Rheinische
Friedrich-Wilhelms-
Universität Bonn

Institute for Computer Science
Department VI
Autonomous Intelligent Systems

Rheinische
Friedrich-Wilhelms-Universität Bonn

Master Thesis

Learning from Human Demonstrations with History-aware Offline Reinforcement Learning

Author:
Kara Moraw

First Examiner:
Prof. Dr. Sven Behnke

Second Examiner:
Prof. Dr. Maren Bennewitz

Date: December 31, 2022

Declaration

Abstract

Dexterous manipulation of everyday objects has become a key interest to allow robots to seamlessly integrate in human-centred environments. As the objects in question are shaped for human hands, anthropomorphic end-effectors are a versatile and effective interface, but they come with a high dimensional control space which hinders an agent's ability to learn robust manipulation strategies. Learning from human demonstrations can aid learning in such complex search spaces.

Offline reinforcement learning (RL) is a data-driven form of RL able to take advantage of large and diverse datasets. Prior work has observed a performance gap of offline RL algorithms when applied on human demonstrations in contrast to machine-generated demonstrations. In this thesis, we investigate whether augmenting state-of-the-art offline RL algorithm CQL with history-aware components can help to close this gap. First, we develop an immersive Virtual Reality (VR) teleoperation system enabling intuitive and fast collection of human demonstrations. We then propose three variants of CQL and evaluate them on our demonstrations. We provide an extensive analysis of the benefits and shortcomings of all considered variants, and find that simple changes to the agent input, such as concatenating subsequent observed states, can improve performance in complex, high precision tasks.

Contents

1. Introduction

As robot capabilities have grown beyond the structured and well-defined application space of industrial settings, dexterous manipulation of everyday objects has become a key interest to allow robots to seamlessly integrate in human-centred environments. However, robot manipulation skills are still far from matching those of humans [1]. Equipping robots with anthropomorphic end-effectors is an intuitive and practical choice, since the objects in question are shaped for human hands. These end-effectors come with a high-dimensional action space which makes it difficult to learn robust manipulation strategies, especially for tasks requiring high precision.

Reinforcement learning is an online interactive learning paradigm where an agent receives information about the current environment state and is rewarded for actions that lead to successful task completion. During training, the agent aims to find actions which will yield high rewards, resulting in a trial-and-error approach to solving the task at hand. Given both enough time to explore the search space and meaningful feedback, RL can find a successful and robust strategy to solve virtually any well-defined task. Indeed, this method has shown great results across different domains [2–5], but its online learning character has several drawbacks when compared to data-driven methods. As the agent starts out with a random policy, it will most likely achieve none or only low rewards initially. To improve the strategy, high reward regions need to be found. For large state-action spaces, it can take many iterations for the agent to encounter such high reward regions. Collecting such large amounts of online samples is not only expensive, but those initial interactions can also be unsafe, especially when they need to be carried out in the real world. For example, taking an action with a high force in a delicate state can lead to damage in the robot or its environment.

Manually designed dense reward functions can help guide learning towards success, therefore effectively decreasing the number of samples required. In the case of an agent that should learn to pick up an object, an increasing reward could be provided for smaller distance between the robot end-effector and the object. This reward shaping can be non-trivial for complex tasks and may even impede the agent: In some settings, it might not be obvious for the person designing the reward function what strategy is the most effective or robust for the agent at hand. For the specific case of dexterous manipulation with an anthropomorphic end-effector, online RL can additionally lead to undesired strategies like opening a door with the wrist instead of using the fingers [6]. While effective, this can be alienating to humans potentially interacting with the robot.

Of course, an agent learning to utilize an anthropomorphic end-effector for dexterous manipulation tasks would ideally not start from scratch. Humans already know how to manipulate those objects with their hands, and we should try to effectively leverage their expertise to teach robots the same. Imitation learning [7] can successfully mimic demonstrations, but performance is thereby limited by the demonstrated behaviour. Alternatively, an RL agent can be initialized with a policy learned with imitation learning. This form of bootstrapping the RL algorithm significantly decreases the sample complexity of standard RL and produces policies with high generalization capabilities [6]. However, these methods also show a so-called unlearning effect at the start of online interaction [8, 9], suggesting that demonstrations are not ideally used during the entire learning process. In this work, we instead employ offline reinforcement learning, a data-driven form of RL which requires no online interaction and instead trains only on pre-recorded demonstrations. In comparison to online RL, this method is much more sample-efficient as the demonstrations provide a good starting point of how higher rewards can be achieved. Moreover, in contrast to imitation learning, this method does not limit the agent performance to that of the demonstrator. Instead, offline RL enables the use of more extensive and diverse datasets, including suboptimal demonstrations or data from other tasks, allowing the agent to effectively generalize, identify particularly good behaviours in the dataset and even recombine demonstrated behaviour. For example, a dataset containing demonstrations of lifting an object out of an open drawer could be increased by few unlabelled (i.e. zero rewards) demonstrations of opening said drawer. The added demonstrations enable the agent to learn how to reach the initial state of the demonstrations showing the lifting motions, allowing for a combination of the two strategies [10].

Mandlekar et al. [11] observe that offline RL algorithms shows better performance when trained on synthetic demonstrations generated by a policy trained with RL than when trained on human demonstrations. While the former are inherently Markovian, actions demonstrated by humans may depend on other factors apart from the current state, such as past observations. The authors observe that imitation learning methods with recurrent components are more successful in learning from human data. They hypothesize that history-aware components might aid offline RL to effectively leverage such datasets as well. To our knowledge, the impact of combining history-aware components with offline RL has not been investigated, and this thesis aims to fill that gap.

The contributions of this work are twofold: First, we present an immersive VR teleoperation framework with haptic feedback to collect demonstrations for dexterous manipulation tasks from human operators. Secondly, we propose three approaches to augment an offline RL algorithm with history-awareness and investigate the performance of these methods on human datasets. We find that some forms of history-awareness can aid performance when learning complex manipulation tasks with high precision requirements from human demonstrations.

2. Preliminaries

In this chapter, we first describe the tasks considered for experimental evaluation and then discuss the theoretical concepts applied in this work.

2.1. Task Overview

The majority of the experiments presented in this thesis are carried out on tasks from gym-grasp [12], a framework for robot grasping and dexterous hand manipulation powered by IsaacGym with tasks representing daily life. We also carry out some experiments on datasets from robomimic [11] which utilize tasks from robosuite [13], a simulation framework for robot learning powered by MuJoCo. In the following, we describe all tasks considered.

2.1.1. Gym-grasp tasks

All environments in the gym-grasp framework feature a robotic arm and hand, specifically a UR5 arm and a Schunk SIH hand. The environment receives relative actions consisting of a 6-dimensional wrist pose and 5 finger positions. Figure 2.1 shows how the arm and hand are actuated. The initial position of the robot arm and hand is randomized within a small cube at the start of each episode.

In our experiments, we utilize low-dimensional observations which can be configured to hold combinations of pose and velocity information of the robot wrist and fingers, finger contacts observations and pose information of task-specific objects. Additionally, gym-grasp offers visual observations which are not considered here.

Both sparse and dense rewards are available, but only the former are considered in this work. The sparse reward given is 0 once the task is completed successfully and -1 otherwise.

All considered tasks are depicted in figure 2.2, and we give short descriptions of each in the following.

OpenDrawer The task starts with a closed drawer and is solved when the agent has opened the drawer by 20 cm. As the box with the drawer is always initialized in the same spot, task-specific observations only include the x-dimension of the

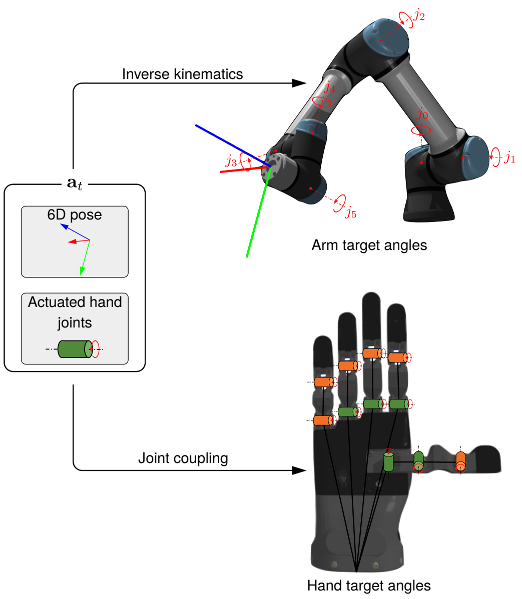
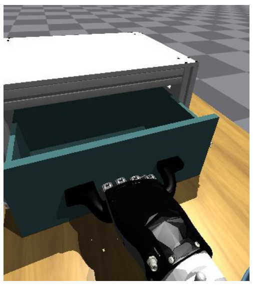
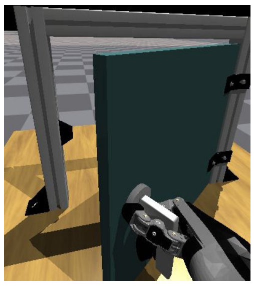
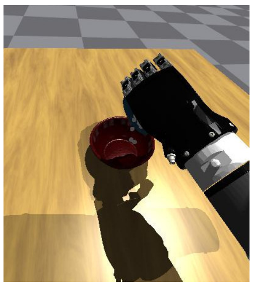


Figure 2.1: SIH actuation. Only the joints marked green can be directly actuated. Figure from [14].


(a) OpenDrawer.


(b) OpenDoor.


(c) PourCup.

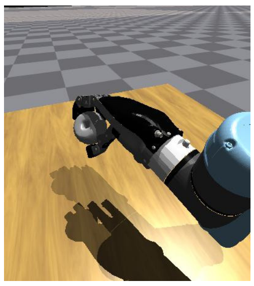

(d) LiftObject.

Figure 2.2: Gym-Grasp Tasks.

handle position. This is the simplest task considered, as it can be completed in one motion and requires little precision.

OpenDoor The door is always initialized at the same position and closed in the beginning. The task is solved when the agent has turned the handle and pulled it open by 45°. This requires the concatenation of two more difficult motions, making the task more complex than OpenDrawer. The current pose of the handle is tracked as a task-specific observation.

PourCup This task consists of an empty bowl and a cup filled with particles representing liquid. Both containers are always initialized at the same positions, and the goal is to pour the particles from the cup into the bowl. The task is solved when at least 90% of the particles are in the bowl. Dropping the full cup into the bowl is not considered a success. This task has a high risk of failure, as the agent cannot recover if too many particles are spilled. Both the grasping and pouring motion require high precision. Task-specific observations comprise the pose of both containers.

LiftObject In this task, the agent must lift a lemon lying on the table to a height of 10 cm. In contrast to the other tasks, both the object pose and the agent's hand

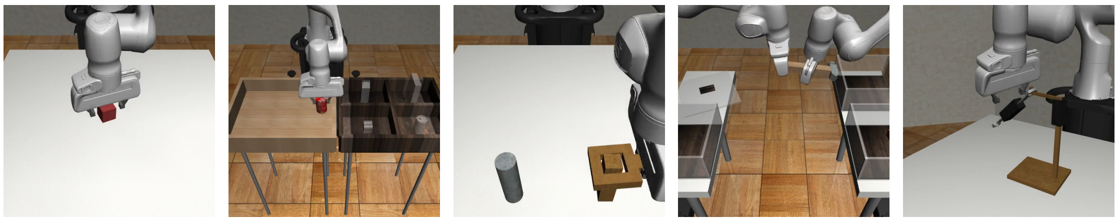

(a) Lift. (b) Can. (c) Square. (d) Transport. (e) Tool Hang.

Figure 2.3: Robomimic Tasks. Figures from [11].

position are randomly initialized as the object is dropped from a random position onto the table. Solving the task requires precise positioning of the fingers for a stable grip as well as generalization capabilities. The current object pose is available as a task-specific observation.

2.1.2. Robomimic datasets

The datasets from robomimic [11] that were considered for this work were recorded in environments with one or two Panda robotic arms. One arm is controlled using 7-dimensional actions consisting of 3 coordinates respectively for the translation and rotation from the end-effector pose and one coordinate controlling the opening and closing of the gripper fingers.

For this work, we only consider low-dimensional observations. These consist of proprioception observations, i.e. the end-effector pose and gripper finger position, as well as object observations. Both sparse and dense rewards are available. The datasets we utilize have binary sparse rewards of 1 once the task is completed and 0 otherwise.

All considered tasks are depicted in figure 2.3, and we give short descriptions of each in the following.

Lift In this task, the agent must lift a small cube. The cube position is randomized within a small square on the table, as is its rotation in the z-axis. Object observations consist of the absolute cube pose and the cube position relative to the end-effector. This is the simplest task from the robomimic datasets.

Can This task comprises a large bin containing a can and a smaller empty bin. The agent must lift the can and place it into the smaller bin. The can position is randomized within the large bin, as well as its z-rotation. Object observations consist of the absolute can pose and its relative position to the end-effector. Solving this task is slightly more difficult than Lift, as it requires the concatenation of two motions, picking and placing, and picking the can is harder than picking the cube.

Square The agent must pick a square nut and place it on a rod. The nut position as well as its z-rotation are randomized in a square region on the table. Object observations consist of the absolute nut pose and the nut pose relative to the robot end-effector. Solving this task requires higher precision than Lift and Can.

Transport This task comprises two robot arms. One must pick a hammer from a closed container on a shelf while the other clears a target bin on another shelf by removing a piece of rubbish. Then, the first arm hands the hammer to the second one which places the hammer in the target bin. The positions of all bins and the lid are randomized within small squares. Additionally, the positions and z-rotations of the hammer and piece of rubbish are randomized. Object observations consist of the absolute poses of the hammer, piece of rubbish, and lid handle, the absolute position of the target and rubbish bin, the relative positions of the hammer and lid handle to the first arm, the relative positions of the hammer and piece of rubbish to the second arm, and binary indicators for the hammer reaching the target bin and the rubbish reaching the rubbish bin. As this task consists of multiple stages, it takes longer than the others and is accordingly difficult to solve with binary task rewards.

Tool Hang The agent must insert a hook into a base and hang a wrench on the hook. The positions and z-rotations of the hook and wrench are randomized within a small square. Object observations consist of the absolute and relative pose of the base frame, the insertion hook and the wrench to the end-effector, and binary indicators for assembly of the base and hook and placement of the wrench on the hook. This task is the most difficult one in this set, as it requires multiple high precision movements.

2.2. Theoretical Background

In this section, we first define the reinforcement learning (RL) problem following Sutton and Barto [15]. Then, we highlight the opportunities and challenges in offline reinforcement learning and outline different approaches to offline RL. Finally, we detail the formalism behind CQL [16] as the main algorithm used in this thesis. For a more in-depth analysis of offline RL methods, please refer to recent surveys by Levine et al. [9] and Prudencio et al. [17].

2.2.1. Reinforcement Learning

Reinforcement learning aims at learning to control a dynamical system through interaction. The dynamical system is modelled as a Markov decision process (MDP):

Definition 2.1 (Markov decision process)**.** A Markov decision process is defined by a tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, T, d_0, r, \gamma)$, where

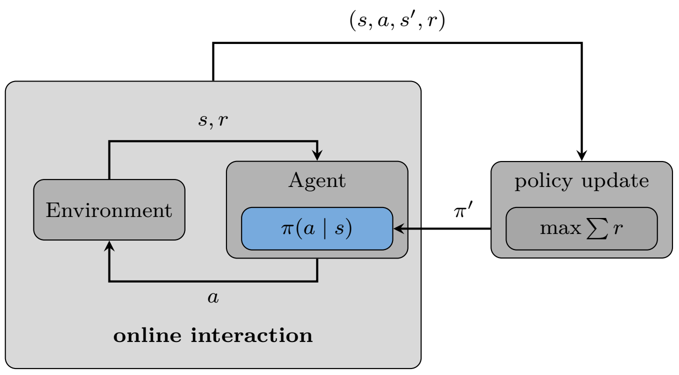
- $\mathcal{S}$ is the set of states,
- $\mathcal{A}$ is the set of actions,
- $T\left(s_{t+1}|s_t, a_t\right)$ describes the system dynamics in terms of the probability of transitioning into state $s_{t+1} \in \mathcal{S}$ when executing action $a_t \in \mathcal{A}$ in state $s_t \in \mathcal{S}$,
- $d_0(s_0)$ defines the initial state distribution,
- $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ defines a reward function and
- $\gamma \in [0, 1]$ is a scalar discount factor.

It is a stochastic process that satisfies the Markov property, meaning that there is no dependency on previous states as the current state contains all necessary information.

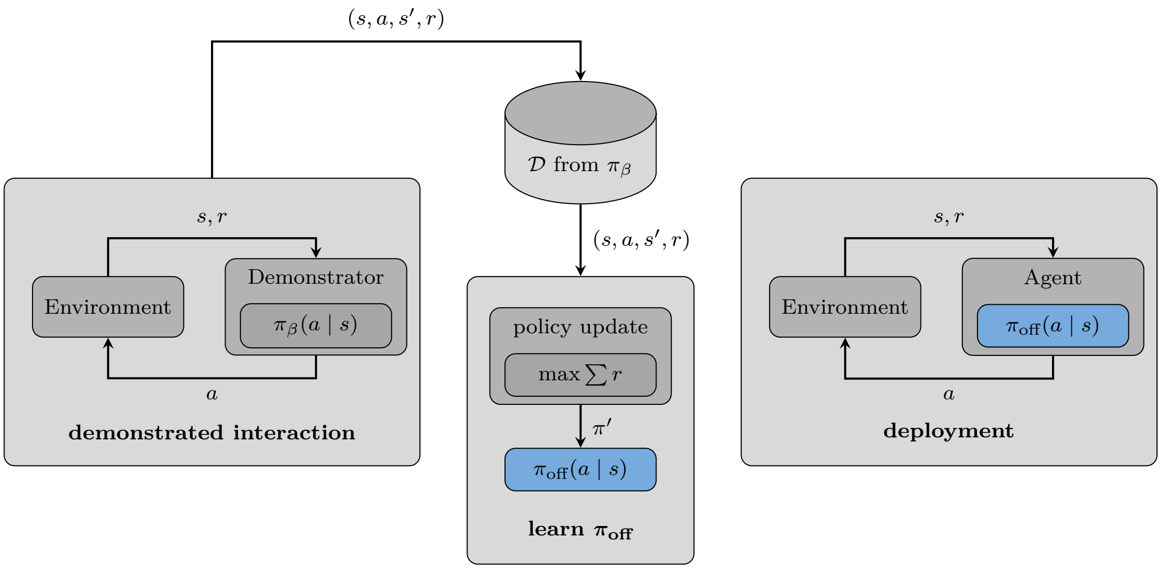
A behaviour policy $\pi\left(a_t|s_t\right)$ is a distribution of actions $a_t \in \mathcal{A}$ conditioned on the current state $s_t \in \mathcal{S}$, inducing a marginal state-action distribution $\rho_\pi(s_t, a_t)$. The goal of reinforcement learning is then to learn a policy which maximizes the expected sum of rewards discounted over time, expressed in the following learning objective:

$$J(\pi) = \mathbb{E}_{(s_t,a_t)\sim\rho_\pi(\tau)}\left[\sum_t \gamma^t r(s_t, a_t)\right] \tag{2.1}$$

Reinforcement learning algorithms begin with an agent carrying out an initial, usually random policy in an environment governed by an MDP as defined above. The agent-environment interaction is depicted in figure 2.4a: At each time step t, the agent receives the current state $s_t \in \mathcal{S}$ of the environment and selects an action $a_t \in \mathcal{A}$ based on the policy π. It then receives the environment's next state $s_{t+1} \in \mathcal{S}$ and a reward signal $r_t = r(s_t, a_t) \in \mathbb{R}$ judging the usefulness of the selected action in this state. Based on these interaction tuples (s_t, a_t, s_{t+1}, r_t), reinforcement learning algorithms employ policy updates to maximize the learning objective.



(a) Online RL Framework. The agent policy is iteratively updated with the objective to maximize the expected sum of rewards with respect to the current interaction.



(b) Offline RL Framework. Samples of demonstrated behaviour π_β are collected into dataset $\mathcal{D}$, then the agent policy π_{off} is trained from $\mathcal{D}$ without additional interaction, with the same objective as in online RL. After training, π_{off} is deployed.

Figure 2.4: Reinforcement Learning framework

Policy gradient methods directly estimate the gradient of the policy and update the policy accordingly. On the other hand, dynamic programming methods maximize the objective by learning value functions that estimate the expected discounted sum of rewards when starting in state s_t and following some policy $\pi\left(a_t|s_t\right)$:

$$\begin{aligned} V^\pi(s_t) &= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty}\gamma^k r(s_{t+k},a_{t+k})\,\middle|\, s_t\right] \\ Q^\pi(s_t,a_t) &= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty}\gamma^k r(s_{t+k},a_{t+k})\,\middle|\, s_t,a_t\right] \end{aligned}$$

where $V^\pi(s_t)$ is the value of state s_t and $Q^\pi(s_t,a_t)$ the value of taking action a_t in state s_t. It follows that

$$V^\pi(s_t) = \mathbb{E}_{a_t\sim\pi(a_t|s_t)}\left[Q^\pi(s_t,a_t)\right]$$

and thus

$$\begin{aligned} Q^\pi(s_t,a_t) &= r(s_t,a_t) + \gamma\mathbb{E}_{s_{t+1}\sim T(s_{t+1}|s_t,a_t)}\left[V^\pi(s_{t+1})\right] \\ &= r(s_t,a_t) + \gamma\mathbb{E}_{s_{t+1}\sim T(s_{t+1}|s_t,a_t),a_{t+1}\sim\pi(a_{t+1}|s_{t+1})}\left[Q^\pi(s_{t+1},a_{t+1})\right] \end{aligned} \tag{2.2}$$

Choosing the action with maximum value will result in an optimal policy. The Q-function update in equation 2.2 is often expressed in terms of the Bellman operator $\mathcal{B}^\pi$, resulting in $\vec{Q}^\pi \leftarrow \mathcal{B}^\pi\vec{Q}^\pi$ with $\vec{Q}$ referring to the representation of Q as a vector of length $|\mathcal{S}|\times|\mathcal{A}|$.

To learn value functions, we can further distinguish between Q-learning methods and actor-critic methods. Q-learning methods implicitly define a greedy policy $\pi\left(a_t|s_t\right) = \arg\max_{a_t} Q(s_t,a_t)$ and learn the corresponding optimal Q-function:

$$Q^*(s_t,a_t) = r(s_t,a_t) + \gamma\mathbb{E}_{s_{t+1}\sim T(s_{t+1}|s_t,a_t)}\left[\max_{a_{t+1}} Q^*(s_{t+1},a_{t+1})\right]$$

This can be again expressed as $\vec{Q} = \mathcal{B}^*\vec{Q}$ in terms of the Bellman optimality operator $\mathcal{B}^*$. Actor-critic methods, on the other hand, separately train a parameterized policy π_θ (the actor) and the corresponding parameterized value function Q^π_ϕ (the critic). First, the Q-function is updated with respect to the current policy:

$$\begin{aligned} & Q^\pi_\phi(s_t,a_t) = r(s_t,a_t) + \gamma\mathbb{E}_{s_{t+1}\sim T(s_{t+1}|s_t,a_t),a_{t+1}\sim\pi_\theta(a_{t+1}|s_{t+1})}\left[Q^\pi_\phi(s_{t+1},a_{t+1})\right] \\ \Leftrightarrow\quad & \vec{Q}^\pi_\phi = \mathcal{B}^\pi_\theta\vec{Q}^\pi_\phi \end{aligned}$$

Then, the policy is updated to select actions with a high value according to the updated Q-function.

2.2.2. Offline Reinforcement Learning

Offline reinforcement learning is a data-driven approach to RL, optimizing the same objective from equation 2.1 but without online interaction (as depicted in figure 2.4b). Instead, these algorithms leverage a static dataset $\mathcal{D} = \{(s_t, a_t, s_{t+1}, r_t)_i\}$ of previously collected transitions. $\mathcal{D}$ is produced by an unknown behaviour policy π_β which induces the marginal state-distribution $d^{\pi_\beta}(s)$, meaning that the state-action tuples $(s, a) \in \mathcal{D}$ are sampled according to $d^{\pi_\beta}(s)\pi_\beta(a|s)$.

The main challenge in offline reinforcement learning is the distributional shift. Recall equation 2.2: the target value $Q^\pi(s_t, a_t)$ is only updated during training for in-distribution actions $a_t \sim \pi_\beta(a_t|s_t)$, but this update uses the estimated value of actions $a_{t+1} \sim \pi(a_{t+1}|s_{t+1})$. For these out-of-distribution (OOD) actions a_{t+1}, the value of $Q(a_{t+1}, s_{t+1})$ could be erroneously high. While such errors would be corrected through interaction in online reinforcement learning, they instead accumulate in the offline setting. Policies trained to maximize the Q-value might be biased towards those OOD actions.

Possible mitigations for this issue include policy constraints and uncertainty estimation. The latter method aims to determine the trust-worthiness of the Q-value prediction. When the uncertainty of the estimate is high, as is expected to be the case for OOD actions, conservative target values are produced. In practice, this method does not perform well as the demands on the fidelity of the uncertainty estimates are immense [9].

Policy constraint methods limit how far the learned policy π deviates from the behaviour policy π_β. The constraint usually looks at the distance between the two policies and therefore requires explicit modelling of π_β. Learning is thus limited by the capability to model the behaviour policy which can be difficult in complex scenarios. In practice, these methods tend to be overly conservative [9].

2.2.3. Conservative Q-Learning

Kumar et al. [16] propose to address the distributional shift by learning a conservative Q-function that lower-bounds the true Q-function. In the following, we derive their method.

In standard Q-function training, we aim to minimize the error $\varepsilon(\mathcal{D}, Q^k)$ with respect to the Bellman objective:

$$Q^{k+1} \leftarrow \arg\min_Q \underbrace{\frac{1}{2}\mathbb{E}_{s,a,s'\sim\mathcal{D}}\left[\left(Q(s,a) - \mathcal{B}^{\pi_k}Q^k(s,a)\right)^2\right]}_{\varepsilon(\mathcal{D},Q^k)}$$

To learn a conservative Q-function, the authors propose to additionally minimize the Q-values under the distribution $\mu\left(a|s\right)$, therefore penalizing high Q-values for actions drawn according to μ:

$$Q^{k+1} \leftarrow \arg\min_Q \alpha\, \mathbb{E}_{s\sim\mathcal{D},a\sim\mu(a|s)}\left[Q(s,a)\right] + \varepsilon(\mathcal{D},Q^k)$$

They show that the resulting function is indeed a point-wise lower bound of the true Q-function Q^π at all $s \in \mathcal{D}, a \in \mathcal{A}$. As this objective would be far too conservative, a maximization term is added to tighten the bound:

$$Q^{k+1} \leftarrow \arg\min_Q \alpha\, \underbrace{\left(\mathbb{E}_{s\sim\mathcal{D},a\sim\mu(a|s)}\left[Q(s,a)\right] - \mathbb{E}_{s\sim\mathcal{D},a\sim\pi_\beta(a|s)}\left[Q(s,a)\right]\right)}_{\mathcal{C}_{CQL}(\mathcal{D},Q^k)} + \varepsilon(\mathcal{D},Q^k)$$

While the result is not a point-wise lower bound as before, it can be shown to be a lower bound in expectation under the current policy. The intuition behind this objective is that for all states in $\mathcal{D}$, high Q-values for actions drawn according to μ will be penalized, while the Q-values for in-distribution actions $a \sim \pi_\beta$ will be penalized less because of the Bellman backup. For an adversarially chosen μ, this impedes overestimation of Q-values of OOD actions, while still encouraging improvement over the observed behaviour.

Regarding the choice of μ, Kumar et al. propose a family of optimization problems over $\mu\left(a|s\right)$ characterized by a regularized adversarial $\mathcal{R}(\mu)$:

$$\min_Q \max_\mu \alpha\, \mathcal{C}_{CQL}(\mathcal{D},Q^k) + \varepsilon(\mathcal{D},Q^k) + \mathcal{R}(\mu) \tag{2.3}$$

An interesting instance of this family is given by setting $\mathcal{R}(\mu) = -D_{KL}(\mu,\rho)$, i.e. the KL divergence against a prior distribution $\rho\left(a|s\right)$. The authors note that for the maximization problem

$$\max_\mu \mathbb{E}_{x\sim\mu(x)}\left[f(x)\right] + D_{KL}(\mu,\rho) \text{ s.t. } \sum_x \mu(x) = 1, \forall x \mu(x) \geq 0$$

the optimal solution is $\mu^*(x) = \frac{1}{Z}\rho(x)\exp\left(f(x)\right)$, where Z is a normalization factor.

We can rewrite equation 2.3 as

$$\min_Q \max_\mu \alpha\, \mathbb{E}_{s\sim\mathcal{D}} \left(\mathbb{E}_{a\sim\mu(a|s)}\left[Q(s,a)\right] - \mathbb{E}_{a\sim\pi_\beta(a|s)}\left[Q(s,a)\right]\right) + \varepsilon(\mathcal{D},Q^k) - D_{KL}(\mu,\rho) \tag{2.4}$$

and focus on the maximization problem regarding μ marked in red which yields $\mu^*(a|s) = \frac{1}{Z}\rho(a|s)\exp(Q(s,a))$. Plugging this into 2.4 yields a variant of CQL denoted by CQL(ρ):

$$\min_Q \alpha\, \mathbb{E}_{s\sim\mathcal{D}} \left(\mathbb{E}_{a\sim\rho(a|s)}\left[Q(s,a)\frac{\exp(Q(s,a))}{Z}\right] - \mathbb{E}_{a\sim\pi_\beta(a|s)}\left[Q(s,a)\right]\right) + \varepsilon(\mathcal{D},Q^k) \tag{2.5}$$

We can choose $\rho = \mathrm{Unif}(a)$ and note that

$$\begin{aligned} D_{KL}(\mu,\mathrm{Unif}) &= \mathbb{E}_{x\sim\mu}\left[\log(\mu(x)) - \log(\mathrm{Unif}(x))\right] \\ &= \mathbb{E}_{x\sim\mu}\left[\log(\mu(x))\right] - \mathbb{E}_{x\sim\mu}\Big[\underbrace{\mathrm{Unif}(x)}\Big]_{\mathrm{const}} \\ &= \underbrace{\mathbb{E}_{x\sim\mu}\left[\log(\mu(x))\right]}_{-\mathcal{H}(\mu)} - \mathrm{const} \\ &\approx -\mathcal{H}(\mu) \end{aligned}$$

which is why this variant is then denoted as CQL($\mathcal{H}$). We can then further simplify equation 2.5:

$$\min_Q \alpha\, \mathbb{E}_{s\sim\mathcal{D}} \left(\log\sum_a \exp(Q(s,a)) - \mathbb{E}_{a\sim\pi_\beta(a|s)}\left[Q(s,a)\right]\right) + \varepsilon(\mathcal{D},Q^k) \tag{2.6}$$

The first term marked in red has to be estimated via importance sampling. The authors propose another variant called CQL(ρ), where $\rho(a|s) = \pi^{k-1}(a|s)$. This variant does not require sampling. The authors find that while CQL($\mathcal{H}$) outperforms CQL(ρ) in most experiments, CQL(ρ) can perform better in higher-dimensional action spaces such as in the Adroit tasks [6] utilizing a 24-DoF robotic hand, where it is difficult to estimate $\log\sum_a \exp$ using importance sampling due to high variance.

In this thesis, we only consider the variant CQL($\mathcal{H}$) as our action space is only 11-dimensional and this variant was also used in the experiments we consider for comparison [11].

3. Related Work

In the following, we summarize recent literature on the different aspects considered in this work.

3.1. Considerations for learning from human demonstrations

Algorithms utilizing demonstrations are commonly evaluated on machine-generated data as such demonstrations are often easy to obtain in a lab setting. However, a real-world application of such algorithms will likely involve human demonstrations, especially in application domains like humanoid robotics. Recent works have indicated that machine-generated demonstrations are not a good substitute for human ones.

In their study [11], Mandlekar et al. explore the challenges when learning from human demonstrations in the context of robot manipulation. They consider tasks of different complexity for a robot with a gripper as end-effector and collect both human and machine-generated demonstrations. Their evaluation examines imitation learning and offline reinforcement learning algorithms, and we leverage their open sourced framework, robomimic, for this work. Among other lessons, they find that state-of-the-art offline RL algorithms CQL [16] and BCQ [18] struggle to learn from human demonstrations and show much better performance on machine-generated data. On the other hand, imitation learning methods that include history-dependent models like RNNs are more successful. While machine-generated datasets are Markovian by design, human actions might depend on more than the current observation alone. The authors hypothesize that in this non-Markovian setting, history-dependent models are better suited to capture the decision process.

In a similarly extensive study, Orsini et al. [19] investigate the impact of design choices specifically in the adversarial imitation learning (AIL) setting. AIL takes inspiration from Generative Adversarial Networks (GANs) and Inverse RL and trains a policy to generate trajectories that are indistinguishable from the teacher demonstration. Interestingly, they also find performance differences between human and machine-generated demonstration, and note that the impact of different design decisions differs between data sources. In their experiments, learning from human demonstrations shows an increased need for regularization and benefits from a reward function that minimizes the Jensen-Shannon divergence instead of the KL divergence. The KL divergence penalizes the policy more for visiting states the teacher has not visited than for not visiting states the teacher has visited. On the other hand, the Jensen-Shannon divergence is symmetric and bounded, which the authors hypothesize fits the human data better as it might not be possible to exactly replicate the demonstrations due to circumstantial factors. That said, the objective in AIL is inherently different from the objective of offline RL which is considered in this work, as AIL aims to match the policy behaviour to the demonstrated behaviour, whereas offline RL tries to improve on it. The specific findings of Orsini et al. are thus likely not transferable to the offline RL setting, but show that investigating algorithmic choices for learning from different data sources is of paramount importance.

3.2. Human demonstrations for dexterous manipulation

Rajeswaran et al. [6] propose a suite of manipulation tasks which comprises complex dexterous manipulation tasks such as in-hand rotation of a pen, opening a door, moving a ball to a target position and hammering a nail into a board. The action space is much higher-dimensional than in our work, as they consider a 24-DoF robotic hand whereas we utilize coupled finger control resulting in an 11-dimensional action space. They collect a dataset of 25 successful human demonstrations leveraging an updated version of the MuJoCo HAPTIX [20] framework described below. Their tasks and datasets have become part of the D4RL [21] benchmark under the name *Adroit*. Furthermore, they propose and evaluate Demo Augmented Policy Gradient (DAPG), an online RL method based on Natural Policy Gradient (NPG) [22]. First, they train a BC policy from the demonstrations and use it to initialize RL training. In contrast to previous works that bootstrap RL with demonstrations, they further add a BC loss to NPG's gradient which is weighted according to a heuristic weighting scheme. This motivates the policy to stay close to the demonstrated behaviour, especially during the beginning of training. Their method significantly outperforms all baselines, especially when using sparse task completion rewards. Moreover, it shows a higher robustness to variations in the mass and size of the considered object, suggesting that the strategies learned from human behaviour have better generalization capabilities. DAPG has also been successfully applied to dexterous manipulation tasks in the real world using kinaesthetic teaching [23].

The Adroit dataset has been further investigated by Nair et al. [24] who propose their algorithm Advantage Weighted Actor Critic (AWAC). AWAC is designed to leverage prior datasets from diverse sources, such as data for the task at hand, suboptimal data, or data from related tasks, and improve policies pre-trained on these datasets: They employ off-policy temporal difference (TD) learning but use a supervised learning style update for the actor. The actor is implicitly constrained to mitigate distribution shift. For evaluation on the Adroit dataset, they combine the human demonstrations with samples from a BC policy to increase the number of available samples. In this setting, AWAC fine-tunes faster than DAPG and outperforms all baselines.

A more recent approach to offline RL are so-called one-step methods [25, 26]. These methods avoid the distribution shift challenge by performing only a single step of policy evaluation, i.e. critic update, followed by a single step of policy improvement, i.e. updating the actor. In this way, the critic is never queried on out-of-distribution actions. These methods show slightly higher success on the Adroit dataset than the considered baseline.

Mandikal and Grauman [27] take a different approach to human demonstrations. Instead of recording trajectories, they leverage a large dataset of video frames from how-to videos on YouTube showing human interactions with different objects. First, they use a state-of-the-art computer vision method to extract human hand poses from the frames and apply inverse kinematics to transfer the pose to their 30-DoF simulated robot hand. They then augment a grasp success reward function with a term accounting for affordance regions predicted from an image-based model and a term encouraging the agent to match the pose extracted from the video data. After training with deep online RL in simulation, they find that their model outperforms DAPG, purely affordance based models and other baselines with respect to success, stability and functionality of the grasp.

3.3. Dexterous Teleoperation

We can separate related work in hand teleoperation into roughly three categories based on how finger movements are tracked: motion capture, vision-based and glove-based.

In the field of motion capture, a human operator wears motion capture markers on the fingers which are tracked by an array of cameras. The labelling of those markers proves a challenge as the fingers they are attached to are self-similar and movements can easily lead to occlusion. Han et al. [28] are very successful in solving this problem by using convolutional neural networks to process the camera images, resulting in a precise and reliable tracking framework that can successfully be used for teleoperation. However, many cameras are required for motion capture approaches, making them expensive and essentially restricting them to lab setups.

Hence, more recent work has mostly focused on vision-based teleoperation, using fewer cameras and no motion capture markers. Handa et al. [29] propose their teleoperation framework DexPilot which consists of a human pilot arena directly adjacent to a robot system. The operator can directly observe the operating space of the robot, and their hand movements are tracked by four depth cameras positioned around the pilot space. Deep neural networks are pre-trained in this setup to process the camera images and estimate the current hand state to improve tracking provided by DART [30], a model-based tracker. Online kinematic retargeting is applied to the tracked motion to be suitable to the robot hand. The system enables the operator to successfully carry out a range of tasks like inserting cups and extracting money from a wallet. Qin, Su, and Wang [31] further reduce the number of cameras required to a single one, specifically an iPad video stream. First, they construct a customized simulated hand for each individual operator. This makes costly online retargeting obsolete, as the operator's movements can be directly mapped to movements in the simulated environment. Afterwards, the collected trajectories are processed and converted to trajectories for different existing robot hands, allowing for reusable trajectories while saving resources during recording. The authors then train DAPG [6] on their converted trajectories and show good results. While these approaches have a much lower cost compared to motion tracking, they are less capable when dealing with occlusion and require careful pre-training or tuning. Moreover, using fewer cameras limits the tracking area and thus the available workspace.

Glove-based approaches forgo cameras and instead track the finger movement using gloves equipped with sensors that the operator wears, like the SenseGlove we utilize in this work. This allows for precise real-time movement tracking without the need of post-processing for state estimation. Moreover, the setup is flexible and easy to reproduce. On the downside, the gloves come with more bulk attached to the user, like the glove itself, add-ons like buzzers, and cables. Kumar et al. [20] propose MuJoCo HAPTIX, a framework based on the simulation engine MuJoCo. The operator is equipped with a CyberGlove [32] to track wrist and finger movements, a 3D-printed wristband for tracking the hand base using infrared motion capture and motion capture markers on glasses to track the head movements. The tracked hand movements are applied to a hand in simulation, and the simulation is shown to the operator using stereoscopic visualization. Rather than an immersive first-person view like in our system, this creates the impression of looking through a window. The authors evaluate the usability of their system using tasks from the Southampton Hand Assessment Procedure [33], a clinically validated test of hand function developed to test the functionality of hand prostheses. They find that users are able to successfully complete the selected tests in a reasonable amount of time using their framework. Rajeswaran et al. [6] used an updated version of the MuJoCo HAPTIX framework to record the demonstrations for their dexterous manipulation dataset Adroit, substituting the motion capture markers with a VR headset and a Vive tracker. Our framework is similar to this updated version in terms of the utilized hardware, but we explicitly focus on creating an *immersive* experience, with a first-person view that exactly matches the operator's view angle and haptic feedback mimicking forces of contact.

4. Dexterous Teleoperation Framework

To collect human demonstrations for dexterous manipulation tasks from gym-grasp, we developed an immersive teleoperation framework. In contrast to recent works in dexterous teleoperation, which largely propose vision-based approaches (see section 3.3), we opt for a combination of Virtual Reality (VR) and glove-based teleoperation. This allows us to incorporate intuitive haptic feedback, and the reliability of VR tracking provides the operator with a large workspace. The human operator is equipped with a Vive VR headset, a SenseGlove to track finger movements and provide haptic feedback, and a Vive Tracker to determine the wrist pose. In the headset, the operator sees an immersive view of the IsaacGym environment defined by a gym-grasp task where they can move their head and look around. The reference pose for the head is positioned such that the distance to the robot hand is realistic, as if it were the operator's own hand. We employ a clutch-like mechanism where the operator is able to freely move their hand without controlling the simulated robot hand until pressing a key combination. This allows the operator to position their hand in a way that feels compliant with the simulation. When the keys are pressed, the current pose is stored as the reference pose. From then, the movements of the operator's hand are translated into control of the robot hand, and they remain locked until the end of the episode. The starting pose of the robot hand is set to a random position for each episode, which makes it necessary to reset the reference pose after each demonstration. Otherwise, pose tracking would be shortly interrupted during loading of the new episode, and subsequent pose errors would accumulate over multiple demonstrations.

During the episode, we collect the observations, reward and done signals provided by the simulation, as well as the operator's actions. At the end of each episode, the data is written to an HDF5 file in a structure compatible with the robomimic framework [11] which we use for training.

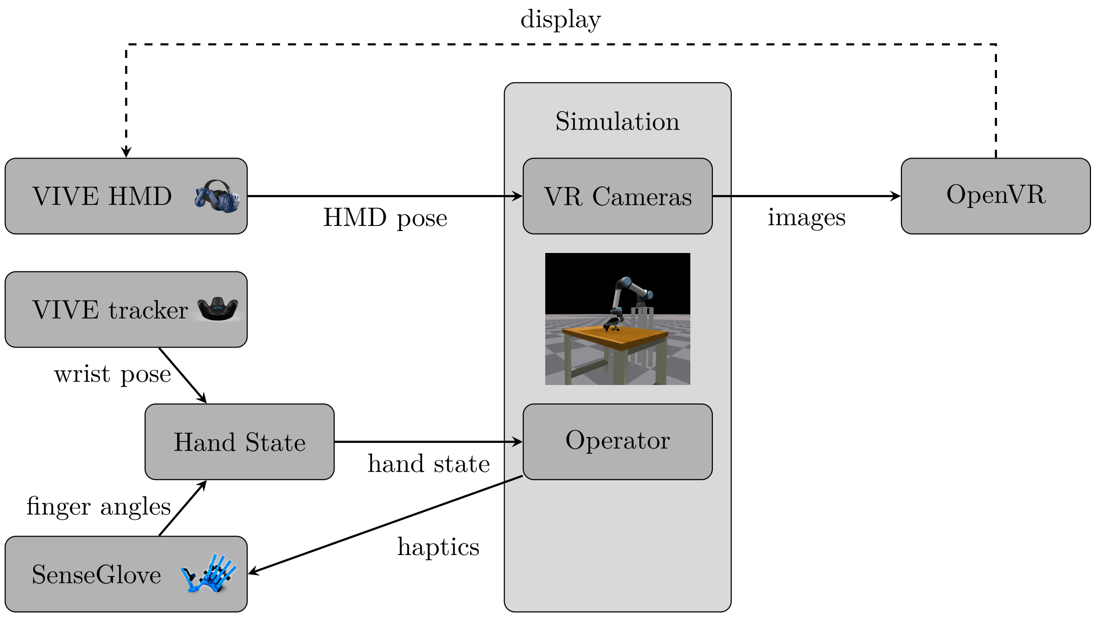


Figure 4.1: Teleoperation framework.

4.1. System Overview

The logical components and the corresponding hardware components are shown in figure 4.1. All logical components communicate via the ROS framework [34] with a frequency of around 90 messages per second.

To enable the operator to view the simulation for gym-grasp tasks, we add two dedicated cameras into the simulation that act as the operator's eyes. The pose of these cameras is updated by applying the tracked headset pose which is transmitted with a frequency of 100 Hz. Concurrently, the sensor information from the SenseGlove and Vive Tracker is combined into a hand state consisting of the 6-dimensional wrist pose, the normalized flexion for thumb, index, middle and ring finger, as well as the rotation of the thumb. This information is used as relative action input with coupled finger control for the simulation. The haptic feedback is calculated from the resulting contact forces of carrying out the simulation. The IsaacGym simulation steps at a frame rate of 90 Hz, transmitting the current view from both cameras to the VR application and the haptic feedback commands to the SenseGlove at each step. 90 Hz is the recommended minimum frequency for VR applications [35, 36], but we may want to select actions at a lower frequency. This both reduces the task length, avoiding otherwise long-horizon tasks and thus speeding up learning, and increases comparability with other robot simulation frameworks like Gym [37], which uses a frame rate of 25 in most environments. We therefore decouple the control frequency of the Markov decision process from the camera and feedback frequency by repeating an action c times, resulting in a control frequency of $\frac{90}{c}$. In practice, we found a control frequency of 30 Hz to be sufficient for intuitive and accurate control. The VR application is built in the OpenVR framework [38] and updates the displayed frame every time a new image message comes in.

Hardware Details In our setup, the IsaacGym simulation ran on a single machine with an AMD Ryzen 9 5950X CPU and an NVIDIA RTX A6000 GPU. The operator is equipped with a Vive Pro Headset, a Vive Tracker and SenseGlove DK1.

4.2. Haptic Feedback

Humans rely heavily on tactile perception when manipulating objects, and haptic feedback has been shown to significantly increase the sense of embodiment VR users experience [39]. Hence, in our framework, we harness the feedback capabilities provided by SenseGlove to provide similar feedback to the operator.[1] The rigid-body contact forces of the fingertips are decomposed into the absolute force F_{abs} and a directional component F_{eff} acting against closing a finger. F_{eff} is mapped to the SenseGlove's braking system, increasing the resistance against moving the finger further and mimicking the sensation of holding an object. Additionally, we activate the SenseGlove's vibration feedback for sudden increases in contact force, e.g. collisions, by mapping the feedback to $F_{abs} - MA(F_{abs})$, where MA is a moving average low-pass filter smoothing the absolute force. In our experiments, we find that enabling haptic feedback helps operators to recognize moments of contact and results in more confident and faster object manipulation.

4.3. Optimization

We thoroughly investigated the timing impact of the processing steps of the camera images to maximize resolution of the VR view while consistently achieving the recommended frame rate of 90 Hz. The camera images have to first be retrieved, then flipped to transfer the origin to the lower left corner as expected by the VR application and finally converted to a byte stream before being handed over to the ROS messaging system. IsaacGym provides direct GPU access to camera images, and utilizing this option significantly speeds up the process as shown in figure 4.2.

[1] The haptic feedback capabilities described here were implemented by Malte Mosbach as part of our submission to the Humanoids 2022 conference [14].

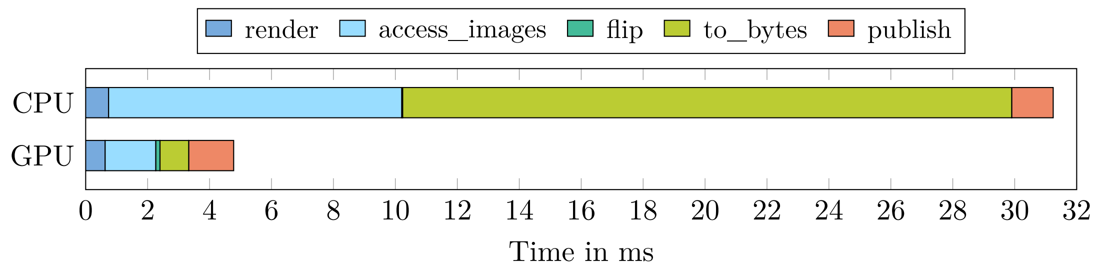


Figure 4.2: Median time measured for image processing steps carried out using either CPU or GPU access.

The maximum possible resolution will vary depending on the hardware and visual complexity of the task. At a fixed resolution, we can observe that the image processing is fast at the start of the simulation, resulting in a high frame rate. However, the frame rate significantly drops once the operator starts moving around and manipulating objects in the simulation. To consistently achieve 90 FPS, we have limited the resolution to 900 × 1000 pixels per eye, which corresponds to 62.5% of the resolution available on the Vive Pro headset. Reducing the frame rate to achieve higher resolution results in a perceivable delay between the operator's movements in the real world and the simulation view, which directly hinders performance.

5. History-aware Offline Reinforcement Learning

We investigate different possibilities for augmenting an offline RL algorithm with history awareness. The simplest way of introducing time dependency into the learning process is by adding relevant observations. Moreover, we explore two variants of offline RL, an extended state variant and a recurrent variant, which incorporate history awareness on an architectural level. In this chapter, we detail all three variants and showcase their implementation on top of CQL.

5.1. History-aware observations

In most experiments, observations only comprise the current hand pose, the five degrees of freedom for the fingers (see figure 2.1), as well as object observations. However, we also consider incorporating history-aware observations. For our gym-grasp demonstration datasets, we collect different types of observations which can be specifically enabled for learning. We consider velocity information, which comprises 17-dimensional joint velocities and, in the LiftObject task, the linear and angular velocities of the object. Another possibility to add history-aware observations is by including the 11-dimensional previous action.

These types of observations are easy to add in the simulation setting, but might be difficult to obtain in more noisy, real-world surroundings.

5.2. Extended State Variant

For the Extended State variant, short EXT, we form a history-aware state input by concatenating the current observation with the last $t-1$ observations. To augment robomimic's CQL implementation, we sample sequences of (s, a, r, d) of length t and adjust the encoder networks to expect observations of shape (t, o), where o is the size of one observation. The input to the agent is composed of the sequence of state observations, $(s)_{1:t}$, and the last action, reward and done signal of the sequence, (a_t, r_t, d_t).

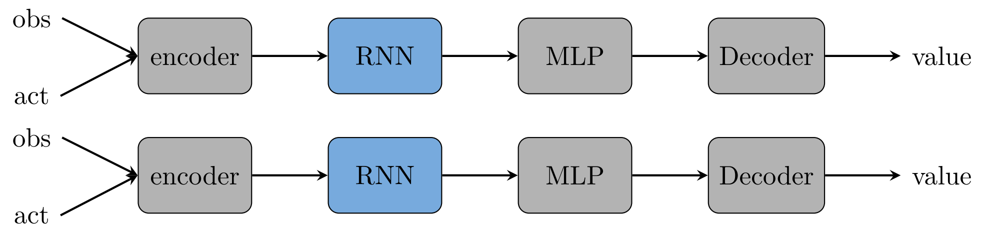
This design leads to a discrepancy between the data seen during training and testing: At the beginning of a test episode, there are no past observations available, but the trained networks cannot be queried with less than t observations. We can either repeat the first observation t times, or wait t time steps. In our specific case, i.e. the simulated tasks described earlier, the result is the same for both methods as the environment does not change while the agent does not act, hence the observations are the same. This might be different in a real-world environment, where the surroundings change even when the agent does not act. Regardless, the resulting combined first state has possibly not been seen during training. This initial state distribution shift is not investigated further in this work as our focus lies on examining the effect of different aspects of history-awareness.

5.3. Recurrent Variant

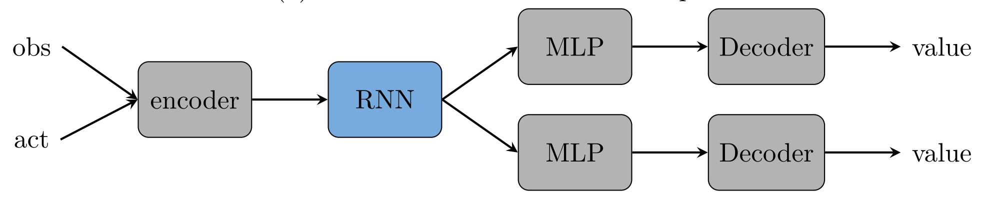
For the recurrent variant, we augment the model architecture with a recurrent neural network (RNN) and consider input sequences $(s, a, r, d)_t$ of length t. Our design is based on the architecture explored by Ni, Eysenbach, and Salakhutdinov [40] in their study of RNNs for online reinforcement learning in different types of partially observable MDPs (POMDPs). They show that a carefully designed and tuned recurrent model-free RL algorithm can perform as well as, and in fact sometimes better than, algorithms specifically designed for the type of POMDP at hand. On the basis of their extensive ablation study, they derive lessons for augmenting RL with recurrent architectures which we take into consideration in our own design, such as implementing separate RNNs for the actor and critic. They note that for RNNs, the chosen context length has major influence on the performance but is task-specific and thus requires additional tuning.

Robomimic's CQL implementation is based on Soft Actor Critic (SAC) [41], which fits well with the findings in [40] where the authors note that off-policy algorithms perform better when recurrent components are introduced. They also note that SAC led to better performance in environments with harder dynamics, whereas in environments with simpler dynamics, TD3 seemed to be a better choice.

SAC maintains separate networks for an actor and an ensemble of critics. Figure 5.1 shows the two options we implemented for adding RNNs to the ensemble of critics: The critics can either share one RNN or maintain one RNN each. While sharing the RNN saves memory, it makes coupling the critic's losses necessary. The actor can also be configured to be recurrent, which results in the architecture shown in figure 5.2. However, previous work [40] has indicated that a non-recurrent actor can lead to better performance. We evaluate the effect of these design choices



(a) Ensemble of two critics with separate RNNs.



(b) Ensemble of two critics with shared RNN.

Figure 5.1: Recurrent critics architectures. Components of standard CQL are depicted in grey, added components in blue.

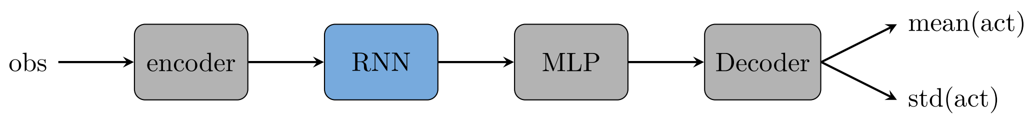


Figure 5.2: Optionally recurrent actor. Components of standard CQL are depicted in grey, added components in blue.

during hyperparameter scanning in chapter 6.

Of all three variants, CQL-RNN has the highest demand for resources and runtime due to the extended network architecture.

6. Evaluation

6.1. Dexterous Teleoperation Framework

In this section, we first validate the system with respect to its purpose, i.e. collecting demonstrations for learning. Then, we evaluate its usability.

6.1.1. Validation

To validate the teleoperation system, we posed two questions:

1. Can we successfully collect demonstrations in a reasonable amount of time?
2. Can we learn from the collected demonstrations?

In the following, we describe our experiments to address these questions.

Can we successfully collect demonstrations in a reasonable amount of time?

We collected 20 demonstrations with an experienced user for the tasks OpenDrawer and PourCup. Each demonstration was subject to a task-specific time limit and considered successful if the task was solved within said time limit. Moreover, we measured the overall time required to collect the demonstrations. The results are reported in table 6.1.

For both tasks, the 20 demonstrations are recorded within a few minutes. A dataset large enough for training, e.g. consisting of 200 demonstrations, can thus be recorded in a reasonable time frame of 15 to 30 minutes. The discrepancy between the mean length of a demonstration and the overall time stems from the environment reset and subsequent reset of the clutch mechanism after each demonstration.

Table 6.1: Success and timing analysis of demonstration collection.

Task	Time limit	Success rate	Overall time	Mean length of demonstration
OpenDrawer	4s	100%	70s	1.5s
PourCup	10s	85%	180s	6s

Table 6.2: Success Rates of behaviour cloning averaged over three seeds. For all configurations we report the mean and standard deviation of at least three runs with 100 test episodes per run.

Method	OpenDrawer	OpenDoor	PourCup	LiftObject
BC	$\mathbf{1.0 \pm 0.0}$	0.96 ± 0.02	0.76 ± 0.23	0.27 ± 0.03

Table 6.3: Success rates and completion times for the tasks performed in the user study.

Haptic feedback	Task	Success rate	Completion time [s]	
			Mean	StdDev
✓	Stack cubes	0.83	**35.17**	**9.98**
	Open door	**1.0**	**6.47**	**1.82**
	Pour cup	**1.0**	**13.86**	**3.46**
✗	Stack cubes	**1.0**	43.26	22.13
	Open door	**1.0**	9.37	5.26
	Pour cup	**1.0**	15.63	3.56

Can we learn from the collected demonstrations?

We collected datasets of 200 demonstrations for all four gym-grasp tasks and split them into 90% training data and 10% validation data. We leveraged robomimic's learning framework to train behaviour cloning policies over 1000 epochs, where each epoch consisted of 100 training and 10 validation steps. For testing, we chose the trained model with the lowest validation loss to avoid policies affected by overfitting, and ran 100 test episodes for each of the models. The achieved success rates are reported in table 6.2.

BC is able to learn highly successful and robust policies for OpenDrawer and OpenDoor. For PourCup, the standard deviation is much higher, probably due to the increased complexity of the motions required to solve the task. LiftObject is the only task considered where random initialization is applied, and BC seemingly struggles to generalize from the provided demonstrations: In rollouts, the end effector sometimes makes grasping motions in the wrong spot or performs no action at all. Overall, we can infer that it is possible to learn meaningful policies from our recorded demonstrations.

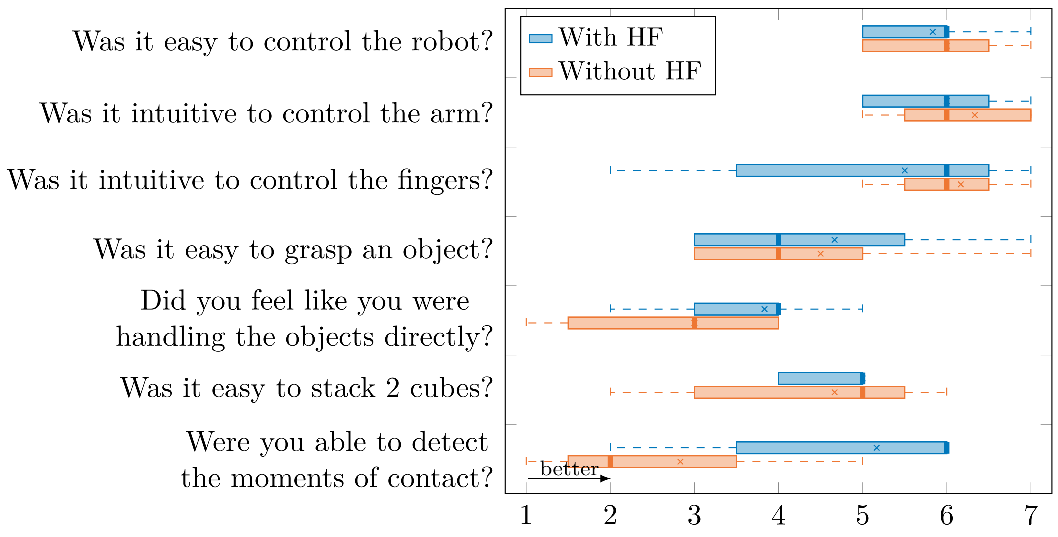


Figure 6.1: Statistical results of the user questionnaire. The median, lower and upper quartile (including interquartile range), lower and upper fence, outliers (marked with •), and average value (marked with ×) are displayed for each item captured in our questionnaire. Figure from [14].

6.1.2. Usability

We conducted a small user study to evaluate both the general user experience and the specific influence of haptic feedback. The six participants had no prior experience with the framework and were asked to perform three tasks from gym-grasp: Stacking three cubes, opening a door and pouring particles from a cup into a bowl. After that, they were given a short questionnaire to rate their experience with seven-level Likert items. This process was completed once with and once without haptic feedback, and the order of these two trials was randomized.

Table 6.3 shows the completion time and success rates observed, and the high success rates indicate that our VR system can be effectively used even without prior experience. Across all tasks, there was only one failure when one of the cubes fell off the table. The stacking task also has a high standard deviation as the tower of cubes is easily knocked over, requiring the user to start again. On average, completion time was lower when haptic feedback was enabled.

Figure 6.1 summarizes the results of the questionnaire. Intuitive control was generally rated highly, and haptic feedback seems to have a positive impact on the ability to grasp an object and the impression of direct interaction. Most notably, haptic feedback significantly improved the ability to detect moments of contact. Recognizing those moments increases confidence in the pose of the hand in simulation and can be helpful to discern different phases of motion such as moving towards a cube, grasping it and releasing it. We hypothesize that this aspect was a main contributor to the overall lower completion times observed when haptic feedback was enabled.

6.2. History-aware Offline Reinforcement Learning

We evaluated our history-aware variants of CQL against standard CQL with the aim of answering the following questions:

Q1: Does CQL benefit from history-awareness when learning from human demonstrations? Mandlekar et al. [11] observe that algorithms with history-aware components performed better on human datasets than those without, and hypothesize that adding such components to an offline RL algorithm such as CQL might improve their performance on such datasets. In our experiments, we look at a total of nine tasks with different levels of complexity and required precision, both from gym-grasp with an anthropomorphic end-effector and robomimic with a simple gripper end-effector.

Q2: How do different history-aware variants of CQL compare? As we have explored in chapter 5, there are multiple ways to provide an algorithm with information about recent history, each with different practical benefits and drawbacks. When it comes to considerations such as availability of observations and resources, we need to be able to estimate the advantage of adding specific types of history-awareness. We thoroughly investigate the effect of our three methods on human gym-grasp datasets.

Q3: What are the implications for learning from machine-generated demonstrations? Machine-generated demonstrations are recorded by a successful policy and are inherently Markovian, meaning that the state at each time step holds all information needed to select the correct action. Augmenting the state by extending it over multiple time steps or introducing a recurrent hidden state should therefore show little effect on this type of data. We evaluate standard CQL and the two variants CQL-EXT and CQL-RNN on machine-generated gym-grasp datasets.

In the following, we describe our procedure from collecting the required datasets to hyperparameter scanning and running experiments to answer the aforementioned questions.

Table 6.4: Average trajectory length of demonstrations for gym-grasp tasks.

Dataset Type	OpenDrawer	OpenDoor	PourCup	LiftObject
machine-generated	11 ± 1	73 ± 62	35 ± 50	19 ± 0
human	47 ± 17	73 ± 15	102 ± 22	91 ± 32

6.2.1. Dataset Collection

For our experiments with robosuite tasks, we used the proficient-human (PH) datasets provided by robomimic. They consist of 200 demonstrations collected from an experienced teleoperator through RoboTurk [42].

For tasks from the gym-grasp framework, we collected both human and machine-generated demonstrations. Table 6.4 shows the average trajectory length of the collected demonstrations.

Machine-Generated We trained RL agents on all four tasks using dense rewards and Proximal Policy Optimization (PPO) for 1000 epochs, where each epoch consists of 32 steps in 16384 parallel environment instances. Then, we played 200 episodes of the trained model and recorded them as demonstrations. All collected demonstrations were successful.

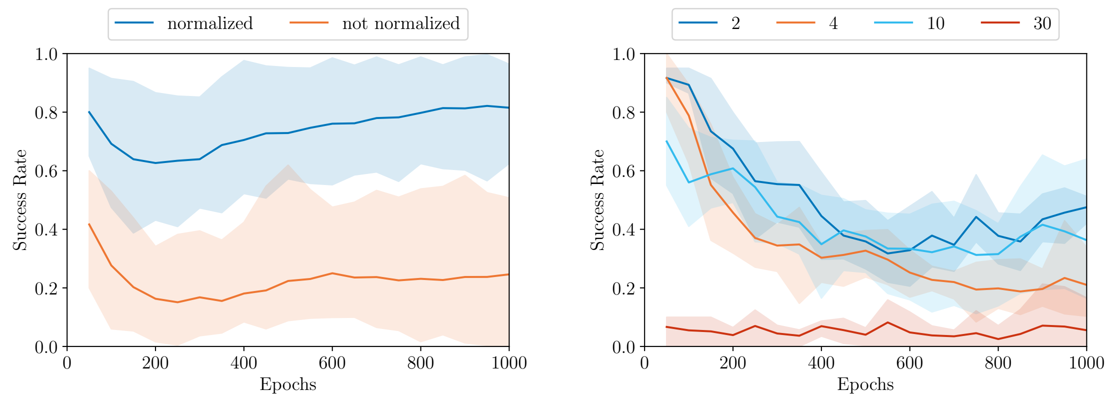
Human For each task, we collected 200 demonstrations with an experienced user in our VR teleoperation framework. Only successful episodes were recorded.

6.2.2. Hyperparameter Scanning

We carried out extensive hyperparameter scanning for all variants of CQL on both gym-grasp and robomimic datasets using Weights and Biases [43]. The results can be viewed online[1].

For gym-grasp tasks, we used the OpenDoor datasets to select hyperparameters separately for machine-generated and human data. We first determined a set of hyperparameters for standard CQL, consulting the hyperparameter choices reported by Mandlekar et al. [11] as a starting point. The most surprising finding was the effect of normalizing observations. Figure 6.2a shows that in our experiments, normalizing observations more than doubled the achieved success rate. For the EXT variant, we then only scanned the sequence length and network sizes. Our hyperparameter scans for the RNN variant were more extensive, considering the architecture and size of the added networks. However, we could not observe

[1] `https://wandb.ai/moraw/thesis-human-OpenDoor/reportlist`



(a) Effect of normalizing observations in standard CQL.

(b) Effect of different sequence lengths for CQL-EXT.

Figure 6.2: Results from hyperparameter scanning. We report the mean smoothed success rate across three seeds on the task OpenDoor. The shaded area corresponds to the smoothed minimum and maximum success rate.

improvements across different configurations, hence opting for the simplest and most resource-saving variant with a non-recurrent actor and recurrent critics with one shared RNN. The resulting sets of hyperparameters were used for all other gym-grasp datasets from the same source, as each run took about 1.5 hours and scanning more tasks would have been infeasible.

For robomimic tasks, we adopted the same hyperparameters as reported by Mandlekar et al. [11] and scanned only the hyperparameters added by our variants of CQL on the Can task. Again, we could not discern gains from different RNN architectures and decided to use the same hyperparameters as for the gym-grasp tasks.

As Ni, Eysenbach, and Salakhutdinov [40] indicate that the appropriate sequence length can be highly task-specific, we carried out additional short sweeps of only 200 epochs on a single seed regarding the sequence length for the EXT variant on all tasks. In our case, the most appropriate sequence length varied not between tasks, but task suites, as robomimic tasks performed better with a sequence length of 4, whereas gym-grasp tasks benefitted from a sequence length of 2. Figure 6.2b shows the impact of the sequence length: Even a small change, like from 2 to 4, leads to poorer overall performance. A detailed breakdown of the scanned hyperparameters and subsequent choices is given in appendix A.

Table 6.5: Results of CQL variants on human gym-grasp datasets. For all methods, we report the mean and standard deviation of the maximum success rates averaged over three seeds.

Method	OpenDrawer	OpenDoor	PourCup	LiftObject
CQL	$\mathbf{1.00 \pm 0.00}$	$\mathbf{0.95 \pm 0.07}$	$\mathbf{0.77 \pm 0.02}$	0.60 ± 0.15
CQL-EXT	$\mathbf{1.00 \pm 0.00}$	0.90 ± 0.00	0.60 ± 0.07	$\mathbf{0.67 \pm 0.06}$
CQL-RNN	0.00 ± 0.00	0.00 ± 0.00	0.02 ± 0.02	0.00 ± 0.00

Table 6.6: Results of additional observation types on human gym-grasp datasets. For all observation types, we report the mean and standard deviation of the maximum success rates averaged over three seeds.

Added observations	OpenDrawer	OpenDoor	PourCup	LiftObject
velocity	0.68 ± 0.17	0.70 ± 0.27	0.02 ± 0.02	0.20 ± 0.11
previous actions	0.43 ± 0.22	0.45 ± 0.04	0.03 ± 0.05	0.17 ± 0.02
velocity, previous actions	0.43 ± 0.33	0.48 ± 0.14	0.00 ± 0.00	0.05 ± 0.04
finger contacts	$\mathbf{1.00 \pm 0.00}$	0.90 ± 0.08	0.67 ± 0.09	0.48 ± 0.06
fingertip positions	$\mathbf{1.00 \pm 0.00}$	$\mathbf{1.00 \pm 0.00}$	$\mathbf{0.93 \pm 0.05}$	$\mathbf{0.73 \pm 0.06}$

6.2.3. Experimental Results

We carried out comprehensive experiments and evaluated them with respect to the achieved maximum success rate averaged across three seeds. Tables 6.5 to 6.8 show the results which we discuss with respect to the questions posed above.

Q1 Tables 6.5 and 6.8 show that using an extended state significantly aids performance in complex tasks that require high precision, such as LiftObject and Square. However, we also note that it seems to hinder performance in tasks like OpenDoor and PourCup that are subject to less randomization. Augmenting the CQL critics with an RNN is substantially harmful to performance. Table 6.6 shows that including history-aware observations such as velocity information and previous actions has a detrimental effect on performance. This is particularly surprising considering that we utilize a sequence length of 2 for gym-grasp tasks, meaning that the extended state considered in CQL-EXT contains implicit velocity information. For the LiftObject task specifically, adding velocity observation causes the performance to drop by two thirds when compared to standard CQL. The extended state, on the other hand, proves a substantial benefit in learning this task. The same phenomenon can be observed for previous actions: These

Table 6.7: Results of CQL variants on machine-generated gym-grasp datasets. For all methods, we report the mean and standard deviation of the maximum success rates averaged over three seeds.

Method	OpenDrawer	OpenDoor	PourCup	LiftObject
CQL	$\mathbf{0.75 \pm 0.04}$	$\mathbf{0.57 \pm 0.08}$	$\mathbf{0.90 \pm 0.00}$	0.00 ± 0.00
CQL-EXT	0.45 ± 0.11	0.53 ± 0.08	0.62 ± 0.05	0.00 ± 0.00
CQL-RNN	0.05 ± 0.04	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00

Table 6.8: Results of CQL variants on robomimic PH datasets. For all methods, we report the mean and standard deviation of the maximum success rates averaged over three seeds. For comparison, we also include the results of BC-RNN reported by Mandlekar et al. [11].

Method	Lift	Can	Square	Transport	Tool-Hang
CQL [11]	0.93 ± 0.05	$\mathbf{0.38 \pm 0.08}$	0.05 ± 0.03	0.00 ± 0.00	0.00 ± 0.00
CQL-EXT	$\mathbf{1.00 \pm 0.00}$	$\mathbf{0.38 \pm 0.05}$	$\mathbf{0.22 \pm 0.08}$	0.00 ± 0.00	0.00 ± 0.00
CQL-RNN	0.15 ± 0.04	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
BC-RNN [11]	1.00 ± 0.00	1.00 ± 0.00	0.84 ± 0.00	0.71 ± 0.07	0.19 ± 0.05

can also be inferred from the two consecutive states available in CQL-EXT, but only show a harmful effect on performance when made available as explicit inputs. Neither the EXT variant nor the RNN variant manage to improve performance in the long-horizon multi-stage tasks Transport and Tool-Hang from robomimic. As Transport and Tool-Hang require a much slower pace in action than the other tasks, we hypothesize that the EXT variant might be able to achieve better results for longer sequences in later epochs than we considered during sweeping of the sequence length. Both variants are also far from reaching the performance reported by Mandlekar et al. for BC-RNN on their tasks.

Q2 We note that neither of our three approaches to history-aware CQL enable consistently improved performance. CQL-RNN and history-aware observations prove significantly harmful across all considered tasks. CQL-EXT is able to boost performance in complex tasks. However, if one were to consider employing CQL-EXT in such a difficult, possibly long-horizon task, successfully tuning the sequence length would likely require more than the 200 epochs we employed here, further adding to the increased effort in time and computation that comes with this variant.

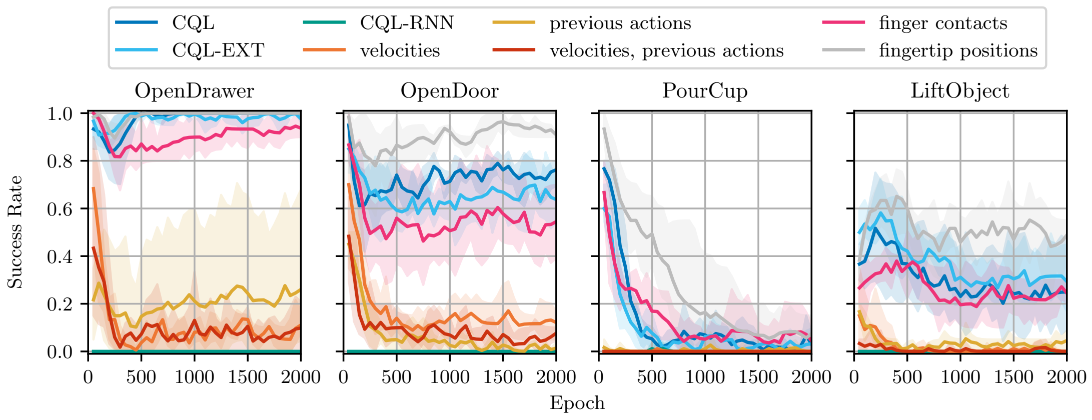


Figure 6.3: Human datasets for gym-grasp tasks.

Q3 Table 6.7 shows that our history-aware variants of CQL do not improve performance for machine-generated demonstrations but instead harm performance. This is expected, as the demonstrated behaviour is Markovian and a single state already contains all information the behaviour policy used to decide on an action. Adding more information in this case occludes the true state the demonstrator was using. Moreover, no variant of CQL is able to learn from machine-generated demonstrations of LiftObject, which we attribute to the comparatively small dataset available considering the complexity of the task.

Additional findings We also explored the effect of adding other types of observations such as fingertip position and fingertip contact forces. Surprisingly, fingertip position observations boosted performance for all tasks and achieved the highest success rates across all gym-grasp experiments, as shown in table 6.6. The fingertip position can in principle be inferred from the standard observations, as the degrees of freedom of the hand and the wrist pose are included. Nevertheless, providing them as explicit inputs to the learning agent seems beneficial. Overall, we find that CQL is highly sensitive not only to hyperparameter choices such as learning rates, but also to different tasks and input types. Figure 6.3 shows how performance usually drops after 50 to 200 epochs, suggesting that the model quickly overfits. However, we can observe that some configurations can recover from this effect while others are unable to achieve any success in later epochs. In the PourCup task, no configuration is able to maintain good success rates.

7. Conclusion

Offline RL methods are a promising technique to effectively learn robust policies from large, diverse datasets. Particularly in the field of dexterous manipulation, where we deal with a vast amount of tasks and a high-dimensional state-action space, the ability to learn from suboptimal data or demonstrations from other tasks would drastically increase the efficiency of the learning process.

In this work, we presented an immersive VR teleoperation system designed to enable intuitive and efficient collection of demonstrations for tasks from the gym-grasp [12] suite. Our setup uses only few hardware components and runs at a high frame rate, allowing for seamless interaction with objects in simulation. The incorporation of haptic feedback further decreases the time a human operator requires to solve a task, hence enabling faster data collection.

Prior work [11] has indicated that offline RL algorithms perform better on machine-generated data than on human data, and suggested that history-aware components might aid performance on potentially non-Markovian data sources such as human demonstrations. We proposed and investigated three ways of including history-awareness in state-of-the-art offline RL algorithm CQL [16]: incorporating history-aware observations, concatenating multiple subsequent states into one (CQL-EXT), and incorporating recurrent neural networks into the architecture (CQL-RNN). Overall, we find that CQL is highly sensitive to both hyperparameter changes and input changes, which makes it challenging to effectively evaluate a large set of tasks such as the one we study here. Our variants struggle to improve performance in the tasks considered, more often hindering success than boosting it. The most promising of our variants is CQL-EXT, which improves performance in complex tasks with high precision demands. For future work, it would be interesting to investigate whether an extended state as in CQL-EXT can lead to similar improvements when applied to other offline RL algorithms.

In the broader picture, developing effective and robust offline RL algorithms will enable turning large and diverse datasets into generalizable policies. As discussed earlier, these algorithms are in principle capable of learning optimal behaviour policies from datasets containing suboptimal and unlabelled (i.e. zero-reward) data. The demands on the demonstrations are thus much lower than in imitation learning, where we usually require a teacher demonstrating optimal behaviour, facilitating the collection of large datasets.

A. Hyperparameter Scans

Each run carried out during hyperparameter scanning consisted of 1000 epochs, and we collected 20 rollouts of the current policy every 50 epochs. Hyperparameter choices were judged by the success rates achieved during rollouts. For standard CQL on gym-grasp datasets, we scanned the hyperparameters shown in table A.1 on the task OpenDoor. Tables A.2 and A.3 show the scanned hyperparameters for our history-aware variants of CQL on gym-grasp datasets. For robomimic datasets, we did not scan the actor and critic network dimensions but kept them equal to those utilized by Mandlekar et al. [11] for their evaluation of standard CQL. Tables A.4 and A.5 show the resulting hyperparameter values which are identical to those used for gym-grasp datasets except for the sequence length.

Table A.1: Scanned hyperparameters for standard CQL on gym-grasp datasets.

Hyperparameter	Scanned values	Choice
Actor Network Dimensions	$(256, 256), (512, 512),$ $(256, 512, 256)$	human: $(256, 512, 256)$, mg: $(256, 256)$
Critic Network Dimensions	$(256, 256), (512, 512),$ $(256, 512, 256)$	$(256, 512, 256)$,
Critic Action Samples	$1, 2, 4, 10, 30$	1
Critic Target Q Gap	$1, 5, 10$	5
Actor LR Decay Factor	$0, 0.1$	0.1
Actor Initial LR	$1 \times 10^{\{-5,-4,-3,-2\}}$, $3 \times 10^{\{-5,-4,-3,-2\}}$	1×10^{-4}
Critic LR Decay Factor	$0, 0.1$	0
Critic Initial LR	$1 \times 10^{\{-5,-4,-3,-2\}}$, $3 \times 10^{\{-5,-4,-3,-2\}}$	3×10^{-5}
Actor Target τ	$5 \times 10^{-3}, 5 \times 10^{-4}$	5×10^{-4}
Actor L2 Regularization	$0, 0.01, 0.1$	0
Critic L2 Regularization	$0, 0.01, 0.1$	0
Actor Gradient Clipping	true, false	false
Critic Gradient Clipping	true, false	false
Normalized Observations	true, false	true

Table A.2: Scanned hyperparameters for CQL-EXT on gym-grasp datasets.

Hyperparameter	Scanned values	Choice
Actor Network Dimensions	$(256, 256), (512, 512),$ $(256, 512, 256)$	human: $(256, 512, 256)$, mg: $(256, 256)$
Critic Network Dimensions	$(256, 256), (512, 512),$ $(256, 512, 256)$	$(256, 512, 256)$,
Sequence Length	$2, 4, 10, 30$	2

Table A.3: Scanned hyperparameters for CQL-RNN on gym-grasp datasets.

Hyperparameter	Scanned values	Choice
Actor Network Dimensions	$(256, 256), (512, 512),$ $(256, 512, 256)$	human: $(256, 512, 256)$, mg: $(256, 256)$
Critic Network Dimensions	$(256, 256), (512, 512),$ $(256, 512, 256)$	$(256, 512, 256)$,
Sequence Length	$2, 4, 10, 30$	2
Recurrent Actor	true, false	false
Critic RNN Dimensions	$(256, 256), (512, 512),$ $(256, 256, 256)$	$(256, 256)$,
Shared Critic RNN	true, false	true

Table A.4: Scanned hyperparameters for CQL-EXT on robomimic datasets.

Hyperparameter	Scanned values	Choice
Sequence Length	$2, 4, 10, 30$	4

Table A.5: Scanned hyperparameters for CQL-RNN on robomimic datasets.

Hyperparameter	Scanned values	Choice
Sequence Length	$2, 4, 10, 30$	4
Recurrent Actor	true, false	false
Critic RNN Dimensions	$(256, 256), (512, 512),$ $(256, 256, 256)$	$(256, 256)$,
Shared Critic RNN	true, false	true

Bibliography

[1] Kilian Kleeberger, Richard Bormann, Werner Kraus, and Marco F Huber. "A survey on learning-based robotic grasping." In: *Current robotics reports* 1.4 (2020), pp. 239–249.

[2] Di Cao, Weihao Hu, Junbo Zhao, Guozhou Zhang, Bin Zhang, Zhou Liu, Zhe Chen, and Frede Blaabjerg. "Reinforcement learning and its applications in modern power and energy systems: a review." In: *Journal of modern power systems and clean energy* 8.6 (2020), pp. 1029–1042.

[3] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. "Mastering the game of go without human knowledge." In: *Nature* 550.7676 (2017), pp. 354–359.

[4] Ngan Le, Vidhiwar Singh Rathour, Kashu Yamazaki, Khoa Luu, and Marios Savvides. "Deep reinforcement learning in computer vision: a comprehensive survey." In: *Artificial intelligence review* (2021), pp. 1–87.

[5] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. "End-to-end training of deep visuomotor policies." In: *The journal of machine learning research* 17.1 (2016), pp. 1334–1373.

[6] Aravind Rajeswaran, Vikash Kumar, Abhishek Gupta, Giulia Vezzani, John Schulman, Emanuel Todorov, and Sergey Levine. "Learning complex dexterous manipulation with deep reinforcement learning and demonstrations." In: *Arxiv preprint arxiv:1709.10087* (2017).

[7] Ajay Mandlekar, Danfei Xu, Roberto Martín-Martín, Silvio Savarese, and Li Fei-Fei. "Learning to generalize across long-horizon tasks from human demonstrations." In: *Arxiv preprint arxiv:2003.06085* (2020).

[8] Aviral Kumar. Data-driven Deep Reinforcement Learning. 2019. URL: `https://bair.berkeley.edu/blog/2019/12/05/bear/`.

[9] Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. "Offline reinforcement learning: tutorial, review, and perspectives on open problems." In: *Arxiv preprint arxiv:2005.01643* (2020).

[10] Avi Singh, Albert Yu, Jonathan Yang, Jesse Zhang, Aviral Kumar, and Sergey Levine. "Cog: connecting new skills to past experience with offline reinforcement learning." In: *Arxiv preprint arxiv:2010.14500* (2020).

[11] Ajay Mandlekar, Danfei Xu, Josiah Wong, Soroush Nasiriany, Chen Wang, Rohun Kulkarni, Li Fei-Fei, Silvio Savarese, Yuke Zhu, and Roberto Martín-Martín. “What matters in learning from offline human demonstrations for robot manipulation.” In: *Arxiv preprint arxiv:2108.03298* (2021).

[12] Malte Mosbach. gym-grasp. 2022. URL: `https://git.ais.uni-bonn.de/mosbach/gym-grasp`.

[13] Yuke Zhu, Josiah Wong, Ajay Mandlekar, and Roberto Martín-Martín. “Robosuite: a modular simulation framework and benchmark for robot learning.” In: *Arxiv preprint arxiv:2009.12293*. 2020.

[14] Malte Mosbach, Kara Moraw, and Sven Behnke. “Accelerating interactive human-like manipulation learning with gpu-based simulation and high-quality demonstrations.” In: *Arxiv preprint arxiv:2212.02126* (2022).

[15] Richard S Sutton and Andrew G Barto. *Reinforcement learning: an introduction.* MIT press, 2018.

[16] Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. “Conservative q-learning for offline reinforcement learning.” In: *Advances in neural information processing systems* 33 (2020), pp. 1179–1191.

[17] Rafael Figueiredo Prudencio, Marcos ROA Maximo, and Esther Luna Colombini. “A survey on offline reinforcement learning: taxonomy, review, and open problems.” In: *Arxiv preprint arxiv:2203.01387* (2022).

[18] Scott Fujimoto, David Meger, and Doina Precup. “Off-policy deep reinforcement learning without exploration.” In: *International conference on machine learning.* PMLR. 2019, pp. 2052–2062.

[19] Manu Orsini, Anton Raichuk, Léonard Hussenot, Damien Vincent, Robert Dadashi, Sertan Girgin, Matthieu Geist, Olivier Bachem, Olivier Pietquin, and Marcin Andrychowicz. “What matters for adversarial imitation learning?” In: *Advances in neural information processing systems* 34 (2021), pp. 14656–14668.

[20] Vikash Kumar and Emanuel Todorov. “Mujoco haptix: a virtual reality system for hand manipulation.” In: *2015 ieee-ras 15th international conference on humanoid robots (humanoids).* IEEE. 2015, pp. 657–663.

[21] Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. “D4rl: datasets for deep data-driven reinforcement learning.” In: *Arxiv preprint arxiv:2004.07219* (2020).

[22] Sham M Kakade. “A natural policy gradient.” In: *Advances in neural information processing systems* 14 (2001).

[23] Henry Zhu, Abhishek Gupta, Aravind Rajeswaran, Sergey Levine, and Vikash Kumar. “Dexterous manipulation with deep reinforcement learning: efficient, general, and low-cost.” In: *2019 international conference on robotics and automation (icra)*. IEEE. 2019, pp. 3651–3657.

[24] Ashvin Nair, Abhishek Gupta, Murtaza Dalal, and Sergey Levine. “Awac: accelerating online reinforcement learning with offline datasets.” In: *Arxiv preprint arxiv:2006.09359* (2020).

[25] David Brandfonbrener, Will Whitney, Rajesh Ranganath, and Joan Bruna. “Offline rl without off-policy evaluation.” In: *Advances in neural information processing systems* 34 (2021), pp. 4933–4946.

[26] Ilya Kostrikov, Ashvin Nair, and Sergey Levine. “Offline reinforcement learning with implicit q-learning.” In: *Arxiv preprint arxiv:2110.06169* (2021).

[27] Priyanka Mandikal and Kristen Grauman. “Dexvip: learning dexterous grasping with human hand pose priors from video.” In: *Conference on robot learning*. PMLR. 2022, pp. 651–661.

[28] Shangchen Han, Beibei Liu, Robert Wang, Yuting Ye, Christopher D Twigg, and Kenrick Kin. “Online optical marker-based hand tracking with deep labels.” In: *Acm transactions on graphics (tog)* 37.4 (2018), pp. 1–10.

[29] Ankur Handa, Karl Van Wyk, Wei Yang, Jacky Liang, Yu-Wei Chao, Qian Wan, Stan Birchfield, Nathan Ratliff, and Dieter Fox. “Dexpilot: vision-based teleoperation of dexterous robotic hand-arm system.” In: *2020 ieee international conference on robotics and automation (icra)*. IEEE. 2020, pp. 9164–9170.

[30] Tanner Schmidt, Richard A Newcombe, and Dieter Fox. “Dart: dense articulated real-time tracking.” In: *Robotics: science and systems*. Vol. 2. 1. Berkeley, CA. 2014, pp. 1–9.

[31] Yuzhe Qin, Hao Su, and Xiaolong Wang. “From one hand to multiple hands: imitation learning for dexterous manipulation from single-camera teleoperation.” In: *Arxiv preprint arxiv:2204.12490* (2022).

[32] CyberGlove Systems Inc. CyberGlove Systems LLC. 2017. URL: `https://www.cyberglovesystems.com/`.

[33] Colin M Light, Paul H Chappell, and Peter J Kyberd. “Establishing a standardized clinical assessment tool of pathologic and prosthetic hand function: normative data, reliability, and validity.” In: *Archives of physical medicine and rehabilitation* 83.6 (2002), pp. 776–783.

[34] Stanford Artificial Intelligence Laboratory et al. *Robotic operating system*. Version ROS Noetic Ninjemys. 2020. URL: `https://www.ros.org`.

[35] Jeff Hecht. “Optical dreams, virtual reality.” In: *Optics and photonics news* 27.6 (2016), pp. 24–31.

[36] Panagiotis Kourtesis, Simona Collina, Leonidas A. A. Doumas, and Sarah E. MacPherson. “Technological competence is a pre-condition for effective implementation of virtual reality head mounted displays in human neuroscience: a technological review and meta-analysis.” In: *Frontiers in human neuroscience* 13 (2019). ISSN: 1662-5161. URL: `https://www.frontiersin.org/articles/10.3389/fnhum.2019.00342`.

[37] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym. 2016. eprint: `arXiv:1606.01540`.

[38] Valve Corporation. OpenVR SDK. URL: `https://github.com/ValveSoftware/openvr`.

[39] Grégoire Richard, Thomas Pietrzak, Ferran Argelaguet, Anatole Lécuyer, and Géry Casiez. “Studying the role of haptic feedback on virtual embodiment in a drawing task.” In: *Frontiers in virtual reality* 1 (2021), p. 573167.

[40] Tianwei Ni, Benjamin Eysenbach, and Ruslan Salakhutdinov. “Recurrent model-free rl is a strong baseline for many pomdps.” In: *Arxiv preprint arxiv:2110.05038* (2021).

[41] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. “Soft actor-critic algorithms and applications.” In: *Arxiv preprint arxiv:1812.05905* (2018).

[42] Ajay Mandlekar, Yuke Zhu, Animesh Garg, Jonathan Booher, Max Spero, Albert Tung, Julian Gao, John Emmons, Anchit Gupta, Emre Orbay, et al. “Roboturk: a crowdsourcing platform for robotic skill learning through imitation.” In: *Conference on robot learning*. PMLR. 2018, pp. 879–893.

[43] Weights & Biases. Weights & Biases - Developer Tools for ML. 2022. URL: `https://wandb.ai/site`.