communication frequencies of 11Hz and 2Hz. Interestingly, the agent operating at 2Hz shows quicker and more stable convergence. We suspect that the lower communication frequency functions similarly to 'frame skip', a widely utilized technique in reinforcement learning. Frame skipping helps to reduce the number of actions an agent takes, thereby simplifying the decision-making processes. This method may explain the more efficient convergence observed with the 2Hz frequency.

A.5 RunAway-v0 Strategies

Figure 8 illustrates the distinct strategies developed by the algorithms. Specifically, Figure 8b shows the final distance measured, while Figure 8a displays the mean action taken over the entire episode.

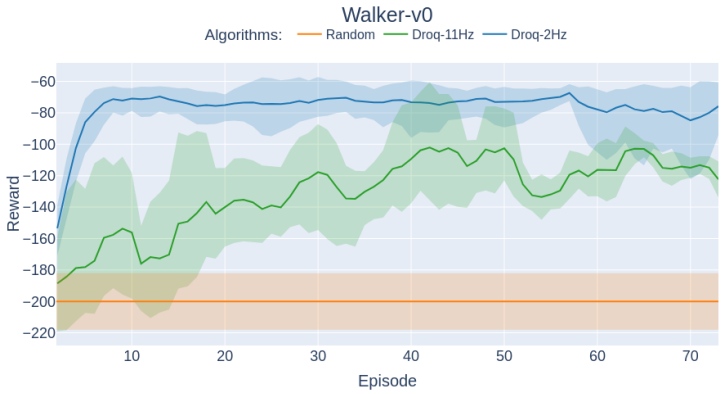


Figure 7: Comparison of communication frequencies for the DroQ agent on the `Walker-v0` task, illustrating the differences between the operational frequencies of 11Hz and 2Hz.

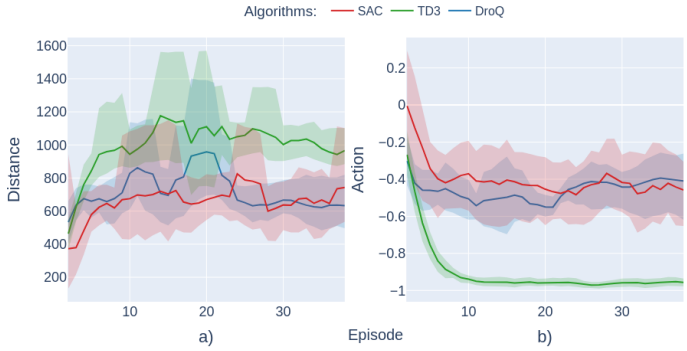


Figure 8: Final distance and (mean) action taken over one episode for the `RunAway-v0` task.

A.6 Online Training Parameter

Table 12 displays the hyperparameter used in all our experiments.

A.7 Dataset Generation Process

The expert dataset for each robot configuration was generated by training a Soft Actor-Critic (SAC) agent to solve the respective task and record transitions over 100 episodes on the real robot. The random dataset was created by executing a random policy for 100 episodes. For example, the collection process for the Walker robot took about an hour, yielding approximately 10,000 transitions. Details such as mean reward, number of transitions, and collection episodes for each dataset can be found in Table 11. The dataset is available on Hugging face.

A.8 Offline Training Parameter

For the online algorithms, we used the same parameters in our offline rl experiments as in the online experiments.

We trained the models for various tasks and datasets with different update counts 13. The `RunAway-v0` task was trained for 2,000 updates on both the expert and random datasets. For the `Spinning-v0` task, we used 5,000 updates across both datasets. The `Walker-v0` task required 10,000 updates for both expert and random datasets. Finally, the `RoboArm-v0` task was trained for 10,000 updates on the random dataset and 5,000 updates on the expert dataset.

https://doi.org/10.52202/079017-2394

Parameter	DroQ	SAC	TD3
Learning Rate (lr)	3×10^{-4}	3×10^{-4}	3×10^{-4}
Batch Size	256	256	256
UTD Ratio	20	1	1
Prefill Episodes	10	10	10
Number of Cells	256	256	256
Gamma	0.99	0.99	0.99
Soft Update ϵ	0.995	0.995	0.995
Alpha Initial	1	1	-
Fixed Alpha	False	False	-
Normalization	LayerNorm	None	None
Dropout	0.01	0.0	0.0
Buffer Size	1000000	1000000	1000000
Exploration Noise	-	-	0.1

Table 10: Hyperparameter for the agents DroQ, SAC, and TD3

Task	Mean Reward	Expert Transitions	Random Transitions	Episodes
Walker-v0	-69.12	$9,244$	$10,000$	100
RoboArm-v0	-9.87	$1,297$	$10,000$	100
RunAway-v0	18.04	$1,987$	$1,612$	100
Spinning-v0	8981.19	$5,000$	$5,000$	100

Table 11: Dataset Statistics

A.9 Network Architecutre

Throughout the experiments, all algorithms utilize the same architecture for the policy, Q-functions, and value functions (where applicable). Each network is structured as a three-layer multilayer perceptron (MLP), with specific `TorchRL` actor modules used for the policy, depending on the algorithm. For more details, we refer readers to the code repository: GitHub.

The only variation in architecture occurs when incorporating pixel-based observations. In this case, a convolutional neural network (CNN) is used to encode the image data. These encodings are then concatenated with the sensor-based encodings, and the combined embeddings are passed through a shared MLP. Detailed implementation specifics can be found in the GitHub repository.

Parameter	BC	IQL	CQL
Learning Rate (lr)	3×10^{-4}	3×10^{-4}	3×10^{-4}
Batch Size	256	256	256
Number of Cells	256	256	256
Gamma	-	0.99	0.99
Soft Update ϵ	-	0.995	0.995
Loss Function	L2	L2	L2
Temperature	-	1.0	1.0
Expectile	-	0.5	-
Min Q Weight	-	-	1.0
Max Q Backup	-	-	False
Deterministic Backup	-	-	False
Num Random Actions	-	-	10
With Lagrange	-	-	True
Lagrange Threshold	-	-	5.0
Normalization	LayerNorm	None	None
Dropout	0.01	0.0	0.0
BC Steps	-	-	1,000

Table 12: Hyperparameter for the agents BC, IQL, and CQL

Task	**Expert**	**Random**
RunAway-v0	2,000	2,000
Spinning-v0	5,000	5,000
Walker-v0	10,000	10,000
RoboArm-v0	5,000	10,000

Table 13: Number of offline training updates for each task and dataset

https://doi.org/10.52202/079017-2394

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The claims made in the abstract and introduction are demonstrated in our results section and can be further observed in the evaluation videos provided on our project page.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The paper discusses the limitations of the work, including the low communication speed and its causes, the lack of millimeter-precise constructions, backlash, and noisy sensors. It also highlights how the limitation of missing vision sensors can be extended.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory Assumptions and Proofs**

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [NA]

Justification: The paper does not provide theoretical results; instead, it focuses on demonstrating practical applications.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental Result Reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: For reproduction, robotic hardware is needed; however, the paper fully discloses all the information needed to reproduce the main experimental results, including building instructions for the robots, specifications for the environment setup, reward functions, and hyperparameter settings for the algorithms. Additionally, full access to the code is provided.

Guidelines:

- The answer NA means that the paper does not include experiments.
- If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

https://doi.org/10.52202/079017-2394

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: The paper provides full access to the code and data and provides detailed instructinos to reproduce the experiments.

   Guidelines:

   - The answer NA means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental Setting/Details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: Yes, the paper specifies all the hyperparameters for the algorithms and the specific settings for each environment.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment Statistical Significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: We run our experiments over 5 seeds for each algorithm and provide plots displaying the mean, minimum, and maximum values across all experiments. Additionally, the evaluation results are provided over 5 seeds, with results displayed using the mean and standard deviation.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments Compute Resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: The paper specifies that a regular affordable laptop is used for the experiments, highlighting the low cost of running the experiments. Additionally, specific training times are provided in the Experiments section.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code Of Ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The research conducted in the paper conforms in every respect with the NeurIPS Code of Ethics. We do not foresee any potential harms or negative social impact arising from our work.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader Impacts**

https://doi.org/10.52202/079017-2394

Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [No]

Justification: We do not discuss negative societal impacts, as we do not foresee any harmful or negative consequences from training LEGO robots. There is no risk of disinformation, generating fake profiles, surveillance, or similar issues. However, the paper focuses on the positive societal impact by democratizing robotics and reinforcement learning for research and education.

Guidelines:

- The answer NA means that there is no societal impact of the work performed.
- If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

Answer: [NA]

Justification: There is no risk of possible misuse of models or data described in the paper, so no specific safeguards are necessary for their release.

Guidelines:

- The answer NA means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: Yes, prior work on which this paper builds is properly mentioned and cited, and the creators or original owners of assets used in the paper are appropriately credited.

Guidelines:

- The answer NA means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New Assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [Yes]

Justification: New assets introduced in the paper are well documented. We provide building plans for the robots used in the experiments and access to the code for training them. Additionally, we offer templates for creating training environments and client scripts to train new custom robots.

Guidelines:

- The answer NA means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and Research with Human Subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [NA]

Justification: The paper does not involve crowdsourcing or research with human subjects.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

https://doi.org/10.52202/079017-2394

15. **Institutional Review Board (IRB) Approvals or Equivalent for Research with Human Subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: The paper does not involve crowdsourcing or research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.