Evaluating PyBullet and Isaac Sim in the Scope of Robotics and Reinforcement Learning

Enaitz Berrocal[1,2], Basilio Sierra[2], and Héctor Herrero[1]
[1]Industry and Movility, TECNALIA, Basque Research and Technology Alliance (BRTA). Spain
{enaitz.berrocal, hector.herrero}@tecnalia.com
[2]Department of Computer Sciences and Artificial Intelligence, University of the Basque Country (UPV/EHU). Spain
b.sierra@ehu.eus

***Abstract*—The choice of a physics simulator is a crucial decision in robotics and reinforcement learning (RL), significantly impacting the efficiency and effectiveness. This study conducts a comprehensive comparison of two popular simulator, PyBullet and Isaac sim, to identify their strengths and limitations in facilitating RL agent training. Each environment offers distinct features in terms of fidelity and computational efficiency, making it essential to evaluate their suitability for robotic reach task and their impact on RL algorithm performance. By benchmarking these environments, we aim to provide valuable insights for researchers and developers in the robotics community, informing them about the most suitable physics simulator for their specific application and ultimately advancing the state of the art in RL-based robotics research.**

***Index Terms*—Reinforcement learning, PPO, Stable-Baselines3, Isaac sim, PyBullet**

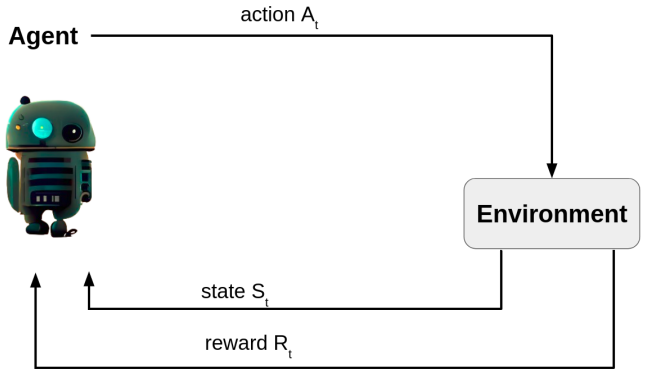

Fig. 1. Agent-environment interaction

I. Introduction

Reinforcement learning (RL) is a machine learning (ML) technique that allows an agent to learn from its environment through trial-and-error interactions [1]. In RL, rewards provide feedback on the agent's actions, guiding it towards optimal behaviour.

The standard RL setting involves an agent interacting with its environment, as shown in Fig. 1. This interaction consists of four key elements: a policy, a reward signal, a value function, and optionally a model of the environment [2], [3]:

- Policy: Defines how the agent behaves at a given time, mapping perceived states to actions.
- Reward signal: The goal of RL is to maximise the cumulative reward, which depends on the current state and the action taken.
- Value function: Specifies what is beneficial for a state in the long run, given a particular policy.
- Model of the environment (optional): Allows inferences to be made about the behaviour of the environment.

In contrast to other machine learning paradigms, RL presents a unique set of challenges and opportunities. A key difference between RL and supervised learning is that RL requires the agent to explicitly explore its environment in order to learn, whereas supervised learning learns from labelled training data without interaction. This exploration involves the agent taking actions and observing the consequences in order to learn about the dynamics of the environment and optimal behaviour. This feature of RL introduces challenges such as the exploration/exploitation trade-off. Exploration consist of trying new actions to learn about the environment, while exploitation means using the learned information to make reward-maximising decision [1].

RL has attracted considerable attention in recent years, and has achieved remarkable success in a variety of domains. Notable examples include outperforming human experts on Atari games [4] and playing Go [5]. RL has also demonstrated impressive capabilities in autonomous driving [6], teaching a robot to walk [7], and drug discovery [8]. These examples demonstrate the versatility and potential of RL.

In recent years, simulations have become a critical topic in robotics and RL. Robotics as an RL problem differs significantly from most well-studied RL benchmark problems [9]. The choice of a physics simulator is crucial for effective research and development. However, with the variety of simulators available, choosing the most appropriate one can be challenging. This paper aims to address this issue by conducting a comparative study of two different physics simulators.

We evaluate PyBullet and Isaac Sim based on their performance on the reach task to assess their RL capabilities. Unity was excluded due to its paid licence, and Mujoco was omitted due to technical issues.

Stable-Baselines3 [10], an open source RL library, is used

to assess simulator suitability by providing implementations of state-of-the-art algorithms such as Proximal Policy Optimization (PPO) [11], Deep deterministic Policy Gradients (DDPG) [12], and Soft Actor Critic (SAC) [13]. Its flexible, modular architecture allows for easy integration and testing in different simulators, algorithms, and environments, ensuring a comprehensive evaluation.

To evaluate the effectiveness of the generated model in various simulators, the Franka Emika Panda robot is used. This robot is a 7-joint robotic arm with a payload capacity of 3 kg and a maximum reach of 850 mm, complemented by a 2-joint gripper[1].

Through our investigation, we aim to explore the strengths and limitations of Isaac sim and PyBullet, contributing to the advancement of RL research and applications. Our evaluation focuses on frames per second (FPS) as a key metric to assess the speed and efficiency of training

II. Related Work

As RL applications have gained popularity, numerous papers have investigated the capabilities of various physics simulators.

Erez et al. [14] evaluated the performance of 5 different simulators (Bullet, Havok, MuJoCO, ODE, and PhysX), introducing a speed-accuracy measure that identifies the point where simulations become unstable.

Körber et al. [15] compared 4 popular physics simulators (Gazebo, MuJoCo, PyBullet, and Webots) for robotics and RL, evaluating stability, speed, and hardware usage. They test the simulators in two scenarios, providing guidance for researchers to choose the best tool for their projects.

Ayala et al. [16] conducted a quantitative comparison of Webots, Gazebo, and V-Rep, focusing on hardware load, using a humanoid robot NAO navigating around a chair controlled by an external controller.

Pitonakova et al. [17] compared Gazebo,V-Rep, and ARGoS in terms of functionalities and simulation speed, with a focus on mobile robot control development, evaluating the *RTF*, CPU, and memory utilization.

Audonnet et al. [18] propose a systematic review of simulation software compatible with ROS 2, benchmarking them under similar parameters, tasks, and scenarios to evaluate their capability for long-term operations, task success, repeatability, and resource usage.

Previous studies on robotic simulators have focused on accuracy and resource utilization, but rarely on frames per second (FPS). Notably, there is a lack of direct comparisons between simulators like Isaac Sim and PyBullet, particularly considering the recent emergence of Isaac Sim in RL. Evaluating their performance on identical RL tasks is essential for determine their suitability for large-scale training.

III. Considered Simulation Environments

The performance of RL agents is strongly influenced by the physics simulator in which they are trained. Each simulator uses a physics engine, which is a representation of the real world with its own strengths and limitations [19]. In this study, we will focus on two popular simulators, PyBullet and Isaac sim, which are widely used in robotics and RL research. By comparing and evaluating these environments, we aim to provide insights into their strengths and limitations.

- PyBullet: PyBullet is a physics simulator built on the Bullet physics engine, specifically designed for robotics simulation and machine learning with a focus on sim-to-real transfer. One of its key features is the ease of importing robot models, supporting a wide range of formats including URDF, SDF and MJCF (MuJoCo's model format). A library called Panda_gym [20] has been identified as a useful resource to simplify task creation.
- Issac sim: Isaac sim is a robotics simulation toolkit built on PhysX that provides essential functionality for building virtual robot worlds and experiments[2]. While the USD format remains the primary choice for importing robot models, users also have the flexibility to use URDF and MJCF formats. Orbit [21] and Omniverse Isaac Gym Environments [22] libraries have been identified as resources that can simplify task development. Their new library is Isaac Lab[3], which replaces the previous IsaacGymEnvs and Orbit frameworks and will be the only robot learning framework for Isaac sim.

IV. Considered Tasks

One of the most basic but essential task a robot can perform is to reach a specific target. This basic movement is a crucial starting point for more complex actions and serves as the basis for robots to interact with their environment.

In this context, the primary objective of our task is to guide the end effector to reach a randomly generated target position within a volume of 30cm x 30cm x 30cm.

To achieve this, the observation consists of three distinct components:

- Achieved goal: This component represents the current position of the end effector, specified by three coordinates (x, y, z).
- Desired goal: This component defines the target position, also consisting of three coordinates (x, y, z).
- Observation: This component comprises six coordinates: end effector position and velocity.

The action corresponds to the individual movement of each joint, allowing the robot to adapt its movement to reach the desired target.

The reward function is structured to encourage the robot to reach the goal as defined in (1):

$$reward = -\sqrt{(x_b - x_a)^2 + (y_b - y_a)^2 + (z_b - z_a)^2}, \quad (1)$$

where a is the achieved goal and b is the desired goal.

[1]https://robodk.com/robot/Franka/Emika-Panda, Last viewed: 2024-07-10

[2]https://docs.omniverse.nvidia.com/isaacsim/latest/overview.html. Last viewed: 2024-07-9

[3]https://github.com/isaac-sim/IsaacLab. Last viewed: 2024-07-11

V. Algorithm Used

PPO [11] is a state-of-the-art DRL algorithm [23] that employs an actor-critic architecture. Building upon the trust-region policy optimization (TRPO) algorithm [24], the primary objective of PPO is to optimise the policy while ensuring that the new policy does not significantly diverge from the old policy.

To update the actor's parameters, TRPO utilizes a Kullback-Leibler (KL) divergence penalty [25] on the loss function. The optimisation objective is to maximise the objective function $L(\theta)$, defined in (2), which is a proxy for the expected return:

$$L(\theta) = \hat{\mathbb{E}}_t \left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \hat{A}_t \right], \tag{2}$$

where π_θ and $\pi_{\theta_{old}}$ are the current and old policies respectively, and $\hat{A}_t$ is the advantage function which measures how much better or worse an action is compared to the average action in state s_t [26]. The advantage function is defined in (3):

$$\hat{A}_t = -V(s_t) + r_t + \gamma r_{t+1} + \ldots + \gamma^{T-t+1} r_{T-1} + \gamma^{T-t} V(s_T), \tag{3}$$

where t is the time index in the interval [0,T], with a given trajectory and γ is the discount factor.

In contrast, PPO uses a clipped surrogate objective to update the actor's parameters, which has been shown to yield better results than the KL penalty. Let $r_t(\theta)$ denote the probability ratio in (4):

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \tag{4}$$

Without a constraint, maximisation of $L(\theta)$ would lead to an excessively large policy update. Therefore, PPO modifies the objective function to penalise policy changes that keep $r_t(\theta)$ away from 1. The objective function is defined in (5):

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t \left[\min \left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right], \tag{5}$$

where ϵ is a hyperparameter controlling the clipping range. By employing this clipped surrogate objective, PPO ensures a more stable and efficient learning process, leading to improved performance in various RL tasks. This algorithm is used mainly for its flexibility and robustness, offering a good balance between efficiency, stability and scalability.

VI. Results

One of the most notable differences between Isaac sim and PyBullet is their system requirements. Isaac Sim requires a powerful machine with at least a 4-core Intel i7, 32GB of RAM, and an NVIDIA RTX 3070 GPU. In contrast, PyBullet has modest requirements, needing only a 2003 C++ compiler and a Python wrapper, making it more accessible on a wider range of hardware.

Another crucial aspect to consider is that Isaac sim has implemented curriculum learning [27], which is a significant advantage. Its curriculum manager adjusts environment variables based on a training curriculum, gradually increasing task difficulty as the agent improves, stabilising the learning process.

To evaluate the performance of these two platforms, tests were conducted on a computer running Ubuntu 22.04.1, equipped with an Nvidia GeForce RTX 4090 GPU (24564 MiB VRAM), and an Intel i9-13900KF CPU with 24 cores.

Frames per Second (FPS) was the key metric used to evaluate the performance of the physics simulator. This metric is a critical indicator of simulator performance as it directly affects the speed and efficiency of the training process. In addition, FPS influences the stability of the simulator as it affects the ability to handle complex environments and scale to increasing numbers of parallel simulations. We use this metric because is was used in the comparative analysis of Orbit, IsaacGym, RoboSuite and ManiSkill2 in the paper that introduced Orbit [21].

More specifically, FPS is important for the following reasons:

- Simulation speed: Higher FPS means faster environment updates, allowing more agent interactions per time unit, speeding up the training process.
- Training efficiency: RL algorithms often require extensive interaction with the environment. A higher FPS allows more data to be collected in the same amount of time, resulting in more efficient training.
- Performance comparison: FPS is a key metric for comparing simulators, with higher FPS indicating better performance and the ability to handle more complex environments.

By comparing the FPS of different simulation environments, we can gain insight into their relative performance and suitability for different applications.

Training was performed using the PPO algorithm with the following hyperparameters:

- Actor and critic networks: 2 layers fully connected with 64 units each.
- Optimizer : Adam optimizer [28].
- Number of time steps: 10^6.
- Learning rate: 0.0003.
- Number of steps: 512.
- Batch size: 32.
- Number of epoch: 4.
- Discount factor (γ): 0.9.
- GAE λ: 0.98.
- Clipping parameter: 0.2.
- Entropy coefficient for the loss calculation: 0.01.
- Value function coefficient for the loss calculation: 0.5.
- Maximum value for the gradient clipping: 0.5.

Fig. 2 and Table I compares the training results for the each task using Isaac sim and PyBullet, both with simulator timesteps representing 2 ms and 8 m. Training was performed with the PPO algorithm in several parallel environments, varying from 1 to 2^9 simultaneous environments. 10 independent

tests were performed for each configuration, using different seeds in each test to ensure the reliability of the results. Table I displays the number of parallel environments in the first column, followed by the average frames per second (FPS) and the standard deviation of FPS, calculated from the 10 trials.

The results of the training experiment show an interesting trend. When the reach task is trained with a small number of parallel environments, PyBullet outperforms Isaac sim in terms of FPS. However, as the number of parallel environments increases, PyBullet's improvement is minimal, while Isaac sim shows a significant increase in FPS. This difference becomes more pronounced as the number of parallel environments increases. It can be seen that PyBullet crashes at around 300-350 environments due to lack of memory, whereas GPU-based parallelisation in Isaac sim scales more effectively to accommodate a larger number of environments, demonstrating its superior scalability.

With a simulator step of 8ms, Isaac sim starts to perform better with 4 parallel environments, showing a 23.34% increase in FPS. Whereas with a size step of 2ms, Isaac sim starts to achieve better performance with 8 parallel environments, showing 34.86% increase in FPS. This performance gap widens as the number of parallel environments increases. These percentages are calculated with respect to Isaac Sim with a step size of 8ms.

There are several reasons why this behaviour may be experienced:

- Initial overhead and system complexity: The initial setup and complexity of Isaac sim, including the configuration of advanced graphics resources and GPU hardware, introduces a significant overhead. This overhead is designed to optimise performance in highly parallel environments. In contrast, PyBullet is a lightweight and straightforward simulator, optimised for efficiency and minimal overhead. This simplicity allows PyBullet to achieve high performance even in a single environment.
- Optimization for parallelism: Isaac sim is specifically designed and optimised to handle massive parallelism, making it well suited to running in multiple environments simultaneously. However, its performance may be less efficient and slower in situations with fewer parallel environments. PyBullet, on the other hand, is designed to be fast and efficient in smaller setups that do not require as much parallel processing.
- GPU or CPU usage: Isaac sim uses GPUs, which are extremely efficient at performing parallel operations but may not be as effective at handling small workloads. GPUs are optimised for throughput (the amount of work done in a given time) rather than latency (the time it takes to complete a single task). PyBullet, on the other hand, utilizes CPUs, which provide constant and high throughput at low workloads. CPUs, with fewer but more powerful cores, are designed to handle sequential tasks and provide fast responses, making them ideal for low workload operations.

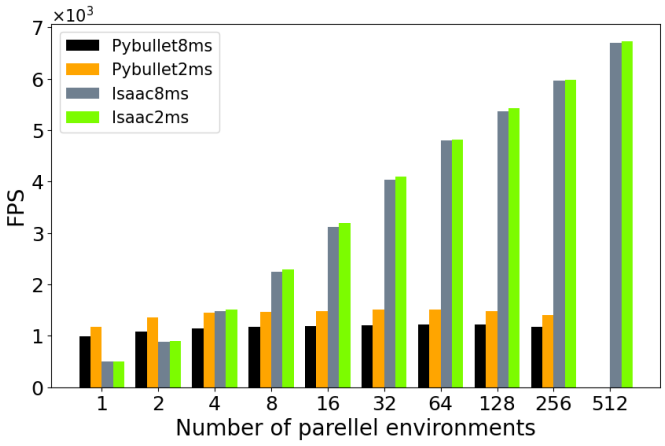

Fig. 2. Comparison of the simulators Isaac sim and PyBullet using the robot panda in the task reach.

TABLE I
COMPARISON OF THE SIMULATORS ISAAC SIM AND PYBULLET USING THE ROBOT PANDA IN THE TASK REACH.

p.env	Isaac8ms (FPS)	Isaac2ms (FPS)	PyBullet8ms (FPS)	PyBullet2ms (FPS)
1	499.0±1.89	100.74±0.13%	198.98±0.28%	235.91±7.2%
2	888.9±3.36	100.80±0.07%	121.82±0.01%	153.19±0.29%
4	1488.2±4.83	101.15±0.05%	76.66±0.28%	97.34±0.19%
8	2251.0±7.26	101.64±0.07%	52.15±0.04%	65.14±0.52%
16	3119.1±12.61	102.28±0.01%	38.13±0.21%	47.47±0.34%
32	4042.0±20.2	101.24±0.04%	29.93±0.07%	37.46±0.23%
64	4797.6±25.85	100.50±0.05%	25.31±0.01%	31.36±0.06%
128	5374.1±41.71	101.03±0.27%	22.95±0.03%	27.61±0.01%
256	5971.1±30.03	100.29±0.2 %	19.61±0.8%	23.40±0.57%
512	6695.5±64.97	100.60±0.11%	*	*

*Result not obtained.

The results also indicate that, for Isaac sim, the simulator timesteps (8ms or 2ms) has no significant impact on the FPS. For PyBullet, however, reducing the step size leads to improves the FPS. Furthermore, despite its ability to handle multiple environments in parallel, Isaac sim achieves an 80-90% success rate with 512 environments running simultaneously. This suggests that with these hyperparameters, optimal performance can not be obtained with a large number of environments in parallel. We also observed that PyBullet achieves a 100% success rate in fewer steps, which may be because the same hyperparameterisation is not optimal for all physics simulator. In addition, PyBullet has simpler and less realistic physics, which could also contribute to this difference.

In our final comparison, we evaluated Isaac sim and PyBullet using camera data as observations. Notably, while Isaac sim can efficiently learn from camera data, PyBullet's performance suffers significantly, with a drastic FPS drop to 3 FPS. This limitation suggests that PyBullet is not well-suited for image-based tasks.

After observing the differences between these two simulators, we ran Isaac sim with different RL libraries for comparison. We chose Isaac sim because the wrappers are already prepared. The new libraries used are rl_games [29], skrl [30] and rsl_rl [31]. The results of this experiment are presented in Fig. 3 and Table II. Although the 8ms step size is

slightly lower in performance, it was used because it provided more stability in the experiments.

Due to implementation differences, we can observe a performance and training time disparity in the results. Skrl, rl_games, and rsl_rl are GPU-optimised, while Stable-Baselines3 is not and performs better with a CPU flag. The performance difference becomes noticeable from 32 parallel environments, with 159.91% more FPS in skrl, 167.97% more in rl_games, and 161.59% in rsl_rl. This increase grows with the number of parallel environments. For example training with 256 environments yields 790.26% more FPS in skrl, 885.01% more in rl_games, and 798.00% in rsl_rl. This performance difference can have a significant impact on the training process for complex task that require extensive computing resources. However, for simpler task such as the reach time, the time difference in relatively small, suggesting that the choice of library may not be critical. Notably, skrl does not produce results running in a single environment.

In addition to all of these benefits, Isaac sim offers even greater flexibility and scalability by providing the option to run environments on multi-GPU training and multi-node training architectures. This means that Isaac sim take full advantage of distributed computing resources, allowing it to scale to meet the demands of complex simulations and large-scale training task. Specifically, multi-GPU training allows each process to run on a dedicated GPU, launching its own instance of Isaac sim. Each process collects its own rollouts during training and its own copy of the policy network. While multi-node training enables training to be scaled beyond multiple GPUs on a single machine. This requires a separate process to be started on each node, allowing training tasks to be distributed across multiple nodes/machines. This makes Isaac sim an ideal choice for large-scale simulations and training tasks that require massive computing resources.

VII. Conclusion

This study compares the performance of Isaac Sim and PyBullet in training a RL model for the reach task with the Franka Emika Panda robotic arm. The results show that PyBullet achieves better performance when training with a small number of environments in parallel, but demonstrates limited improvement as more environments are added. In contrast, Isaac sim demonstrates poor performance with a single environment, but its performance increases significantly as the number of parallel environments increases. This difference can be due to to the design differences between the two simulation platforms: PyBullet is optimised for speed and efficiency, but its architecture is not designed to run multiple environments simultaneously, leading to bottlenecks. PyBullet runs out of memory when trying to run 300-350 environments in parallel. Conversely, Isaac sim is designed for high parallelism, but has a high initial overhead.

It is also worth noting that the step size affects performance significantly in PyBullet. Another important aspect to consider is curriculum learning. In PyBullet this would have to be implemented manually, whereas in Isaac sim it is already integrated. This opens up the possibility of training more complex environments which help to stabilise learning by making the task progressively more challenging as the agent improves.

In addition, Isaac Sim can train using camera data for observations, unlike PyBullet, which suffers FPS drops with image-based data.

Isaac Sim shows a significant FPS boost with GPU-optimized libraries, demonstrating superior performance and flexibility for complex training scenarios.

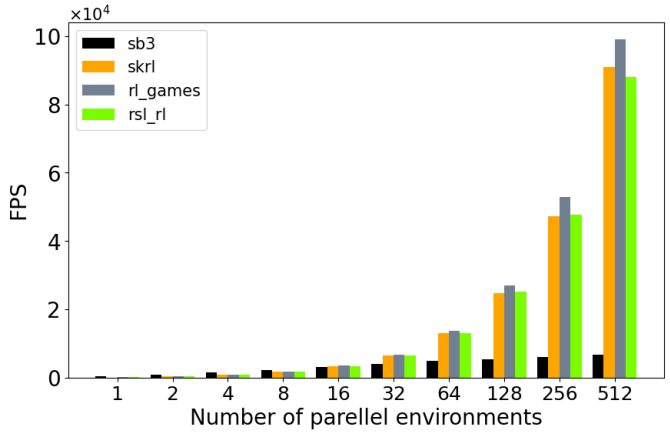

Fig. 3. Comparison of different RL libraries in Isaac sim with the robot panda in the task reach with a step size of 8ms.

TABLE II
Comparison of different RL libraries in Isaac sim with the robot panda in the task reach with a step size of 8ms.

p.env	sb3 (FPS)	skrl (FPS)	rl_games (FPS)	rsl_rl (FPS)
1	499.00±1.89	*	45.07±0.09%	44.73±0.05%
2	888.90±3.36	33.82±0.03%	51.67±0.18%	49.62±0.27%
4	1488.20±4.83	52.43±0.40%	61.41±0.36%	59.21±0.34%
8	2251.00±7.26	73.31±1.11%	80.31±0.49%	77.29±0.17%
16	3119.10±12.61	107.36±0.46%	113.08±0.17%	109.11±0.42%
32	4042.00±20.2	159.91±0.13%	167.97±1.55%	161.59±3.34%
64	4797.60±25.85	269.01±0.49%	284.60±0.09%	269.74±1.00%
128	5374.10±41.71	460.50±0.75%	499.92±1.09%	468.26±1.23%
256	5971.10±30.03	790.26±12.61%	885.01±8.26%	798.00±15.74%
512	6695.50±64.97	1358.27±22.30%	1480.57±1.28%	1314.36±28.11%

*Result not obtained.

A. Recommendations

On the basis of our findings, we would recommend each of the simulation environments in the following cases:

- PyBullet: One of the main advantages of PyBullet is its large community. With PyBullet we are able to train simple tasks at high speed. However, it is not designed to handle multiple parallel environments, so you will not notice a significant difference in performance. Nevertheless, it is highly recommended for beginners in RL using robotics arms, as it is very easy to create a new environment thanks to the Panda_gym library.
- Isaac sim: Isaac sim is a very powerful physics simulator to train task using RL. With Isaac sim we can train task in less time than with PyBullet, thanks to its use

of GPU acceleration. However, it also has it drawbacks, one of which is that the code is significantly more complex. Another is that it requires at high hardware requirements, which not everyone may have access to. It is also still under development, which means that could have some bugs in some of its features. Nevertheless, it is recommended for complex task due to is capabilities and the training speed.

B. Future work

Future work will validate the sim-to-real transfer. This will be done by using the trained RL model on a real robot. We also plan to extend the system's capabilities to handle more complex tasks, such as those involving deformable objects, which are notoriously challenging. Furthermore, we intend to invest the integration of imitation learning with RL, which has the potential to accelerate learning by using demonstrations from humans or other agents. This hybrid approach can exploit the strengths of both paradigms, allowing the RL agent to learn from expert demonstrations while still adapting to novel situations through trial and error. By combining imitation learning with RL, we can potentially reduce exploration time and improve the overall robustness of the learned policy. In addition, it would be interesting to explore the applicability of our approach to different robots, to further demonstrate its versatility and adaptability. It would be also interesting to investigate the effectiveness of different RL algorithms, such as Deep Deterministic Policy Graient (DDPG), Trucated Quantile Critics (TQC), to name a few, on our simulation environments, potentially leading to new insights and improvements in the field of robotics and artificial intelligence.

ACKNOWLEDGMENT

This paper has been supported by the Basque program BIKAINTEK 2023, grant agreement No. 012-B2/2023, in support of the PhD thesis *Deep Reinforcement Learning Para Robótica Basada En Visión Artificial*.

REFERENCES

[1] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey", In *Journal of Artificial Intelligence Research (JAIR)*, 1996, vol. 4, pp. 237-285.

[2] H. Nguyen and H. La, "Review of deep reinforcement learning for robot manipulation", in *3rd IEEE International Conference on Robotic Computing (IRC)*, 2019, pp. 590-595.

[3] R. S. Sutton and A. G. Barto, "Reinforcement learning: An introduction, 2nd edition", in *MIT Press*, 2018.

[4] V. Mnih et al., "Playing Atari with deep reinforcement learning", *arXiv preprint arXiv:1312.5602*, 2013.

[5] D. Silver et al., "Mastering the game of Go without human knowledge", in *Nature*, 2017, vol. 550, no. 7676, pp. 354-359.

[6] B. R. Kiran et al., "Deep reinforcement learning for autonomous driving: A survey", in *IEEE Transactions on Intelligent Transportation Systems*, 2021, vol. 23, no. 6, pp. 4909-4926.

[7] T. Haarnoja, S. Ha, A. Zhou, J. Tan, G. Tucker, and S. Levine, 'Learning to walk via deep reinforcement learning". in *Robotic Science and Systems (RSS)*, 2018

[8] M. Popova, O. Isayev, and A. Tropsha, "Deep reinforcement learning for de novo drug design", in *Science Advances*, 2018, vol. 4, no. 7, eaap7885.

[9] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey", in *The International Journal of Robotics Research*, 2013, vol. 32, no. 11, pp. 1238-1274.

[10] A. Raffin et al., "Stable-baselines3: Reliable reinforcement learning implementations", in *Journal of Machine Learning Research (JMLR)*, 2021, vol. 22, no. 268, pp. 1-8.

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

[12] T. P. Lillicrap eta al., "Continuous control with deep reinforcement learning", *International Conference on Learning Representations (ICLR)*, 2016.

[13] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor", in *35th International Conference on Machine Learning (ICML)*, 2018, pp. 1861-1870.

[14] T. Erez, Y. Tassa, and E. Todorov, "Simulation tools for model-based robotics: Comparison of bullet, Havok, MuJoCo, ODE and PhysX", in *IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 4397–4404.

[15] M. Körber, J. Lange, S. Rediske, S. Steinmann, and R. Glück, "Comparing popular simulation environments in the scope of robotics and reinforcement learning", in *arXiv preprint arXiv:2103.04616*, 2021.

[16] A. Ayala, F. Cruz, D. Campos, R. Rubio, B. Fernandes, and R. Dazeley, "A Comparison of Humanoid Robot Simulators: A Quantitative Approach", in *10th IEEE International Conference on Development and Learning and Epigenetic Robotics (ICDL-EpiRob)*, 2020, pp. 1–6.

[17] L. Pitonakova, M. Giuliani, A. Pipe, and A. Winfield, "Feature and performance comparison of the V-REP, Gazebo and ARGoS robot simulators", in *Annual Conference Towards Autonomous Robotic Systems (TAROS)*, 2018, pp. 357-368.

[18] F. P. Audonnet, A. Hamilton, and G. Aragon-Camarasa, "A systematic comparison of simulation software for robotic arm manipulation using ros2", in *22nd International Conference on Control, Automation and Systems (ICCAS)*, 2022, pp. 755-762.

[19] I. Millington, "Game Physics Engine Development", in *CRC Press*, 2007.

[20] Q. Gallouédec, N. Cazin, E. Dellandréa, and L. Chen, "panda-gym: Open-source goal-conditioned environments for robotic learning", in *4th Robot Learning Workshop: Self-Supervised and Lifelong Learning at NeurIPS*, 2021.

[21] M. Mittal et al., "Orbit: A unified simulation framework for interactive robot learning environments", in *IEEE Robotics and Automation Letters (RA-L)*, 2023, vol. 8, no. 6, pp. 3740-3747.

[22] V. Makoviychuk et al., "Isaac gym: High performance GPU-based physics simulation for robot learning", in *35th Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021.

[23] I. Goodfellow, Y. Bengio, and A. Courville, "Deep learning", in *MIT Press*, 2016.

[24] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization", in *32nd International Conference on Machine Learning (ICML), PMLR*, 2015, pp. 1889-1897.

[25] S. Kullback and R. A. Leibler, "On information and sufficiency", in *The Annals of Mathematical Statistics*, 1951, vol. 22, no. 1, pp. 79-86.

[26] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation", in *International Conference on Learning Representations (ICLR)*, 2016.

[27] R. Portelas, C. Colas, L. Weng, K. Hofmann, and P. Y. Oudeyer, "Automatic curriculum learning for deep RL: A short survey", in *International Joint Conference on Artificial Intelligence (IJCAI)*, 2020.

[28] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization", in *3rd International Conference on Learning Representation (ICLR)*, 2015.

[29] A. Serrano-Muñoz, D. Chrysostomou, S. Bøgh, and N. Arana-Arexolaleiba, "skrl: Modular and flexible library for reinforcement learning", *Journal of Machine Learning Research (JMLR)*, 2023, vol. 24, no. 254, pp. 1-9.

[30] D. Makoviichuk and V. Makoviychuk, "rl-games: A high-performance framework for reinforcement learning", available in *https://github.com/Denys88/rl_games (accessed on 18 July 2024)*, 2022.

[31] N. Rudin, D. Hoeller, P. Reist, and M. Hutter, "Learning to walk in minutes using massively parallel deep reinforcement learning", in *Conference on Robot Learning (CoRL)*, 2022, pp. 91-100.