panda-gym : Open-source goal-conditioned environments for robotic learning

Quentin Gallouédec, Nicolas Cazin, Emmanuel Dellandréa, Liming Chen
École Centrale de Lyon
LIRIS, CNRS UMR 5205, France
{first.last}@ec-lyon.fr

Abstract

This paper presents panda-gym, a set of Reinforcement Learning (RL) environments for the Franka Emika Panda robot integrated with OpenAI Gym. Five tasks are included: reach, push, slide, pick & place and stack. They all follow a Multi-Goal RL framework, allowing to use goal-oriented RL algorithms. To foster open-research, we chose to use the open-source physics engine PyBullet. The implementation chosen for this package allows to define very easily new tasks or new robots. This paper also presents a baseline of results obtained with state-of-the-art model-free off-policy algorithms. panda-gym is open-source and freely available at https://github.com/qgallouedec/panda-gym.

1 Introduction

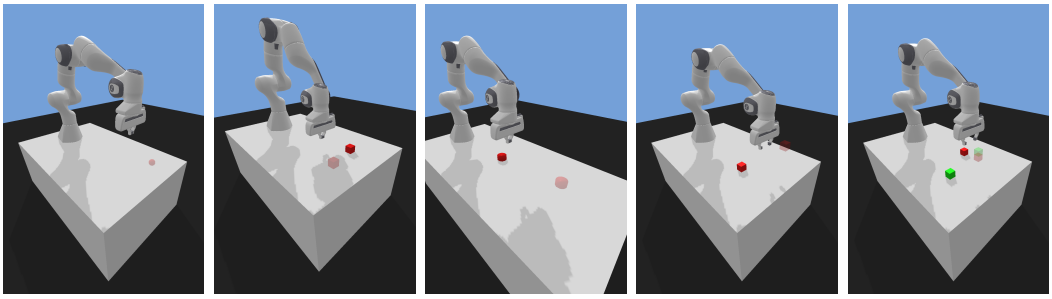
Recent advances in reinforcement learning applied to robotics have enabled to learn complex manipulation tasks. Nevertheless, current algorithms still struggle to solve tasks for which rewards are very sparse. Recent algorithms have contributed to the advancement in this area, but the number of interactions required to learn a satisfactory model is still very high. For the moment, learning complex tasks with sparse reward functions requires learning in simulation. A large number of physics simulator exists for various applications [Collins et al., 2021] . For robotic manipulation, the Panda robot arm of Franka Emika is widely used. For this reason, we propose a simulated environment of this robot arm for common tasks used to evaluate RL algorithm. Contrary to what the name of the package suggests, it is possible to easily define new robots, which can be used directly with the tasks already available.

2 Environments

The environments presented consist of a Panda robotic arm from Franka Emika[1], already widely used in simulation as well as in real in many academic works. It has 7 degrees of freedom and a parallel finger gripper. The robot is simulated with the PyBullet physics engine [Coumans and Bai, 2016], which has the advantage of being open-source and shows very good simulation performance. The environments are integrated with OpenAI Gym [Brockman et al., 2016], allowing the use of all learning algorithms based on this API. All the tasks presented by Plappert et al. [2018] and Andrychowicz et al. [2018] have their equivalent in this package. We have also added a stacking task, which is harder to solve than the other tasks, since two objects must be moved (instead of one for the pick & place task). The proposed environments all follow the Multi-Goal RL framework [Plappert et al., 2018]. At each episode, a new goal is randomly generated. The type of this goal depends on the task. For example, for the *reach* task, it is the position to be reached with the gripper which is

[1] https://www.franka.de/

NeurIPS 2021 Workshop on Robot Learning: Self-Supervised and Lifelong Learning, Virtual, Virtual



(a) Reach (b) Push (c) Slide (d) Pick And Place (e) Stack

Figure 1: Panda environments. The target positions are shaded (red and green).

randomly generated. The observation is therefore augmented with two additional vectors: the *desired goal*, and the *achieved goal*.

2.1 Tasks

A task consists in moving either the gripper or one (or more) object(s) to a target position. A task is completed when the distance between the entity to move and the target position is less than 5 cm. The tasks have an increasing level of difficulty. For each task, a rendering is presented in the Figure 1.

`PandaReach-v1` A target position must be reached with the gripper. This target position is randomly generated in a volume of 30 cm $\times$ 30 cm $\times$ 30 cm.

`PandaPush-v1` A cube, placed on a table, must be pushed to a target position also on the table surface. The gripper is blocked closed. The target position and the initial position of the cube are randomly generated in a 30 cm $\times$ 30 cm square around the neutral position of the robot.

`PandaSlide-v1` A flat cylinder (like an ice hockey puck) must be moved to a target position on the surface of a table. The gripper is blocked closed. The target position is randomly generated in a 50 cm $\times$ 50 cm square located out of reach of the robot, in front of the neutral position. Thus, is necessary to give an impulse to the object, instead of just pushing it.

`PandaPickAndPlace-v1` A cube must be brought to a target position generated in a volume of 30 cm $\times$ 30 cm $\times$ 20 cm above the table.

`PandaStack-v1` Two cubes must be stacked at a target position on the table surface. The target position is generated in a square of 30 cm $\times$ 30 cm. The stacking must be done in the correct order: the red cube must be under the green cube.

2.2 Observation and action space

The observation space varies depending on the task. For all tasks, the observation contains the position and speed of the gripper (6 coordinates). The control of the gripper does not allow to change its orientation. Its state is therefore completely determined by these 6 coordinates. If the task involves one or more objects, the observation space contains the position, the orientation, the linear and rotational speed (12 coordinates) for each object. When the gripper is not constrained to be closed, the opening of the gripper (i.e. the distance between the fingers) is part of the observation space (1 coordinate).

The action space is composed of the gripper movement command (3 coordinates, one for each axis of movement x, y and z) and the fingers movement (1 coordinate, corresponding to the variation of the gripper opening). For some tasks, the gripper is blocked closed. For these tasks, the action space is only composed of the gripper motion command.

At each action of the agent, the simulator runs 20 timesteps, before giving the control back to the agent, and waiting for the next action. On the other hand, one simulator timestep represents 2 ms.

The interaction frequency is thus 25 Hz. An episode is made of 50 interactions, so the duration of an episode is 2 seconds (for the stacking task, an episode lasts 100 interactions, so 4 seconds). These durations are empirically sufficient for the realization of the corresponding tasks.

2.3 Reward

By default, the reward is *sparse*: a reward of 0 is obtained if the entity to move is at the desired position (with a tolerance of 5 cm), and -1 otherwise. For each environment, a variant exists in which the reward is *dense*: this reward is the opposite of the distance between the entity to move and the desired position[2].

In general, a sparse reward function is easier to define, since it is only a question of assessing whether the task is completed in the current state. Conversely, defining a dense reward function can be a tricky process, especially when the task implies several completion criteria. For example, for a task consisting in moving and rotating a cube (task considered for a future version of the package, see Section 5), defining a dense reward function requires to assign a weight of preference to each criterion [Gábor et al., 1998, Natarajan and Tadepalli, 2005]. These preferences constitute additional hyperparameters.

3 Design Decisions

A robotic environment consists of a robotic arm and a task. Conceptually, a robotic arm can perform different tasks. Similarly, a task can be performed by different robots. To allow for this flexibility, we have separated the task class from the robot class. This allows to easily define a new task without worrying about the robot that will execute it. In the same way, it is possible to define a new robot without worrying about the task to be executed. Figure 2 shows the chosen implementation.

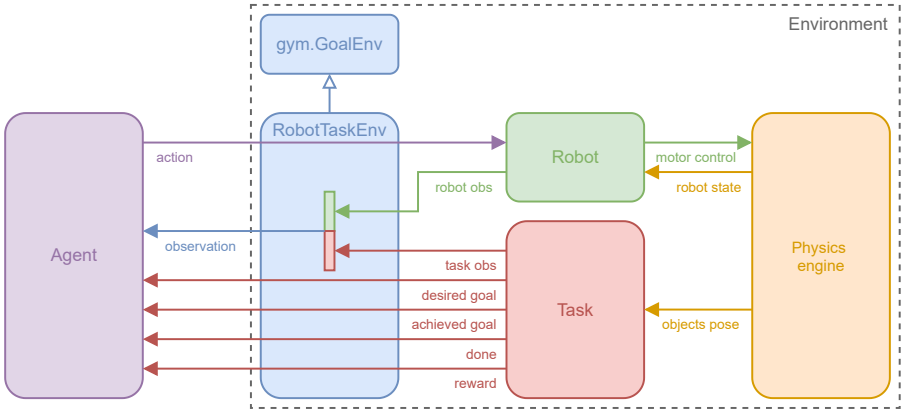


Figure 2: Code design. The task and the robot are separate, which allows them to be modular. The agent's actions are sent to the robot.

The main class, called `RobotTaskEnv` contains a `robot` attribute, and a `task` attribute. When the agent takes an action, send an action to the environment, it transfers it to the robot. The collected observation is the concatenation of the observations specific to the robot (the pose of the gripper, for example) and the observations specific to the task (the pose of the objects, for example). Finally, to follow the Multi-Goal framework, the desired goal and the achieved goal are derived from the task attribute.

The proposed environments allow fast learning, even on a computer with limited computing capacity. The PyBullet physics engine allows the parallel simulation of several scenes. Thus, the environments are compatible with learning methods that use multiple CPU cores. Tests show that the environments are on average 9.2% faster than their equivalents developed on MuJoCo[3].

[2]For the stacking task, the reward is $-\sqrt{{d_1}^2 + {d_2}^2}$, where d_i is the distance between the object i and its desired position.

[3]We measured time required to simulate 10^5 timesteps using a single CPU core.

4 Experimental results

The length of a trajectory is 50 timesteps, except for the stacking task, for which we chose a length of 100 timesteps, due to its higher complexity. At the end of each trajectory, the environment is reset and a new goal is randomly generated. The learning has been distributed on 8 CPU cores. Each core generates trajectories and all these trajectories are stored in a common replay buffer. The results are the success rate evaluated over 80 test episodes, regularly over the course of learning. We give a baseline of the results we obtain for three off-policy algorithms from the recent literature used with Hindsight Experience Replay (HER) [Andrychowicz et al., 2018]: Deep Deterministic Policy Gradient (DDPG) [Lillicrap et al., 2015], Soft Actor-Critic (SAC) [Haarnoja et al., 2018] and Twin Delayed DDPG (TD3) [Fujimoto et al., 2018]. The implementation of DDPG used for the training is the one proposed by Dhariwal et al. [2017]. The appropriate modifications have been made to DDPG to implement TD3 and SAC[4]. The hyperparameters are available in Appendix A. The learning curves are shown in Figure 3. Note that the horizontal axis corresponds to the total number of interaction with the environment. The learning curves are therefore independent of the number of workers used to collect these interactions.

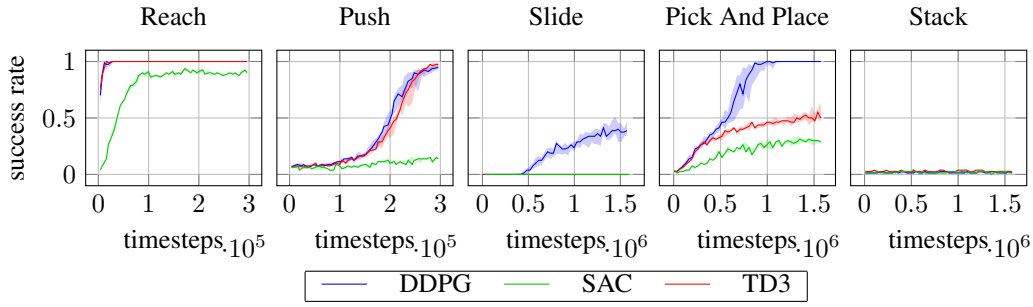


Figure 3: Success rates for the five Panda environments. We repeat each experiment with 21 different random seeds. Median rates are solid lines and interquartile range are shaded areas. We represent the results for the DDPG, SAC and TD3 algorithms, all three ran with HER. The horizontal axis corresponds to the total number of interaction with the environment.

The number of timesteps needed to resolve a task depend on the task and the algorithm. For DDPG, the success rate reaches 100% for the reach and push tasks, after 10^4 and 3×10^4 timesteps, respectively. It reaches about 50% for the slide and pick and place tasks, after 6×10^5 and 1.6×10^6 timesteps, respectively. The success rate for the stacking task remains close to 0 after 1.6×10^6 timesteps of training. The presented algorithms do not allow to solve it in this amount of timesteps.

We notice that for TD3 and SAC, the ablation of the clipped double-Q trick allows a significant increase of the results in multiple environments. The set of curves representing the results of the distinct ablations is given in Appendix B. Appendix D shows an overview of the policies at the end of the training for the four task that are solved or partially solved.

5 Conclusion and future works

In this paper, we have described `panda-gym`, a free and open-source package which allows to define robotic tasks, 5 of which are present in the current version. They allow to evaluate the reinforcement learning algorithms in the context of complex robotic tasks. The architectural choices allow to define very easily new tasks and new robots. Finally, the state of the art algorithms allow to solve some tasks, while others remain unsolved.

We are planning to add to the presented tasks some new and very used tasks such as the peg-in-hole insertion [Lee et al., 2019] or the cube flipping [OpenAI, 2020]. On the other hand, we also plan to give the possibility to control the robot directly with joint values. This would require the agent to learn by itself the inverse robot dynamics. Finally, to better fit with reality, we plan to add the possibility to use an observation space that would include several modalities, such as a RGB camera, a depth camera, a force sensor or a tactile sensor.

[4] `https://github.com/qgallouedec/baselines`

References

J. Collins, S. Chand, A. Vanderkop, and D. Howard. A review of physics simulators for robotic applications. *IEEE Access*, 9:51416–51431, 2021. doi: 10.1109/ACCESS.2021.3068769.

Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. 2016.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

Matthias Plappert, Marcin Andrychowicz, Alex Ray, Bob McGrew, Bowen Baker, Glenn Powell, Jonas Schneider, Josh Tobin, Maciek Chociej, Peter Welinder, et al. Multi-goal reinforcement learning: Challenging robotics environments and request for research. *arXiv preprint arXiv:1802.09464*, 2018.

Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay, 2018.

Zoltán Gábor, Zsolt Kalmár, and Csaba Szepesvári. Multi-criteria reinforcement learning. In *ICML*, volume 98, pages 197–205. Citeseer, 1998.

Sriraam Natarajan and Prasad Tadepalli. Dynamic preferences in multi-criteria reinforcement learning. In *Proceedings of the 22nd International Conference on Machine learning*, pages 601–608, 2005.

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, pages 1861–1870. PMLR, 2018.

Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, pages 1587–1596. PMLR, 2018.

Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, Yuhuai Wu, and Peter Zhokhov. Openai baselines. `https://github.com/openai/baselines`, 2017.

Michelle A Lee, Yuke Zhu, Krishnan Srinivasan, Parth Shah, Silvio Savarese, Li Fei-Fei, Animesh Garg, and Jeannette Bohg. Making sense of vision and touch: Self-supervised learning of multimodal representations for contact-rich tasks. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8943–8950. IEEE, 2019.

OpenAI. Robogym. `https://github.com/openai/robogym`, 2020.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Boris T Polyak and Anatoli B Juditsky. Acceleration of stochastic approximation by averaging. *SIAM journal on control and optimization*, 30(4):838–855, 1992.

A Hyperparameters

All experiments in this paper use the hyperparameters presented in Table 1. They have been chosen identical to those submitted by Plappert et al. [2018].

Actor and Critic	Network type	Multi-layer perceptron
	Network size	3 layers of 256 nodes
	Optimizer	Adam [Kingma and Ba, 2014]
	Learning rate	0.001
	Polyak-averaging [1992]	0.95
	L2 normalisation coefficient	1.0
Observation	Clipping	$[-200, 200]$
Action	Clipping	$[-1, 1]$
	Probability of random action	0.3 (DDPG and TD3 only)
	Scale of additive gaussian noise	0.2 (DDPG and TD3 only)
	Number of HER per transition (k)	4
Training	Episode length	50 (100 for `PandaStack-v1`)
	Testing	Every 80 episodes
	Number of testing episodes	80
	Replay buffer size	10^6 transitions
	Batch size	256
	Policy delay	2 (TD3 only)
	Policy noise	0.2 (TD3 only)
	Policy noise clip	$[-0.5, 0.5]$ (TD3 only)
	α	0.2 (SAC only)

Table 1: Hyperparameters used for the experiments

B Full results and ablations study

Ablating the clipped double-Q trick In the TD3 algorithm, a transition (s, a, r, s', d) is sampled from the replay buffer, where s is a state, a the action, r the reward, s' the next state, and d a boolean value indicating if s' is terminal. The target value y is computed as follows:

$$y(r, s', d) = r + \gamma(1 - d) \min_{i=1,2} Q_{\phi_{\text{targ},i}}(s', a'(s')) \tag{1}$$

where γ is the discount factor, $Q_{\phi_{\text{targ},1}}$ and $Q_{\phi_{\text{targ},2}}$ are the target Q-networks, and $a'(s')$ is the target action, resulting from the target policy smoothing. The ablation of the clipped double-Q trick in TD3 consists in replacing the target value by

$$y(r, s', d) = r + \gamma(1 - d) Q_{\phi_{\text{targ},1}}(s', a'(s')) \tag{2}$$

In the SAC algorithm, the target value y is computed as follow:

$$y(r, s', d) = r + \gamma(1 - d) \left(\min_{i=1,2} Q_{\phi_{\text{targ},i}}(s', \tilde{a}') - \alpha \log \pi_\theta(\tilde{a}' \mid s') \right), \quad \tilde{a}' \sim \pi_\theta(\cdot \mid s') \tag{3}$$

where α is the entropy regularization coefficient and π_θ the stochastic policy of the actor. The ablation of the clipped double-Q trick in SAC consists in replacing the target value by

$$y(r, s', d) = r + \gamma(1 - d) \left(Q_{\phi_{\text{targ},1}}(s', \tilde{a}') - \alpha \log \pi_\theta(\tilde{a}' \mid s') \right), \quad \tilde{a}' \sim \pi_\theta(\cdot \mid s') \tag{4}$$

Once the clipped double-Q trick is ablated, the second Q-network (denoted $Q_{\phi_{\text{targ},2}}$) no longer exists.

Ablating HER After each episode, the agent stores in the replay buffer each transition with the initial goal. For each stored transition, it stores k identical transitions, with a different goal, corresponding to a goal achieved later, during the episode. The ablation of HER consists in removing this part of the algorithm.

The results are presented in Figure 4.

Although the clipped double-Q trick allows in some environments to increase the learning performance (`Hopper-v2`, `Walker2d-v2`, `HalfCheetah-v2`, `Humanoid-v2`, [Haarnoja et al., 2018]), by limiting the value overestimation, its ablation leads here either to no effect, or to an increase of the results. It is possible that the scattering of the reward prevents overestimation of value. This trick would thus be counterproductive, by preventing the value from spreading properly. This intuition should be verified in future work.

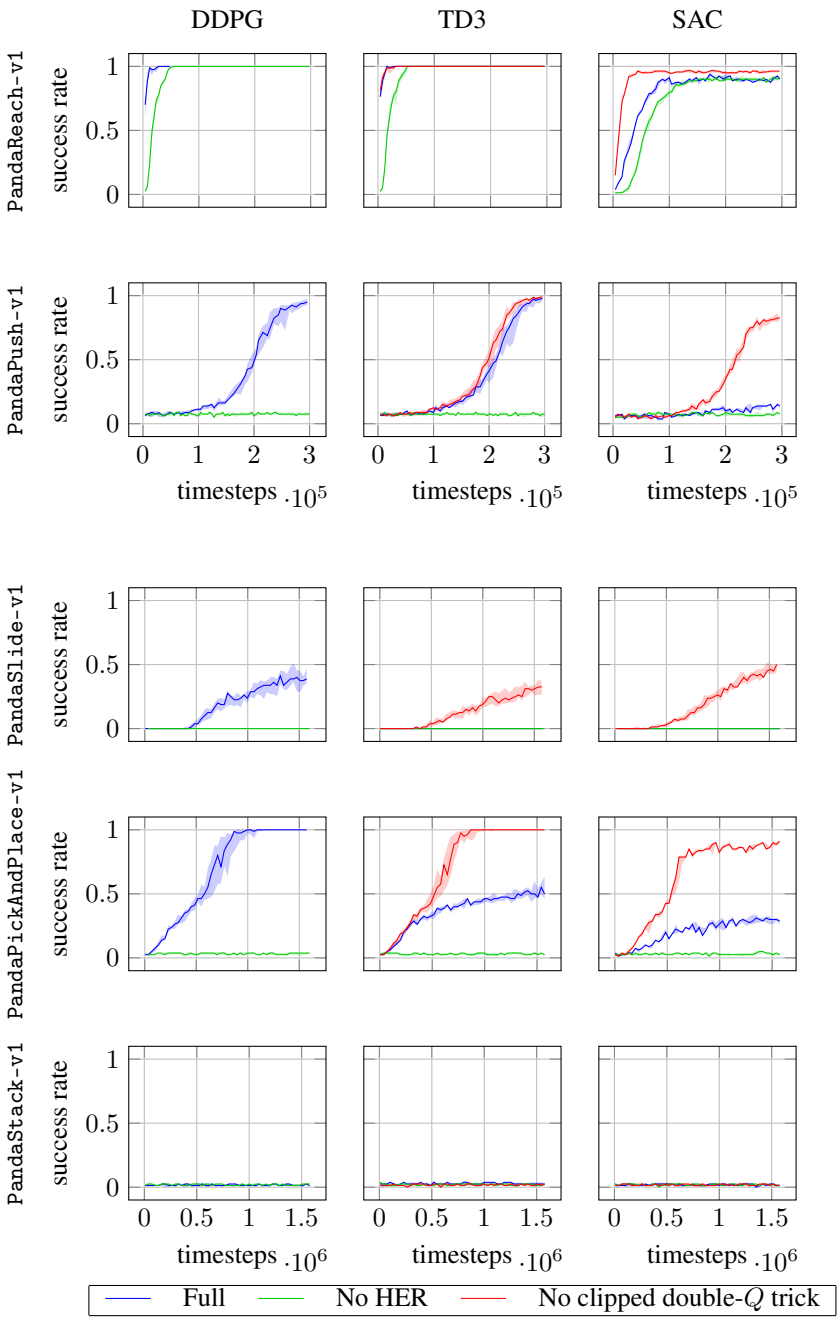


Figure 4: Ablation study for HER and for the clipped double-Q trick. We repeat each experiment with 21 different random seeds. Median success rates are solid lines and interquartile range are shaded areas.

C Environments specifications

Task	Observation				Action		
	Gripper pose	Object(s) pose	Gripper opening	Size	Gripper displacement	Gripper opening	Size
Reach	✓	✗	✗	6	✓	✗	3
Push	✓	✓	✗	18	✓	✗	3
Slide	✓	✓	✗	18	✓	✗	3
PickAndPlace	✓	✓	✓	19	✓	✓	4
Stack	✓	✓	✓	31	✓	✓	4

Table 2: Components of the observation and the action space.

D Overview of the learned policies

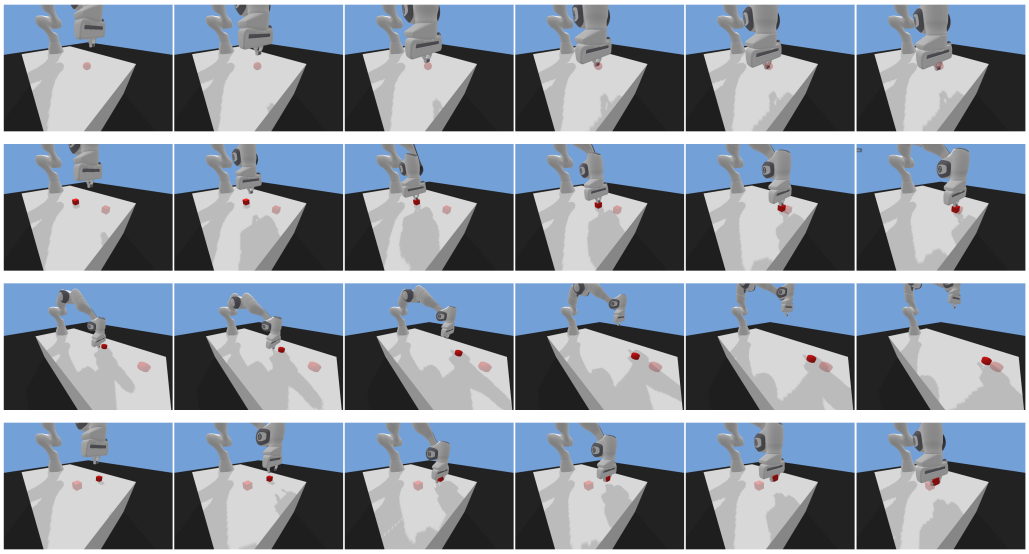


Figure 5: Overview of policies at the end of the training. Each line represents a task. From top to bottom: reach (one timestep between two successive images), push (two timesteps between two successive images), slide (four timesteps between two successive images) and pick & place (two timesteps between two successive images).