Karlsruhe Institute of Technology

Autonomous Learning Robots Lab

Benchmarking Reinforcement Learning Algorithms on Realistic Simulated Environments

Bachelor's Thesis
of

Jan Küblbeck

KIT Department of Informatics
Institute for Anthropomatics and Robotics (IAR)
Autonomous Learning Robots (ALR)

Referees:	Prof. Dr. Techn. Gerhard Neumann
	Prof. Dr.-Ing. Tamim Asfour
Advisors:	Onur Celik
	Ge Li

Duration: October 25th, 2022 — February 27th, 2023

Erklärung

Ich versichere hiermit, dass ich die Arbeit selbstständig verfasst habe, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, die wörtlich oder inhaltlich übernommenen Stellen als solche kenntlich gemacht habe und die Satzung des Karlsruher Instituts für Technologie zur Sicherung guter wissenschaftlicher Praxis beachtet habe.

Karlsruhe, den 27. Februar 2023

[illegible]

Jan Küblbeck

Zusammenfassung

Reinforcement learning (verstärkendes Lernen, RL) ist eine Methodik des machinellen Lernens, in der ein Agent lernt, in einer bestimmten Umgebung die jeweils optimalen Aktionen auszuführen um langfristig die größtmögliche Belohnung zu erhalten. Für die Entwicklung von neuen RL-Algorithmen und -Technologien sind Benchmarks äußerst wichtig, damit die Qualität verschiedener Methoden bewertet und verglichen werden kann. Für die Robotik ist es insbesondere wichtig, dass Benchmarks so akkurat and realistisch wie möglich echte Roboter wiederspiegeln, um Algorithmen für diese zu testen und Modelle zu trainieren.

In dieser Bachelorarbeit verwende ich eine realistische Simulation eines Franka Emika Panda-Roboters, das ALR Simulation Framework, um drei Aufgaben mit unterschiedlichen Schwierigkeitsgraden für diesen Roboterarm zu implementieren: Der Roboter soll lernen sich an eine zufällig generierte Position bewegen, eine Tür zu öffnen oder einen Fußball in ein Tor zu stoßen. Eine praktische Anwendung dieser drei Umgebungen zeige ich, indem ich vier bestehende Algorithmen (PPO, DDPG, TD3, SAC) experimentell teste und die Ergebnisse auswerte. Dabei stelle ich fest, dass die Fähigkeit der verschiedenen Algorithmen, die drei Aufgaben zu lösen, sich teilweise deutlich underscheidet. Unter anderem zeigt die Auswertung, dass SAC meist leistungsstärker ist als DDPG und TD3, und außerdem, dass PPO häufig langsamer lernt als die anderen Algorithmen.

Abstract

Reinforcement learning (RL) is a method of machine learning wherein an agent is trained to take the optimal actions in an environment to achieve the greatest possible long-term returns. It is a very active research field with many new technologies being developed at all times. Benchmarks are a necessity to test, evaluate and compare the performance of new RL algorithms and methods. For robotics applications, it is especially important that benchmarks accurately and realistically simulate real robots.

In this bachelor's thesis I use the ALR Simulation Framework — a realistic simulation of a Franka Emika Panda robot arm — to implement three tasks of varying difficulty for the robot to solve: Reaching a specific random position, opening a door, and kicking a soccer ball into a goal. To demonstrate the three environments, I use them to benchmark four well-known RL algorithms (PPO, DDPG, TD3, SAC) and evaluate the results of each. I find that the performance of the algorithms can vary substantially between environments, as some are better suited for the challenges than others. The results show that SAC performs better than DDPG and TD3 in most cases, and that PPO usually learns more slowly than the others but can outperform them in some situations.

Table of Contents

Chapter 1

Introduction

1.1 Motivation

During recent years there have been many new developments in artificial intelligence and machine learning in general, including especially in the field of reinforcement learning (RL) and deep reinforcement learning. New techniques and algorithms like PPO (Schulman et al., 2017) and SAC (Haarnoja et al., 2018a) have demonstrated ever increasing performance. But while they overcame some weaknesses of older RL approaches, they also introduce new challenges and problems to be solved.

One of the challenges when developing and evaluating a novel RL algorithm is the need for reliable benchmarking. Researchers rely on tests and benchmarks to assess how the performance of their algorithm compares to other works. Without good benchmarks, it would be difficult to determine the strengths and weaknesses of different technologies and to make decisions on which algorithm to choose for a particular use case. Algorithms should be tested on as many tasks as possible to measure their performance in different situations. These tasks should ideally be accurate representations of problems that the algorithm may be used to solve in the real world. Benchmarks must also be consistent and reliable, so that researchers can report reproducible results in line with scientific best practices.

While there are several well-established sets of benchmark environments (Bellemare et al., 2013; Duan et al., 2016), these often work with a high level of abstraction. They are therefore simple to use but limited in how accurately they portray real-world problems, meaning that an algorithm's performance in a benchmark may not always translate fully to real applications. A realistic simulation is particularly important in the context of robotics because extensive training and testing of an algorithm on a real robot is associated with significant effort, time and financial costs (Kober et al., 2013). Using a simulated environment substantially improves the efficiency of working on software for robots. Concepts like sim-to-real transfer require a robot to be simulated with a very high level of accuracy. The design specifics of a particular robot system can substantially impact the ability of a RL algorithm to learn a task, and must be reflected in the simulation.

For these reasons, the Autonomous Learning Robots (ALR) lab has developed a realistic Simulation Framework modeling a Franka Emika Panda robot arm, representing a real robot counterpart in the lab. This thesis aims to implement a set of benchmark environments that represent different real-world robot tasks of varying complexity using the ALR Simulation Framework: Reaching for a random goal, opening a door, and kicking a soccer ball into a goal. To demonstrate their validity and usefulness, the environments are then used to benchmark four well-known RL algorithms: Deep Deterministic Policy Gradient (DDPG), Twin Double DDPG (TD3), Soft Actor Critic (SAC), and Proximal Policy Optimization (PPO).

1.2 Structure

In Chapter 2 I will explain the fundamentals of reinforcement learning, introduce the algorithms used for benchmarking in this work, as well as some general best practices for benchmark design and usage. Chapter 3 will show how some other authors have approached similar challenges as this thesis. I will share details about the Simulation Framework and its application in the development and implementation of the three new task environments in Chapter 4. Chapter 5 will explain the setup of the experiments done and present the results, which will be evaluated and discussed in Chapter 6. Finally, in Chapter 7 I will sum up the the thesis and propose how future works may improve upon it.

Chapter 2

Fundamentals

2.1 Reinforcement Learning

The typical framework of reinforcement learning (RL) consists of an agent and an environment. The environment is modeled as a Markov decision process, which consists of a state space $\mathbb{S}$, an action space $\mathbb{A}$, a set of transition probabilities $P(s'|s,a)$, and a reward function $r(s,a)$. In each step t, the agent is aware of the current state s_t and uses policy π (which may be deterministic or stochastic) to choose an action a_t. The transition probability $P(s_{t+1}|s_t,a_t)$ determines the the next state, which is observed by the agent along with the reward $r_t = r(s_t,a_t)$ (see Figure 2.1). Through successive steps, the series of observed states and performed actions is called a trajectory $\tau = (s_0,a_0,s_1,a_1,...)$. While each step is associated with a single reward r_t, the long-term performance of the agent is measured by the return

$$R = \sum_{t=0}^{\infty} \gamma^t r_t,$$

where the hyperparameter $0 \leq \gamma \leq 1$ is called the discount factor and represents the reduced benefit of distant future rewards compared to immediate rewards (Sutton and Barto, 2018). The return can also be calculated for a given trajectory of T steps as

$$R(\tau) = \sum_{t=0}^{T} \gamma^t r(s_t,a_t).$$

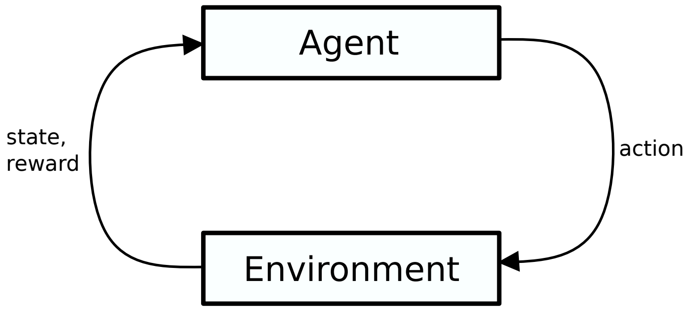


Figure 2.1: The agent chooses and performs an action. The environment returns the next state alongside a step reward.

The goal of RL algorithms is to determine the optimal policy π^* for a given environment, which is frequently done using a value function. The state-value function

$$V^{\pi}(s) = \mathbb{E}_{\pi}[R|s_0 = s]$$

describes the expected return of policy π when starting in state s. Likewise, the action-value function

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}[R|s_0 = s, a_0 = a]$$

is the expected return of π when starting in state s with action a. Additionally, the advantage

$$A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s)$$

can be used to evaluate the value of action a compared to the overall expected return in state s (François-Lavet et al., 2018). A policy is considered optimal if it maximizes the expected return

$$J(\pi) = \mathbb{E}[R]$$

and therefore also $V^{\pi}(s)$ for all $s \in \mathbb{S}$ and $Q^{\pi}(s,a)$ for all $s \in \mathbb{S}$ and $a \in \mathbb{A}(s)$. RL algorithms commonly work by repeatedly updating their estimates of the value function during training, eventually creating a policy that can approximate the optimal action in each state (Sutton and Barto, 2018).

Reinforcement learning algorithms can be categorized as either on-policy or off-policy. An off-policy algorithm makes its estimates without considering the policy it is following for its predictions. For example, Q-learning (Watkins and Dayan, 1992) uses the update function

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha[r_{t+1} + \gamma \max_{a} Q(s_{t+1},a) - Q(s_t,a_t)]$$

which uses the maximum Q-value out of all possible following actions to update its estimate for the current state-action pair. (α is called the learning rate and determines how significantly the value function is changed in each step.) On the other hand, SARSA (Rummery and Niranjan, 1994) uses the update function

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

which only considers the next action a_{t+1} provided by the current policy in each step. As such, SARSA is classified as an on-policy algorithm because updates to the policy are directly influenced by the output of the current policy (Sutton and Barto, 2018).

An alternative approach to the use of value function is to directly optimize the policy, for which a parametric policy π_θ is used with θ being the policy's parameter vector (Sutton et al., 1999; Silver et al., 2014). This strategy makes it possible to directly update the policy via its parameters using the gradient ascent method

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\pi_\theta),$$

for which the gradient can be estimated by sampling N trajectories and calculating

$$\nabla_\theta J(\pi_\theta) \approx \frac{1}{N} \sum_i (\sum_t \nabla_\theta \log \pi_\theta (a_{i,t}|s_{i,t})) R(\tau_i),$$

as is done by the REINFORCE algorithm (Williams, 1992). By disregarding the past return and only focusing on future rewards, this can be further modified to

$$\nabla_\theta J(\pi_\theta) \approx \frac{1}{N} \sum_i \sum_t \nabla_\theta \log \pi_\theta (a_{i,t}|s_{i,t}) (\sum_{k=t}^{T} \gamma^{k-t} r(s_{i,k}, a_{i,k})).$$

It is also possible to subtract a baseline b from the return and modify the estimate to

$$\nabla_\theta J(\pi_\theta) \approx \frac{1}{N} \sum_i (\sum_t \nabla_\theta \log \pi_\theta (a_{i,t}|s_{i,t}))(R(\tau_i) - b),$$

which has the effect of reducing variance without damaging the convergence of the gradient ascent. If one now uses an estimated Q-function as an approximation of the future reward and the current state value as the baseline, the calculation can be restated in relation to the advantage function

$$\nabla_\theta J(\pi_\theta) \approx \frac{1}{N} \sum_i \sum_t \nabla_\theta \log \pi_\theta (a_{i,t}|s_{i,t}) A^{\pi_{\text{old}}}(s_{i,t}, a_{i,t}).$$

Methods that separately learn both the policy and the value function are referred to as actor-critic algorithms (Konda and Tsitsiklis, 1999). Usually the actor will use a policy gradient approach to directly learn the policy, while the critic updates a V- or Q-function to evaluate the actor's performance and inform future policy updates (Arulkumaran et al., 2017; François-Lavet et al., 2018).

Deep reinforcement learning combines traditional RL with deep learning techniques. Deep neural networks are able to work with much greater quantities of data such as larger state and action spaces (François-Lavet et al., 2018). The weights defining a parametrized policy or value function can also be represented as neural networks, allowing them to be efficiently updated (Mnih et al., 2013). These benefits have led to significant advancements in RL research as novel deep RL algorithms, which are designed to exploit the benefits of deep learning, continue to be developed and published. Off-policy deep RL algorithms typically make use of an experience replay buffer which stores previously encountered transitions to be sampled at a later date, which greatly improves sample efficiency (Mnih et al., 2013; Arulkumaran et al., 2017).

2.2 Select Algorithms

2.2.1 Proximal Policy Optimization

Proximal Policy Optimization (PPO) introduced by Schulman et al. (2017) is an on-policy algorithm using the policy gradient method. It is related to the earlier Trust Region Policy Optimization (TRPO) (Schulman et al., 2015a). PPO uses multiple parallel actors in the exploration stage, making sample collection more efficient.

On-policy methods cannot use experience replay, but always having to sample new trajectories for every single policy update would be extremely inefficient. To work around this problem, TRPO and PPO make use of the importance weight ratio

$$\rho_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\text{old}}(a_t|s_t)},$$

which quantifies the difference in the policy before and after it is updated. Effectively, by keeping track of this ratio, it is possible to make multiple policy updates based on the old policy's trajectories. By combining this ratio with advantage-based policy optimization, it is possible to derive the maximization objective

$$L = \mathbb{E}_t[\rho_t(\theta) A^{\pi_{\text{old}}}(s_t, a_t)],$$

$A^{\pi_{\text{old}}}$ being an estimated advantage function based on the old policy. $A^{\pi_{\text{old}}}$ can be calculated by any method, often using generalized advantage estimation (GAE) (Schulman et al., 2015b). To ensure that only relatively small policy updates are made in each step, PPO additionally clips $\rho(\theta)$ to the interval $[1-\varepsilon, 1+\varepsilon]$ (with ε being a small hyperparameter). The result is the clipped objective

$$L^{\text{clip}}(\theta) = \mathbb{E}_t[\min(\rho_t(\theta) A_t^{\pi_{\text{old}}}, \text{clip}(\rho_t(\theta), 1-\varepsilon, 1+\varepsilon) A_t^{\pi_{\text{old}}}],$$

which can now be maximized using gradient ascent.

In addition, TRPO and PPO improve their performance by applying the trust region method. It is used to choose the optimal size of update steps in policy gradient ascent, which should be neither too large nor too small for best performance. Trust regions are defined using the Kullback–Leibler (KL) divergence and PPO will choose the greatest possible step that still satisfies $KL(\pi_\theta \| \pi_{\text{old}}) \leq \delta$ (with δ being a small hyperparameter).

2.2.2 Deep Deterministic Policy Gradient

Deep Deterministic Policy Gradient (DDPG) as introduced by Lillicrap et al. (2015) is an off-policy algorithm that is designed specifically for environments with continuous action spaces. As the name suggests, DDPG results in a deterministic policy. It combines features of deep Q-learning (Mnih et al., 2013) and policy gradient optimization in an actor-critic structure.

The actor learns the deterministic policy π_θ with parameters θ and the critic's learned Q-function is represented by the weights ϕ. Additionally, the algorithm maintains copies of both parameters (θ' and ϕ') which are updated more slowly and are used to calculate the target values of the Q-function updates. Updating the target parameters at a slower rate helps reduce the algorithm's volatility.

During training, a stochastic exploration policy is created by adding random noise to the deterministic actions of π_θ. Doing so is necessary to make sure that the algorithm explores sufficiently despite training a deterministic policy. All transitions encountered during exploration are added to the experience replay buffer B in the format of (s_t, a_t, r_t, s_{t+1}), which can then be used for off-policy updates. If a terminal state is encountered, the environment is reset to start a new episode.

To update its estimates, DDPG samples a batch of N transitions from the replay buffer B and calculates the target value

$$y_i = r_i + \gamma Q_{\phi'}(s_{i+1}, \pi_{\theta'}(s_{i+1}))$$

for each. The algorithm then updates its estimated Q-function parameters ϕ with gradient descent to minimize the loss function

$$L = \frac{1}{N} \sum_i (Q_\phi(s_i, a_i) - y_i)^2$$

and applies gradient ascent to the policy parameters θ using

$$\nabla_\theta J \approx \frac{1}{N} \sum_i \nabla_a Q_\phi(s_i, \pi_\theta(s_i)) \nabla_\theta \pi_\theta(s_i).$$

Finally, the target networks are updated to $\phi' \leftarrow \beta\phi' + (1-\beta)\phi$ and $\theta' \leftarrow \beta\theta' + (1-\beta)\theta$, with $0 < \beta < 1$. This updating method is called Polyak averaging and stabilizes the target networks.

2.2.3 Twin Delayed Deep Deterministic Policy Gradient

Twin Delayed DDPG (TD3) is a modification of DDPG introduced by Fujimoto et al. (2018). It differs from the original DDPG in three significant ways:

- It uses two Q-function critics at once instead of one, represented by the weights ϕ_1 and ϕ_2. Only the smaller Q-value is considered for updates. This variation is known as "clipped double Q-learning" (Fujimoto et al., 2018) and produces more pessimistic estimates for the Q-function, therefore avoiding overestimation bias which regular DDPG can suffer from.

- The policy and the target networks are updated less frequently than the Q-function, for example only on every second step. Such delayed updates are aimed at reducing the build-up of estimation errors between critic and actor. Any error in the value function calculation also distorts the policy gradient calculation, so by making multiple Q-function updates before every policy update the algorithm gets more a refined value estimate with less variance.

- A small amount of random noise ε is added to the actions in the target value calculation, which smooths the target policy and addresses problems with overfitting the Q-function.

In summary, exploration is the same as in DDPG. In the update step, after sampling the transitions from the replay buffer B, the random noise

$$\varepsilon \sim \text{clip}(\mathcal{N}(0,\sigma), -c, c)$$

is sampled (hyperparameter c is a small number limiting the magnitude of the noise, σ is the standard deviation). The target value is now calculated as

$$y_i = r_i + \gamma \min_{j=1,2} Q_{\phi'_j}(s_{i+1}, \pi_{\theta'}(s_{i+1}) + \varepsilon)$$

and used to update the respective Q-function parameters with the slightly modified loss functions

$$L_{\theta_j} = \frac{1}{N}\sum_i (Q_{\phi_j}(s_i,a_i) - y_i)^2.$$

Then, only in every d-th update step, the policy parameters θ are updated using the Q-function defined by ϕ_1, i.e. with the gradient

$$\nabla_\theta J \approx \frac{1}{N}\sum_i \nabla_a Q_{\phi_1}(s_i, \pi_\theta(s_i)) \nabla_\theta \pi_\theta(s_i).$$

TD3's target networks ϕ'_1, ϕ'_2 and θ' are also only updated after the policy updates and by using the same Polyak averaging method applied in regular DDPG.

2.2.4 Soft Actor-Critic

Haarnoja et al. (2018a) introduced Soft Actor-Critic (SAC), which is also an off-policy algorithm with some similarities to DDPG and TD3. However, because the deterministic algorithms have some drawbacks (e.g. suboptimal exploration, a tendency to get stuck in local optima, and sensitivity to hyperparameter tuning), SAC instead trains a stochastic policy that is optimized for maximum entropy. Entropy measures the policy's randomness, therefore high entropy ensures high exploration and accelerates learning.

In the exploration stage, due to using a stochastic policy, SAC does not require the action noise used by DDPG and can instead explore on-policy. Its exploration is otherwise similar and also adds all transitions to the experience replay buffer B.

For updating its actor and critic, SAC uses so-called "soft value functions" that incorporate the entropy

$$H(\pi(\cdot|s)) = \mathbb{E}_{a\sim\pi}[-\log\pi(a|s)].$$

Using this apprach, we can regularize the objective for learning an optimal policy

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{\pi}[\sum_{t=0}^{\infty} \gamma^t (r_t + \alpha H(\pi(\cdot|s_t))]$$

and also redefine the value functions as

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}[\sum_{t=0}^{\infty} \gamma^t (r_t + \sum_{t=1}^{\infty} \alpha H(\pi(\cdot|s_t))|s_0 = s, a_0 = a]$$

and

$$V^{\pi}(s) = \mathbb{E}_{a\sim\pi}[Q^{\pi}(s,a) + \alpha H(\pi(\cdot|s))].$$

Following these principles, SAC in practice learns two soft Q-functions $Q_{\phi_1}(s,a)$ and $Q_{\phi_2}(s,a)$ (using the same clipped double Q-learning method as TD3), and a policy $\pi_\theta(a|s)$ (Haarnoja et al., 2018b). Their respective parameters ϕ_i and θ are updated using gradient ascent/descent. SAC uses target copies of the Q-function parameters (ϕ'_j), but not of the policy parameters, which are once again updated following Polyak averaging. The Q-function target value calculation is now

$$y_i = r_i + \gamma(\min_{j=1,2} Q_{\phi'_j}(s_{i+1}, a') - \alpha \log\pi_\theta(a'|s_{i+1})), a' \sim \pi_\theta(\cdot|s_{t+1}),$$

with the Q-function update otherwise remaining the same as in TD3. The policy optimization objective is defined by

$$J(\pi_\theta) = \frac{1}{N}\sum_i \mathbb{E}_{a\sim\pi_\theta}[Q_{\phi_1}(s_i,a) - \alpha\log\pi_\theta(a|s_i)],$$

the gradient of which can be more easily estimated by applying a reparametrization trick that maps the distribution of π_θ to a Gaussian distribution $\mathcal{N}(0,1)$ and transforms the actions accordingly. That makes the objective differentiable w.r.t. θ and gradient ascent can be applied for direct policy updates.

2.3 Benchmarking

Benchmarking in RL is the process of comparing the performance of different algorithms by testing them on the same environments and under fair conditions (François-Lavet et al., 2018). In order for results to be consistent and reproducible in accordance with best scientific practices, it is important to consider several key aspects for benchmark design:

- Both environments and algorithms are frequently affected by random effects and stochasticity, and these factors can cause significant variance in results between benchmark trials. A common recommendation is to run many trials (as many as possible) with different random seeds and average their results (Islam et al., 2017; Henderson et al., 2018).

- Considerable differences can exist even between different implementations of the same algorithm, as some are more optimized than others (Engstrom et al., 2020). For fair benchmarks it is crucial to pick reliable and consistent algorithm implementations (Raffin et al., 2021).

- Hyperparameter tuning can also cause large differences in an algorithm's ability to learn tasks (Henderson et al., 2018). A benchmark where the hyperparameters of one algorithm were more carefully tuned than another would lead to an unfair comparison. Researchers testing algorithms should always disclose all hyperparameters used in their experiments (Islam et al., 2017).

- The selection of environments impacts the performance of algorithms. Some environments are more challenging than others, and some algorithms are well-suited for a particular type of environment only (Henderson et al., 2018). A reliable benchmark should use a comprehensive selection of tasks with varied difficulty and different objectives (Duan et al., 2016; François-Lavet et al., 2018).

- It matters which metrics are chosen to evaluate and compare the performance of algorithms. For instance, reporting maximum returns instead of averages may be biased in favor of unstable or unreliable algorithms (François-Lavet et al., 2018). Furthermore, algorithms may find a local optimum that appears to maximize returns but does not actually succeed at solving the task (Henderson et al., 2018). Researchers should ensure they understand what the reported metrics actually indicate.

Chapter 3

Related Work

There are many non-robotic benchmarking frameworks for RL and other machine learning applications. One well-known example is the Arcade Learning Environment (ALE), which tests agents on their ability to play Atari 2600 video games (Bellemare et al., 2013). The ALE has seen very widespread use for testing new machine learning technologies, and while some improvements to it have been suggested, it remains a very useful benchmark tool (Machado et al., 2018). However, Duan et al. (2016) highlighted that the ALE is designed for discrete action spaces and often fails to accurately measure the performance of algorithms designed for continuous actions (Lillicrap et al., 2015). Because robot movements are best described by continuous action spaces (Kober et al., 2013), different benchmarks are required for robotics applications and examples were proposed by Duan et al. (2016).

While Mahmood et al. (2018) found that benchmarking algorithms on real robots would be preferable to demonstrate real-world performance, it is also clear that such testing requires significant cost, effort and time. Even when using simple and low-cost robots as proposed by Ahn et al. (2020), there are benefits to using a simulated robot for faster prototyping and initial testing. Kober et al. (2013) discuss the benefits of first training, evaluating and tuning algorithms on simulated environments before applying them to the real robot. When following this sim-to-real transfer approach, it is important that the simulation represents the real robot as closely as possible (Zhao et al., 2020).

During the past years there have been a number of newly developed realistic robot benchmarks, which all differ in their scope and implementation details. Authors chose different physics engines (Collins et al., 2021), simulated many robot types which are controlled in different ways, and designed a variety of tasks.

Yu et al. (2020) introduced Meta-World, a benchmark for multi-task meta-RL algorithms with fifty different robot tasks. In Meta-World, a single policy is trained on one set of tasks (multi-task RL) and tested on another set of tasks (meta-learning). The practical implementation uses the MuJoCo engine and simulates a Sawyer robot arm. Meta-World tasks are designed in a way that allows the same agent to perform all of them interchangeably. This approach is aimed to test the ability of a RL algorithm to learn multiple distinct tasks, transfer its knowledge and adapt to novel tasks that were not seen in training. Yu et al. tested six algorithms on their environments, including variants of PPO and SAC, and found that they had created a very challenging benchmark suite. Wołczyk et al. (2021) subsequently demonstrated how Meta-World can be used as the basis for other, new benchmarks, in that case the Continual World which is aimed at benchmarking algorithms for continual RL.

Robosuite (Fan et al., 2018; Zhu et al., 2020) is a framework which is capable of simulating several different robots on a number of different tasks. This approach has the benefit of allowing user to not only compare different algorithms and tasks, but to also evaluate the benefits and drawbacks of specific robots in otherwise equal circumstances. Because some robots have unique capabilities as well as limitations, this presents researchers with more options to ensure that their algorithm can perform tasks under a wider variety of conditions. Further options are provided by robosuite's support for a variety of controller types such as Cartesian coordinate positions, joint velocities and joint torques. Because the choice of controller determines the action space of the RL agent, it can have a considerable impact on the learning performance of RL algorithms.

Another powerful robotics benchmark titled RLBench (James et al., 2020) emphasizes the importance of using as many tasks as possible with 100 unique tasks designed for the Franka Emika Panda robot. The architecture of RLBench is intended to allow applications beyond reinforcement learning, such as in imitation learning, and makes it easy to transfer policies from the simulation to real-world use.

Mirza et al. (2020) propose embedding abstract, discrete 2D tasks such as Sokoban and tic-tac-toe within a realistic 3D physics simulation. This design combines the continuous robot simulation approach with that of the ALE and other simple RL environments. The agent controls a robot arm and must learn to manipulate physical objects in a way that also solves the represented games. It is then possible to test the ability of machine learning methods to accomplish the transfer between abstract tasks and realistic robotics.

CausalWorld (Ahmed et al., 2020) is a benchmark based on simulating the open-source TriFinger robot (Wüthrich et al., 2020). Its tasks center on object manipulation (e.g. pushing, picking, placing, stacking) with options to modify individual properties of the simulation as needed. The benchmark's viability was demonstrated with tests of PPO, SAC and TD3, showing that the performance of each algorithm can depend greatly on the choice of task as well as factors like goal randomization.

Panda-gym (Gallouédec et al., 2021) uses a Franka Emika Panda robot simulation based on the physics engine PyBullet and implements five simple task environments (reach, push, slide, pick and place, and stack) that can be used to benchmark RL algorithms. Panda-gym focuses on multi-goal RL applications and primarily uses sparse reward functions, commenting on the challenges of defining dense reward functions. Evaluating DDPG, TD3 and SAC on their benchmark, Gallouédec et al. found that the algorithms' performance would differ significantly between tasks, demonstrating a gradient of difficulty between tasks.

Lee et al. (2021) proposed an environment for assembling IKEA furniture, a difficult task that requires a complex series of actions and long-term planning. The framework simulates six different robot arms and 60 models of furniture. In benchmarking multiple algorithms with this environment, the authors found that SAC and PPO were unable to learn the furniture assembly task. While they succeeded at gripping and holding the furniture parts, they were usually unable to attach them together and could not complete the assembly.

Since their introduction, many of these different benchmarks have been used to evaluate novel algorithms and technologies (Bharadhwaj et al., 2020; Sinha et al., 2022; Seo et al., 2022; He and Lv, 2023), demonstrating their value to the scientific community.

Chapter 4

Methods

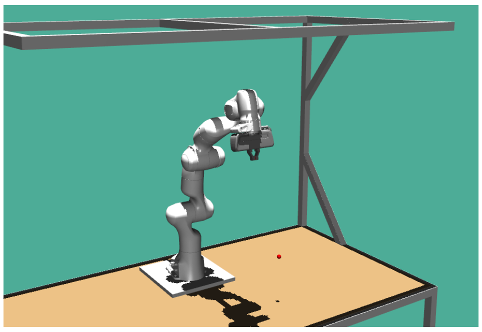
4.1 ALR Simulation Framework

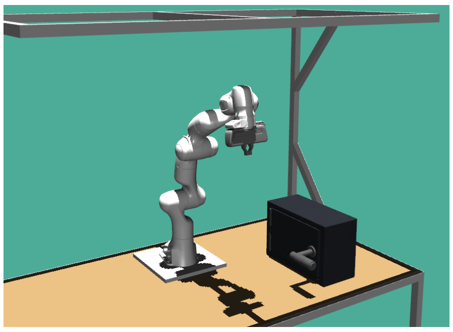
The Autonomous Learning Robots (ALR) Lab at the KIT Institute for Anthropomatics and Robotics has developed its own framework for realistically simulating a Franka Emika Panda robot arm with seven degrees of freedom and a gripper mounted as its end effector (pictured in Figure 4.1) (ALR Team, 2022). The Panda robot is a widely used system in research for both simulated and real applications (Gallouédec et al., 2021).

The robot arm is controlled by direct torque inputs to its joints, leading to a continuous 7-dimensional action space. This method allows for more direct control of the robot and reduces computational overhead compared to the alternative approach of providing Cartesian coordinates as inputs, as used for instance by Meta-World and panda-gym (Yu et al., 2020; Gallouédec et al., 2021).

The simulation framework supports the Bullet and MuJoCo (Todorov et al., 2012) physics engines. This work uses MuJoCo for its experiments with a simulation frequency of 1000 Hz (ALR Team, 2022). For the purposes of the RL environments, each environment step consists of one torque command which is executed for 10 simulation steps (total of 0.01 seconds). All observations returned by the environments have in common that they contain the following 27-dimensional data about the robot's state: Positions and velocities of all seven arm joints, opening width and velocity of the gripper component, and position and velocity information about the end effector.

4.2 Tasks


(a) Reach


(b) Door Opening

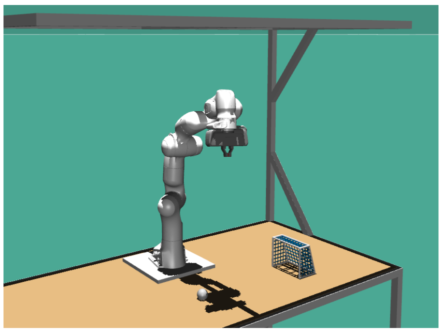

(c) Soccer

Figure 4.1: Screenshots of the Panda robot and the tasks implemented in the Simulation Framework

Meta-World (Yu et al., 2020) serves as the source for the tasks in this work. The task designs were adapted from their Meta-World versions and re-implemented from scratch with updated observations, rewards and other details that fit the ALR Simulation Framework. The following tasks were chosen, in order of increasing complexity (see also Figure 4.1):

- Reach: The robot must move its end effector to a goal location. The goal location and the initial position of the robot arm are randomized.
- Door Opening: The robot must pull on the handle to swing open a door. The starting position of the door door and robot arm are static.
- Soccer: The robot must kick or push a ball into a soccer goal. The starting position of the ball is randomized while the goal and robot arm remain static.
- Hammer: The robot must grasp and pick up a hammer and use it to push a nail into a block of wood. The hammer task was originally intended to be the final task of this project, but it was omitted due to technical limitations of the Simulation Framework (specifically, torque control over the gripper component) and time constraints.

All environments were implemented using the `gym` library and wrappers included with the Simulation Framework. In the API provided by the `gym` library, the agent primarily interacts with the environment by passing its chosen action to the `step(action)` function. The environment then internally executes this action and responds with an observation of the environment's state after the action, a reward resulting from the action, and a Boolean signal indicating whether the current episode has ended (e.g. by reaching its goal or exceeding the maximum number of steps) (Brockman et al., 2016). The following sections document how each of these values is calculated by the three implemented task environments. All tasks work with negative dense rewards, the reward space is capped at 0.

4.2.1 Reach Environment

When the environment is initialized, and every time it is reset, the starting position of the robot arm and the goal position are newly randomized. The goal position is uniformly sampled from the box space between the points $[0.2, -0.3, 0.1]$ and $[0.5, 0.3, 0.5]$. All possible positions are well within the arm's reach. For the robot arm's initial position, a value is sampled from between $[-0.05, -0.05, -0.05, -0.05, -0.025, -0.025, -0.025]$ and $[+0.05, +0.05, +0.05, +0.05, +0.025, +0.025, +0.025]$ and added to the default initial position (the 7-tuple values correspond to the 7 DoF in order from the arm's base to its end effector). Alternatively, the user may choose to keep one or both positions static. Keeping the goal static makes the task substantially easier.

Each step reward is calculated as

$$r_t = -\exp(||g - p||^2)$$

where p and g are the 3-dimensional Cartesian coordinates of the robot end effector and goal positions, respectively. Placing the squared Euclidean distance in an exponential function strongly penalizes the agent for being far from the target, but the reward rapidly grows as the end effector approaches the goal.

The observation space is 34-dimensional, and has the following contents: The end effector position p and other robot state data (27-dimensional), the goal position g (3-dimensional), as well as the absolute distance $||g - p||$ (1-dimensional) and relative distance (3-dimensional). Here, the relative distance between two points x, y is defined as

$$\frac{x - y}{||x - y||}. \tag{4.1}$$

The threshold for an episode's success is given as $||g - p|| < 0.025$, at which point it terminates early. Otherwise the maximum episode length is 250 environment steps or 2.5 seconds.

4.2.2 Door Opening Environment

This environment uses the same door assets as Meta-World (Yu et al., 2020). The door is mounted on the side of a large box (safe) and swings open outwards (see Figure 4.1(b)). A handle is located on the far side of the door, large enough for the robot to pull on and open the door.

The reward for the task is composed of two parts. The first component is the hinge penalty

$$r^{\text{hinge}} = exp(\theta - \frac{\pi}{2}) - 1,$$

where θ is the opening angle of the door in radians. r^{hinge} is negative for small values of θ and becomes positive if the door is opened by more than 90°. The second part may be called the handle distance penalty

$$r^{\text{handle}} = \mathbb{I}_{p \notin B_h} ||p - h||$$

which denotes the Euclidean distance between the end effector position p and a point h located behind the door handle. This component serves as an incentive for the robot to move its hand into the correct position for opening the door. For this reason, the penalty is only applied when p is not within a box-shaped 3-dimensional interval of size $[0.16, 0.16, 0.05]$ centered on the point h, denoted as B_h. This condition is expressed with the indicator function

$$\mathbb{I}_{p \notin B_h} = \begin{cases} 1 \text{ if } p \notin B_h \\ 0 \text{ if } p \in B_h \end{cases} . \tag{4.2}$$

From these preliminaries, the step reward is finally calculated as $r_t = \min(-r, 0)$ with

$$r = w_1(r^{\text{hinge}} + w_2 r^{\text{handle}}),$$

where w_1 and w_2 are weights to modify the relative impact of the two components. For the experiments in Chapter 5, these were set to $w_1 = 25$ and $w_2 = 30$.

This environment's observation space is 47-dimensional and is composed of the following data: Current robot state information (27-dimensional) and end effector's Cartesian position p (3-dimensional), position of the door handle d (3-dimensional) and distance $||p - d||$ between the end effector and door handle (1 dimension), the position of the target point h behind the handle (3-dimensional), the size of the target box B_h which is centered on h (3 dimensions), the difference $p - h$ (3-dimensional) and the distance $||p - h||$ (1-dimensional) between the end effector and the target point, as well as the current opening angle θ (in radians, 1-dimensional) of the door measured at the hinge, the target angle of $\frac{\pi}{2}$ and the difference $\theta - \frac{\pi}{2}$ (each 1-dimensional).

The success threshold is reached when the door hinge angle is greater than 30° ($\theta > \frac{\pi}{6}$). In case of no early termination, the maximum episode length is 625 environment steps (6.25 seconds).

4.2.3 Soccer Environment

For its goal, this environment uses the soccer goal assets from Meta-World (Yu et al., 2020). The goal is located in a fixed position (see Figure 4.1(c)). The goal posts are located at the coordinates $[0.3, 0.44]$ and $[0.5, 0.44]$. The inside of the goal is 0.15 units tall and 0.10 units deep (in positive y-direction). These dimensions define the goal box B_g. The ball is a sphere of radius 0.026 and mass 0.04, which is placed at a random position that is newly sampled in each episode from the box space between the points $[0.2, -0.5, 0.01]$ and $[0.6, 0.3, 0.01]$. Optionally, the user may choose to keep the starting position of the ball static, which makes the task less challenging to learn.

This environment's 50-dimensional observation includes the following data: Current robot state (27 dimensions), the end effector's Cartesian position p (3-dimensional), ball position b (3-dimensional), the center of the goal g (3-dimensional) and the boundaries of the goal box B_g (6 dimensions), the absolute ($||p-b||$) and relative distance (see Equation (4.1)) between end effector and ball (together 4 dimensions), and finally the absolute ($||b-g||$) and relative distance between ball and goal (4-dimensional).

The reward is composed of three main components: First, the distance between ball and goal $||b-g||$, which is always part of the calculation. Second, a penalty is added for the distance between end effector and ball $||p-b||$, which is intended to help the robot find the ball's initial position. This penalty is only applied if the distance between p and b exceeds 0.1. Finally, another penalty is added if the ball is behind the goal line (y-coordinate $b_y > 0.44$). If however the ball is inside the goal box ($g \in B_g$), none of the above applies because the episode is immediately considered a success and the reward is set to the maximum of 0.

In summary, the step reward is calculated as $r_t = \min(-r, 0)$ with

$$r = \mathbb{I}_{b \notin B_g}(w_1||b-g|| + \mathbb{I}_{||p-b||>0.1} w_2 ||p-b|| + \mathbb{I}_{b_y>0.44} w_3),$$

where w_1, w_2 and w_3 are weights determining the relative importance of the three components. For the experiments in Chapter 5, these were set to $w_1 = 50$, $w_2 = 15$, $w_3 = 50$. Once again, $\mathbb{I}$ denotes indicator functions whose definitions are analogous to Equation (4.2).

The maximum episode length before a reset for this environment is 500 simulation steps, meaning 5 seconds when running in real time.

Chapter 5

Experiments

All experiments were run on the *BwUniCluster 2.0* high-performance computing cluster (bwHPC Wiki Maintainers, 2023). PPO was trained on 8 CPUs to allow for 8 parallel actors, while the other algorithms were each trained on a single CPU.

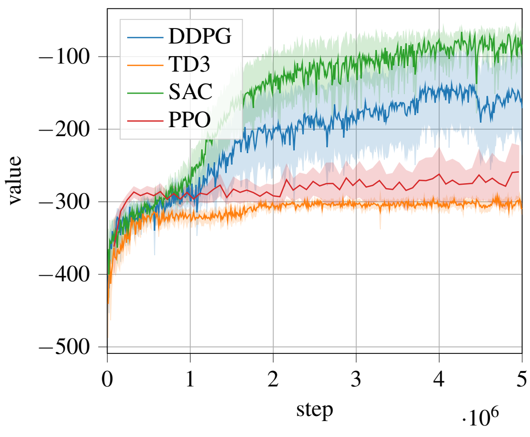
The four algorithms were run on the reach and door opening tasks with 20 random seeds, and on the soccer task with 10 random seeds (due to time limitations). Each run lasted for a total of 5 million simulation steps. The result plots in the following sections show the mean of all seeds as the solid lines, with the shaded areas representing twice the standard error of the mean.

All algorithms were used in their standard implementation found in Stable-Baselines3 (Raffin et al., 2021), which is intended to ensure a fair comparison between the algorithms. An overview of some relevant hyperparameters is provided in Table 5.1.

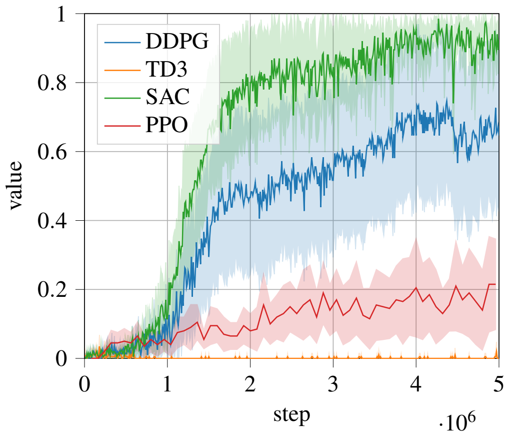
Hyperparameter	DDPG	TD3	SAC	PPO
Discount rate γ	0.99	0.99	0.99	0.99
Learning rate α	0.001	0.001	0.0003	0.0003
Polyak update value β	0.005	0.005	0.005	—
Clipping range ε	—	—	—	0.2
GAE parameter λ	—	—	—	0.95
Policy delay d	—	2	—	—
Sampled minibatch size N	100	100	256	64

Table 5.1: Algorithm hyperparameters

5.1 Reach Results



(a) Mean episodic rewards



(b) Success rates

Figure 5.1: Reach experiment results of all four algorithms.

The results of all four algorithms evaluated on the reach environment are shown in Figure 5.1. Figure 5.1(a) shows the mean episodic rewards encountered during evaluation, while Figure 5.1(b) plots the success rates as a percentage value.

DDPG's mean episodic rewards exceed -300 after 1 million steps, reach -200 at around 2 million steps and level out at approximately -150 between 4 and 5 million steps. Its success rate grows most significantly in the period between 1 and 2 million steps, from approximately 0.1 to 0.5. It further increases and finishes at about 0.7 at 5 million steps. Both evaluations of DDPG show a very large standard error, which does not significantly decrease with time. Between the 4 and 5 million step points, both the mean rewards and success rate for DDPG noticably drop temporarily, counter to the overall increasing trend.

TD3 sees its mean rewards improve at the very beginning of training, but they level out and do not improve beyond -300 after 2 million steps. TD3's success rate remains at or near 0 throughout. The standard error of both metrics is insignificant.

SAC displays a similar behavior to DDPG. Its mean episodic rewards are approximately -250 after 1 million steps. After 3 million steps they have reached -100 and the algorithm appears to converge to a value above -100. SAC's success rate grows rapidly during the beginning stage of training, passing 0.2 at the 1 million point and reaching 0.8 after 2 million steps. By the end of 5 million steps, the mean success rate is in excess of 0.9. While the standard error of SAC is relatively large during the middle stage of training, it shrinks considerably by the 4 million step mark. SAC's standard error is much smaller than that of DDPG overall.

PPO's mean rewards cross the threshold of -300 very quickly, but only grow slowly after that point. At the end of 5 million steps the mean episodic reward remains below -250. Its mean success rate also makes slow progress, passing 0.1 at 2 million steps and only reaching approximately 0.2 after 5 million steps. The standard error of the mean for PPO increases as time goes on.

5.2 Door Opening Results

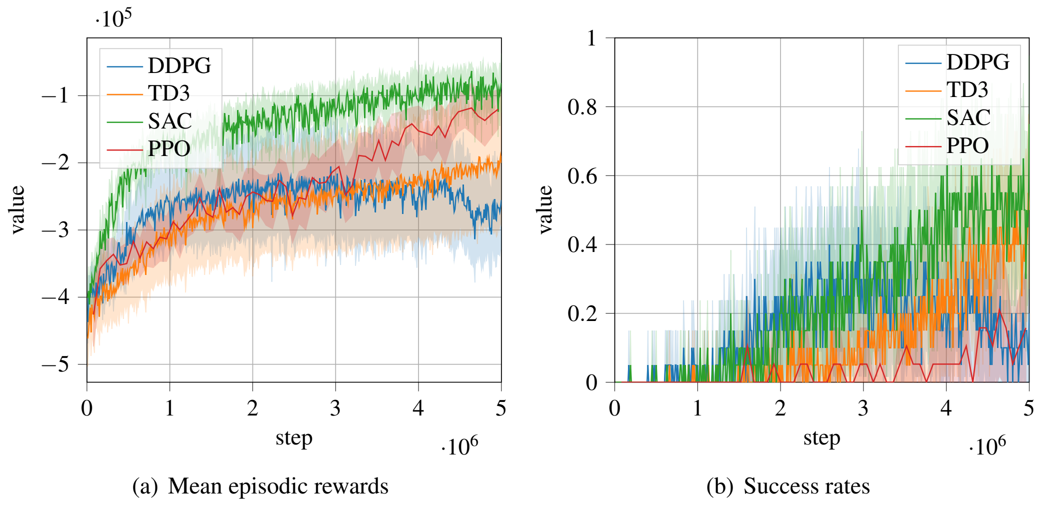


Figure 5.2: Door opening experiment results of all four algorithms.

Figure 5.2 shows the results of evaluating the four algorithms on the door opening environment. Because of the limited legibility of the overlapping success rate curves in Figure 5.2(b), the same data is shown with a Gaussian filter applied in Figure A.1 of the appendix.

DDPG at first sees its mean episodic rewards grow relatively quickly, passing -3×10^5 after 500,000 steps. After passing 1 million steps however, the value does not increase much further. It levels out at less than -2×10^5 and decreases slightly during the second half of training and ends at less than -2.5×10^5 after 5 million steps. DDPG's success rate first passes 0.2 at 2 million steps and reaches its maximum of approximately 0.3 after 3 million steps. Following that point, the success rate steadily decreases and finishes at below 0.2 at the end of training. The standard error recorded for both values is relatively large in the middle stages, but shrinks towards the end of training.

TD3's mean rewards grow steadily past -3×10^5 at 1 million steps and reach -2×10^5 at the end of 5 million steps. The algorithm's success rate only increases slowly at the beginning, surpassing 0.1 after 3 million steps. Growth accelerates at that point and the success rate surpasses 0.4 at 5 million steps. Standard errors for both mean rewards and success rate are consistently large for TD3.

SAC exhibits very fast growth of the mean rewards, crossing -2×10^5 in less than 1 million steps. After 3 million steps it approaches -1×10^5, converging slightly above that value at the end of training. The success rate grows steadily to 0.2 at 2 million steps, 0.4 at 3.5 million steps, and finishing at approximately 0.5 after 5 million steps. While SAC's mean reward standard error is small, its success rate has a significantly larger error.

PPO's mean episodic rewards increase steadily, passing -3×10^5 at 1 million steps and nearly increasing to -1×10^5 at the end of 5 million steps. Its success rate does not grow as quickly. It remains near 0 for most of the training duration and only increases past 0.1 occasionally between the 4 and 5 million step markers. The standard error of the mean rewards is stable throughout training but the success rate's standard error appears to increase with time.

5.3 Soccer Results

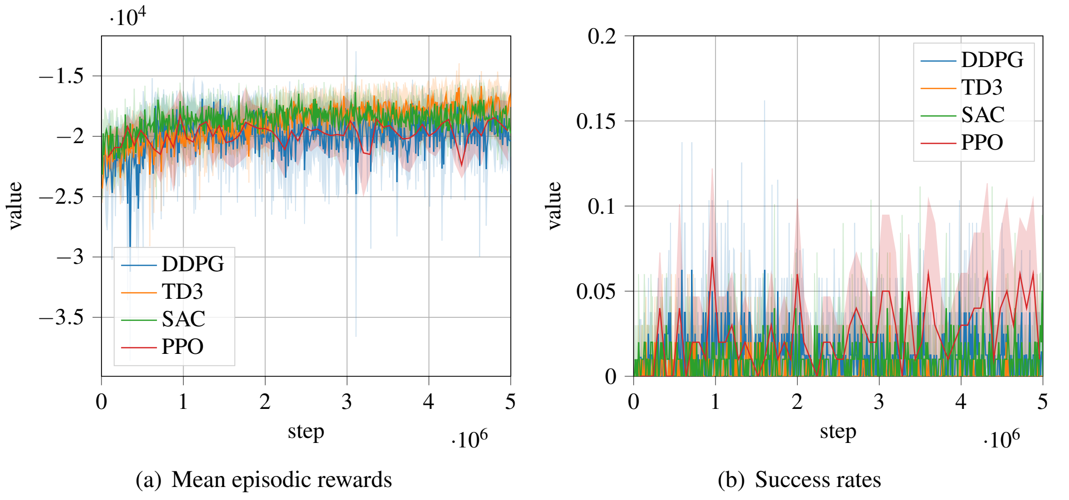


Figure 5.3: Soccer (randomized) experiment results of all 4 algorithms.

The soccer task's results with randomized ball positions are shown in Figure 5.3. Note that, unlike for the other two tasks, the success rate plot has its y-axis limited to only 0.2 instead of 1. Furthermore, both data sets are shown in Figure A.2 of the appendix with a Gaussian filter applied to help the reader more easily distinguish between the overlapping curves.

DDPG's mean episodic rewards reach -2×10^4 after 1 million steps but do not grow substantially after that point and appear to converge at only slightly more than that value. The algorithm's success rate records several spikes early in training around the 1 million step mark but remains below 0.05 at almost all other times, including at the end of 5 million steps. The standard error of the mean however appears to slightly decrease later in training.

TD3 sees its mean rewards grow consistently past -2×10^4 at 1 million steps to approximately -1.75×10^4 at the end of training. Its success rate however remains at or very near 0 throughout training. The mean rewards recorded by SAC increase the most quickly at the beginning, passing the -2×10^4 mark after less than 0.5 million steps. The episodic rewards then level out and remain at a value between those of DDPG and TD3.

PPO's mean rewards grow inconsistently, alternating above and below the -2×10^4 level. The standard error is also noticeable and does not shrink as time goes on. The algorithm's average success rate increases overall and reaches or exceeds 0.05 multiple times after 4 million steps. PPO has a significant standard error that does not substantially decrease even as time goes on.

For the sake of comparison, results of an experiment using the static version of the environment (where the ball's starting position is not randomized) and trained on a shorter period of only 2 million steps are included in Appendix A. Note that, due to the shorter training period, those results may not be as conclusive.

Chapter 6

Evaluation

This chapter contains analyses and evaluations of the results obtained in Chapter 5. The first section discusses the results of each task by itself, the second section then evaluates the individual algorithms based on their overall performance across tasks.

6.1 Tasks

The reach task was most successfully learned by SAC. The results indicate that the algorithm achieved a very high success rate of 90% or beyond on all seeds. DDPG is the second best performer with an average success rate of around 70%, but the large standard error indicates that the algorithm was much more successful on some seeds than on others. Such inconsistency, learning very well in some cases but failing with only slightly different circumstances, can indicate a problem with the reliability of the algorithm. TD3 was entirely unable to learn this task, with episodic rewards plateauing at a low level and the success rate staying at zero. This is a surprising result, given that TD3 is mean to be an improvement over DDPG. It may be that the weaknesses that caused DDPG to fail on some seeds were amplified in TD3. Alternatively, this result could be caused by unsuitable hyperparameters on the algorithm's side or poor reward scaling within the environment. PPO shows that it is able to somewhat learn this task, but its metrics only improve at a slow rate.

In summary, this was intended to be the easiest task and the quick success of SAC and in some cases DDPG seems to demonstrate that it can be solved very successfully. On the other hand, the high variance of DDPG's performance, the slow improvement exhibited by PPO and especially the complete failure of TD3 may be indicative of design problems with the environment. Future work might experiment with higher reward scaling, i.e. applying a multiplier to the rewards to increase the impact of small differences in the distance between end effector and goal.

SAC also appeared to be the most successful algorithm in the door opening task. Mean rewards grew faster than any other algorithm and reached the highest overall values. Its success rate initially appeared to be slightly behind DDPG, but continued to grow after DDPG reached its maximum. TD3 learned more slowly than SAC but reached nearly the same success rate of 50% by the end of training. It took 4 million steps to surpass simple DDPG, but did not suffer from the same problem of decreasing performance. DDPG's performance during the first half of training seemed promising, but the fact that it reached a maximum and decreased its success rate as time went appears to be a massive flaw. PPO saw steady growth of mean episodic rewards that almost caught up with SAC at the 5 million step mark, but this apparent success was not reflected in the success rate statistic. This result may indicate that the algorithm found local maxima without achieving the actual goal of opening the door, which would suggest insufficient exploration. On the other hand, the increase in the success rate towards the end of the training period may signify that its success could have grown significantly if training had continued for a slightly longer time.

Ultimately, the door opening task enabled all four algorithms to learn. In that regard it was more consistent than the reach task, which had a large variance between successful and unsuccessful trials. This difference may be caused by the more complex observation and the shape of the reward, but this cannot be conclusively stated without additional experimentation.

None of the four algorithms was able to reliably solve the soccer task in its randomized version. PPO had the highest average success rate which climbed above 5% with increasing frequency during the later parts of training, but the standard error is too large to draw reliable conclusions. On the other hand, even out of four unsuccessful algorithms, TD3 had a noticeably lower success rate despite a higher mean episodic reward than the rest. DDPG and SAC both produced mediocre results, with DDPG once again recording a higher success rate in the first half of training and failing to improve on that later.

Overall, this task was evidently the most challenging of the three. Yet this circumstance should not be mistaken as a failure of the benchmark, as it was part of the design for the difficulty of each task to increase. In fact, this can make future progress very apparent if other algorithms than the four tested here are able to solve this environment reliably. The comparison with the static-ball version (see Appendix A) shows how much this task's complexity is amplified by the randomization of the ball's starting position. PPO was able to learn this version of the task quite successfully, and DDPG and SAC also saw improved performance even after a training period less than half of the usual duration. Further experimentation may examine versions of the environment where the ball position is still random but limited to a smaller area, or sampled from a set of a few discrete position. Such a modification would likely lead to an environment with intermediate difficulty.

6.2 Algorithms

In all three tasks, DDPG performed relatively well in the earlier stages of training but appeared to struggle with long training runs. This weakness is especially apparent in the door opening task, where DDPG reached a maximum and average performance actually degraded as time went on. The temporary drop in performance in the reach task at around 4.5 million steps may have the same cause. Even in the challenging soccer task, DDPG's success rate spiked around the 1 million step mark and yet it failed to reproduce or improve upon these maxima later.

TD3 proved to be unreliable in these benchmarks. It performed much better on the door opening task than the others, despite the reaching task being considerably simpler and less challenging for the other algorithms. This inconsistency may be explained by the algorithm's default hyperparameters not being suitable for the simpler task. On the other hand, when it succeeded in the door opening task, TD3 was able to outperform regular DDPG in the long term by avoiding the latter's apparent performance degradation. It even approached similar success rates to SAC, even if it took slightly longer to reach that point.

SAC stands out as a top performer in all three tasks, though especially the reach and door opening tasks. It was consistently better than DDPG, which was to be expected given that it was designed to be an improvement over the latter. SAC learned faster and did not suffer from the same performance degradation issues as DDPG. In the soccer task SAC appeared to slightly outperform DDPG and TD3 as well, but was surpassed by PPO. This was especially apparent in the trial of the static soccer version, which may show a need for further optimization of SAC for that task.

PPO struggled somewhat with the reach and door opening tasks, though it still made some progress and never completely failed. It learned considerably slower than the other algorithms, likely due to reduced sample efficiency caused by its on-policy design. Given enough time, PPO's performance may be improved. Its poor sample efficiency is however partially counteracted by the ability to utilize multiple parallel agents for sample collection, which reduced the real time spent on training if at the cost of dedicating more cores to the task. This pattern of slower learning also appeared to be reversed in the soccer task, especially the static version where PPO succeeded after a much smaller amount of samples. It is not clear why PPO appears to be so relatively well-suited for this task compared to the others, it may be the result of its on-policy architecture but no clear conclusion can be drawn without further investigation.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

The aim of this thesis has been to develop and demonstrate a set of environments for benchmarking reinforcement learning algorithms using the ALR's custom Simulation Framework. I explained the basics of RL — including the four well-established algorithms PPO, DDPG, TD3 and SAC — as well as benchmarking, and explored how other works have approached this topic. I introduced the fundamentals of the Simulation Framework and detailed how I used it and other tools to develop three benchmark task environments of different complexity. I conducted scientific experiments, training the four aforementioned algorithms on the three environments. Finally, I evaluated the results of the experiments, discussing the strengths and weaknesses of each algorithm as shown by their ability to learn the three benchmark tasks. I also proposed explanations for unexpected behaviors and suggested how the benchmark environments may still be improved in the future.

7.2 Future Work

This work may be expanded upon by implementing additional task environments using the established methods and the ALR Simulation Framework. As a simple example, the door opening environment can easily be modified to a door closing task. Additionally, if the technical limitations of the Simulation Framework can be overcome in the future, the hammer environment that had to be omitted from this work would be a viable candidate. Furthermore, Meta-World and other benchmarks referred to in Chapter 3 can once again serve as the inspiration for adapting other tasks.

Besides new environments, improvements may also be made to the existing environments. Given the well-known difficulty of designing good dense reward functions (Gallouédec et al., 2021), it is possible that a different reward design could improve the reliability of the environments. While the rewards in this work have been tuned with the goal of making a diverse set of challenging yet solvable tasks, further optimization and rescaling may help some of the algorithms overcome unexpected problems with training. Some specific suggestions for environment improvements can be found in Chapter 6.

Similarly, some the tested algorithms may benefit from detailed hyperparameter tuning for optimal performance. In addition, it would be valuable to test a greater number of RL algorithms on the benchmark tasks (existing and new). One likely candidate would be A2C, the Stable-Baselines3 variant implementation of Asynchronous Advantage Actor Critic (Mnih et al., 2016; Raffin et al., 2021).

Future work may also be done on sim-to-real transfer of the benchmark tasks by training agent policies in the simulated environments, then evaluating how well their performance translates to the real world. While some modifications to the environments would likely be necessary to reflect additional constraints imposed by a real setting, the ALR Simulation Framework is designed as a realistic simulation and seeks to make transfer to the real robot as easy as possible. Testing sim-to-real transfer using this work's tasks could demonstrate these capabilities or, if unsuccessful, reveal new challenges.

Acknowledgments

The author acknowledges support by the state of Baden-Württemberg through bwHPC.

Bibliography

O. Ahmed, F. Träuble, A. Goyal, A. Neitz, Y. Bengio, B. Schölkopf, M. Wüthrich, and S. Bauer. Causalworld: A robotic manipulation benchmark for causal structure and transfer learning. *arXiv preprint arXiv:2010.04296*, 2020.

M. Ahn, H. Zhu, K. Hartikainen, H. Ponte, A. Gupta, S. Levine, and V. Kumar. Robel: Robotics benchmarks for learning with low-cost robots. In *Conference on robot learning*, pages 1300–1313. PMLR, 2020.

ALR Team. Alr simulation framework documentation, 2022. URL `https://github.com/ALRhub/SimulationFramework`. Unpublished, last accessed 27 February 2023.

K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath. Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34(6):26–38, 2017.

M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47: 253–279, 2013.

H. Bharadhwaj, A. Kumar, N. Rhinehart, S. Levine, F. Shkurti, and A. Garg. Conservative safety critics for exploration. *arXiv preprint arXiv:2010.14497*, 2020.

G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

bwHPC Wiki Maintainers. Bwunicluster 2.0 documentation, 2023. URL `https://wiki.bwhpc.de/e/BwUniCluster2.0`. Last accessed 26 February 2023.

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

Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel. Benchmarking deep reinforcement learning for continuous control. In *International conference on machine learning*, pages 1329–1338. PMLR, 2016.

L. Engstrom, A. Ilyas, S. Santurkar, D. Tsipras, F. Janoos, L. Rudolph, and A. Madry. Implementation matters in deep policy gradients: A case study on ppo and trpo. *arXiv preprint arXiv:2005.12729*, 2020.

L. Fan, Y. Zhu, J. Zhu, Z. Liu, O. Zeng, A. Gupta, J. Creus-Costa, S. Savarese, and L. Fei-Fei. Surreal: Open-source reinforcement learning framework and robot manipulation benchmark. In *Conference on Robot Learning*, pages 767–782. PMLR, 2018.

V. François-Lavet, P. Henderson, R. Islam, M. G. Bellemare, J. Pineau, et al. An introduction to deep reinforcement learning. *Foundations and Trends® in Machine Learning*, 11(3-4): 219–354, 2018.

S. Fujimoto, H. Hoof, and D. Meger. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pages 1587–1596. PMLR, 2018.

Q. Gallouédec, N. Cazin, E. Dellandréa, and L. Chen. panda-gym: Open-source goal-conditioned environments for robotic learning. *arXiv preprint arXiv:2106.13687*, 2021.

T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018a.

T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018b.

X. He and C. Lv. Robotic control in adversarial and sparse reward environments: A robust goal-conditioned reinforcement learning approach. *IEEE Transactions on Artificial Intelligence*, 2023.

P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger. Deep reinforcement learning that matters. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

R. Islam, P. Henderson, M. Gomrokchi, and D. Precup. Reproducibility of benchmarked deep reinforcement learning tasks for continuous control. *arXiv preprint arXiv:1708.04133*, 2017.

S. James, Z. Ma, D. R. Arrojo, and A. J. Davison. Rlbench: The robot learning benchmark & learning environment. *IEEE Robotics and Automation Letters*, 5(2):3019–3026, 2020.

J. Kober, J. A. Bagnell, and J. Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013.

V. Konda and J. Tsitsiklis. Actor-critic algorithms. *Advances in neural information processing systems*, 12, 1999.

Y. Lee, E. S. Hu, and J. J. Lim. Ikea furniture assembly environment for long-horizon complex manipulation tasks. In *2021 ieee international conference on robotics and automation (icra)*, pages 6343–6349. IEEE, 2021.

T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

M. C. Machado, M. G. Bellemare, E. Talvitie, J. Veness, M. Hausknecht, and M. Bowling. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *Journal of Artificial Intelligence Research*, 61:523–562, 2018.

A. R. Mahmood, D. Korenkevych, G. Vasan, W. Ma, and J. Bergstra. Benchmarking reinforcement learning algorithms on real-world robots. In *Conference on robot learning*, pages 561–591. PMLR, 2018.

M. Mirza, A. Jaegle, J. J. Hunt, A. Guez, S. Tunyasuvunakool, A. Muldal, T. Weber, P. Karkus, S. Racanière, L. Buesing, et al. Physically embedded planning problems: New challenges for reinforcement learning. *arXiv preprint arXiv:2009.05524*, 2020.

V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR, 2016.

A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *The Journal of Machine Learning Research*, 22(1):12348–12355, 2021.

G. A. Rummery and M. Niranjan. *On-line Q-learning using connectionist systems*, volume 37. University of Cambridge, Department of Engineering Cambridge, UK, 1994.

J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR, 2015a.

J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015b.

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

Y. Seo, K. Lee, S. L. James, and P. Abbeel. Reinforcement learning with action-free pre-training from videos. In *International Conference on Machine Learning*, pages 19561–19579. PMLR, 2022.

D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller. Deterministic policy gradient algorithms. In *International conference on machine learning*, pages 387–395. Pmlr, 2014.

S. Sinha, A. Mandlekar, and A. Garg. S4rl: Surprisingly simple self-supervision for offline reinforcement learning in robotics. In *Conference on Robot Learning*, pages 907–917. PMLR, 2022.

R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT Press, second edition, 2018.

R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12, 1999.

E. Todorov, T. Erez, and Y. Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pages 5026–5033. IEEE, 2012.

C. J. Watkins and P. Dayan. Q-learning. *Machine learning*, 8:279–292, 1992.

R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Reinforcement learning*, pages 5–32, 1992.

M. Wołczyk, M. Zając, R. Pascanu, Ł. Kuciński, and P. Miłoś. Continual world: A robotic benchmark for continual reinforcement learning. *Advances in Neural Information Processing Systems*, 34:28496–28510, 2021.

M. Wüthrich, F. Widmaier, F. Grimminger, J. Akpo, S. Joshi, V. Agrawal, B. Hammoud, M. Khadiv, M. Bogdanovic, V. Berenz, et al. Trifinger: An open-source robot for learning dexterity. *arXiv preprint arXiv:2008.03596*, 2020.

T. Yu, D. Quillen, Z. He, R. Julian, K. Hausman, C. Finn, and S. Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Conference on robot learning*, pages 1094–1100. PMLR, 2020.

W. Zhao, J. P. Queralta, and T. Westerlund. Sim-to-real transfer in deep reinforcement learning for robotics: a survey. In *2020 IEEE symposium series on computational intelligence (SSCI)*, pages 737–744. IEEE, 2020.

Y. Zhu, J. Wong, A. Mandlekar, and R. Martín-Martín. robosuite: A modular simulation framework and benchmark for robot learning. *arXiv preprint arXiv:2009.12293*, 2020.

Appendix A

Additional Results and Figures

This appendix contains additional figures and results beyond what was included in Chapter 5.

A.1 Smoothed Plots

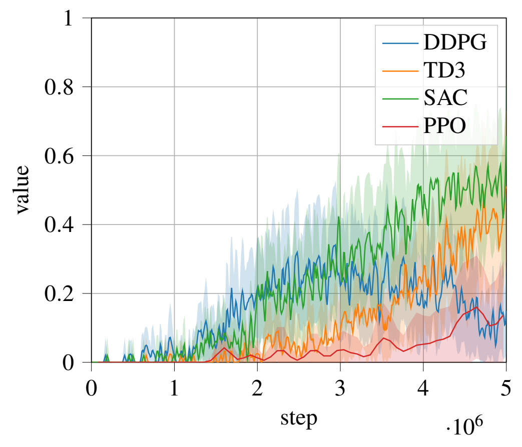
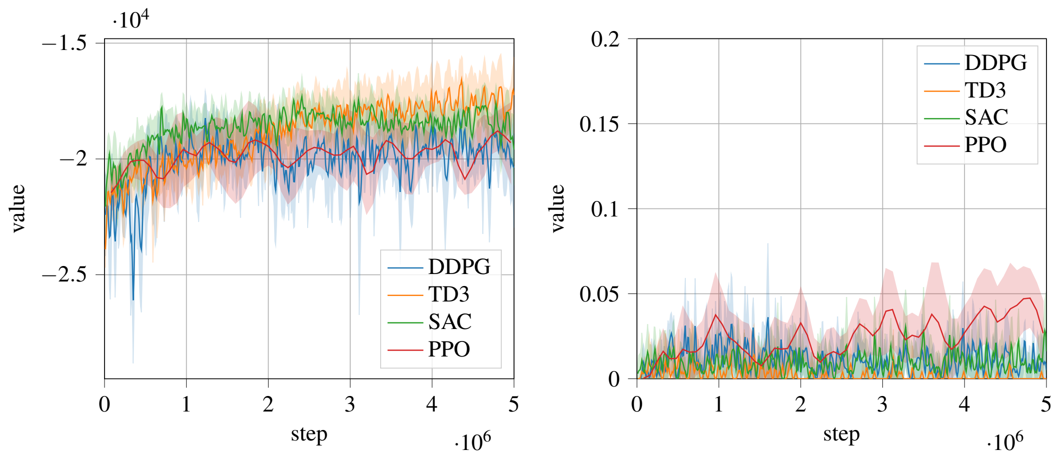


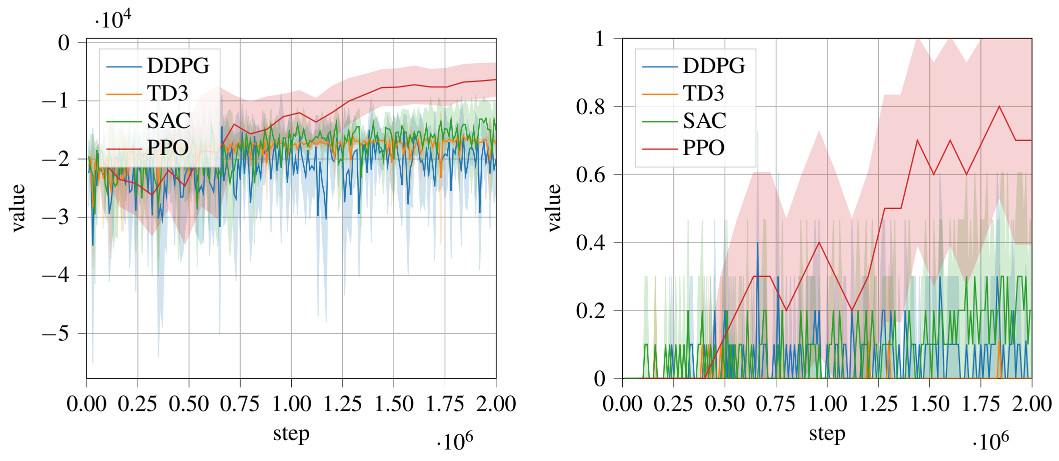
Figure A.1: Door opening experiment, success rates of all four algorithms. Plot is smoothed by a Gaussian filter ($\sigma = 1$).



(a) Mean episodic rewards (b) Success rates

Figure A.2: Soccer (randomized) experiment results of all 4 algorithms smoothed by a Gaussian filter ($\sigma = 1$).

A.2 Static Soccer Experiment



(a) Mean episodic rewards (b) Success rates

Figure A.3: Soccer (static) experiment results of all 4 algorithms.

Figure A.3 shows the results of an additional experiment on the static version of the soccer environment. Each algorithm was run on the environment 10 times with different random seeds and for 2 million simulation steps each.

The mean rewards for DDPG, TD3 and SAC all increase very little. Regarding success rates, TD3's remains at or near 0 for the entire duration. DDPG makes some progress to an average of around 0.1 at the end of 2 million steps. SAC has the most success of the three, reaching an average success rate of 0.2 and recording an increased standard error after 1.5 million steps, suggesting that the algorithm had considerable success on some runs but failed on others out of the 10 seeds.

PPO's mean episodic rewards, after initially dropping to a lower level of -2.5×10^4, then grow significantly and pass -1×10^4 after 1.25 million steps. Its average success rate is also much higher than the other algorithms, above 0.2 after 0.75 million steps and above 0.6 after 1.5 million steps, reaching a maximum of 0.8. The standard error of the mean is relatively large, however.