Learning Autonomous Robot Grasping

Oliver Day

4th Year Project Report
Artificial Intelligence and Computer Science
School of Informatics
University of Edinburgh

2021

Abstract

This work explores how robust and generalisable robotic grasping behaviours can be learned from data. A pick and reach task is used in which a simulated robot arm grasps a cylindrical object and lifts it out of a tray. Offline reinforcement learning is a class of methods that train only from pre-gathered data. This work contributes an evaluation of state-of-the-art offline reinforcement learning techniques in comparison with potential-integral-derivative control and online reinforcement learning. Conservative Q-learning (CQL), the state-of-the-art in offline reinforcement learning, is able to successfully learn to grasp and complete the task. CQL is shown to closely match the data in terms of task success rate and learned behaviours. CQL also shows some robustness to external forces pushing the object from the end-effector and ability to generalise to perturbed simulation physics and dynamics settings. These offline reinforcement learning results outperform online reinforcement learning that learns with additional interaction data and is not able to complete the task.

Acknowledgements

I would like to thank Zhibin Li and his lab for their supervision and guidance. I also give thanks to my parents for supporting me through University. Finally, I would like to acknowledge the PyBullet team and Aviral Kumar, Aurick Zhou, George Tucker and Sergey Levine for their open-source robotics and reinforcement learning libraries that made this work possible.

Table of Contents

Chapter 1

Introduction

Robots are defined as "physical agents that perform tasks by manipulating the physical world" [1]. Robot engineering draws on a wide range of sources including perception, planning and control theory. Robotics is increasingly impacting society from manufacturing to drones and self-driving cars. Industrial robotics is a field of robotics concerned with manufacturing tasks such as assembly and welding. The first robotic arm went on sale in the 1961 [1] and the industry is set to more than double in the next 5-10 years [2].

Approaches to robotics manipulation typically use classical planning and control methods. These methods have proved efficient and effective on the assembly line for controlled tasks. However, they require manual tuning for new tasks and can struggle in unstructured domains. Learning based approaches aim to overcome these limitations by developing behaviours from data. The hope is that learned behaviours can be more robust and generalisable, as well as making it easy to train on new tasks by just providing data.

This project aims to explore how robotic grasping can be learned from behaviour data. It aims to evaluate the robustness and generalisability of the learned grasping. The task is a pick and reach robotics task in which an object must be picked out of a tray.

The contributions in this work are:

- Developed a suite of simulated robotics tasks with datasets to evaluate learning to grasp.
- Showed that offline reinforcement learning is able to learn robotic grasping from data.
- Benchmarked a proportional-integral-derivative (PID) controller and a state-of-the-art online reinforcement learning algorithm.
- Evaluated the robustness and generalisability of the learned policies on the robotics tasks.

The aims of this work were completed successfully and conservative Q-learning

(CQL) is shown to learn from offline data to grasp the object and complete the task. CQL learns to behave similarly to the behaviour data and completes the task 73% of the time with the object randomly placed in the tray. CQL also proves to be able to generalise to perturbed physics and dynamics settings and robust under external forces, comparable to the dataset. Further work beyond the original goals was completed in comparing to a state-of-the-art online reinforcement learning algorithm, soft actor-critic (SAC) [3]. CQL trains better and outperforms SAC, despite SAC gathering additional online data. The code base for this work is open-source[1] and builds on outstanding open-source work from [4], [5] and [6].

This report starts with chapter 2, a background chapter introducing the problem settings and related work in reinforcement learning and robotic manipulation. Next, in chapter 3, the approach used in this work is elaborated, with details about the algorithms and models used. The experiment setup is explained in chapter 4, describing the simulation, data and tasks. Chapter 5 discusses the experiment results, showing the models' training and evaluation performance and examining the learned behaviours. Finally, chapter 6 concludes the report and proposes future work.

[1]The code is available here and in the additional materials.

Chapter 2

Background

Reinforcement learning is a framework for learning control from interaction. The aim is to learn a policy to interact with an environment in a way that maximises the values of a reward signal. Reinforcement learning is widely applicable in areas including robotics, dialogue systems and finance. The field has seen considerable progress in the last decade, notably with beating the world champion at the board game Go [7], achieving human-level performance at Atari games [8] and algorithms such as proximal policy optimisation [9]. Much of this development involves combining reinforcement learning with deep learning.

Deep learning is a subfield of machine learning that uses neural networks to learn functions from data. With an increase in the availability of data and computing resources as well as improvements in algorithms, there have been major developments in deep learning including image recognition [10], image generation [11] and natural language generation [12]. These breakthroughs have led to a growing artificial intelligence industry with real-world applications as broad as health [13], recommender systems [14] and speech technologies [15].

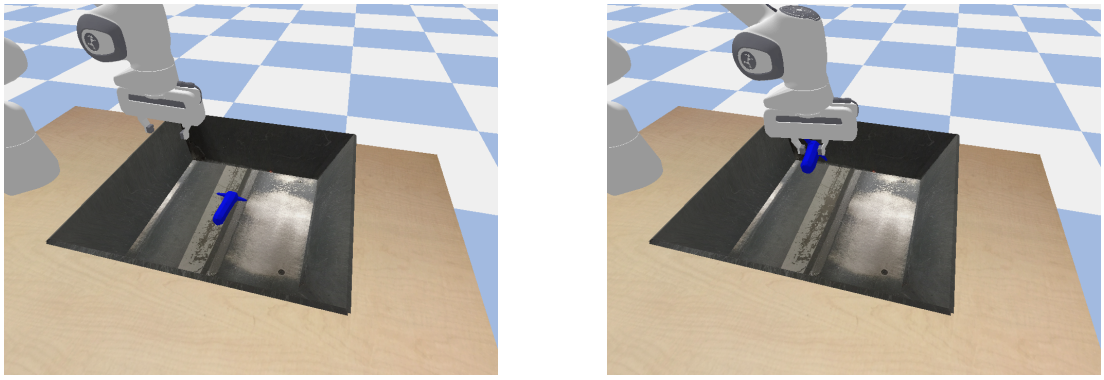
Despite its potential, reinforcement learning has not seen as much real-world impact as deep learning. Largely because the techniques used today require impractical amounts of experience in the environment to learn from. This is especially an issue when interaction data are expensive (e.g. robotics) or dangerous (e.g. healthcare) [16].

In standard reinforcement learning these experience data are collected throughout training, this is referred to as the *online* setting. In contrast, *offline* reinforcement learning utilises data sets gathered before training, to reduce or even eliminate the need for online interaction to learn optimal behaviours.

Several alternatives to offline reinforcement learning have been developed to make better use of interaction data such as simulation to real-world transfer [16] and iterative labelling of on-policy data as a form of imitation learning in DAgger [17]. Offline reinforcement learning is a promising direction for real-world applications where, despite the difficulty in gathering new interaction

data, there are data sets available to learn. Example applications include dialogue systems and healthcare [16].

The application domain in this work is learning robotic grasping. A continuous control pick and reach task is used, requiring a robot arm to successfully grip an object in the environment and lift it out of a tray (figure 2.1). This has applications to manufacturing and assistive robotics. For evaluation, a regrasping task and perturbed physics task are used.



(a) Start of the task

(b) Completion behaviour of the task

Figure 2.1: The pick and reach task: a PyBullet simulation of a Franka robotic arm is controlled to grasp and lift a cylindrical object.

2.1 Problem Definition

2.1.1 Reinforcement Learning Problem

Reinforcement learning is a framework for learning a control policy from interaction with a system called the environment. This section defines the problem setting in reinforcement learning as formulated in [16] and [18]. The policy receives a state, s, from the environment and decides on an action, a, to send to the environment. The environment updates based on the action and sends back the new state, s' and a scalar reward signal, $r(s,a)$. This sequence $s, a, r(s,a), s'$ is called a transition.

In reinforcement learning, sequential decision making is formalised as a Markov decision process (MDP). An MDP is defined as: $\mathcal{M} = (\mathcal{S}, \mathcal{A}, T, d_0, r, \gamma)$ where $\mathcal{S}$ is the set of states $s \in \mathcal{S}$; $\mathcal{A}$ is the set of actions $a \in \mathcal{A}$; T is the transition probability distribution, $T(s_{t+1}|s_t, a_t)$, that defines the dynamics of the system; d_0 is the distribution of initial states $d_0(s_0)$; $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function and $\gamma \in [0,1]$ is the discount factor. The key assumption in the MDP setting is the Markov property, in the formulation of the transition probability T, that the probability distribution over a state s_{t+1} depends only on the previous state s_t and action a_t [18].

Given an MDP, the goal of reinforcement learning, as described in the reward hypothesis, is to maximise the expected sum of discounted rewards received

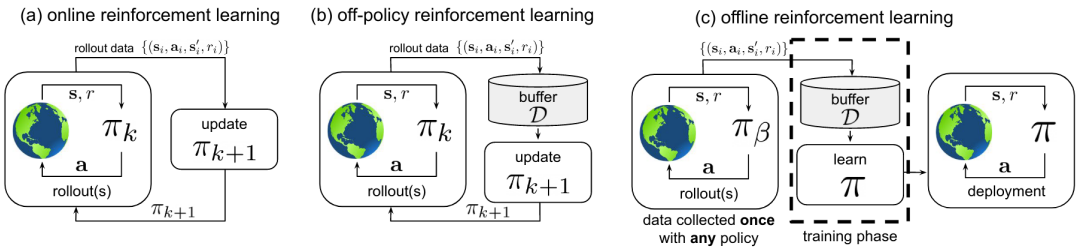


Figure 2.2: Illustration comparing the main reinforcement learning settings discussed showing environment interaction and how data are processed in: (a) on-policy online reinforcement learning, (b) off-policy online reinforcement learning and (c) offline reinforcement learning. From [16].

from the reward function [18]. The discounting factor, γ, of the reward hypothesis is comparable to the interest rate in finance and controls the value of present and future rewards.

To construct this formally, first define the policy, $\pi(a|s)$, a conditional probability distribution over actions, $a \in \mathcal{A}$, to take in each state, $s \in \mathcal{S}$. The policy interacts with the environment over a series of separate interactions, called episodes, of length H. A policy and MDP then define a trajectory distribution, p_π, of sequences of states and actions:

$$p_\pi(s_0, a_0, ..., s_H, a_H) = d_0(s_0) \prod_{t=0}^{H} \pi(a_t|s_t) T(s_{t+1}|s_t, a_t)$$

The reinforcement learning objective (the reward hypothesis) $J(\pi)$ can then be written:

$$J(\pi) = \max_\pi \mathbb{E}_{(s_0, a_0, ..., s_H, a_H) \sim p_\pi(\cdot)} \left[\sum_{t=0}^{H} \gamma^t r(s_t, a_t) \right] \tag{2.1}$$

An array of algorithms have resulted from the MDP formulation of control. In online reinforcement learning, algorithms gather and train on new interaction data. This data can be on-policy, where the data are collected under the current policy itself or off-policy where the data come from a buffer gathered under another policy, typically previous updates of the policy during training. In offline reinforcement learning, the data are gathered separately by a behaviour policy that interacts with the MDP. These types of reinforcement learning are illustrated in figure 2.2. Another class of techniques is model based reinforcement learning which explicitly models the system dynamics $T(s_{t+1}|s_t, a_t)$ and reward function $r(s_t, a_t)$ and uses these to plan behaviours.

To solve the reinforcement learning problem defined above, it is often useful to represent the objective $J(\pi)$. One such representation is the Q-value, $Q_\pi(s, a)$,

which is the expected sum of discounted rewards given you are in state s, take action a and thereafter act according to the policy:

$$Q^{\pi}(s,a) = \mathbb{E}_{(s_i,a_i)\sim p_{\pi}(\cdot)}\left[\sum_{k=0}^{\infty}\gamma^k r(s_{t+k},a_{t+k})\middle| s_t = s, a_t = a\right] \tag{2.2}$$

The optimal Q-function, $Q^*(s,a)$ represents the Q-values under the optimal policy (that maximises $J(\pi)$ in equation 2.1):

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a) \tag{2.3}$$

Combining this definition with equation 2.2 gives the Bellman optimality equations for Q. These equations express the fact that the Q-value of a state and action under the optimal policy must be the next reward received and then the best possible expected Q-value after that. Let a' be the next action and s' be the resulting next state:

$$Q^*(s,a) = \mathbb{E}\left[r(s,a) + \gamma \max_{a'} Q^*(s',a')\right]$$

The optimal policy can be defined as that which takes the best action in each state:

$$\pi^*(a|s) = \max_{a'} Q^*(s,a') \tag{2.4}$$

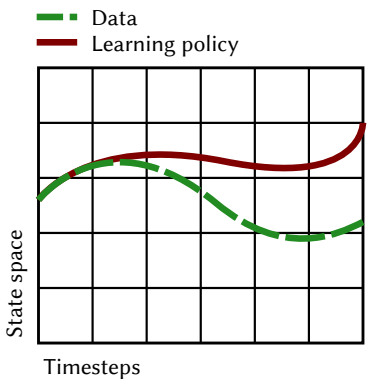
This is a cyclical definition since Q^* was defined based on the policy in equation 2.3. This relationship gives rise to bootstrapping and backup algorithms for iteratively computing Q-functions and policies.

2.1.2 Offline Reinforcement Learning

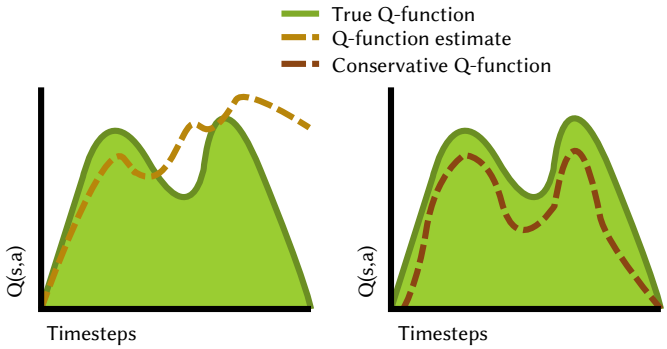
Offline reinforcement learning is different from the online case in that it trains from only a dataset $\mathcal{D}$, collected under a behaviour policy π_b (figure 2.2). The behaviour policy collecting data can be a pre-trained online reinforcement learning model or other controller. Even random environment interaction data can be used but less optimal data make offline reinforcement learning more challenging.

In theory, off-policy online reinforcement learning methods should apply to the offline case, since they can learn from data collected under a different policy. In practice, off-policy algorithms are typically used with online data from previous iterations of the policy during training. These off-policy methods have not worked well when directly applied to offline data [16].

The main issue with applying online reinforcement learning to the offline setting is the distribution shift. This shift is between the data under the behaviour



(a) The distribution shift problem.



(b) Q-values for out of distribution states and actions are overestimated (left), conservative regularisation can alleviate this (right).

Figure 2.3: Illustrative example of the distribution shift and resulting Q-value overestimation.

policy (the dataset) and the data under the learned policy [16]. As illustrated in figure 2.3, the policy will take different actions to the data leading to a different state-action distribution. This effect compounds over time as each deviation takes the policy further from the data distribution. Naive Q-function estimators then overestimate the Q-values for out of distribution states and actions. This over-optimism in values leads to poor choice of actions. This work uses conservative Q-learning (CQL). CQL regularises the Q-function updates to ensure out of distribution states and actions have more conservative Q-value estimates [4] (figure 2.3 (b, right)).

This distribution shift is also the main reason why standard deep learning techniques don't work well in the offline setting. However, a more fundamental issue is that the goal is to extract an optimal policy from the data which may not be the same as the behaviour policy. For example, offline reinforcement learning could learn to splice parts from two trajectories to create a path between two trajectories that was not seen in the data [19]. Deep learning instead aims to learn to match or imitate the behaviour in the data directly.

2.1.3 Robotic Grasping

This work explores the pick and reach robotic grasping task. The aim is to send a continuous control signal to a robot arm to grasp an object and lift it out of a tray to a threshold height. A PyBullet simulation [5] of a Franka robotic arm with a two finger gripper (see figure 2.1) is used.

Similar assumptions to [20] are made: the environment exposes robot joint angles, end-effector position and orientation and object position and orientation. Desired joint positions are the actions for the simulation. An oracle provides a task completion reward. The data used are gathered by a proportional-integral-derivative (PID) controller. More details about the experiment setup can be found in section 4.

To evaluate the robustness of the policies, the pick and reach task is extended by applying an external force requiring error recovery and by using perturbed physics engine parameters. These tasks are only used in the evaluation and so explore how well the trained models work in new dynamics and physics settings.

2.2 Related Work

2.2.1 Reinforcement Learning

A rich field of reinforcement learning has developed around the problem as described above. This section will introduce the online actor-critic approach based on deep deterministic policy gradients (DDPG) [21]. DDPG can be seen as an extension of deep Q-networks [8] to the continuous action space. The seminal text on reinforcement learning [18] provides more detail and information on other approaches.

Actor-critic utilises an actor, the policy $\pi_\phi(a|s)$ parameterised by ϕ, and a critic, the Q-function $Q_\theta(s,a)$ parameterised by θ, which evaluates actions.

The Bellman equation for Q is based on the fact that the definition of the Q-function (equation 2.2) can be rewritten recursively. Let s,a be the current states and actions and s',a' the next timestep states and actions, the Bellman equation is:

$$Q^\pi(s,a) = r(s,a) + \mathbb{E}_{(s',a')\sim p_\pi}[Q(s',a')] \tag{2.5}$$

The temporal difference (TD) error is the discrepancy of the Q-function estimates in the Bellman equation. This error gives an update objective for the Q-function parameters:

$$J^{TD}(\theta) = \min_\theta r(s,a) + \gamma Q_\theta(s',a') - Q_\theta(s,a) \tag{2.6}$$

In DDPG, the policy is then updated based on the Q-function definition of the optimal policy (equation 2.4). Intuitively, the Q-values are how good an action is in a state, so the policy should be updated to choose the value that is best in each state. Let $\mathcal{D}$ be a dataset of states:

$$J^{DDPG}(\phi) = \max_\phi \mathbb{E}_{s\sim\mathcal{D}}[Q_\theta(s,\pi_\phi(s))] \tag{2.7}$$

Twin delayed DDPG improves DDPG by applying double-Q learning and target policy smoothing [22]. Soft actor-critic also uses similar techniques [3].

Other actor-critic methods include asynchronous advantage actor-critic (A3C) [23] which uses multiple workers in parallel that accumulate gradient updates.

This was improved in actor-critic with experience replay (ACER) [24] which uses several modifications to A3C to construct a more sample efficient, off-policy method.

Another class of reinforcement learning algorithms that has proven effective is policy gradients. These methods update the policy directly without the need for learning a value or Q-value function, such as proximal policy optimisation (PPO) [9].

2.2.2 Soft Actor-Critic (SAC)

Soft actor-critic (SAC), is a state-of-the-art, off-policy, online actor-critic algorithm that extends the above with the maximum entropy reinforcement learning framework [3]. Entropy is a measure of the uncertainty in a random variable, $\mathcal{H}(p) = \mathbb{E}_{x\sim p}[-\log(p(x)]$. By explicitly maximising the entropy of the policy, SAC aims to improve exploration and learn more generalised policies. This is formulated as adding to the reward hypothesis objective (equation 2.1) a policy entropy term [3]:

$$J(\phi) = \max_{\phi} \mathbb{E}_{(s_t,a_t)\sim p_\pi(.)}\Big[\sum_{t=1}^{T} r(s_t,a_t) + \alpha\mathcal{H}(\pi_\phi(\cdot|s_t))\Big] \tag{2.8}$$

Such stochastic energy-based methods have been shown to work well in transfer learning settings [25] and thus might have some resistance to the distribution shift in offline reinforcement learning. SAC is used as a benchmark technique in this work, more details are in section 3.

2.2.3 Offline Reinforcement Learning

The following is an overview of the main directions in offline reinforcement learning research, a more in-depth survey is [16].

Policy constraint methods such as [26] mitigate the distributional shift by constraining the policy updates to stay close to the behaviour policy. This way, the Q-function is queried for states closer to the data distribution which can be more reliably estimated. This can be implemented as a penalty on the update targets or as a constraint on the policy updates.

Uncertainty-based offline reinforcement learning methods attempt to estimate the epistemic or model uncertainty of Q-function estimates. This uncertainty should be larger for out of distribution actions and can thus be used to produce conservative target values for the Q-function on such updates. The uncertainty can be modelled using, for example, bootstrap ensembles [27].

Similarly to off-policy reinforcement learning, model-based reinforcement learning methods can in theory apply to the offline setting by simply training the system dynamics model on the dataset. Some improvement can be made on this, such as incorporating model uncertainty to mitigate bias [28]. However,

model-based offline techniques suffer from model exploitation, a problem related to distribution shift, where the model handles out of distribution states and actions poorly.

Off-policy evaluation attempts to evaluate policies using data from another policy. This is useful for offline reinforcement learning because it allows evaluation and model selection without online interaction, further improving the data efficiency. Approaches to off-policy evaluation typically use importance sampling, but a recent state-of-the-art method instead uses a binary classification metric on the Q-function [29]. This evaluation correlates well with performance across several environments, including a robotics task. Another approach focuses on interpretable evaluation by highlighting transitions that have the most impact on the value estimates when removed [30]. Despite this progress, off-policy evaluation remains an open challenge with deep neural network policies so online evaluation is used in this work (that is, the policy is run in the environment).

Datasets for reinforcement learning (D4RL) [19], is a recent set of benchmark tasks, including several robotics tasks, with datasets and simulations that help to compare and evaluate offline reinforcement learning methods.

2.2.4 Conservative Q-Learning (CQL)

The offline reinforcement learning method used in this work is conservative Q-learning (CQL) [4]. It is a recent work that performs state-of-the-art on several D4RL offline reinforcement learning benchmarks. CQL augments standard actor-critic objectives described above with a Q-value regularisation term. This change leads to learning a Q-function under which the value of a policy lower-bounds its true value, thus mitigating the overestimation of Q-values. More details of CQL and its implementation in this work can be found in section 3.

CQL has been applied in to dexterous in-hand manipulation and kitchen robotics tasks [4]. An extension of CQL has also been applied to robotic manipulation tasks where it is shown to learn complex behaviours from a combination of unlabelled and task-specific datasets and to chain together behaviours on new tasks [31].

2.2.5 Robotic Grasping

Robotic grasping is a widely researched topic. Classical planning and control is a class of explicitly programmed approaches including proportional-integral-derivative (PID) control [32] and model predictive control [33] that have proved effective on a range of complex tasks. Engineering robotic manipulation requires knowledge of object motion and friction as well as the dynamics and kinematics of the robot hand, as developed in the Stanford/JPL robot hand [34]. Without modeling the object, tactile feedback sensing is used to determine grasping position and detect slipping [35].

This work focuses on learning behaviours from data. Data-driven learning approaches have proved effective when applied at a large scale. In [36], they gathered over 800,000 grasp attempts with 14 robotic arms over two months. Using this data, they learned a controller that was able to grasp novel objects and demonstrate error correction.

Reinforcement learning has been applied to robotics, such as [37] which trains robotic manipulation end-to-end from vision to control. Online Q-learning has been used with both on- and off-policy data to learn vision-based robotic manipulation [38]. In [39], multiple expert models are combined to produce adaptive locomotion behaviours. Each model is trained using reinforcement learning to achieve a different task. This hierarchical reinforcement learning approach is shown to generalise to new tasks. Deep reinforcement learning has also been combined with imitation learning of human demonstrations to learn humanoid locomotion tasks [40]. This is achieved by adding a data tracking reward to the task reward.

In this work, the robustness and generalisability of the models are evaluated in domains with varied physics and dynamics. A regrasping task is used where the object is forced out of the gripper. Learning such regrasping behaviour directly is explored in [41]. The second task used has randomised physics settings such as gravity, object mass and friction parameters. Domain randomisation [42] uses similar randomisation however they apply the perturbation during training whereas here it is applied only at test time.

This work focuses on a setting using one robotic manipulator, but the state-of-the-art in wider research uses bimanual manipulation. In [43], a set of base skills, such as top-grasp, are combined by a learned policy to complete a series of bimanual manipulation tasks, including several pick and reach tasks. Bimanual manipulation can also help in grasping objects that are otherwise difficult or impossible to grasp [44]. The two manipulators are used collaboratively to perform pregrasp manipulation that affords better grasping. These pregrasp behaviours are learned in a deep reinforcement learning framework and are effective over a range of objects both in simulation and transferred onto real hardware.

2.2.6 Human and Animal Behaviour

Although the aim is to extract optimal policies from data and not to mimic behaviours, it is insightful to compare the learning with the natural example of animals. In psychology, learning from other's behaviours is called observational learning [45]. Analogously to the reward signal in reinforcement learning, psychologists define vicarious reinforcement and punishment based on the outcomes observed of others' behaviours [45].

One study of chimpanzees demonstrated their mechanism of social learning [46]. A group of chimpanzees was seen to observe and model the drinking behaviour of another group. Young children are also able to learn behaviour from

demonstrations. In particular, a grasping task has been studied [47]. They found that demonstrations significantly improved the strategies the children employed to complete the task, hypothesising that it affected their task-relevant motor strategies.

Chapter 3

Approach

The approach used in this work is conservative Q-learning (CQL) [4]. The CQL system (figure 3.1), consists of three parts: the environment, the behaviour data and the model. The environment consists of the physics simulation and the task setup and is explained in section 4. The behaviour data are the offline demonstrations used to train CQL. These data are gathered by running a behaviour policy which in this case is a proportional-integral-derivative (PID) controller.

The training and evaluation procedure of the D4RL benchmark is used [19]. CQL is trained offline on the behaviour data but evaluated online in the environment.

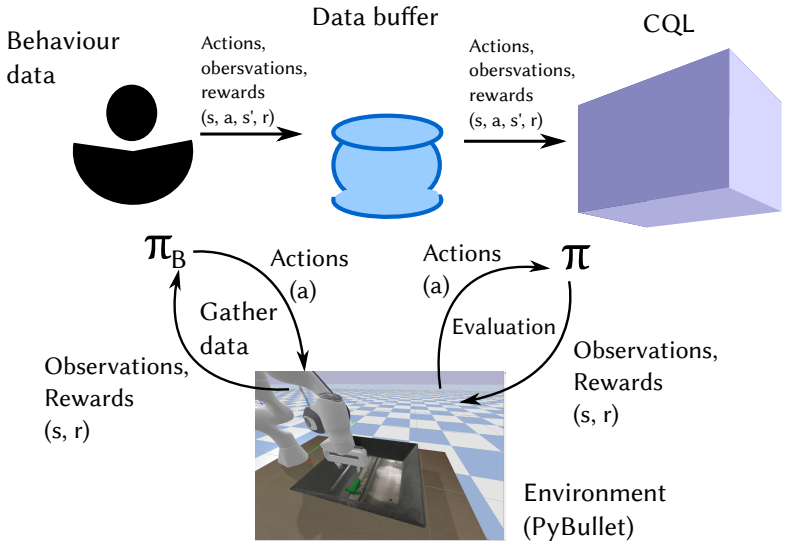


Figure 3.1: A system overview showing how the data flow. CQL training starts with gathering the behaviour data using the behaviour policy π_β. These data are then used to train a CQL policy which is evaluated online in the environment.

3.1 Proportional-Integral-Derivative Contoller

In this work, a proportional-integral-derivative (PID) controller is used for basic grasping behaviour. The PID controller in this work is based on [6]. It only uses the proportional and derivative terms not the integral term but the name PID is used for consistency.

The PID output computes a desired end-effector position. This position is combined with the desired orientation and an inverse kinematics solver [5] is used to obtain desired joint positions as the control signal.

Each dimension i is independently controlled by PID (without the integral term) as the proportional and derivative of the error [32]:

$$u(t)_i = K_p^i e_i(t) + K_d^i \frac{de_i}{dt}$$

Where $K_p \in \mathbb{R}^3$ and $K_d \in \mathbb{R}^3$ are the gains, $\mathbf{u}(t) \in \mathbb{R}^3$ is the output PID control for the end effector position and $\mathbf{e}(t) \in \mathbb{R}^3$ is the error signal at timestep t.

The error signal, $\mathbf{e}(t)$ is just the difference between the position of the end effector $\mathbf{x}_{ee} \in \mathbb{R}^3$ and the target $\mathbf{x}_{tgt} \in \mathbb{R}^3$.

$$\mathbf{e}(t) = \mathbf{x}_{tgt} - \mathbf{x}_{ee}$$

The PID gains, K_p and K_d, are parameters that balance the proportional and derivative terms and control the resulting behaviour. These gains were manually tuned to give the best performance of the system in the training environment.

In PID control, the proportional term is proportional to the error and used to reduce it directly. The derivative term provides stability through damping of oscillatory behaviour and helping prevent overshoot. The integral term is not needed with the simulation as it is for reducing steady-state error, it could be introduced for real-world settings. Overall, the proportional-derivative controller can be interpreted as the predicted system output, where the predictions are extrapolated into the future error using the error gradient [32].

Initially, for picking up the object, the PID controller is set to target the object position. When a threshold distance is reached, the fingers of the gripper are closed and the target position is set to a raised position to achieve the reaching behaviour. The desired end-effector orientation is set based on the object's orientation directly.

3.2 Soft Actor-Critic (SAC)

Soft actor-critic (SAC) [3] is used in this work as a benchmark to compare to conservative Q-learning (CQL), the main offline learning algorithm used. SAC

is an online reinforcement learning algorithm however it is also off-policy, so it can learn from data gathered under different policies. Therefore, it is a good comparison as it can be trained online whilst also being provided with the behaviour data.

SAC is part of the maximum entropy reinforcement learning framework. This augments the policy objective with a term rewarding the entropy of the policy (equation 2.8). The introduction of entropy encourages more exploration and capturing a more diverse set of behaviours [3].

SAC augments the actor-critic policy update (equation 2.7) with the policy entropy:

$$J_{SAC}(\phi) = \min_{\phi} \mathbb{E}_{(s_t,a_t)\sim p_\pi(.)}\Big[\alpha \log \pi_\phi(a_t|s_t) - Q_\theta(s_t,a_t)\Big] \tag{3.1}$$

In this work, SAC is used with only Q-functions and a policy (no value functions) to match the architecture of CQL. The resulting procedure is the same as algorithm 1 below except the update for the Q-functions, $J_{CQL}(\theta)$. The SAC objective for Q includes the policy entropy in the Bellman equation (equation 2.5) [48]:

$$J_{SAC}(\theta) = \mathbb{E}_{(s_t,a_t)\sim\mathcal{D}}\Bigg[\frac{1}{2}\Bigg(Q_\theta(s_t,a_t) - \Big(r(s_t,a_t) + \gamma Q_{\hat{\theta}}(s_{t+1},a_{t+1}) - \alpha \log \pi_\phi(a_{t+1}|s_{t+1})\Big)\Bigg)^2\Bigg]$$

The implementation used is based on the original authors' open-source code but uses a different network structure to more closely match the CQL implementation [3].

3.3 Conservative Q-Learning (CQL)

Recall that the core issue with offline reinforcement learning is the distribution shift between the behaviour policy data and the learning policy. This leads to an overestimation of Q-values for the out of distribution states and actions and so poor actions are chosen. Conservative Q-learning (CQL) addresses this overestimation by ensuring low-value estimates on out of distribution states and actions. In this way, the learned policy will be trained to keep closer to the known actions in the behaviour data rather than erroneously favouring the overestimated values.

To obtain such a lower-bound on the true Q values, CQL uses an objective, $\hat{Q}^\pi_{CQL}$, combining the squared temporal difference error (equation 2.6), with a

regularisation term (first line):

$$\hat{Q}^{\pi}_{CQL} := \arg\min_{Q} \alpha \cdot \Big(\mathbb{E}_{s\sim\mathcal{D}, a\sim\mu(a|s)}[Q(s,a)] - \mathbb{E}_{s\sim\mathcal{D}, a\sim\hat{\pi}_{\beta}(a|s)}[Q(s,a)] \Big) + \frac{1}{2}\mathbb{E}_{s,a,s'\sim\mathcal{D}} \left[\Big(r(s,a) + \gamma \mathbb{E}_{\pi}[\hat{Q}(s',a')] - Q(s,a) \Big)^2 \right] \tag{3.2}$$

Where (s, a, s') are state, action, next state triples; $r(s,a)$ is the reward; $\hat{\pi}_{\beta}(a|s)$ is the behaviour policy which produced the dataset $\mathcal{D}$; $\mu(a|s)$ is a distribution over actions (used in the regulariser) and $\hat{Q}(s,a)$ is the Q-function approximator. Intuitively, the CQL regularisation term minimises the Q-value estimates over the $\mu(a|s)$ distribution and maximises them over those of the behaviour policy, $\hat{\pi}_{\beta}(a|s)$.

Fitting a Q-function to this objective, the expected value of a policy provably lower-bounds its true value. That is (for sufficiently large α) [4]:

$$\mathbb{E}_{s\sim d_0, a\sim\pi(s)}[\hat{Q}^{\pi}_{CQL}(s,a)] \leq \mathbb{E}_{s\sim d_0, a\sim\pi(s)}[Q^{\pi}(s,a)]$$

Developing this into an offline reinforcement learning algorithm, a family of CQL update equations can be formulated based on the choice of action distribution $\mu(a|s)$ in equation 3.2 which impacts the regularisation. In this work, $\mu(a|s) \propto \text{Unif}(a) \cdot \exp(Q(s,a))$ is used which gives the Q-function update objective (called CQL($\mathcal{H}$) in [4]):

$$J_{CQL}(\theta) = \min_{Q} \alpha \mathbb{E}_{s\sim\mathcal{D}} \Big[\log \sum_{a} \exp(Q(s,a)) - \mathbb{E}_{a\sim\hat{\pi}_{\beta}(a|s)}[Q(s,a)] \Big] + \frac{1}{2}\mathbb{E}_{s,a,s'\sim\mathcal{D}} \left[\Big(r(s,a) + \gamma \mathbb{E}_{\pi}[\hat{Q}(s',a')] - Q(s,a) \Big)^2 \right] \tag{3.3}$$

The $\log \sum_a \exp(Q(s,a))$ term is potential problematic for over/underflow so an importance sampling approximation is used with $N = 10$ samples from $\text{Unif}(a)$ and from the training policy $\pi(a)$ [4]:

$$\log \left(\frac{1}{2N} \sum_{a_i\sim\text{Unif}(a)}^{N} \left[\frac{\exp(Q(s,a_i))}{\text{Unif}(a)} \right] + \frac{1}{2N} \sum_{a_i\sim\pi(a|s)}^{N} \left[\frac{\exp(Q(s,a_i))}{\pi(a_i|s)} \right] \right)$$

The policy is updated using the entropy regularised policy gradient from SAC, $J_{SAC}(\phi)$ (equation 3.1).

This summarises the main contribution of CQL and results in a relatively small change to standard actor-critic algorithms. In this case, CQL is adapted from SAC [3]. The CQL training procedure is outlined in algorithm 1.

The implementation of CQL used in this work is based on an open-source implementation by the original authors [4].

Algorithm 1: Conservative Q-learning training (red shows difference from actor-critic algorithms such as SAC) [4]

Input: A Q-function, Q_θ with parameters θ_0; a policy, π_ϕ, with parameters ϕ_0; learning rate η and number of training epochs N

Output: Trained policy π_{ϕ_N}

1 **for** *step* t *in* $\{1,...,N\}$ **do**

2 Update the Q-function using the CQL regularised TD-error (equation 3.3): $\theta_t = \theta_{t-1} - \eta \nabla_\theta J_{CQL}(\theta)$;

3 Update the policy π_ϕ with SAC-style entropy regularisation (equation 3.1): $\phi_t = \phi_{t-1} + \eta \nabla_\phi J_{SAC}(\phi)$;

4 **end**

3.4 Architecture

Both the implementations of SAC and CQL in this work makes use of double Q-networks and target Q-networks (figure 3.2) [4], [3].

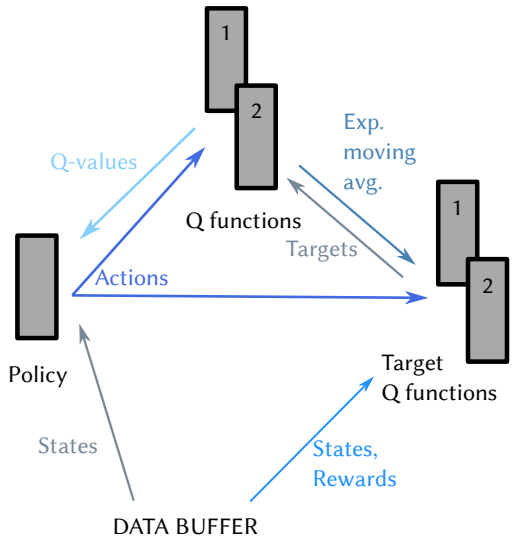


Figure 3.2: CQL uses multiple neural networks. It uses a policy, double Q functions and target Q networks during training. Only the policy network is used for evaluation.

Double Q-networks were originally introduced in [49] to address overestimation of action values in online reinforcement learning and can speed up training. Here the variant from [22] is used. Two Q-functions, $Q_{\theta^1}(s,a)$ and $Q_{\theta^2}(s,a)$, are maintained with separate parameters, θ^1 and θ^2. These Q functions are then independently updated based on $J_{CQL}(\theta)$ (line 2 in algorithm 1). The minimum of their estimates is used to update the policy (line 3 in algorithm 1), $Q_\theta(s,a) = min\{Q_{\theta^1}(s,a), Q_{\theta^2}(s,a)\}$.

Target Q-networks were developed in [8], the actor-critic variant is used here

[50]. This method introduces target networks, $Q_{\theta^i}^{TARG}$, for each of the Q-networks. These target networks are updated as exponentially moving averages, parameterised by τ, of the Q-networks and thus track the Q-network updates at a delay. The target Q-networks are used to compute the target values for the Q-network updates (line 2 in algorithm 1). This helps improve stability since the Q-network being updated is itself used in the target for the update which can lead to instability and divergences in training.

All four Q-networks are feed-forward neural networks, as is the policy network. They all share the same architecture, shown in figure 3.3, with three hidden layers of dimension 256 using ReLU non-linearity ($f(x) = max(x, 0)$) [51]. The Q-networks' input is the current state from the environment and the action to be evaluated (from the data or the policy). The Q-networks output a scalar estimating the Q-value, $Q(s, a)$. The policy takes the state as input and outputs the action, the desired joint positions for the robot. Details of the states and actions used in this work can be found in section 4.

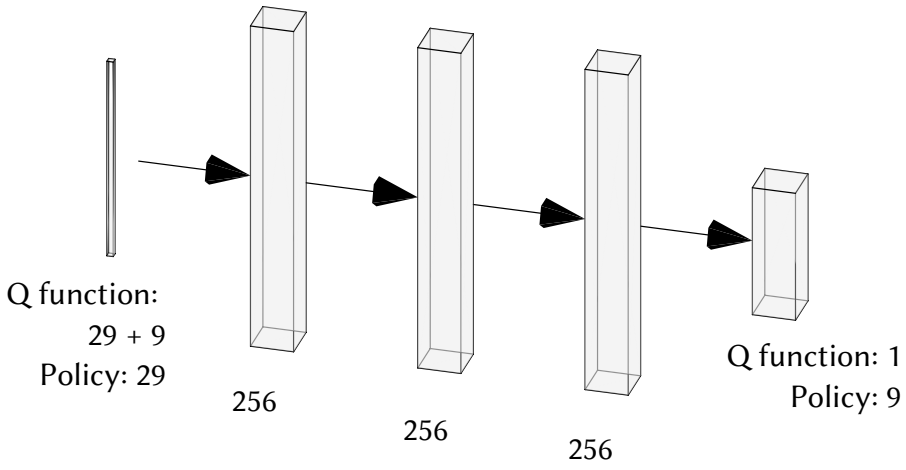


Figure 3.3: The feed-forward neural network architecture and layer sizes used in the Q-functions and policy. The input size is 29 for the state and 9 for the actions. The output size is 1 for the Q-value estimate and 9 for the actions. Generated with [52].

Chapter 4

Experiments

4.1 Simulation

The experiments are based on an open-source simulation [6] of a Franka Emika Panda robot arm with a two-finger gripper shown in figure 2.1. The simulation uses PyBullet [5] for the physics engine and rigid body dynamics. The manufacturers provide the official universal robot description file (URDF) for the Panda arm [53]. In the simulation, a small cylindrical object is dropped in the tray at a random position and orientation (within constraints to ensure it is reachable). The task is for the robot to pick up this object and raise it above a threshold height.

As defined in section 2, reinforcement learning requires defining states, actions, and rewards (figure 4.1).

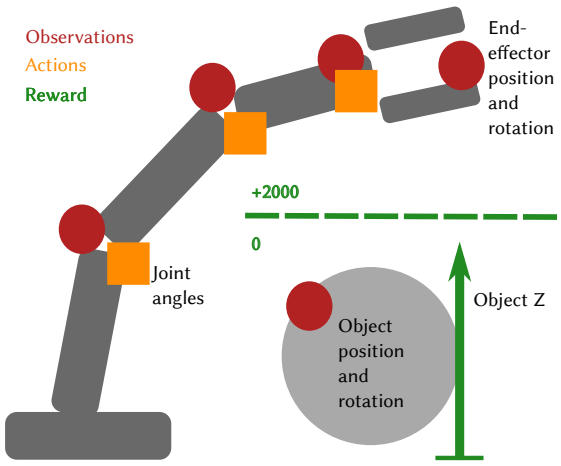


Figure 4.1: The reinforcement learning environments have states as the robot and object position, actions as desired joint positions and reward based on a threshold of the object Z position.

The states are based on similar assumptions to prior manipulation reinforcement learning work [20]: a good proprioceptive sensor system to get good joint

states (the real Panda arm has absolute joint position encoders); good localisation of the end-effector position and a good exteroceptive sensor system (e.g. LiDAR) to locate the object in world space. This information for the state is exposed in the simulation and the environment gathers it directly each timestep. The state is a a vector, $s \in \mathbb{R}^{29}$ (dimensions in brackets):

- (9) Joint positions for the controlled joints
- (3) End effector position in world space
- (4) End effector orientation quaternion
- (3) Object position in world space
- (4) Object orientation quaternion
- (6) Distance from object to each of the two fingers in world space

The actions, $a \in \mathbb{R}^9$, sent by the policy to the environment are the desired joint angles. 9 out of 11 of the joints are controlled. Internally these are passed to the joint position controller of the PyBullet simulation.

Rewards are sparse, that is, zero for all timesteps except the last which is 1 on successful completion. An oracle provides the reward from the environment when the object Z coordinate in world space (its height) reaches a threshold of 0.4 m. This threshold was manually set to be reachable whilst being high enough to validate effective pick and reach behaviour. This relies on object position data and so requires no more assumptions than the state description. There is a 700 timestep limit to complete the task.

4.2 Environments

Three variations of the above simulation framework are used. Panda-v0 is the standard environment for training and evaluation. PandaForce-v0 is a regrasping task and PandaPerturbed-v0 is a task with perturbed physics simulation parameters, both used for evaluation. The simulation and physics parameters are listed in appendix A.

Panda-v0 is the simulation as described with fixed physics parameters set to realistic values and no external forces.

PandaForce-v0 is a regrasping task such that on the initial pickup, at a height of 0.2 m (below the threshold), an external force is applied to the object to force it out of the grasp. The force is only applied once and so to solve this environment the policy must learn to regrasp the object and pick it up a second time. This is useful in examining the policy's robustness and how it handles errors.

In PandaPerturbed-v0, each new episode of the environment is initialised with randomly sampled physics parameters. Specifically, the gravity can range from -20 to 0.5 m/s^2, the object mass is sampled from 0.01-10 kg and the lateral friction coefficient from 0.01 to 1. Some of these boundaries, in particular positive

gravity, are unrealistic but intended to test the model under new and different dynamics. In this way, the ability of the models to generalise is tested.

4.3 Experiment Setup

As a scientific community, reproducibility is an important part of reinforcement learning research, but some results have proved challenging to replicate [54]. To ensure rigorous experimentation all results are averaged over five random seeds with hyperparameters set based on well-tested values adjusted with a small search. The reinforcement learning models are run for 100 epochs with 1,000 training steps per epoch. CQL is trained offline whereas SAC receives the offline data but also trains online so gathers more data. Further details of the parameters can be found in appendix A.

The PID agent is not trained and is a deterministic function. Nonetheless, it is run for each random seed both to gather data and to evaluate, since the environment dynamics contain some stochasticity such as the initial object position. 10,000 episodes (approximately 3 million timesteps) of data are collected under the PID controller. Every episode in the data completes the task successfully.

CQL is trained for each random seed offline using the PID data. It is evaluated online during and after training in an environment instance with the same seed. Hyperparameters are manually adjusted from the original authors' choices and technical report [4].

The SAC benchmark is trained for each random seed online in an environment with the random seed set. The evaluation during training occurs in another instance of the environment with the same random seed. The hyperparameters used are based on the original author's work and adjusted with a small search [3]. Since this work is exploring data-driven approaches, the replay buffer of SAC is initialised with the PID training data to learn from. Unlike CQL, SAC also interacts with the environment online throughout training to gather more data.

For evaluation, models are saved and evaluated for 1,400 timesteps every epoch. The model with the best mean reward in evaluation during training is reported. This model is run in each of the test environments for 100 episodes with the same random seed as training.

As a reinforcement learning problem, for both training and evaluation, the metric of interest is the mean episode return (sum of rewards in an episode). Since a sparse reward is used a return of 1 indicates success and zero indicates a failure on that episode. As such, the mean return can also be seen as the mean success rate for that environment, with similar interpretations for the confidence intervals.

Chapter 5

Results

CQL is successfully able to learn to grasp from data. It completes the training task successfully 73% of the time compared to 100% by the PID agent. The learned CQL policy also performs similarly to the PID controller in generalising to new physics settings and outperforms PID control on robustness under external forces. CQL significantly outperforms SAC which is not able to complete the task.

5.1 Training

Training CQL proved to be quite unstable but did somewhat follow a pattern across random seeds. Figure 5.1 shows the online evaluation (in the environment) during training. There are a few early spikes as it begins to learn and then the main period of improvement from around epoch 20. For all random seeds, the best model was at epoch 27, this was used for evaluation. Later on in training, the performance degrades quite abruptly after 30 epochs. This appears to be due to overfitting to the offline data leading to policy and Q-value estimates that do not match the environment.

Figure 5.2 shows the policy loss and Q-function predictions during training, this gives us more insight into the potential overfitting. The loss decreases initially as the reward increases in the evaluation plot. The loss spikes around epoch 20 then decreases again from around epoch 30. This spike and decline is evidence of overfitting because the loss is decreasing, so the training is ostensibly improving but at the same time, the evaluation reward is getting worse. In the Q-function plots, around this time the Q-function predictions also start to spike. This reaches over 1 at around 40 epochs. This spike is indicative of overconfidence since the highest sum of returns possible is 1 for successful task completion. Thus, when the evaluation mean reward decreases, the policy loss and Q-function predictions both appear to be improving so this instability at epoch 30 is likely due to overfitting.

SAC appears to learn very little during training (figure 5.1) compared to CQL,

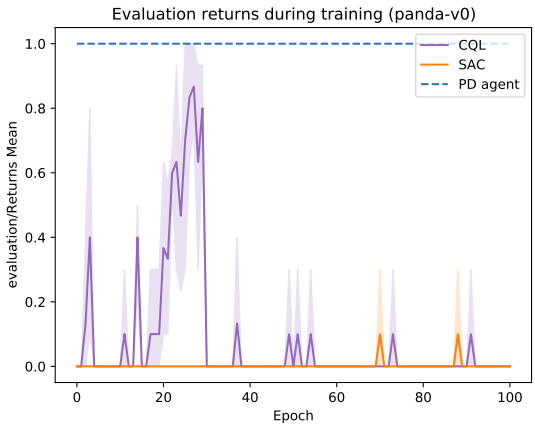


Figure 5.1: The mean and 95% confidence interval (shaded) of episode returns in online evaluation during training. This can be interpreted as success rate. These are evaluated over 5 random seeds. The PID agent is not trained but is included as a benchmark.

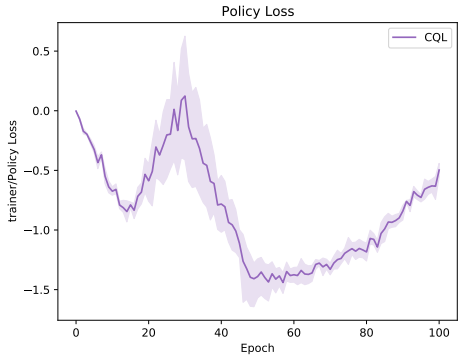
despite the additional online data it gathers. It has a few small spikes, but the mean reward never surpasses 0.1. This suggests SAC as implemented cannot effectively make use of the data to learn this complex domain. As an off-policy algorithm, SAC is capable of learning from the behaviour data, but this result shows that in practice, using data in this way is not effective for learning complex behaviours offline. SAC is also not able to learn online in this case likely due to the sparsity of the reward and the complexity of the task. As it achieves very few successes during training, it receives a very limited reward signal to train from.

5.2 Evaluation

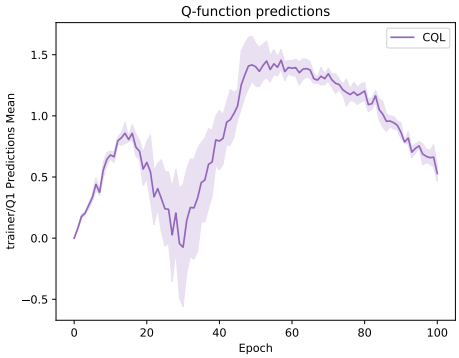
Figure 5.3 and table 5.1 show the performance of the best models during training in an online evaluation of 100 episodes and averaged across the 5 random seeds. Overall, CQL performs almost as well as the PID controller across all evaluation tasks and slightly outperforms it on PandaForce-v0. SAC evaluated poorly on all tasks.

On Panda-v0, CQL achieves a 73% success rate where the PID agent achieves 100% success. On the best random seed, CQL achieved 90% success. This suggests that CQL has learned effectively from the behaviour data and can complete the task consistently.

In PandaPerturbed-v0, the PID agent achieves a 34.8% success rate and CQL achieves 19.5%. These results are indicative of the range of physics settings sampled under which the systems can complete the task. Thus, CQL has learned a policy that can generalise well since it was trained only on the standard physics settings and has learned to react to different settings. The PID agent is quite



(a) The policy loss during training (line 3 in algorithm 1)



(b) The Q-function predictions during training.

Figure 5.2: Shows the mean and 95% confidence interval (shaded) of the CQL policy loss and Q-function predictions during training (across 5 random seeds).

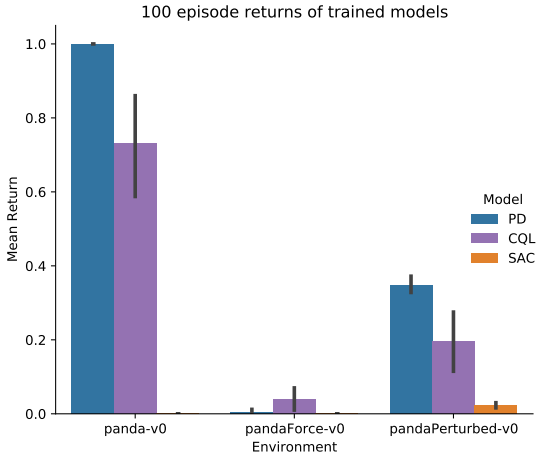


Figure 5.3: The evaluation mean and 95% confidence interval (lines) of the episode returns in the three environments. Evaluation is over 100 episodes and averaged over 5 random seeds. This can be interpreted as task success rate.

a simple, hard-coded controller and so there are cases in which the standard grasping behaviour does not work well. Some settings were quite unrealistic to test the limits of the policies. Two particularly difficult settings are with positive gravity which makes the object harder to track before grasping and very low friction coefficients which leads to the object sliding when touched making grasping much harder.

The PandaForce-v0 environment was the most challenging. CQL achieves 4%, the PID agent achieves 0.4% and the SAC controller achieves none. When the object is forced out of the gripper it moves around in the tray which can lead the PID controller to get into difficult configurations as it tracks the object closely. The object can often bounce out of the tray or into a difficult to reach corner. CQL's significantly superior performance shows that it has learned a more ro-

	Panda-v0		PandaForce-v0		PandaPerturbed-v0	
	Mean	Std	Mean	Std	Mean	Std
CQL	0.73	0.17	**0.040**	0.036	0.195	0.099
PID	**1.00**	0.00	0.004	0.009	**0.348**	0.028
SAC	0.00	0.00	0.000	0.000	0.024	0.009

Table 5.1: Mean and standard deviation of evaluation returns of the best models. Run for 100 episodes and averaged across 5 random seeds. Bold indicates the best result in each environment.

bust policy than the PID agent and is better able to recover from errors. Furthermore, this demonstrates the ability of CQL to extract a more optimal policy from the dataset, based on the rewards, than the behaviour policy that generated the data.

The final performance of SAC is poor showing the challenge of these tasks. It is not able to complete the Panda-v0 or PandaForce-v0 tasks successfully at all. SAC does achieve a small success rate of 2.4% on PandaPerturbed-v0. This indicates that it did manage to complete this task successfully, but this is due to exploiting the randomly sampled physics settings such as positive gravity cases.

5.3 Learned Behaviours

Figure 5.6 (at the end) shows the learned behaviours of each model in the standard Panda-v0 environment. Videos showing multiple runs of each model in each environment are also available[1]. These videos show the first three episodes in full for the same random seed.

One of the first observations of the behaviour is that, as expected from the training and evaluation results, the baseline SAC model is not able to effectively complete the task. It often demonstrates a lack of control and misses the tray as it does in figure 5.6.

In contrast, this figure also shows how well the CQL model matches the grasping behaviour of the PID agent it learned from, with both completing the task well. CQL has learned to get a good initial placement of the end effector and to grasp the object well, learning only from the PID offline data.

It is interesting to compare these learned behaviours to natural examples in animal and behavioural research. In figure 5.4 a study with chimpanzees' grasping behaviours and preferences [55], shows similar reaching and grasping to the results with CQL in figure 5.6. Both can be seen reaching over the object, then closing the fingers to grasp and pick it. The animal studies also examined the end-state comfort effect, that is whether, like humans, primates adapt

[1]Videos of each trained model here and in the additional materials.

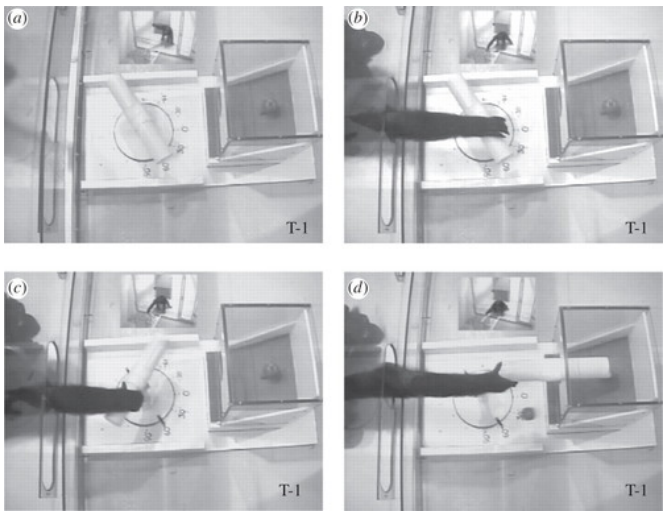


Figure 5.4: Chimpanzee grasping behaviour on a pick and place task. Taken from [55].

their initial grip in anticipation of further task-orientated actions. In a similarity to reward-based CQL, they found that chimpanzees can predict the costs of future actions and use this to select their movements [55]. In [47], young children were examined in a similar grasping task. They found children from 3 years old were able to adapt the type of grip used in response to the demands of the task.

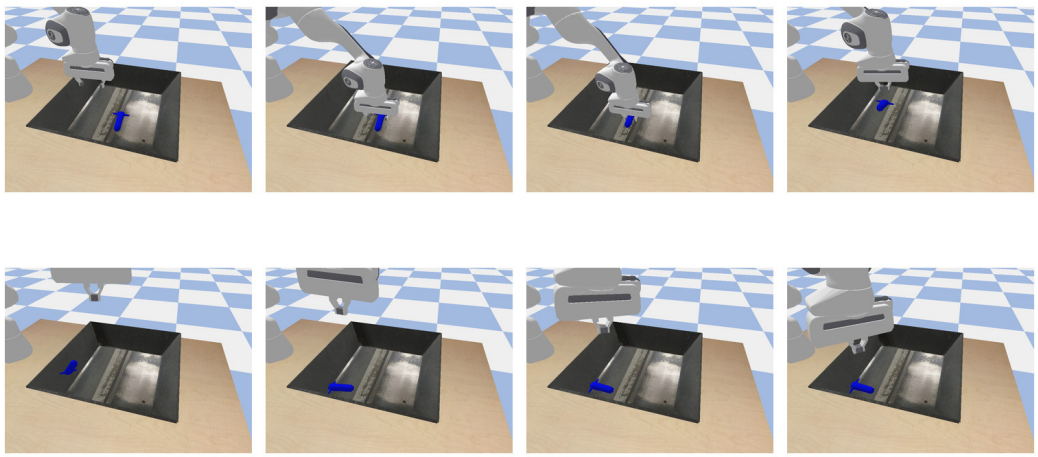
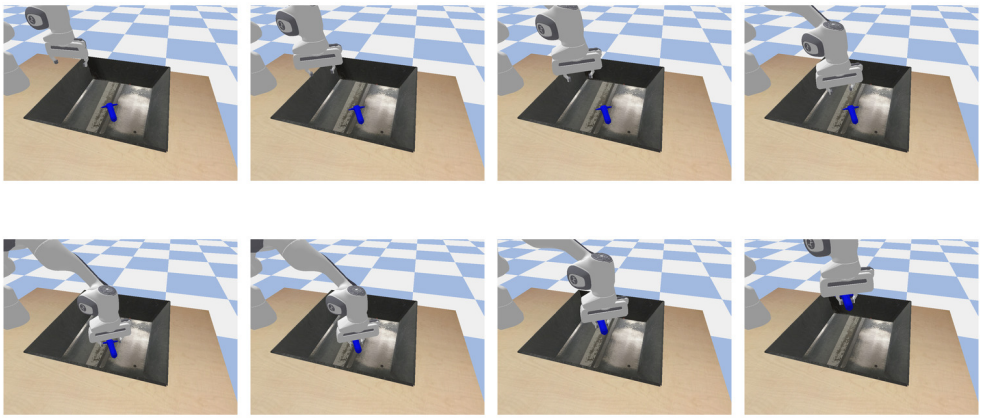


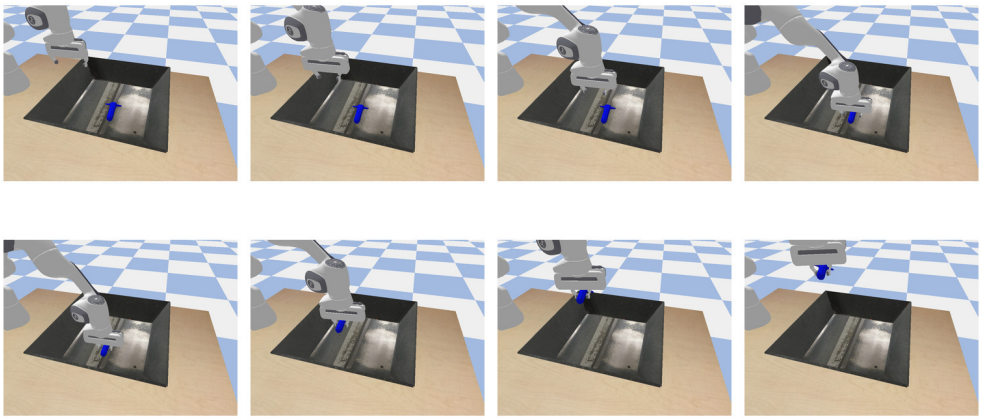
Figure 5.5: Sequence of images showing the behaviour of CQL on the PandaForce-v0 task.

Figure 5.5 shows a behaviour sequence of the CQL agent in the PandaForce-v0 (regrasping) environment. After a successful initial pick and reach, the object is pushed out of the gripper by the simulation. Although it did not manage to regrasp and complete the task in this case, CQL demonstrates good tracking behaviour as it continues to approach the moving object. Sometimes the CQL

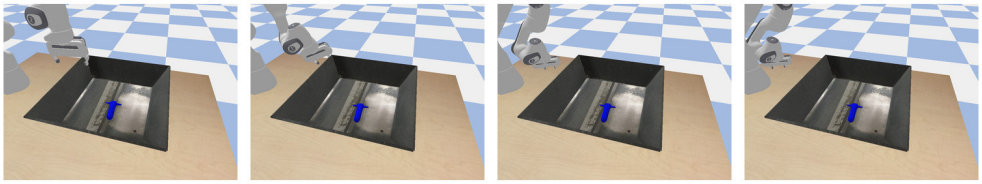
model was able to grip the object throughout the external force and not drop the object. The external force was set based on the PID agent, so this shows that perhaps the CQL model was able to learn a better grasp behaviour that is more resistant to such forces.



(a) PID agent



(b) CQL



(c) SAC

Figure 5.6: Sequences of images showing the behaviour of the trained models in the Panda-v0 task. The PID agent and CQL both complete the task. With SAC the robot misses the tray. Note the snapshots are taken at different timestep intervals for each model.

Chapter 6

Conclusion

This work started by asking whether robust and generalisable autonomous grasping behaviour can be learned from data. In a review of the literature, offline reinforcement learning was chosen as the framework to address this question, in particular, conservative Q-learning (CQL) was chosen [4]. Three pick and reach tasks were used to evaluate the model based on a PyBullet robot arm simulation [5]. CQL was trained on data from a proportional-integral-derivative (PID) controller [32].

In these experiments, CQL was successfully shown to perform the tasks comparably to the PID controller both in terms of the learned behaviours and the task success rate. CQL even outperformed PID in a regrasping task showing it is able to extract a more optimal policy than that in the data. This task also shows that CQL has learned a policy that is robust under external forces and able to recover from errors. CQL was also able to generalise to some extent under highly varied physics settings. Soft actor-critic (SAC) [3], a state-of-the-art online reinforcement learning algorithm on which CQL is built was shown to not be able to learn effective behaviour from the same data even with additional online interaction. Overall, it was shown that CQL does contribute to improved training in the offline setting as it was able to learn grasping behaviour from data. This approach is promising for learning more complex robot behaviours from data.

6.1 Future Work

Following on from the success CQL showed in this domain, several research directions can take this work further.

6.1.1 Curriculum Learning & Combining Tasks

An interesting area in deep and reinforcement learning is multi-task learning. In [39] multiple expert models are combined to create complex quadruped locomotion behaviours that can adapt to unseen scenarios. To achieve this, each

model is trained by reinforcement learning and combined with a gating neural network. A similar approach could be applied to combine offline reinforcement learning models such as those trained with different grasping tasks.

Large, unlabelled datasets also present an opportunity to learn diverse and generalisable behaviour. Offline reinforcement learning can be used to stitch together sub-trajectories from the data. An interesting extension would be to uses datasets containing multiple tasks to examine the learned task combinations. This approach has been used to complete robotic grasping tasks from new initial conditions that do not have a full demonstration in the data [31]. An example of this combination is when the target object is in a drawer requiring a combination of draw opening and object grasping tasks.

6.1.2 Regrasping and Error Recovery Behaviour

Task combination could be used to learn error-correcting behaviours and regrasping. This is done with the fall recovery behaviour in [39]. Regrasping is explored in more detail in [41]. They show regrasping can be learned by training an online reinforcement learning algorithm from different initial states so that the model experiences failures and can learn to recover.

Emergent examples of regrasping behaviour were examined in this work with the PandaForce-v0 task that forced the object out of the grasp. Explicitly handling regrasping' in the offline setting could be achieved through augmenting the data with error recovery examples or through additional online training after the offline training. Incorporating error recovery behaviour could lead to more robust policies as they can persevere and regrasp if the initial grasp fails.

Another extension in this direction could utilise bimanual manipulation. Deep reinforcement learning has been applied to bimanual pregrasping behaviours that help to grasp in difficult or ungraspable settings [44]. Similar work could be used to improve the resilience to and recovery from errors, especially in difficult configurations that the object can reach when dropped.

6.1.3 Sub-Optimal Data

Finally, the data used in this work was from a PID controller which, whilst simple, achieved a 100% success rate on the training task, Panda-v0. A key aspect of offline reinforcement learning is to be able to extract the optimal policy (that maximises the reward signal). As such, it is interesting to explore the use of sub-optimal data, such as human motion data. Human data are used in [40] to combine reinforcement and imitation learning to bias the learning towards human behaviour. Human data also presents an interesting variety where different participants have different approaches to achieve the task.

In the extreme, random data in the environment could be used. In this way, the learning agent has to extract the policy based on the reward signal as the data are not based on any meaningful policy. This would also be useful in

certain domains where data from an expert is difficult to achieve. Datasets for deep data-driven reinforcement learning, a set of benchmark tasks with data for offline reinforcement learning, include multiple tasks with data gathered through taking random actions [19]. This is a challenging but interesting area to explore.

Appendix A

Experiment Details

Tables A.1 and A.2 show the main parameters used in this work. Full code is available for further review[1].

Hyperparameter	Value
Epochs	100
Train steps per epoch	1000
Max steps per episode	700
Eval steps per epoch	1400
Steps before training	1400
Batch size	352
Replay buffer size	2×10^6
Discount	1000
Policy learning rate	3×10^{-5}
Q learning rate	3×10^{-4}
Q target update τ	5×10^{-3}

Table A.1: The hyperparameters for training SAC and CQL. Found using a small hyperparameter sweep starting from initial settings used in other works [4], [3].

[1]The code is available here and in the additional materials.

	Panda-v0	PandaForce-v0	PandaPerturbed-v0	
			Low	High
Gravity	-10	-10	-20	0.5
Mass of object	0.1	0.1	0.01	10
Lateral friction of object	0.1	0.1	0.01	1

Table A.2: The physics parameters used in the environments. Panda-v0 and PandaForce-v0 use the same settings except for the external force (which Panda-v0 doesn't use). PandaPerturbed-v0 samples from the ranges shown.

Bibliography

[1] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, USA, 3rd edition, 2009.

[2] Tanya M. Anandan. Robotics industry insights, 2020. https://www.robotics.org/content-detail.cfm/Industrial-Robotics-Industry-Insights/Industry-Trends-and-Market-Potential-What-s-Next/content_id/9391 (Accessed: 15th March 2021).

[3] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. volume 80 of *Proceedings of Machine Learning Research*, pages 1861–1870, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

[4] Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning, 2020.

[5] Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. http://pybullet.org, 2016–2019.

[6] Mahyar Abdeetedal. Gym panda. https://github.com/mahyaret/gym-panda, 2020.

[7] David Silver et. al. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[8] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. 518(7540):529–533, February 2015.

[9] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

[10] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[11] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014.

[12] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.

[13] Dominic King. Deepmind's health team joins google health. https://deepmind.com/blog/announcements/deepmind-health-joins-google-health, 2019. Accessed: 21st October 2020.

[14] Paul Covington, Jay Adams, and Emre Sargin. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*, New York, NY, USA, 2016.

[15] Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alexander Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. In *Arxiv*, 2016.

[16] Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems, 2020.

[17] Stéphane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell. No-regret reductions for imitation learning and structured prediction. *CoRR*, abs/1011.0686, 2010.

[18] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. A Bradford Book, Cambridge, MA, USA, 2018.

[19] Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4rl: Datasets for deep data-driven reinforcement learning, 2020.

[20] 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 *Proceedings of Robotics: Science and Systems (RSS)*, 2018.

[21] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ICML'14, page I–387–I–395. JMLR.org, 2014.

[22] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods, 2018.

[23] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning, 2016.

[24] Ziyu Wang, Victor Bapst, Nicolas Heess, Volodymyr Mnih, Remi Munos, Koray Kavukcuoglu, and Nando de Freitas. Sample efficient actor-critic with experience replay, 2017.

[25] Tuomas Haarnoja, Haoran Tang, Pieter Abbeel, and Sergey Levine. Reinforcement learning with deep energy-based policies. volume 70 of *Proceedings of Machine Learning Research*, pages 1352–1361, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR.

[26] Aviral Kumar, Justin Fu, George Tucker, and Sergey Levine. Stabilizing off-policy q-learning via bootstrapping error reduction. *CoRR*, abs/1906.00949, 2019.

[27] Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. Deep exploration via bootstrapped dqn, 2016.

[28] Marc Peter Deisenroth and Carl Edward Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ICML'11, page 465–472, Madison, WI, USA, 2011. Omnipress.

[29] Alex Irpan, Kanishka Rao, Konstantinos Bousmalis, Chris Harris, Julian Ibarz, and Sergey Levine. Off-policy evaluation via off-policy classification, 2019.

[30] Omer Gottesman, Joseph Futoma, Yao Liu, Sonali Parbhoo, Leo Anthony Celi, Emma Brunskill, and Finale Doshi-Velez. Interpretable off-policy evaluation in reinforcement learning by highlighting influential transitions, 2020.

[31] Avi Singh, Albert Yu, Jonathan Yang, Jesse Zhang, Aviral Kumar, and Sergey Levine. Cog: Connecting new skills to past experience with offline reinforcement learning, 2020.

[32] Karl Johan Astrom and Richard M. Murray. *Feedback Systems: An Introduction for Scientists and Engineers*. Princeton University Press, USA, 2008.

[33] M. Logothetis, G. C. Karras, S. Heshmati-Alamdari, P. Vlantis, and K. J. Kyriakopoulos. A model predictive control approach for vision-based object grasping via mobile manipulator. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1–6, 2018.

[34] Matthew T. Mason and J. Kenneth Salisbury. *Robot Hands and the Mechanics of Manipulation*. MIT Press, Cambridge, MA, May 1985.

[35] Ronald S. Fearing and John M. Hollerbach. Basic solid mechanics for tactile sensing. *The International Journal of Robotics Research*, 4(3):40–54, 1985.

[36] Sergey Levine, Peter Pastor, Alex Krizhevsky, Julian Ibarz, and Deirdre Quillen. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *The International Journal of Robotics Research*, 37(4-5):421–436, 2018.

[37] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. 17(1):1334–1373, January 2016.

[38] Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, and Sergey Levine. Scalable deep reinforcement learning for vision-based robotic manipulation. volume 87 of *Proceedings of Machine Learning Research*, pages 651–673. PMLR, 29–31 Oct 2018.

[39] Chuanyu Yang, Kai Yuan, Qiuguo Zhu, Wanming Yu, and Zhibin Li. Multi-expert learning of adaptive legged locomotion. *Science Robotics*, 5(49), December 2020.

[40] Chuanyu Yang, Kai Yuan, Shuai Heng, Taku Komura, and Zhibin Li. Learning natural locomotion behaviors for humanoid robots using human bias. *IEEE Robotics and Automation Letters*, 5(2):2610–2617, April 2020.

[41] Wenbin Hu, Chuanyu Yang, Kai Yuan, and Zhibin Li. Reaching, grasping and re-grasping: Learning multimode grasping skills, 2020.

[42] Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. *CoRR*, abs/1710.06537, 2017.

[43] Rohan Chitnis, Shubham Tulsiani, Saurabh Gupta, and Abhinav Gupta. Efficient bimanual manipulation using learned task schemas, 2020.

[44] Z. Sun, K. Yuan, W. Hu, C. Yang, and Z. Li. Learning pregrasp manipulation of objects from ungraspable poses. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9917–9923, 2020.

[45] Rose M. Spielman, Kathryn Dumper, William Jenkins, Arlene Lacombe, Marilyn Lovett, and Marion Perlmutter. *Psychology*. OpenStax, Houston, Texas, 2014.

[46] Shinya Yamamoto, Tatyana Humle, and Masayuki Tanaka. Basis for cumulative cultural evolution in chimpanzees: Social learning of a more efficient tool-use technique. *PLOS ONE*, 8(1):1–5, 01 2013.

[47] Bianca Jovanovic and Gudrun Schwarzer. Learning to grasp efficiently: The development of motor planning and the role of observational learning. *Vision Research*, 51(8):945–954, 2011. Perception and Action: Part II.

[48] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter

Abbeel, and Sergey Levine. Soft actor-critic algorithms and applications, 2019.

[49] Hado Hasselt. Double q-learning. In J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems*, volume 23. Curran Associates, Inc., 2010.

[50] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In Yoshua Bengio and Yann LeCun, editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016.

[51] Abien Fred Agarap. Deep learning using rectified linear units (relu). *CoRR*, abs/1803.08375, 2018.

[52] Alexander LeNail. Nn-svg: Publication-ready neural network architecture schematics. *Journal of Open Source Software*, 4(33):747, 2019.

[53] Franka Emika GmbH. Franka ROS. https://frankaemika.github.io, 2017.

[54] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters, 2019.

[55] Scott H. Frey and Daniel J. Povinelli. Comparative investigations of manual action representations: evidence that chimpanzees represent the costs of potential future actions involving tools. *Philosophical Transactions of the Royal Society*, 367(1585):48–58, 2012.