REINFORCEMENT LEARNING EXPERIMENTS AND BENCHMARK FOR SOLVING ROBOTIC REACHING TASKS

Pierre Aumjaud*
University College Dublin
Dublin, Ireland
pierre.aumjaud@ucd.ie

David McAuliffe
Resero Ltd
Dublin, Ireland
david.mcauliffe@resero.io

Francisco Javier Rodríguez Lera
Universidad de León
León, Spain
fjrodl@unileon.es

Philip Cardiff
University College Dublin
Dublin, Ireland
philip.cardiff@ucd.ie

ABSTRACT

Reinforcement learning has shown great promise in robotics thanks to its ability to develop efficient robotic control procedures through self-training. In particular, reinforcement learning has been successfully applied to solving the reaching task with robotic arms. In this paper, we define a robust, reproducible and systematic experimental procedure to compare the performance of various model-free algorithms at solving this task. The policies are trained in simulation and are then transferred to a physical robotic manipulator. It is shown that augmenting the reward signal with the Hindsight Experience Replay exploration technique increases the average return of off-policy agents between 7 and 9 folds when the target position is initialised randomly at the beginning of each episode.

Keywords Reinforcement learning · Robotics · Benchmark · Model-free · Sim-to-real

1 Introduction

Reinforcement learning (RL) is a paradigm in the field of machine learning that has recently gained tremendous interest [1]. Unlike supervised learning, RL is capable of learning sequential decision-making policies by interacting with an environment, while sometimes achieving super-human performance [2]. RL requires a reward function to be defined, which is why it has naturally been applied to games or toy problems [3]. However, real-world applications are still scarce and challenging. Robotics allow researchers to define controlled training environments relatively easily, thus RL has found many successful applications in this field.

Reinforcement learning approaches can be grouped in two categories: model-based – where the agent first learns a model of the environment and then uses it to make predictions; and model-free – where the agent directly learns a policy based only on its interactions with the environment. Historically, model-based RL has first been applied to robotics [4] due to its high sample efficiency, allowing agents to solve tasks with a limited number of policy iterations. Nevertheless, it is sometimes challenging to build accurate models for complex robotics problems, causing model-based approaches to often suffer from poor asymptotic performance. The invention of Deep Deterministic Policy Gradients [5] has paved the way for other model-free RL algorithms to successfully solve problems with continuous state and action space. Since then, considerable improvements have occurred in the field with the invention of Twin Delayed Deep Deterministic Policy Gradient [6], Soft Actor Critic [7, 8] and Hindsight Experience Replay [9], which addresses the problem of sparse rewards.

*Corresponding author

One of the most common robotic tasks studied in an RL context is trajectory planning to reach a target position. A number of approaches have been adopted to solve this problem, including model-based approaches [10, 11], model-free approaches [12–19], a combination of both [20], or learning acceleration via closed-loop policies [21].

In this article, we will consider solving the problem of reaching target positions both using a robotic simulator and a physical manipulator. The equipment, training environments and methods adopted to perform the performance benchmark of a number of model-free RL algorithms are described in Section 2. The comparative results are reported and discussed in Section 3, both in terms of training convergence and evaluation metrics. Finally, a general conclusion is given and directions for future works are outlined in Section 4.

2 Environments and Methods

2.1 Manipulator Description

The robotic arm used in this work is the WidowX MKII manipulator by Trossen Robotics [22], see Fig. 1b. It is a 6-joints robotic arm possessing an 82 cm diameter reach and equipped with Dynamixel servo actuators and parallel grippers. The robot arm is compatible with the Robot Operating System (ROS) [23] and is shipped with its associated ROS packages. The Replab project [24] has previously employed the same robot arm to solve grasping tasks and provide a reproducible benchmark platform for robotics learning. The learning environments used here are adapted from those in the Replab project. The Pybullet physics engine [25] simulates the response of the robot arm in a virtual environment, which accelerates the training process compared to training on the physical robot alone.

(a) Virtual environment (b) Physical environment

Figure 1: The simulated (Pybullet) and physical (WidowX MKII manipulator) learning environments in their initial episode position. The target position is indicated by a green sphere.

2.2 Reaching Task and Environment Definition

The reaching task considered here is a well-known problem in the robotic manipulation field. It consists of determining the optimal sequence of actions required to bring the robot arm's end effector as close as possible to given target coordinates within its workspace.

In order to solve the reaching task in an RL context, the problem is formalised as a Markov Decision Process. The environment's *state* s_t is defined as an array holding the current spatial coordinates of the robot's end effector and the angular position of each joint:

$$\forall s_t \in S, \qquad s_t = [x_e, y_e, z_e, \theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6]_t \tag{1}$$

where (x_e, y_e, z_e) are the Cartesian coordinates of the end effector and θ_i is the angular position of joint i in radians. The *actions* a_t sent to the environment are defined as an array holding the changes in joint angle from the previous position:

$$\forall a_t \in A, \qquad a_t = [\delta_1, \delta_2, \delta_3, \delta_4, \delta_5, \delta_6]_t \tag{2}$$

where δ_i is the change in angular position from the previous position of joint i. S and A denote the set of possible states and actions, respectively. These spaces are finite and continuous in this context. The *reward* r_t is defined as the negative, squared, L2 norm of the vector between the current end effector position and the goal position (dense reward setting):

$$\forall r_t \in \mathbb{R}^-, \qquad r_t = -[(x_e - x_g)^2 + (y_e - y_g)^2 + (z_e - z_g)^2]_t \tag{3}$$

where (x_g, y_g, z_g) are the Cartesian coordinates of the goal pose. Each learning iteration is composed of an action sent, a state and a reward received by the agent, which defines a *timestep* t. A sequence of 100 timesteps defines one *episode*, unless the termination condition – the episode ends if the distance between the end effector and the goal is less than 0.5 mm – is reached before this. The sum of the rewards over one episode defines the *return* R_t, see equation below.

$$R_t = \sum_{t=1}^{100} r_t \tag{4}$$

The *policy* maps the environment's current state to the next action with the objective of maximising the return. A deterministic policy is a function defined as $\pi_d : S \mapsto A$ and a stochastic policy is a probability distribution defined as $\pi_s(A|S)$, meaning that the probability of selecting an action depends on the current state.

The robot arm joints are always initialised with the same angle positions at the beginning of each episode both in the virtual and the physical environments, as shown in Fig. 1. The goal position is either constant across all training and evaluation episodes (fixed goal) or initialised randomly at the beginning of each episode (random goal). Either way, the goal position is always within the reach of the robot arm.

The learning environments are implemented with OpenAI Gym [26], a standard toolkit for developing and comparing reinforcement learning algorithms. A goal-oriented Gym environment [26] is also implemented in order to explore the observation space more efficiently using Hindsight Experience Replay. In this case, the desired goal and achieved goal coordinates are also included in the state definition. In total, four learning environments are implemented:

- **Env1**: Pybullet simulation + fixed goal
- **Env2**: Pybullet simulation + random goal
- **Env3**: Physical robot + fixed goal
- **Env4**: Physical robot + random goal

The policies are trained on the Pybullet environments only (i.e. Env1 and Env2), however they are evaluated both in simulation and on the physical robot.

2.3 Experiments and Benchmark

It is proposed to compare the performance of a number of RL algorithms at solving the reaching task using the environments described above. The RL algorithms considered in the benchmark are:

- Advantage Actor Critic (A2C) [27]
- Actor Critic using Kronecker-Factored Trust Region (ACKTR) [28]
- Deep Deterministic Policy Gradients (DDPG) [5]
- Proximal Policy Optimization (PPO) [29]
- Twin Delayed Deep Deterministic Policy Gradient (TD3) [6]
- Soft Actor Critic (SAC) [7, 8]
- Trust Region Policy Optimization (TRPO) [30]
- SAC + Hindsight Experience Replay (HER) [9]

- TD3 + Hindsight Experience Replay [9]

All these algorithms are compatible with continuous state and action spaces environments, which is a requirement for the reaching task considered here. DDPG, SAC, TD3 and HER learn a deterministic policy and a stochastic policy is learnt by the rest of the algorithms. The two off-policy algorithms – SAC and TD3 – are combined with HER in an effort to reduce sample complexity. The algorithms are applied using the Stable Baselines implementation [31].

The hyperparameters of each algorithm are first optimised with the Optuna framework [32] over 100 training runs of 100 episodes each. In each run, the hyperparameters are sampled with the Tree-structured Parzen Estimator and unpromising runs are stopped early using a median pruner. Subsequently, RL agents are trained in the virtual environments Env1 and Env2 for 200,000 timesteps using each algorithm described above. The A2C, ACKTR and PPO agents are trained over 8 parallel environments in order to speed up the learning process (the other algorithm's implementations do not currently support parallelisation). The training is performed on University College Dublin's Sonic HPC cluster [33], which is equipped of two NVIDIA Tesla V100 16 GB GPUs per node. The training is repeated over 10 different initialisation seeds and the metrics are averaged in order to strengthen the robustness of the experiments. Once the optimal policies are learnt in the virtual environments, they are deployed and tested both in the virtual (Env1 and Env2) and physical environments (Env3 and Env4). All policies are evaluated with their associated agent for 100 episodes. A random policy is also evaluated to serve as a reference. In the case of physical environments, the model weights of the best-performing seed run (in terms of return) are transferred to the agent controlling the physical robot. In the case of the virtual environment, all seed runs are evaluated and the metrics are averaged. The following evaluation metrics are reported:

- **Average return**: Return averaged over the 100 evaluation episodes.
- **Train time**: Average training time over the 10 random seeds in minutes.
- **Success ratio @X mm**: number of episodes where the end effector ended within a distance threshold of X mm from the goal position, divided by the total number of evaluation episodes. The distance thresholds considered are 50 mm, 20 mm, 10 mm and 5 mm.
- **Reach time @X mm**: number of timesteps required to reach the target during a successful episode within a distance threshold of X mm.

The source code repository and an explanatory video are available at `https://git.io/JJJdu` and `https://youtu.be/-N-6Me8UkFk`.

3 Results and Discussion

3.1 Training Convergence

The training convergence curves averaged over the 10 initialisation seeds are shown in Figs. 2 and 3 for the two virtual environments considered, Env1 (fixed goal) and Env2 (random goal). The curves are smoothed with a rolling window average of 50 for clarity purposes. A comparison of the convergence curve between Env1 and Env2 is also reported for each algorithm individually, see Fig. 4. In these plots, the middle line represents the average reward over the 10 seed runs at each timestep and a shaded area is drawn at $\pm$ the standard deviation.

It can be noted that Env2 (random goal) is a more challenging task to solve since the average return is generally lower than that of Env1 (fixed goal) at the end of the training process. All algorithms manage to learn a successful policy in Env1 after 200,000 timesteps. DDPG, TRPO and SAC achieve the highest sample efficiency as their learning curves plateau the earliest after less than 20,000 timesteps, whereas A2C exhibits the highest sample complexity. Moreover, TRPO, TD3 and ACKTR produce the most stable and repeatable training as their standard deviations are the lowest at the end of training, see Fig. 4.

When the goal is initialised randomly at the beginning of the episode, most algorithms fail to learn a successful policy. Some algorithms perform even worse than a random policy, for example DDPG in Env2. However, combining SAC and TD3 with HER yields a healthy training, even in Env2, as illustrated in Fig. 3. Accelerating the observation space exploration this way tends to reduce slightly the sample efficiency and the training stability of the off-policy algorithms SAC and TD3, see Figs. 4h and 4i.

3.2 Evaluation Metrics

Although the training convergence curve can be a good proxy to measure the performance of an RL agent, it is more precise to consider evaluation metrics. The evaluation metrics of all trained agents and all four environments considered

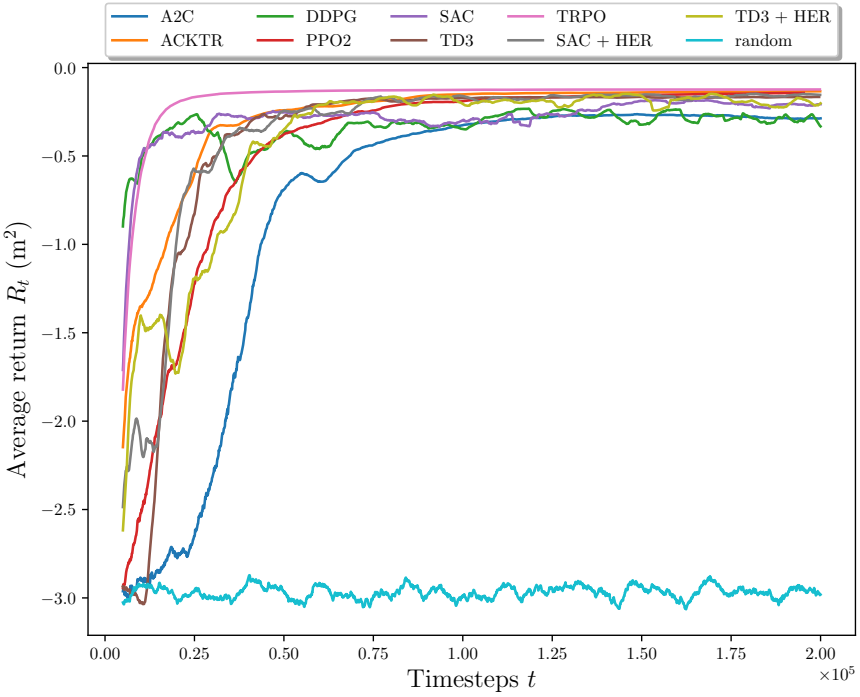


Figure 2: Convergence curves of the algorithms solving Env1 (fixed goal)

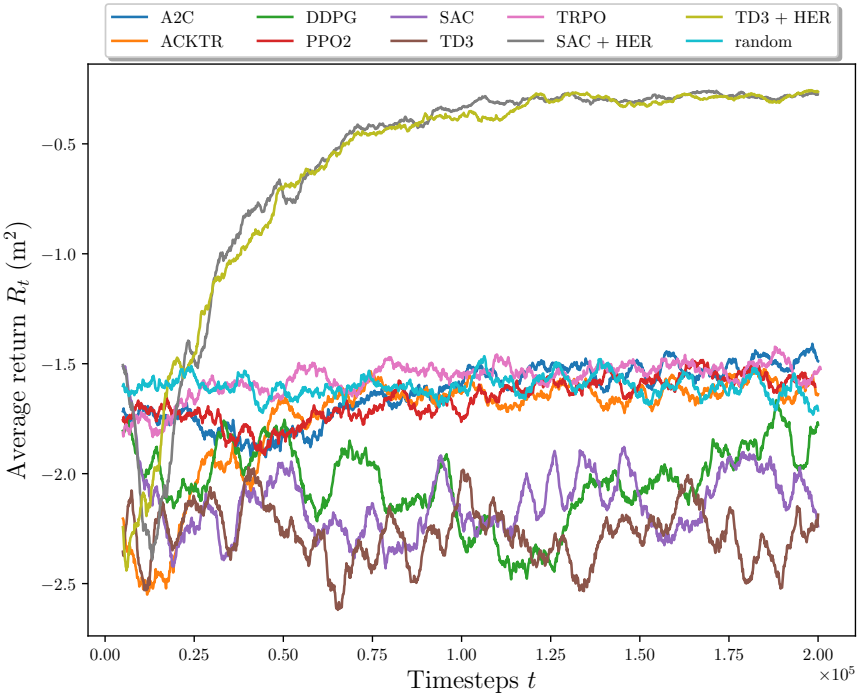
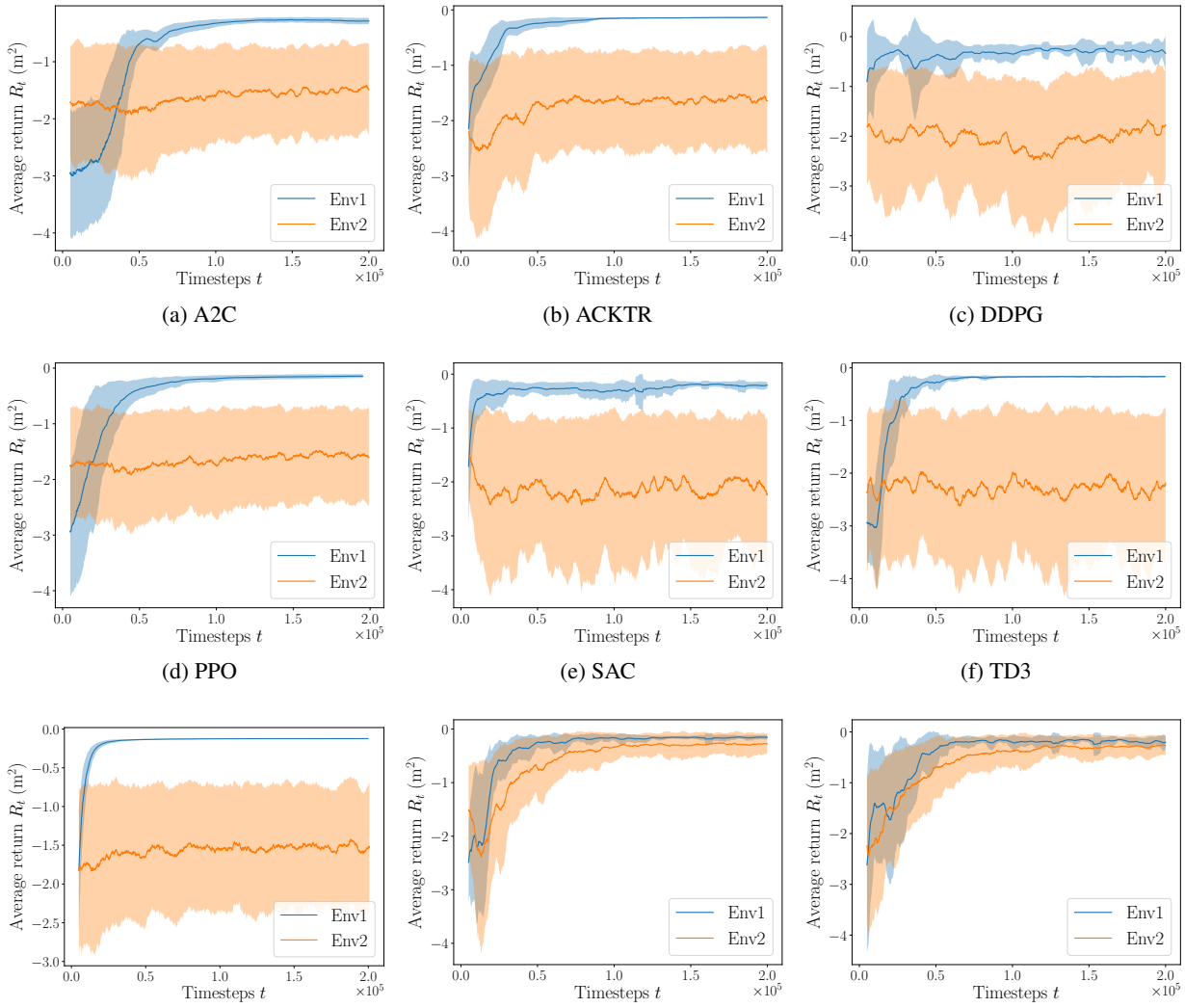


Figure 3: Convergence curves of the algorithms solving Env2 (random goal)



(a) A2C (b) ACKTR (c) DDPG

(d) PPO (e) SAC (f) TD3

(g) TRPO (h) HER + SAC (i) HER + TD3

Figure 4: Convergence curve comparison between Env1 (fixed goal – in blue) and Env2 (goal initialised randomly – in orange)

are reported in Tables 1 and 2. The metrics are averaged over 10 trainings initialised with different seeds; the standard deviations are not reported here for clarity but are available upon request or in the associated source code repository.

When the goal position is fixed, the average return of successful agents is much higher than that of the random policy, which agrees with the observations from the convergence curves, see for example TRPO in Env1. In this case, the best-performing algorithms in terms of return are TRPO and TD3. When the goal position is initialised randomly however, only the algorithms combined with HER achieve an average return significantly higher than random.

The training walltime is substantially shorter for algorithms supporting parallelisation, as expected. For instance with a train walltime of 68 seconds, PPO is almost 19 times faster to train than SAC + HER in Env2. It is also about twice as fast as a random policy that does not exploit any deep learning architectures nor uses parallel environments.

The success ratio measures the accuracy of an agent at reaching the target and the reach time measures their speed at reaching the target during an episode (note that the maximum reach time is equal to the length of an episode, 100).

In the least challenging environment – Env1 (fixed goal), all algorithms achieve a high success ratio and a fast reach time when the distance threshold is large (i.e. 50 mm). As expected, the agent's accuracy decreases as the distance threshold is becoming tighter, and it drops dramatically beyond a distance threshold of 20 mm. In this situation, TD3 is

the most accurate algorithm with a success ratio @5 mm of 0.5. The agents capable of reaching their target quickly are generally those exhibiting a high success ratio and average return. The fastest agents are TD3 and TRPO with a reach time of around 9 timesteps at the 50 mm threshold, although the other successful agents are only slightly slower.

When the goal is initialised randomly (Env2), fewer agents managed to learn a successful policy than in Env1, which is reflected by the much lower average return. The best-performing agents are those leveraging the HER exploration strategy, i.e. SAC + HER and TD3 + HER with a success ratio @50 mm of 0.67 and 0.64, respectively. This phenomenon was also observed in the training convergence, see Fig. 4. Similar to Env1, the success ratio diminishes rapidly as the distance threshold reduces. The negligible success ratios @50 mm observed in the other algorithms are comparable to that of a random policy, meaning that the agent has occasionally and fortuitously reached the target.

A sharp decrease in performance is globally observed when trained policies are transferred to the physical robot and evaluated in Env3 and Env4, both in terms of average return and success ratio. This could be further improved by continuing the training from the virtual environments in the physical environments, as the Pybullet simulation does not take into account the joint friction and other imperfections of the physical manipulator. Such transfer from a simulated to the physical world is a well-known challenge in robotics, often referred to as sim-to-real.

However, it is worth noting that PPO, TD3 and TD3 + HER achieved a relatively successful policy transfer from the simulated (Env1) to the physical environment (Env3) as exhibited by their success ratio @50 mm of 1 and a comparable reach time. Finally in the case of the physical environment with random goal (Env4), the only successful policies are those combined with HER. This was also noted in the associated virtual environment with random goal, Env2..

4 Conclusion and Future Work

A benchmark procedure is described in the present paper to compare the performance of various model-free RL algorithms at solving the reaching task with a robot manipulator. It is found that it is generally more challenging to solve a reaching task where the target coordinates are initialised randomly at the beginning of each episode. In a fixed goal setting, the highest performance in terms of average return, success ratio and reach time is achieved by TD3 and TRPO. The highest sample efficiency is achieved by SAC, TRPO and DDPG. Augmenting the intrinsic reward signal of off-policy algorithms such as SAC and TD3 using the HER exploration strategy tends to destabilise slightly the training – both in terms of repeatability and sample efficiency; however it also allows the agent to learn an efficient policy in environments where the goal is initialised randomly. Transferring policies from virtual to physical environments proved to be challenging, however similar conclusions were drawn as to which algorithms perform best. Combining TD3 with HER proved to be the most efficient strategy in physical environments.

Although the methodology described here provides a systematic and reproducible experimental procedure, the authors acknowledge that the benchmark results could be further improved by additional work.

Firstly, some algorithms failed to solve the reaching task completely when the goal is initialised randomly, despite the dense reward setting, see Fig. 2. This is likely due to failing to identify optimal hyperparameters for the configuration in question. Selecting appropriate hyperparameters for training RL agents is notoriously difficult [7]. A more exhaustive hyperparameter search may improve the performance. Additionally, scaling the environment's action space to lie in the interval [-1, 1] could help with the training convergence.

Secondly, the policy transfer from the simulated to the physical environment appears to be challenging as policies often fail to generalise to the physical system (also known as sim-to-real). By nature, there are inherent discrepancies between the dynamics of a simulator and the real-world. In particular, a physical robotic arm is subject to the following noise sources and non-stationarities: a) the position and speed resolution inherent to the Dynamixel servo actuators used by the WidowX MKII; and b) overheating of the actuators and wear of the body parts. These physical imperfections tend to increase the compliance margin between the desired goal position and the actual position. Performing a rigorous calibration of the physical robot would ensure that the arm's dynamics in the simulator closely match that of the real world, thus reducing this reality gap.

Another possible solution to account for these irregularities consists in introducing noise in the simulator by altering joint friction coefficients or link masses for example, thus effectively creating a collection of imperfect training environments. The most realistic virtual environment can be identified by training RL agents on these imperfect environments, deploying the learnt policies on the physical robot and selecting the one with the smallest observed difference.

The discrepancies observed between virtual and physical environments can be further reduced by executing part of the training on the physical environment in order to adjust the policy to the real world. In addition, the robot's accuracy could be further increased by restricting the joint's amplitude motion between each timestep (i.e. reducing the range of the action space).

Table 1: Evaluation metrics of all trained agents for the four environments

Environment	Algorithm	Average return R_t (m^2)	Average train wall-time (s)	Success ratio @50 mm	Reach time @50 mm	Success ratio @20 mm	Reach time @20 mm
Env1 (simulation + fixed goal)	A2C	-0.30	55.0	0.94	18.01	0.26	37.0
	ACKTR	-0.13	55.3	1.00	9.26	0.99	13.5
	DDPG	-0.29	635.0	0.90	9.2	0.30	12.3
	PPO	-0.14	65.0	1.00	9.7	0.71	14.2
	SAC	-0.18	973.8	1.00	10.6	0.60	24.5
	TD3	-0.12	712.8	1.00	9.0	1.00	12.0
	TRPO	-0.12	267.3	1.00	9.0	1.00	12.0
	SAC + HER	-0.15	1097.4	1.00	9.2	0.80	12.1
	TD3 + HER	-0.20	984.1	0.80	9.0	0.70	12.3
	Random policy	-2.98	134.1	0.00	N/A	0.00	N/A
Env2 (simulation + random goal)	A2C	-1.49	72.3	0.03	30.8	0.00	N/A
	ACKTR	-1.64	74.2	0.04	48.1	0.00	N/A
	DDPG	-1.94	523.2	0.02	39.8	0.00	N/A
	PPO	-2.54	68.1	0.04	31.7	0.00	N/A
	SAC	-2.59	1063.9	0.02	49.7	0.00	N/A
	TD3	-2.19	783.7	0.03	34.9	0.00	N/A
	TRPO	-1.43	191.8	0.05	25.7	0.00	N/A
	SAC + HER	-0.27	1285.7	0.67	11.4	0.10	27.2
	TD3 + HER	-0.32	980.7	0.64	11.1	0.17	19.3
	Random policy	-1.68	132.6	0.05	22.0	0.00	N/A
Env3 (physical robot + fixed goal)	A2C	-1.08	N/A	0.00	N/A	0.00	N/A
	ACKTR	-0.28	N/A	0.95	13.0	0.00	N/A
	DDPG	-0.71	N/A	0.00	N/A	0.00	N/A
	PPO	-0.25	N/A	1.00	16.9	0.00	N/A
	SAC	-0.33	N/A	0.35	10.0	0.00	N/A
	TD3	-0.19	N/A	1.00	10.0	0.00	N/A
	TRPO	-0.53	N/A	0.00	N/A	0.00	N/A
	SAC + HER	-0.68	N/A	0.00	N/A	0.00	N/A
	TD3 + HER	-0.28	N/A	1.00	10.0	0.00	N/A
	Random policy	-2.19	N/A	0.00	N/A	0.00	N/A
Env4 (physical robot + random goal)	A2C	-1.29	N/A	0.10	13.0	0.00	N/A
	ACKTR	-1.82	N/A	0.05	35.0	0.00	N/A
	DDPG	-1.55	N/A	0.05	27.0	0.00	N/A
	PPO	-1.31	N/A	0.00	N/A	0.00	N/A
	SAC	-1.43	N/A	0.00	N/A	0.00	N/A
	TD3	-1.63	N/A	0.00	N/A	0.00	N/A
	TRPO	-1.76	N/A	0.00	N/A	0.00	N/A
	SAC + HER	-0.39	N/A	0.50	17.9	0.05	9.0
	TD3 + HER	-0.28	N/A	0.75	18.5	0.20	17.5
	Random policy	-1.57	N/A	0.00	N/A	0.00	N/A

Table 2: Evaluation metrics of all trained agents for the four environments (continued)

Environment	Algorithm	Success ratio @10 mm	Reach time @10 mm	Success ratio @5 mm	Reach time @5 mm
Env1 (simulation + fixed goal)	A2C	0.04	63.8	0.01	91.8
	ACKTR	0.73	22.1	0.20	43.3
	DDPG	0.20	12.5	0.00	N/A
	PPO	0.41	17.1	0.08	40.3
	SAC	0.10	14.0	0.10	15.0
	TD3	1.00	12.5	0.50	18.2
	TRPO	0.81	12.4	0.22	38.5
	SAC + HER	0.50	12.6	0.00	N/A
	TD3 + HER	0.10	13.0	0.00	N/A
	Random policy	0.00	N/A	0.00	N/A
Env2 (simulation + random goal)	A2C	0.00	N/A	0.00	N/A
	ACKTR	0.00	N/A	0.00	N/A
	DDPG	0.00	N/A	0.00	N/A
	PPO	0.00	N/A	0.00	N/A
	SAC	0.00	N/A	0.00	N/A
	TD3	0.00	N/A	0.00	N/A
	TRPO	0.00	N/A	0.00	N/A
	SAC + HER	0.01	37.1	0.00	N/A
	TD3 + HER	0.03	25.9	0.01	28.6
	Random policy	0.00	N/A	0.00	N/A
Env3 (physical robot + fixed goal)	A2C	0.00	N/A	0.00	N/A
	ACKTR	0.00	N/A	0.00	N/A
	DDPG	0.00	N/A	0.00	N/A
	PPO	0.00	N/A	0.00	N/A
	SAC	0.00	N/A	0.00	N/A
	TD3	0.00	N/A	0.00	N/A
	TRPO	0.00	N/A	0.00	N/A
	SAC + HER	0.00	N/A	0.00	N/A
	TD3 + HER	0.00	N/A	0.00	N/A
	Random policy	0.00	N/A	0.00	N/A
Env4 (physical robot + random goal)	A2C	0.00	N/A	0.00	N/A
	ACKTR	0.00	N/A	0.00	N/A
	DDPG	0.00	N/A	0.00	N/A
	PPO	0.00	N/A	0.00	N/A
	SAC	0.00	N/A	0.00	N/A
	TD3	0.00	N/A	0.00	N/A
	TRPO	0.00	N/A	0.00	N/A
	SAC + HER	0.00	N/A	0.00	N/A
	TD3 + HER	0.00	N/A	0.00	N/A
	Random policy	0.00	N/A	0.00	N/A

Lastly, the complexity of the task to solve could be further increased so that the algorithm's performance can be compared in more detail. Such increases in complexity could be brought by considering a sparse reward setting, where the reward is only incremented if the end effector is positioned close enough to the goal position. Solving such a task would greatly benefit from exploration enhancement techniques such as HER or curiosity-driven exploration [34] where the intrinsic reward signal coming from the environment is augmented. The task difficulty may also be increased by defining a more challenging reward function. For example, the orientation of the arm's end effector may be included in the reward definition. Additionally, obstacles could be included in the environment with a reward penalty applied at each collision. Finally, the environment's complexity could be increased by allowing the target to move randomly at each timestep for instance.

Acknowledgements

This Career-FIT project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 713654.

References

[1] Sutton, R., Barto, A.: Reinforcement Learning: An Introduction, 2nd edition, vol. 258 (2018). doi:10.1109/TNN.1998.712192. URL https://ieeexplore.ieee.org/document/712192

[2] Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., Hassabis, D.: Mastering the game of Go with deep neural networks and tree search. Nature **529**(7587), 484–489 (2016). doi:10.1038/nature16961. URL http://dx.doi.org/10.1038/nature16961

[3] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., Hassabis, D.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529–533 (2015). doi:10.1038/nature14236. URL http://dx.doi.org/10.1038/nature14236

[4] Deisenroth, M.P., Rasmussen, C.E.: PILCO: A Model-Based and Data-Efficient Approach to Policy Search. In: Proceedings of the 28 th International Conference on Machine Learning, pp. 465–472 (2011). doi:10.1080/0034408960910404. URL https://dl.acm.org/doi/10.5555/3104482.3104541

[5] Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., Wierstra, D.: Continuous control with deep reinforcement learning. 4th International Conference on Learning Representation (2016). URL http://arxiv.org/abs/1509.02971

[6] Fujimoto, S., Van Hoof, H., Meger, D.: Addressing Function Approximation Error in Actor-Critic Methods. 35th International Conference on Machine Learning, ICML 2018 **4**, 2587–2601 (2018). URL http://arxiv.org/abs/1802.09477

[7] Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P., Levine, S.: Soft Actor-Critic Algorithms and Applications. Computing Research Repository CoRR (2018). URL http://arxiv.org/abs/1812.05905

[8] Haarnoja, T., Zhou, A., Abbeel, P., Levine, S.: Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. International Conference on Machine Learning (2018). URL http://arxiv.org/abs/1801.01290

[9] Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Abbeel, P., Zaremba, W.: Hindsight Experience Replay. Advances in Neural Information Processing Systems 30 (NIPS) (2017). URL http://arxiv.org/abs/1707.01495

[10] Devin, C., Gupta, A., Darrell, T., Abbeel, P., Levine, S.: Learning Modular Neural Network Policies for Multi-task and Multi-robot Transfer. Proceedings - IEEE International Conference on Robotics and Automation pp. 2169–2176 (2017). doi:10.1109/ICRA.2017.7989250. URL https://ieeexplore.ieee.org/document/7989250

[11] Gupta, A., Devin, C., Liu, Y., Abbeel, P., Levine, S.: Learning Invariant Feature Spaces to Transfer Skills with Reinforcement Learning. International Conference on Learning Representations (2017). URL http://arxiv.org/abs/1703.02949

[12] Plappert, M., Andrychowicz, M., Ray, A., McGrew, B., Baker, B., Powell, G., Schneider, J., Tobin, J., Chociej, M., Welinder, P., Kumar, V., Zaremba, W.: Multi-Goal Reinforcement Learning: Challenging Robotics Environments

and Request for Research. Computing Research Repository CoRR (2018). URL http://arxiv.org/abs/1802.09464

[13] Chen, T., Murali, A., Gupta, A.: Hardware Conditioned Policies for Multi-Robot Transfer Learning. 32nd Conference on Neural Information Processing Systems (NeurIPS) (2018). URL http://arxiv.org/abs/1811.09864

[14] Rupam Mahmood, A., Korenkevych, D., Komer, B.J., Bergstra, J.: Setting up a Reinforcement Learning Task with a Real-World Robot. IEEE International Conference on Intelligent Robots and Systems pp. 4635–4640 (2018). doi:10.1109/IROS.2018.8593894. URL https://ieeexplore.ieee.org/document/8593894

[15] Tavakoli, A., Pardo, F., Kormushev, P.: Action Branching Architectures for Deep Reinforcement Learning. 31st Conference on Neural Information Processing Systems (NIPS) (2017). URL http://arxiv.org/abs/1711.08946

[16] Gu, S., Holly, E., Lillicrap, T., Levine, S.: Deep Reinforcement Learning for Robotic Manipulation with Asynchronous Off-Policy Updates. Computing Research Repository CoRR (2016). URL https://arxiv.org/abs/1610.00633

[17] Luo, S., Kasaei, H., Schomaker, L.: Accelerating Reinforcement Learning for Reaching using Continuous Curriculum Learning. International Joint Conference on Neural Networks (IJCNN) pp. 1–8 (2020). URL https://ieeexplore.ieee.org/document/9207427

[18] Pham, T.H., De Magistris, G., Tachibana, R.: OptLayer - Practical Constrained Optimization for Deep Reinforcement Learning in the Real World. Proceedings - IEEE International Conference on Robotics and Automation pp. 6236–6243 (2018). doi:10.1109/ICRA.2018.8460547

[19] Lucchi, M., Zindler, F., Mühlbacher-Karrer, S., Pichler, H.: robo-gym – An Open Source Toolkit for Distributed Deep Reinforcement Learning on Real and Simulated Robots. In: 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2020) (2020). URL http://arxiv.org/abs/2007.02753

[20] Pong, V., Gu, S., Dalal, M., Levine, S.: Temporal difference models: Model-free deep RL for model-based control. 6th International Conference on Learning Representations (ICLR) pp. 1–14 (2018). URL https://arxiv.org/abs/1802.09081

[21] Pinto, L., Mandalika, A., Hou, B., Srinivasa, S.: Sample-Efficient Learning of Nonprehensile Manipulation Policies via Physics-Based Informed State Distributions. Computing Research Repository CoRR (2018). URL http://arxiv.org/abs/1810.10654

[22] Trossen Robotics (website accessed on 02/07/20)

[23] Quigley, M., Gerkey, B., Conley, K., Faust, J., Foote, T., Leibs, J., Berger, E., Wheeler, R., Ng, A.: ROS: an Open-Source Robot Operating System. Proceedings Open-Source Software workshop of the International Conference on Robotics and Automation (ICRA) pp. 4754–4759 (2009). URL https://pybullet.org/

[24] Yang, B., Zhang, J., Pong, V., Levine, S., Jayaraman, D.: REPLAB: A Reproducible Low-Cost Arm Benchmark Platform for Robotic Learning. International Conference on Robotics and Automation (ICRA) (2019). URL http://arxiv.org/abs/1905.07447

[25] Coumans, E., Bai, Y.: PyBullet, a Python Module for Physics Simulation for Games, Robotics and Machine Learning

[26] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., Zaremba, W.: OpenAI Gym (2016). URL http://gym.openai.com/

[27] Mnih, V., Badia, A.P., Mirza, L., Graves, A., Harley, T., Lillicrap, T.P., Silver, D., Kavukcuoglu, K.: Asynchronous Methods for Deep Reinforcement Learning. In: 33rd International Conference on Machine Learning (ICML), vol. 48, pp. 1928–1937 (2016). URL https://arxiv.org/abs/1602.01783

[28] Wu, Y., Mansimov, E., Liao, S., Grosse, R., Ba, J.: Scalable Trust-Region Method for Deep Reinforcement Learning Using Kronecker-Factored Approximation. 31st Conference on Neural Information Processing Systems (NIPS) pp. 5280–5289 (2017). URL https://arxiv.org/abs/1708.05144

[29] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal Policy Optimization Algorithms. Computing Research Repository (CoRR) pp. 1–12 (2017). URL http://arxiv.org/abs/1707.06347

[30] Schulman, J., Levine, S., Moritz, P., Jordan, M.I., Abbeel, P.: Trust Region Policy Optimization. 31st International Conference on Machine Learning (ICML 2015) (2015). URL http://arxiv.org/abs/1502.05477

[31] Hill, A., Raffin, A., Ernestus, M., Gleave, A., Kanervisto, A., Traore, R., Dhariwal, P., Hesse, C., Klimov, O., Nichol, A., Plappert, M., Radford, A., Schulman, J., Sidor, S., Wu, Y.: Stable Baselines. GitHub repository (2018). URL https://github.com/hill-a/stable-baselines

[32] Akiba, T., Sano, S., Yanase, T., Ohta, T., Koyama, M.: Optuna: A Next-generation Hyperparameter Optimization Framework. Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining pp. 2623–2631 (2019). doi:10.1145/3292500.3330701. URL https://dl.acm.org/doi/10.1145/3292500.3330701

[33] University College Dublin, U.R.O., Services, I.: ResearchIT Sonic HPC cluster. URL https://www.ucd.ie/itservices/ourservices/researchit/sonichpc/

[34] Pathak, D., Agrawal, P., Efros, A.A., Darrell, T.: Curiosity-Driven Exploration by Self-Supervised Prediction. IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops pp. 488–489 (2017). doi:10.1109/CVPRW.2017.70. URL https://ieeexplore.ieee.org/document/8014804