POLITECNICO DI TORINO

Master's Degree in MEHATRONIC ENGINEERING

Politecnico di Torino

Master's Degree Thesis

Imitation Learning for Dexterous Robotic Manipulation

Supervisors

Prof. Giuseppe AVERTA

Prof. Josie HUGHES

Prof. Maryam KAMGARPOUR

Dott. Andreas SCHLAGINHAUFEN

Dott. Cheng PAN

Candidate

Gregorio VALENTI

October 2025

Abstract

Dexterous robotic manipulation with hands is one of the most challenging tasks in robotics control. This thesis addresses learning-based approaches for manipulation tasks from 8 to 22 degrees of freedom (DoF), using 6 and 7-DoF robotic arms, a standard 1-DoF gripper, and the 15-DoF Adapt Hand developed at EPFL's Create Lab.

Current methods for dexterous manipulation have significant limitations: Reinforcement Learning (RL) can achieve good results but requires reward engineering, which scales in complexity with the type, length and dimensionality of the task, often yielding suboptimal results, while Behavioural Cloning (BC) formulates the problem as supervised learning from demonstrations, but suffers from poor generalisation and compounding error problems when deployed in the real world.

This work analyses hybrid approaches which aim to leverage the strengths of RL and BC while mitigating their weaknesses, and proposes a novel BC-PPO algorithm that combines Maximum Likelihood Behavioural Cloning with Proximal Policy Optimisation Reinforcement Learning in a single objective function, optimised end-to-end by an actor-critic neural network framework.

Additionally, expert trajectories are recorded with a custom setup integrating the Apple Vision Pro headset with NVIDIA Isaac Sim, and experiments are conducted on manipulation tasks of increasing complexity: Franka Lift (8-DoF), Franka Reach with Hand Pose (22-DoF), UR5 with Adapt Hand Lift (21-DoF), UR5 with Adapt Hand Pick and Place (21-DoF).

Results show that BC-PPO eliminates the overfitting problem characteristic of BC methods, while maintaining competitive performance with the state-of-the-art. This work characterises the conditions under which hybrid methods can provide value, and establishes a foundation for future work of algorithmic improvement and simulation to reality transfer.

Keywords: Reinforcement Learning, Imitation Learning, Dexterous Manipulation, Robotic Manipulation

Table of Contents

Abstract I

1 Introduction 1

2 Literature Review 3
2.1 Reinforcement Learning for Robotic Manipulation 3
2.1.1 Theoretical Foundations 3
2.1.2 Policy Gradient Methods and Proximal Policy Optimization 4
2.1.3 Gradient Direction and Generalized Advantage Estimation . 5
2.2 Imitation Learning for Robotic Manipulation 5
2.2.1 Motivation and Overview 5
2.2.2 Behavioural Cloning 6
2.2.3 Inverse Reinforcement Learning and Generative Methods . . 6
2.3 Hybrid Approaches 8
2.3.1 Motivation and Overview 8

3 Methodology 9
3.1 Experimental Framework and Simulation Environment 9
3.1.1 Isaac Lab Setup 9
3.1.2 Robotic Platforms and Arm Control Architecture 10
3.2 RL Tasks and MDP Formulation 11
3.2.1 Common MDP Formulation 11
3.2.2 Task 1: Franka Lift with Parallel-Jaw Gripper 12
3.2.3 Task 2: Franka Reach with Hand Pose 14
3.2.4 Training Setup 16
3.3 Behavioral Cloning Implementation 18
3.3.1 Overview 18
3.3.2 Expert Demonstrations 18
3.3.3 Teleoperation and Adapt Hand Sim-to-Real Transfer 20
3.3.4 Task Definitions for Behavioral Cloning 22
3.3.5 Training Setup 25

3.4 Hybrid Learning Approaches . . . 27
3.4.1 Motivation and Overview . . . 27
3.4.2 Inverse Reinforcement Learning Background . . . 28
3.4.3 Generative Adversarial Imitation Learning Implementation . 29
3.4.4 Behavioral Cloning with Proximal Policy Optimization . . . 31

4 Experimental Results 35
4.1 Evaluation Metrics . . . 35
4.1.1 Success Rate . . . 35
4.1.2 Convergence time . . . 36
4.1.3 Training Time . . . 36
4.2 Reinforcement Learning Results . . . 36
4.2.1 Franka Lift Task . . . 36
4.2.2 Franka Reach with Hand Pose Task . . . 38
4.3 Behavioural Cloning Results . . . 40
4.3.1 BC with Synthetic Demonstrations - Franka Lift . . . 40
4.4 Hybrid Approach Results . . . 46
4.4.1 Wasserstein GAIL Performance . . . 47
4.4.2 BC-PPO Hybrid Approach . . . 49

5 Conclusion and Discussion 56
5.1 Methods Comparison . . . 56
5.1.1 Training Time and Efficiency . . . 56
5.1.2 Task Complexity and Performance . . . 57
5.1.3 The Demonstration Sampling Problem . . . 58
5.1.4 Failure Mode Analysis and Generalisation . . . 58
5.1.5 Wd-GAIL's Instability . . . 59
5.2 Limitations . . . 59
5.2.1 Demonstration Quantity and Quality . . . 59
5.2.2 Hyperparameter sensitivity and Scalability . . . 60
5.3 Simulation to Reality Gap . . . 60
5.4 Closing Remarks . . . 61
.1 Appendix A: Equivalence of MLE and MSE for Gaussian Policies . 64
.2 Appendix B: BC+PPO Algorithm . . . 64

Bibliography 66

List of Figures 69

List of Tables 70

Chapter 1

Introduction

Dexterous robotic manipulation represents one of the most challenging tasks in robotics control, requiring precision and coordination of high-dimensional systems. This challenge is particularly relevant when using humanoid robotic hands, where complexity is proportional with the degrees of freedom involved. In this work, we address the problem of controlling six and seven degrees-of-freedom (DoF) robotic arms with traditional one DoF grippers, as well as with a fifteen DoF robotic hand, where dimensionality and joint coupling constraints make traditional control approaches computationally expensive and often ineffective.

The high-dimensionality in dexterous manipulation introduces challenges that incentivise the use of learning-based approaches that can discover complex control strategies from data, rather than relying on hand-designed controllers.

Reinforcement Learning (RL) has emerged as a promising approach for solving dexterous manipulation tasks through interaction with the environment. However, the complexity of multi-finger manipulation tasks greatly increases the difficulty of reward engineering, a process which is at the base of RL algorithms. Designing a reward function that fully captures the requirements of manipulation is usually a lengthy iterative process with often suboptimal results.

Behavioural Cloning (BC) offers an alternative approach that handles the problem as supervised learning from expert demonstrations. BC algorithms can achieve state-of-the-art performance when provided with a comprehensive set of demonstrations covering the task space of the robot, but this coverage is rarely achievable in practice, especially for high-DoF systems. Moreover, BC suffers from limitations when deployed in real-world applications, including covariate shift and compounding error problems, where deviations from the demonstrations data distribution can accumulate over time, leading to task failure or undesired behaviours.

This thesis analyses whether modern physics simulation can enable faster and more efficient training of hybrid approaches that leverage both BC and RL. We

use NVIDIA Isaac Sim to parallelise thousands of environments learning simultaneously, accelerating training by orders of magnitude and enabling the integration of imitation and reinforcement learning.

Our contribution is a training framework that combines Behavioural Cloning with Proximal Policy Optimization (PPO) through synchronised sampling from both expert demonstrations and the simulated environment rollouts within the same neural network update loop. Our method uses BC to guide the policy towards demonstrated behaviours while RL rewards incentivise task completion and refinement. The reward engineering process is thus limited to simple functions focusing on basic aspects of the task and, most importantly, its success.

Our experiments are performed on multiple tasks and robotic setups, and demonstrate that this approach achieves competitive success rates with the state-of-the-art methods, although slightly outperformed by BC in high-dimensional tasks, and completely eliminates the overfitting problem characteristic of supervised learning approaches. We characterize the conditions under which hybrid methods provide value over pure approaches, and show that the combination of parallel environment simulation and minimal reward engineering with expert demonstrations establishes a foundation for future algorithmic improvements toward learning robust policies for high-DoF robotic manipulation systems. We also briefly discuss the capabilities of our framework to be transferred to real-world applications in future work.

Chapter 2

Literature Review

2.1 Reinforcement Learning for Robotic Manipulation

2.1.1 Theoretical Foundations

Reinforcement Learning formulates the control problem in terms of a Markov Decision Process (MDP), a mathematical framework to represent sequential decision making. In an MDP, at time step t, an *agent* in state s_t selects an action a_t, interacting with the *environment* and receiving a reward r_t, as well as transitioning to a new state s_{t+1}. The reward signal is a numerical value to be maximised over time, and the new state is both a consequence of the action and of the dynamics of the system. The balance of the trade off between immediate and delayed rewards is essential to increase the agent's return, or the sum of its rewards over time [1].

Formally, an MDP is defined by the tuple (S, A, P, R, γ) where [1]:

- S is the set of states
- A is the set of actions
- $P : S \times S \times A \to [0,1]$ is the state transition probability function, where $P(s'|s, a)$ represents the probability of transitioning to state s' from state s after taking action a
- $R : S \times A \to \mathbb{R}$ is the reward function for state-action pairs
- $\gamma \in [0,1]$ is the discount factor that determines the importance of future rewards

The agent's goal is to find a policy $\pi : S \rightarrow A$ that maximises the expected cumulative reward:

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{2.1}$$

where G_t is the return from time step t [1].

In the context of robotic manipulation, the state space S typically includes joint positions, velocities, and object positions if the task requires interaction with them, while the action space A represents commands given to the robot's actuators. The actuators of both our robot arm and hand are entirely controlled with position signals (joint angles). Reward functions for manipulation tasks vary widely with the application, and can typically involve distance-base rewards, success-based rewards, and penalty terms for undesired behaviour, such as excessive joint velocities and collisions.

The design of reward functions, or reward engineering, remains one of the main challenges in applying RL to manipulation, as covering all the requirements for a complex task can be a lengthy process, often yielding suboptimal results [2].

2.1.2 Policy Gradient Methods and Proximal Policy Optimization

Function approximators like neural networks are essential when trying to solve an MDP for a robotic application, and policy gradient methods aim to directly optimize a parametrized policy $\pi_\theta(a|s)$, which outputs a probability distribution over actions for each available state within the set S [1]. The policy gradient theorem provides a compact formulation for computing the gradient of the objective function as an expected value under policy π_θ:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) G_t\right] \tag{2.2}$$

where $J(\theta)$ is the objective function to be maximised and G_t is the return from time step t [1]. This expresses that the gradient points in the direction of actions that lead to higher returns (gradient ascent), weighted by the returns themselves.

Among policy gradient methods, Proximal Policy Optimization (PPO) [3] has emerged as a robust choice for continuous control. PPO builds from Trust Region Policy Optimization (TRPO) [4], but replaces the KL divergence computation required by TRPO, as it is inefficient and computationally heavy.

PPO aims to maximise a *surrogate* objective denoted as $\hat{\mathbb{E}}_{\approx}[r_t(\theta)\hat{A}_t]$, where r_t measures the relative probability of actions under new and old policies, and can be expressed as $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$. $\hat{A}_t$ is an estimated advantage function, used to guide the direction of the gradient and decrease variance in the policy update. It can be

seen that without constraining this objective, even small gradient changes can lead to excessively large policy variations, rendering the training process unstable.

PPO's key innovation is the clipped surrogate objective:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t \left[\min \left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right] \tag{2.3}$$

where the ratio $r_t(\theta)$ is constrained to the interval $[1-\epsilon, 1+\epsilon]$, with ϵ typically set to 0.2. This clipping achieves the same goal as TRPO's KL divergence, while being easier to implement and often yielding better performance [3].

2.1.3 Gradient Direction and Generalized Advantage Estimation

The advantage function $A(s,a)$ represents how much better an action is compared to the average action for that state. By defining the value function $V_\pi(s)$ as the expected return when starting in state s and following policy π, and the action value function $Q_\pi(s,a)$ as the expected return when starting from state s, taking action a, and then following policy π, the advantage can be seen as $A(s,a) = Q_\pi(s,a) - V_\pi(s)$, and it is usually used to compute the sign of the gradient during policy gradient updates, with a reduction in variance at the cost of some additional bias, as $A(s,a)$ has to be estimated by a neural network and cannot be measured directly [5].

PPO employs Generalized Advantage Estimation, which provides a bias-variance trade off with an exponentially-weighted average of value function estimations:

$$\hat{A}_t^{GAE(\gamma,\lambda)} = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}^V \tag{2.4}$$

where $\delta_t^V = r_t + \gamma V(s_{t+1}) - V(s_t)$ is the Temporal Difference (TD) residual [1], V is a learned value function and $\lambda \in [0,1]$ controls the bias-variance trade off. When $\lambda = 0$, this is a one-step TD estimate (low variance, high bias), while when $\lambda = 1$ this is the Monte Carlo return (high variance, low bias), which considers the full trajectory until its completion. GAE with intermediate λ values is what's usually employed in practice and provides better learning stability [5].

2.2 Imitation Learning for Robotic Manipulation

2.2.1 Motivation and Overview

Imitation learning is an alternative to reinforcement learning that learns directly from a set of expert demonstrations, rather than its own interactions with the environment and the reward signal [6]. This avoids the reward engineering process, which has many clear limitations, and instead relies on the quality and quantity of demonstrations available [7].

2.2.2 Behavioural Cloning

Behavioural Cloning (BC) is the most straightforward approach to imitation learning, solving the problem of finding a policy with a supervised learning approach [8]. Given a set of expert demonstrations $\mathcal{D} = \{(s_i, a_i)\}_{i=1}^{N}$, usually composed of different trajectories each containing state-action pairs for every time step, this approach learns a policy π_θ by minimising the difference between predicted and expert actions.

The two most common loss functions for BC are Mean Squared Error (MSE) and Maximum Likelihood Estimation (MLE). The MSE loss, which is also suitable for continuous action spaces, can be formulated as follows:

$$\mathcal{L}^{MSE}(\theta) = \mathbb{E}_{(s,a)\sim\mathcal{D}}\left[\|\pi_\theta(s) - a\|^2\right] \tag{2.5}$$

For stochastic policies, the MLE loss maximizes the likelihood of expert actions:

$$\mathcal{L}^{BC}(\pi) = -\mathbb{E}_{(s,a)\sim\mathcal{D}}[\log \pi(a|s)] \tag{2.6}$$

In the case of the MLE approach, given that the policy outputs a Gaussian distribution over actions, MLE becomes equivalent to MSE scaled by the variance (see appendix A for derivation). This formulation makes MLE more suitable to be ised in combination with reinforcement learning algorithms, which require a stochastic policy, as demonstrated in hybrid approaches, and in the final algorithm I will present in this thesis [9].

2.2.3 Inverse Reinforcement Learning and Generative Methods

Behavioural Cloning directly fits a policy to expert actions, depending heavily on the quality and quantity of demonstrations available. The main downside of BC is the presence of compounding errors and deviations from the expert demonstrations when deployed, which can cause undesired behaviours or even the failure of the task.

Inverse Reinforcement Learning (IRL) aims to solve the issues in BC by learning a cost function which makes expert behaviour prevail over other choices. RL is then used in an inner loop to obtain a policy from the learned cost function, by minimizing the cost over time. This provides better generalisation to unseen states, because the recovered function can guide learning even outside the available demonstrated trajectories [7].

In general, the IRL objective can be written as the minimization of the distance between expert and agent distributions over actions, and can be compactly written as:

$$\mathcal{L}^{IRL} = \min D\left(d^E \,\|\, d^\pi\right) \tag{2.7}$$

where D is the chosen statistical distance measurement between the two distributions. Different choices of D and the regularisation function used lead to different IRL algorithms.

One of the most popular and used formulations is Generative Adversarial Imitation Learning (GAIL) [7], which treats imitation learning as a minmax occupancy measure matching problem, with a similar structure to Generative Adversarial Networks (GANs). In GAIL, instead of recovering a cost function first, and then running RL to obtain a policy, a generator network G formulates a policy π_θ which is fed to a discriminator network D. The discriminator compares the generated policy with the expert and gives feedback to the generator. The balance between the two networks and their updates must be controlled carefully to maintain stability.

This formulation can be seen as occupancy measure matching: instead of directly minimising the difference between actions, GAIL considers the distribution of state-action pairs visited by the agent and tries to match that to the expert. The occupancy measure $d^{\pi_\theta}(s, a)$ is formulated as the discounted frequency of reaching state s and taking action a, following policy π_θ, and can be mathematically described as follows:

$$d^{\pi_\theta}(s, a) = (1 - \gamma) \sum_{t=0}^{\infty} \gamma^t \, P(s_t = s, a_t = a \mid \pi_\theta) \tag{2.8}$$

where P is the state transition probability function of the considered MDP [1].

The GAIL objective is then written as:

$$\mathcal{L}^{GAIL} = \mathbb{E}_{(s,a)\sim\pi_\theta} \left[\log D(s, a)\right] + \mathbb{E}_{(s,a)\sim\pi_E} \left[\log(1 - D(s, a))\right] - \lambda \, H(\pi) \tag{2.9}$$

where the first part of the loss is sampled from the environment, and the second part is sampled from the expert demonstrations. $H(\pi)$ is the entropy of the Gaussian policy π_θ, and it is commonly used to incentivise exploration, which is both weighted by the *temperature* parameter λ, and intrinsically controlled by the standard deviation of the distribution, which will shrink as the agent has explored its environment and the policy becomes more deterministic.

Overall, GAIL bypasses reward engineering typical of RL, but is is known to require careful tuning and can be unstable for high-dimensional tasks [10]. One major modification which attempts to increase the stability of the algorithm stems from the same imitation learning interpretation of minimizing occupancy measure distribution distance, and it replaces the GAIL discriminator loss based on Jensen-Shannon (JS) divergence with a Wasserstein-based critic, with the same technique that was proposed in Wasserstein GANs [11, 10].

In this variation, the logarithmic term of the discriminator component in the loss function is removed, and the discriminator is constrained to be a 1-Lipschitz function via neural network gradient penalty.

2.3 Hybrid Approaches

2.3.1 Motivation and Overview

Imitation learning methods can achieve good results in many tasks, but often struggle as the difficulty increases, due to either compounding errors caused by covariate shift, or under-representation of specific scenarios within the expert dataset (limited coverage of the state space) [9]. Hybrid methods try to overcome the weaknesses of both BC and RL by combining the two approaches.

A common way to achieve this is by performing a policy *warm start* using BC, and then train with RL to fine-tune behaviour and incentivise task success, avoid collisions, and add other relevant constraints [12].

A similar approach is to run RL and BC within the same neural network update loop, combining the two loss functions in a single formulation, that alows for end-to-end training of the policy leveraging both expert demonstrations and interactions with the environment. The work *"Imitation Is Not Enough: Robustifying Imitation with Reinforcement Learning for Challenging Driving Scenarios"* combines the Soft Actor Critic (SAC) [13] RL objective with a maximum likelihood estimation BC approach, leading to the following function to be minimised:

$$\mathcal{L}^{BC+SAC} = \mathbb{E}_{(s,a)\sim\pi_\theta}\left[Q(s,a) + \mathcal{H}(\pi_\theta(\cdot|s))\right] + \lambda\mathbb{E}_{(s,a)\sim\pi_E}\left[\log(\pi_\theta(a|s))\right]$$

where λ is a task-dependent hyperparameter to be tune according to the numerical values of the BC loss, and how much guidance is desired from imitation. The authors of this algorithm applied it for autonomous driving, and its main idea is to have both RL and BC guiding learning when the vehicle is in a state within the expert distribution, and to have RL take over when the vehicle reaches out-of-distribution states. The method shows to improve the reliability of the agent in challenging scenarios, highlighting the refinement of the policy given by the RL component [13].

Chapter 3

Methodology

3.1 Experimental Framework and Simulation Environment

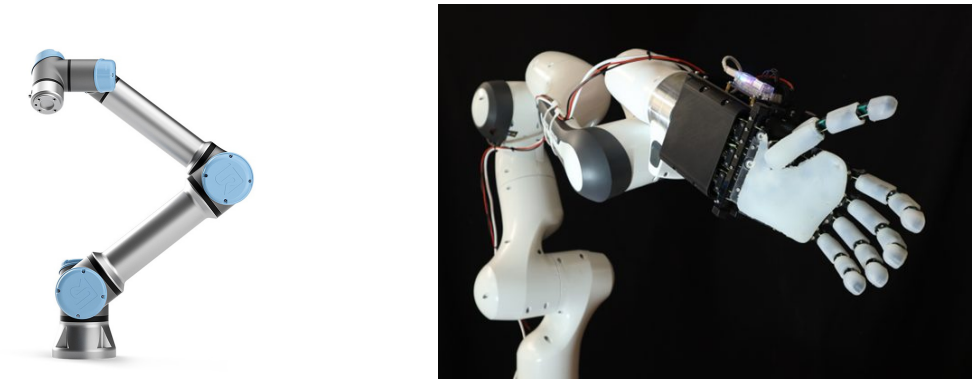
3.1.1 Isaac Lab Setup

Our experimental setup involves NVIDIA Isaac Lab as the simulation environment for training and visualising trained policies. Isaac Lab is built on NVIDIA Omniverse [14], which enables GPU-accelerated physics computation and high-quality rendering.

Isaac Lab is an open-source modular framework characterised by its accurate physics simulation capabilities and possibility to efficiently scale up training by parallelising thousands of robotic environments learning simultaneously. NVIDIA's PhysiX API supports vectorised rendering and domain randomisation, which are essential for robustness and sim-to-real transfer in robotics [15].

The API provides libraries that integrate with standard deep learning frameworks like PyTorch. Our work station is equipped with an NVIDIA RTX 4090 GPU with 24GB VRAM, which allows the parallel simulation of up to 4096 environments with very little performance degradation.

The different robot models are imported in the simulator using standard URDF files with custom modifications made to combine the arm base with the robotic hand. In particular, the kinematic chain of the wrist and hand were added to the arm base, including joint limits, collision geometries and visual meshes. All hand meshes were converted from the original CAD model developed by Kai Junge, the creator of the Adapt Hand [16].



(a) UR5 robotic arm (6-DoF)

(b) Franka Panda with Adapt Hand (7-DoF + 15-DoF)

Figure 3.1: Robotic arm bases used in experiments.

3.1.2 Robotic Platforms and Arm Control Architecture

Our experiments use two distinct robotic arm platforms: the Franka Emika Panda (7-DoF) and the Universal Robots UR5 (6-DoF). Experiments were conducted both with a standard parallel jaw gripper (1-DoF) to test stability and achieve task completion with a simpler configuration, and with Create Lab's Adapt Hand, a multi-finger robotic hand (15-DoF, divided in 13 for the hand and 2 for the wrist) for more complex manipulation tasks. The Adapt hand is tendon-actuated and has a total of 22 joints, 2 of which are responsible of moving the wrist. Due to the tendon actuation, only 15 joints of the hand and wrist setup are active, while the remaining have their motion coupled with other joints and links. In particular, each distal finger joint is coupled with their respective proximal joint, and the carpometacarpal thumb joint is coupled with the metacarpophalangeal joint, miming closely the movements of a real human hand.

Figure 3.1 shows a ur5 robot arm on the left, and a Franka Panda arm with the Adapt Hand mounted on the right.

Robot control for both arms is implemented using inverse kinematics task-space control, where desired end-effector poses are specified as position and orientation (3D position coordinates and quaternion orientation) in Cartesian space. The controller computes joint angle position commands through inverse kinematics to achieve the specified poses.

For a generic n-DOF manipulator, the forward kinematics relationship can be expressed as follows:

$$\mathbf{x} = f(\mathbf{q}) \tag{3.1}$$

where $\mathbf{x} \in \mathbb{R}^6$ represents the end-effector pose (position and orientation) and $\mathbf{q} \in \mathbb{R}^n$

represents the joint configuration.

The velocity relationship is given by the Jacobian:

$$\dot{\mathbf{x}} = \mathbf{J}(\mathbf{q})\dot{\mathbf{q}} \tag{3.2}$$

where $\mathbf{J}(\mathbf{q}) \in \mathbb{R}^{6\times n}$ is the manipulator Jacobian matrix.

The Franka Panda has 7-DoF, and therefore it is kinematically redundant with one redundant DoF. For controlling this type of system and perform manipulation in space, we opted to use null-space control, which handles the extra degree of freedom by finding the null-space of the Jacobian matrix, which represents all the joint velocities that produce no velocity for the end effector. The solution for $\dot{q}$ becomes:

$$\dot{\mathbf{q}} = \mathbf{J}^{\dagger}\dot{\mathbf{x}} + (\mathbf{I} - \mathbf{J}^{\dagger}\mathbf{J})\dot{\mathbf{q}}_0 \tag{3.3}$$

where $\mathbf{J}^{\dagger}$ is the Moore-Penrose pseudoinverse, $\mathbf{I}$ is the identity matrix, and $\dot{\mathbf{q}}_0$ represents joint velocities projected into the null space for secondary objectives [17]. The secondary objective we chose is joint-centering, performing the requested motion while driving joints towards their mid-range position (specified in joint angles), to avoid reaching joint limits or undesired configurations after long trajectories.

Therefore, the complete control law for the redundant Franka system becomes:

$$\dot{\mathbf{q}} = \mathbf{J}^{\dagger}\dot{\mathbf{x}} + (\mathbf{I} - \mathbf{J}^{\dagger}\mathbf{J})k(\mathbf{q}_{mid} - \mathbf{q}) \tag{3.4}$$

where $\mathbf{q}_{mid} = \frac{\mathbf{q}_{max}+\mathbf{q}_{min}}{2}$ represents the joint mid-range positions, $k > 0$ is the null-space gain factor.

The 6-DOF UR5 utilizes standard inverse kinematics without null-space projection, as it is not a redundant system in space.

3.2 RL Tasks and MDP Formulation

Our experimental setup evaluates learning performance across two manipulation scenarios that possess different levels of complexity. The first one is the Franka Lift task performed with the standard 1-DoF gripper, where the robot has to pick up a cube and reach a sampled target position in space. The second is a combination between the Franka Reach task, where the Panda robot moves such that its end effector reaches a sampled pose in space, and a hand configuration task, where the tip of the hand's thumb has to touch the tip of the index finger, leaving the rest of the fingers unconstrained.

3.2.1 Common MDP Formulation

Both tasks share a common MDP structure while differing in environment configuration and reward function design. Results consider learning efficiency in terms

of reward values achieved and visual behaviour, considering also that the hand configuration part of the second task is a custom task that we added to the standard Reach environment.

Both tasks are formulated within Isaac Lab as Markov Decision Processes using the Python programming language. The tuple (S, A, P, R, γ) is thus structured as follows:

The observation space S varies based on end-effector configuration, and includes:

- For the Franka with Adapt Hand: 29 total values consisting of 7 arm joint positions, 7 arm joint velocities, and 15 wrist and hand joint positions
- For the Franka with standard gripper: 15 total values consisting of 7 arm joint positions, 7 arm joint velocities, and 1 gripper position
- End-effector pose: $\mathbf{x}_{ee} \in \mathbb{R}^7$ (3D position + quaternion orientation, thus 7 elements)
- Target end-effector pose of 7 elements but 6-DoF: $\mathbf{x}_{target} \in \mathbb{R}^7$
- Task-specific observations: object position in space for the lift task, fingertip positions for the hand pose task
- The full set of actions from the previous time step

The total number of observations with the standard gripper is 40, while with the Adapt Hand there are 71 elements in the tensor.

The action space A represents target joint angles sent directly to Isaac Lab's Operational Space Controller to move the arm, together with the joint angles to control the standard gripper or the Adapt Hand in position, depending on the configuration:

$$\mathbf{a} = [q_1, q_2, ..., q_7, \text{end-effector joint angles}] \tag{3.5}$$

where q_i are the target pose sent to the inverse kinematics controller to move the arm. The total number of actions is 8 for the standard gripper configuration, and 22 for the Adapt Hand configuration.

The transition probability function $P(s'|s, a)$ is determined by Isaac Lab's physics simulation using the PhysX API, with dynamics randomization applied to improve policy robustness.

The discount factor γ is set to 0.98 for all experiments of both tasks.

3.2.2 Task 1: Franka Lift with Parallel-Jaw Gripper

Environment Configuration

The cube is randomly positioned within the workspace bounds: $x \in [0.4, 0.6]$m, $y \in [-0.25, 0.25]$m, with the goal position sampled from the same distribution for

each episode.

Reward Function Design

The total reward combines four components, with three static terms and one dynamic term which changes over time with curriculum learning:

$$R_{TOT} = R_{reach} + R_{lift} + R_{track} + R_{penalties} \tag{3.6}$$

The reach component incentivises the end effector to move closer to the object, and uses the tanh distance framework:

$$R_{reach} = w_1 \left(1 - \tanh\left(\frac{d_{ee\_obj}}{\sigma_1}\right)\right) \tag{3.7}$$

where $d_{ee\_obj} = \|\mathbf{p}_{ee}^{i} - \mathbf{p}_{obj}\|_2$ is the distance between the end-effector centre point (tool centre point) and the object, with $w_1 = 1.0$ and $\sigma_1 = 0.1$. The standard deviation σ_1 can be interpreted as the distance at which the reward signal is relevant. In fact, when the end effector is 0.1m away from the cube, the reaching component of the reward function is at around 24% of its maximum value.

The lift component of the reward function is a binary value for lifting object above minimum height:

$$R_{lift} = w_2 \cdot \mathbb{I}(z_{obj} > h_{min}) \tag{3.8}$$

where $\mathbb{I}(\cdot)$ is the indicator function, z_{obj} is the object height, $h_{min} = 0.04$m, and $w_2 = 15.0$. The large weight of this component highly encourages task success.

The goal tracking reward motivates moving the lifted object toward the sampled target, and has a normal and a fine-grained component, which has the same formulation as the former but different weight and standard deviation:

$$R_{track} = w_3 \cdot \mathbb{I}(z_{obj} > h_{min}) \cdot \left(1 - \tanh\left(\frac{\|\mathbf{p}_{goal} - \mathbf{p}_{obj}\|_2}{\sigma_2}\right)\right) \tag{3.9}$$

where $\mathbf{p}_{goal}$ is the target position in world frame, $w_3 = 16.0$, and $\sigma_2 = 0.3$. The fine-grained component has $w_4 = 5.0$, and $\sigma_2 = 0.05$, which indicates that this term becomes relevant only when the behaviour of the robot is accurate enough to closely reach the target position, and some fine-tuning is required to adjust the final end-effector pose.

The penalty terms are mainly for regularization and smooth control, focusing

on large actions and large joint velocities:

$$R_{penalties} = R_{action} + R_{velocity} \tag{3.10}$$

$$R_{action} = -w_5 \sum_{i}^{n_a} (a_i^t - a_i^{t-1})^2 \tag{3.11}$$

$$R_{velocity} = -w_6 \sum_{j}^{n_j} (\dot{q}_j)^2 \tag{3.12}$$

where n_a is the action dimension, n_j is the number of joints, and penalty weights start at $w_5 = w_6 = 1 \times 10^{-4}$ and increase to 1×10^{-1} over 10,000 training steps through curriculum learning.

Curriculum learning increases the penalty weights according to the following formulation:

$$w_{penalty}(t) = w_i + \frac{t}{T_{curr}} \cdot (w_f - w_i) \tag{3.13}$$

where t is the current training step, $T_{curr} = 10{,}000$ steps, $w_{initial} = 1 \times 10^{-4}$, and $w_{final} = 1 \times 10^{-1}$.

This addition to the reward function increases the movement smoothness of the final policy, as the penalty terms increase considerably once the fine-tuning process begins, but this is entirely done as a *"black box approach"* over the same training cycle.

3.2.3 Task 2: Franka Reach with Hand Pose

Environment Configuration

This task combines the standard Isaac Lab Reach task with a custom finger coordination task, using the 15-DoF Adapt Hand. Target end-effector poses are sampled within a cylindrical workspace, which closely corresponds to the reachable task space of the Franka manipulator, defined by:

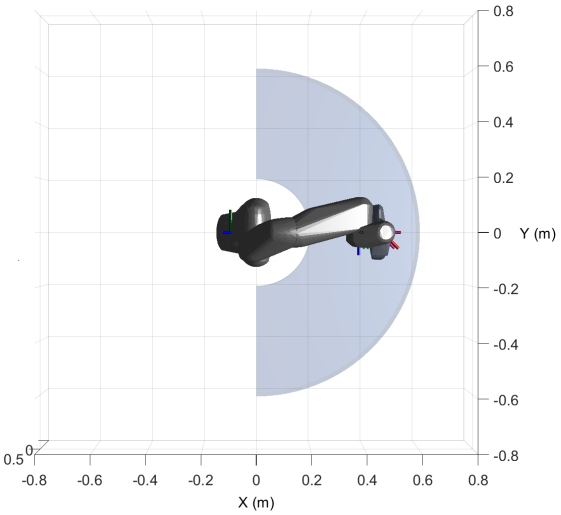
$$r \sim \mathcal{U}[\sqrt{r_i^2 + s(r_o^2 - r_i^2)}] \text{ where } s \sim \mathcal{U}[0,1] \tag{3.14}$$

$$\theta \sim \mathcal{U}[-\pi/2, \pi/2] \tag{3.15}$$

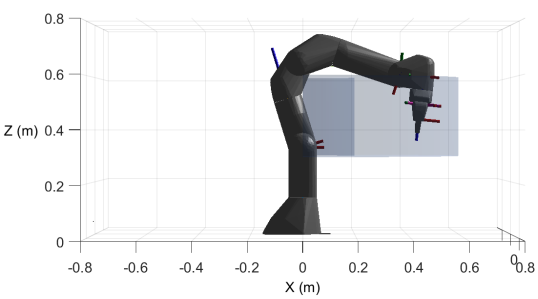
$$x = r\cos(\theta), \quad y = r\sin(\theta) \tag{3.16}$$

$$z \sim \mathcal{U}[0.3, 0.6]\text{m} \tag{3.17}$$

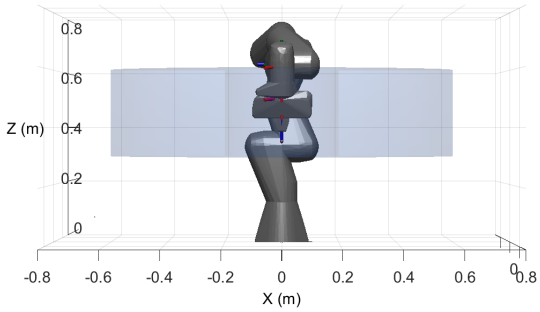
where $r_i = 0.2$m and $r_o = 0.6$m are the inner and outer radii of the cylinder. The radius r_i is chosen to be higher than zero to avoid self collisions between the hand and the arm, and z is chosen to be higher than 0.3 to avoid collisions between the hand and the table. Target orientations are randomized within ±20° for roll and pitch, and ±90° for yaw. The robot must reach the target pose and achieve



(a) Top view



(b) Side view



(c) Front view

Figure 3.2: Cylindrical workspace for Task 2 shown from multiple perspectives.

a configuration where the tip of the thumb touches the tip of the index finger. Feedback on the thumb and index finger links is given to the agent every time step, while the remaining fingers are unconstrained.

Figure 3.2 shows multiple perspectives of the sampling region for the reach target. The choice of z is quite conservative due to the length of the Adapt Hand, as it has both a wrist and a casing containing all the motors.

Reward Function Design

The total reward combines six components, which address both the requirements for the arm movement and hand finger movement:

$$R_{TOT} = R_{alive} + R_{term} + R_{reach} + R_{pinch\_dist} + R_{pinch\_success} + R_{penalties} \quad (3.18)$$

The first two reward components have a term that encourages episode completion, and a term that highly penalises episode failure:

$$R_{alive} = w_7 \cdot (1 - \mathbb{I}(\text{terminated})) \tag{3.19}$$

$$R_{term} = w_8 \cdot \mathbb{I}(\text{terminated}) \tag{3.20}$$

where $w_7 = 1.0$ and $w_8 = -10.0$. In this custom reward function, we tried to successfully complete the task without employing curriculum learning. To this end, we defined episode failure as joint velocities and actions out of bounds, and added a large penalty for exceeding these bounds. The position bounds were set to ± 3.14rad, and the velocity bounds to ± 3.14rad/s.

The reach term is composed of position and orientation tracking:

$$R_{reach\_pos} = w_9 \cdot \|\mathbf{p}_{ee} - \mathbf{p}_{target}\|_2 \tag{3.21}$$

$$R_{reach\_rot} = w_{10} \cdot \text{quat\_error\_magnitude}(\mathbf{q}_{ee}, \mathbf{q}_{target}) \tag{3.22}$$

where $\mathbf{p}_{ee}$ and $\mathbf{q}_{ee}$ are the current end-effector position and quaternion, $\mathbf{p}_{target}$ and $\mathbf{q}_{target}$ are the target position and quaternion, $w_9 = w_{10} = -1.0$. Note that these are negative rewards that decrease as the robot approaches the target configuration. This was done since positive rewards are obtained each step by remaining "alive" during the task.

Pinch rewards focus on the hand target configuration:

$$R_{pinch\_dist} = w_{11} \cdot d_{thumb\_index} \tag{3.23}$$

$$R_{pinch\_success} = w_{12} \cdot \mathbb{I}(d_{thumb\_index} < \epsilon_{pinch}) \tag{3.24}$$

where $d_{thumb\_index} = \|\mathbf{p}_{thumb} - \mathbf{p}_{index}\|_2$ is the distance between fingertips, $w_{11} = -3.0$ penalizes larger distances, $w_{12} = 10.0$ rewards task success, and finally $\epsilon_{pinch} = 0.005$m is the chosen threshold that defines task completion.

The penalty components use the same formulation as Task 1's $R_{velocity}$ (Equation 10), but with constant weight $w_{13} = -0.001$ and applied to all 29 joints (7 arm + 22 hand/wrist joints).

Unlike Task 1, this reward structure does not employ curriculum learning, maintaining constant weights throughout training.

The dual-objective of this task requires a more complex hand-designed reward function, and highlights the main limitation of RL. More complicated and longer-horizon tasks than our Task 2 would require an extensive trial and error procedure to define each reward component and to balance each weighting term according to the objective.

3.2.4 Training Setup

All reinforcement learning experiments utilize RSL-RL's implementation of Proximal Policy Optimization [18, 3], which has the mathematical formulation presented in

Section 2.1.2. The algorithm utilises two separate neural networks to carry out the actor-critic framework: one estimates the policy and the other the value function to compute the advantage and reduce variance in the updates [5].

Network Architecture

Both actor and critic networks utilize three-layer fully connected architectures:

- Actor network: $256 \rightarrow 128 \rightarrow 64$ hidden units with ELU activation
- Critic network: $256 \rightarrow 128 \rightarrow 64$ hidden units with ELU activation
- Policy initialization: Gaussian noise with $\sigma = 1.0$

PPO Hyperparameters

The main PPO hyperparameters are set as follows:

$$\text{Clipping parameter: } \epsilon = 0.2 \tag{3.25}$$

$$\text{Learning rate: } \alpha = 1 \times 10^{-4} \text{ with adaptive scheduling} \tag{3.26}$$

$$\text{Discount factor: } \gamma = 0.98 \tag{3.27}$$

$$\text{GAE parameter: } \lambda = 0.95 \tag{3.28}$$

$$\text{Entropy coefficient: } \beta = 0.006 \tag{3.29}$$

All the values above were the default implementation of the algorithm within Isaac Lab's documentation, and we note that all policies we trained with this setup converge smoothly, and the biggest difference is made by the design choices in the reward function.

Training Schedule

Training is performed for 1500 iterations using the following batch structure:

- Environment steps per iteration: 24 per environment
- Learning epochs per iteration: 5
- Mini-batches per epoch: 4
- Gradient clipping: $\|\nabla\|_{max} = 1.0$

Considering that each policy is trained using 4096 parallel learning environments, this corresponds to 98304 training steps for iteration, which yields around 147.5 million total steps over 1500 iterations. This shows the massive scalability and performance allowed by GPU-accelerated vectorised simulations, and permits to reach convergence for any of the selected tasks rather quickly.

3.3 Behavioral Cloning Implementation

3.3.1 Overview

Reinforcement learning demonstrates good results on the manipulation tasks presented in Section 3.1, but the complexity and length of the reward engineering process increases with the difficulty of the task. In particular, tasks that are more complicated than the Franka Lift, especially with the addition of the Adapt Hand, become prohibitively complicated to solve with RL. To address this limitation and scale to more complex manipulation environments, we implement behavioral cloning, leveragine expert demonstrations rather than hand-crafted reward signals.

Our BC implementation focuses first on repeating the lift task, and then on performing pick-and-place manipulation with the 15-DoF Adapt Hand, requiring the coordination of the robotic arm base and the five fingers to grasp the object. The complexity of this task derives from its multi-phase nature (object reach, grasping, transport, release), which would require a reward function to describe consecutive motion and conflicting objectives, given that the object needs to be initially grasped and then released.

Our approach employs two expert data sources: synthetic demonstrations generated from trained RL policies for baseline tasks, and human teleoperation data for advanced dexterous manipulation. In particular, all behaviour cloning performed on the standard 1-DoF gripper uses synthetic demonstrations, while all Adapt Hand movements were recorded through teleoperation.

3.3.2 Expert Demonstrations

Synthetic Data Collection

For tasks where reliable RL policies exist, for example Franka Lift performed with the parallel-jaw gripper, we use these trained policies as experts and record several trajectories to build a dataset. This approach has the advantage of obtaining consistent demonstrations, as the RL agent will always behave the same way, and allows to collect much more data compared to manually teleoperating the system, as no human intervention is required and this process can be automated within NVIDIA Isaac Sim.

The recorded observation space consists of the following elements:

- Joint states: 7 arm joint positions and velocities, and 1 gripper position
- End-effector pose: 7-element pose vector (3D-position + quaternion)
- Task object position: 3D-coordinates of the target
- Action history: previous time step actions for temporal consistency

The total size of the observation tensor is 33. This calculation does not include history in the observations, for which additional tensors of the same size would be stacked based on the amount of history time steps that have to be recorded.

The action space records actions corresponding to the expert policy output at each time step:

- Arm control: 7 joint angle targets for Isaac Lab's Operational Space Controller
- End-effector control: gripper position (1-DoF)

A total of 8 actions are recorded with the parallel-jaw gripper each time step.

The data collection script automatically generates demonstrations by running the expert policy in custom Isaac Lab environments, recording state-action pairs at each timestep, and storing trajectories for each episode in HDF5 format, which is easily accessible by PyTorch data loaders and possesses visualisation tools within Visual Studio Code that can help debugging and analysing the quality of the collected demonstrations. This method can generate hundreds of consistent demonstrations rapidly (this depends on the available GPU VRAM, but it took less than 3 minutes on an NVIDIA RTX 4090), providing substantial training data for BC algorithms.

Each demonstration episode contains: complete observation tensors which match the RL policy inputs, RL policy outputs, episode termination and task success signals. This approach is also used as a baseline to compare with human-collected demonstrations.

For each experiments, we use 100 total demonstrations, 80 of which employed in training, and 20 in validation. The average trajectory length may vary depending on the task, but synthetic recording was only employed for the lift task explained in section 3.3.4, and its average length is of around 4 seconds. The success rate of the expert policy is 100%.

Human Teleoperation Setup

For more advanced dexterous manipulation tasks with the Adapt Hand, like pick-and-place, we implement a teleoperation system using the Apple Vision Pro (AVP) headset, and with an open-source library [19] for tracking hand joint angles and wrist pose in space, only a simple linear mapping of signals to the simulator is needed to record the trajectories of the robot. This approach enables collection of human expert demonstrations for behaviours which would be very hard to replicate using reward engineering and RL algorithms.

It is important to not that from these experiments onwards, the robot arm platform is switched from a Franka Panda to a UR5. This is done for a higher availability of the robot within the laboratory and intent to deploy these policies on the real setup. The only difference is that the arm now has 6 degrees of freedom, and no longer requires null-space projection to handle over actuation.

The AVP captures human hand joint configurations and estimates 20 finger joint angles corresponding to all the degrees of freedom of the human hand. Because the actuated Adapt Hand joints are 13, a linear coupling is applied for each of the coupled joints, making the real hand movement virtually identical to the simulated model. A custom mapping function converts human joint angles to the robot joint space. This constitutes adding movement multipliers and offsets based on visual feedback of motion. The wrist pose is also mapped to Isaac Sim, and by defining a base reference frame corresponding to the base of the robot arm in simulation, the device is able to send spatially consistent signals. All signals are processed and transmitted to the simulation at 100 Hz, which is the default frequency set by the physics solver within the simulator.

While the data structure for teleoperated recording remains similar to synthetic collection, the teleoperation system collects more data as all trajectories are performed with the 15-DoF Adapt Hand and not with the 1-DoF parallel-jaw gripper.

The observation space consists of the following data:

- Joint states: 6 arm joint positions and velocities, and 15 hand joint angles
- End-effector pose: 7-element pose vector (3D-position + quaternion)
- Task object position: 3D-coordinates of the target
- Action history: 21 joint angles corresponding to previous time step actions

The total number of observations is 58, highlighting the increment in complexity that the Adapt hand adds to the task.

3.3.3 Teleoperation and Adapt Hand Sim-to-Real Transfer

The simulation-to-reality (sim2real) gap represents one of the fundamental challenges in robotics, as policies trained in simulation can fail when deployed on real hardware due to differences in dynamics, noise, actuation errors, and interactions with the real environment that cannot be entirely represented in simulation [20]. For our teleoperation system, we evaluate the sim2real transfer performance on the Adapt Hand.

The UR5 arm control has minimal sim2real gap when using position control, as it is an industrial-grade robot and it provides precise joint position tracking with built-in feedback controllers. The arm model implemented in Isaac Sim is a kinematic model and it closely matches the real robot, therefore the arm will be able to accurately match its simulated motion.

Since the Adapt Hand is a custom system designed and built at EPFL's Create Lab, the sim2real evaluation becomes more critical compared to the arm. The

Figure 3.3: AVP teleoperation system and Adapt Hand sim2real setup for recording

complexity of tendon-driven actuation and the coupling between multiple joints create friction forces within the distal joints and could cause problems when deploying simulation-based policies.

We validate the teleoperation pipeline by comparing human movement, simulated movement, and real Adapt Hand movement as shown in figure 3.3.

Figure 3.4 illustrates the complete signal flow from human hand tracking to robot actuation. The AVP system estimates the human hand configuration at 100Hz, and its output is mapped to the 13 actuated joints of the simulated hand through a Python script. The real Adapt Hand motors then read the manually mapped signals from the simulator and position control targets are applied. The simulator uses lower-level position controllers to perform motion, while the real Adapt Hand motors communicate to ROS packages on a Raspberry Pi.

Figure 3.4 illustrates the complete signal flow from human hand tracking to robot actuation. The AVP system estimates human hand configurations at 100Hz, and its output is mapped to the 13 actuated joints of the Adapt Hand through a custom joint mapping function. This mapping converts human joint angles to robot joint space, and consists of a linear mapping (value multiplier and offset) to account for the kinematic differences with the simulated hand.

The mapped joint targets are applied to the physical Adapt Hand hardware by reading the signals from the simulator, and not from the AVP. This is done to make sure that the Adapt Hand is able to copy the exact movements of its counterpart in Isaac Sim. In simulation, Isaac Lab's position controllers execute the joint commands using the physics solver, while the real hardware receives identical position control targets through ROS packages running on a dedicated Raspberry Pi.

Visual validation is used in absence of hand sensors for accurate measurements. As long as the motion is visually consistent, grasping of objects will be successful given the compliance of the robotic hand and the presence of a silicon skin [16].

The demonstrated sim2real transfer validates our approach for collecting human demonstrations and is therefore directly applicable to real-world deployment.

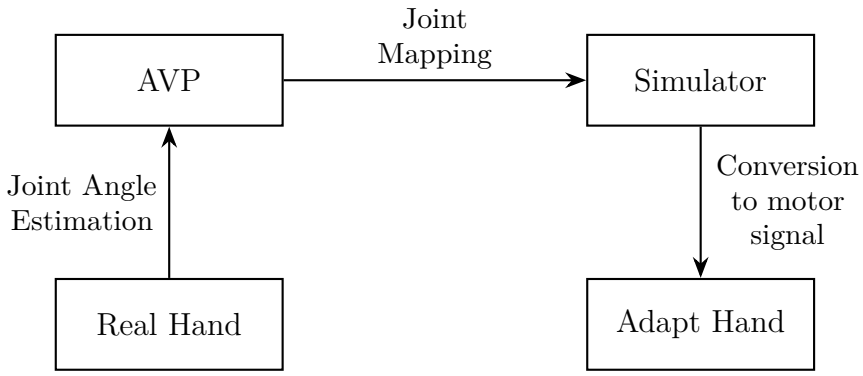


Figure 3.4: AVP teleoperation system and Adapt Hand sim2real diagram and signal flow

3.3.4 Task Definitions for Behavioral Cloning

Our BC implementation is applied to two environments of increasing complexity. The first one is the lift task, performed both by the Franka arm with the 1-DoF gripper and by the UR5 arm with the 15-DoF Adapt Hand, where we used synthetic expert demonstrations for the Franka and human teleoperated demonstrations for the UR5. The second task is a custom pick-and-place environment with the Adapt Hand, requiring coordination and a multi-phase trajectory.

Common Data Structure

Both tasks share a consistent data collection framework while differing in complexity and demonstration source. The state-action pairs of each trajectory are stored in HDF5 format, which can be easily visualised and accessed by PyTorch data loading features.

For the UR5 arm platform (6-DoF), the observation space includes:

- Joint states: 6 arm joint positions
- Hand configuration: 1 gripper position (parallel-jaw) or 15 hand joint angles (Adapt Hand)
- End-effector pose: 7-element pose vector (position + quaternion)
- Task-specific observations: object position for manipulation tasks (3 elements)
- Action space information: previous action history
- Temporal information: joint state history from previous time steps

The total amount of observation data for the UR5 with Adapt Hand configuration is of dimension 52 without the inclusion of temporal information. Some components like joint velocities were omitted due to their low impact in performance in order to reduce the computational complexity. In particular, the addition of joint state history increases the observation space dimension by 31 if actions are not included, and 52 if actions are included, leading to an considerable increase in computation effort for each added past frame. To further simplify this framework, we observed that using joint states of arm and Adapt Hand, and the target object position is enough to achieve success on the proposed tasks. This is likely because pick and place remains a relatively easy task in the field of dexterous manipulation.

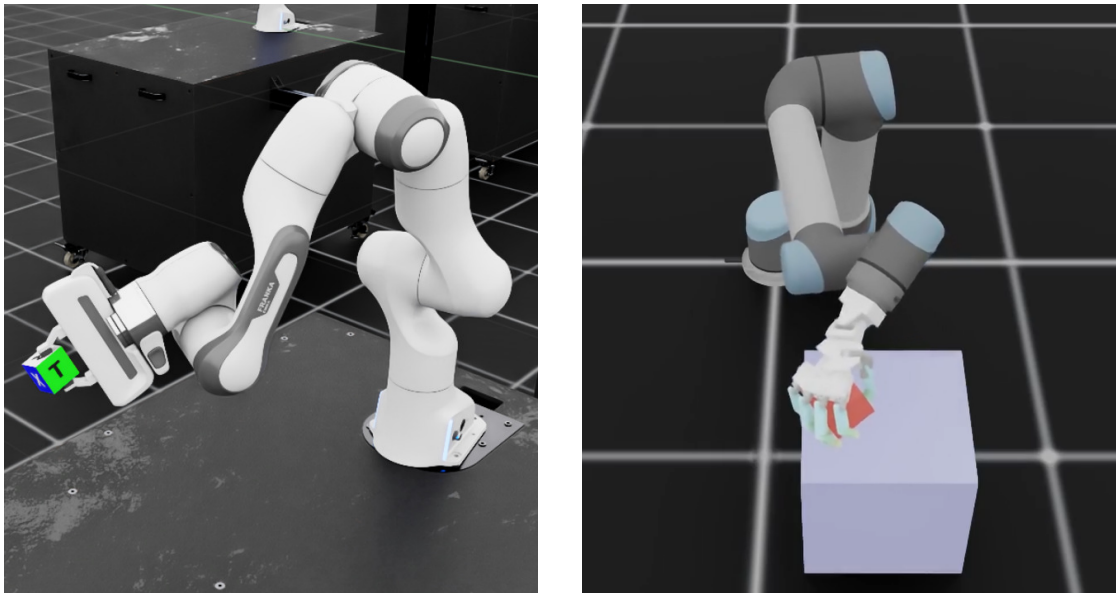
The action space represents joint position targets sent to the robot controllers:

$$\mathbf{a} = [q_{arm}, q_{hand/gripper}] \tag{3.30}$$

where $q_{arm} \in \mathbb{R}^7$ are the arm pose targets and $q_{hand/gripper}$ varies based on end-effector configuration.

Task 1: Lift Task

The lift task is used as a baseline for comparing synthetic demonstrations, generated from the expert RL policy, with human teleoperation data. The task involves picking up a 6cm cube and lifting it above a minimum height threshold of 4 cm. The most important feature of this task for the Adapt Hand is the grasping motion and making sure the cube remains inside the hand after being lifted.



(a) Lift task with parallel-jaw gripper (b) Lift task with Adapt Hand

Figure 3.5: Behavioural cloning lift task environments: parallel-jaw gripper and Adapt Hand

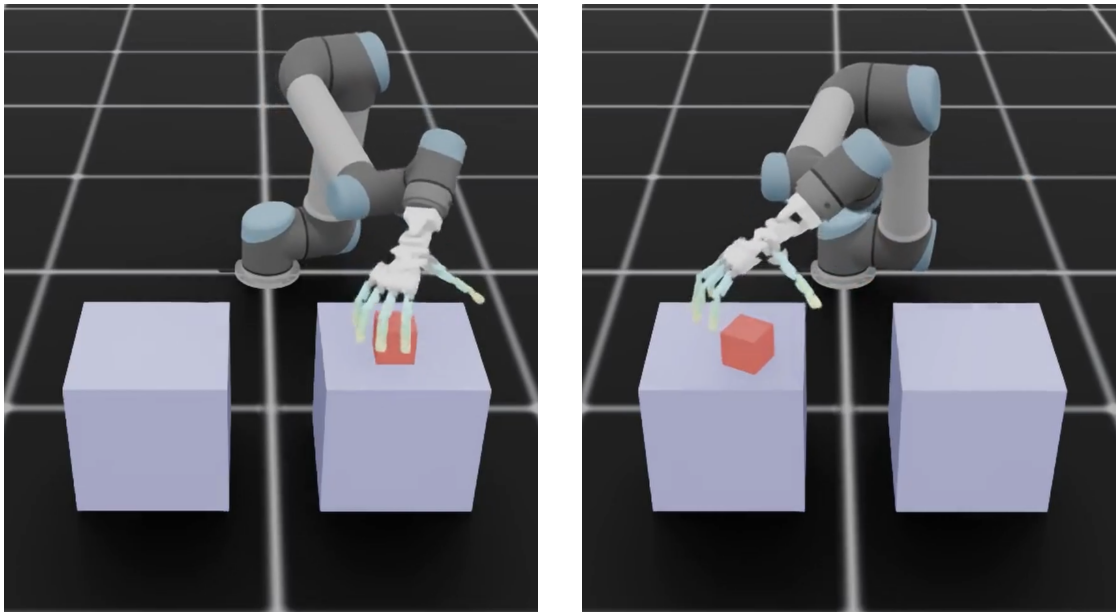
The workspace uses two rigid objects: a large fixed-base box (25cm × 25cm × 25cm) and a small cube (6cm × 6cm × 6cm, 0.1kg mass) which represents the lift target. The cube is positioned on top of the base box with randomized initial position within ±6cm of the base's centre.

Figure 3.5 shows the setup of the two environments, where the Franka Lift environment is pre-defined within Isaac Lab, and the Adapt Hand Lift is entirely custom-made.

Task 2: Pick-and-Place

The pick and place task is significantly more complex than lift, requiring timing and coordination, especially considering that to achieve successful grasping, the concurrent motion of all finger joints is required. The robot starts around 15cm away from the cube to be transported, which spawns on top of the same base box as in Task 1. The base box is now centred in (0.65, 0.2, 0.125), and has a side of 25cm. The target cube is randomly sampled from a 6 by 6 cm distribution on top of its base, and the final position target is at the top of a second box, centred in (0.65, -0.2, 0.125), symmetrical to the base box with respect to the x axis and with the same size.

Figure 3.6 shows the pick and place environment, specifically a frame from the



(a) Pick and place task: initial part (b) Pick and place task: final part

Figure 3.6: Behavioural cloning pick and place task environment: beginning and end of task frames

beginning of the task when the robot is about to grasp the cube (left), and a frame towards the end of the task where the robot has released the target cube at its destination.

Because of these displayed steps, it's clear that the task involves conflicting objectives for an ideal reward function: the hand must first grab the cube and later release it, so any reward signal that incentivises getting close to the target of maintaining the grasping motion will cause the policy to struggle learning to release the cube once the target destination is reached. This is therefore a problem that is easily circumvented by a behavioural cloning approach, where by learning a specific mapping between state-action pairs, the network is able to distinguish the robot states in which grasping is required from the ones in which releasing is required.

Human teleoperation allowed us to perform the required motions with the Adapt Hand with ease, without the possibility to access an expert RL policy.

3.3.5 Training Setup

Our behavioural cloning experiments use a custom implementation of both mean square error and maximum likelihood estimation algorithms as supervised learning

from expert demonstrations. We use a neural network to map observations to actions, employing the expert trajectories as ground truth.

Network Architecture

The BC policy network utilizes three fully connected layers structured as follows:

- Policy network: $512 \rightarrow 512$ hidden units with ReLU activation
- Input layer: Simplified observation tensor of dimension 24 for UR5 and Adapt Hand
- Output layer: Matches action dimension (21 for position control of UR5 and Adapt Hand)
- Regularization: Weight decay ($\lambda = 1 \times 10^{-4}$) to reduce overfitting

The network is wider than the one used for RL, as an increase in capacity helps with the complexity and high-dimensionality of the supervised learning problem, especially for tasks like pick and place which have a longer horizon compared to lift (approximately twice the number of time steps per trajectory).

BC Training Configuration

Two loss formulations are implemented and compared for each task: MSE and MLE behavioural cloning. Considering the aim for this thesis is to implement a novel hybrid algorithm and perform comparison with state-of-the-art baselines, we choose to conduct our experiments using the MLE framework, as its probabilistic formulation is more suited to be combined with reinforcement learning algorithms like PPO.

$$\mathcal{L}^{MSE}(\theta) = \mathbb{E}_{(s,a)\sim\mathcal{D}}\left[\|\pi_\theta(s) - a\|^2\right] \tag{3.31}$$

$$\mathcal{L}^{MLE}(\theta) = -\mathbb{E}_{(s,a)\sim\mathcal{D}}[\log \pi_\theta(a|s)] \tag{3.32}$$

The main BC hyperparameters are configured as follows:

$$\text{Learning rate: } \alpha = 1 \times 10^{-4} \tag{3.33}$$

$$\text{Batch size (train): } = 512 \tag{3.34}$$

$$\text{Batch size (validation): } = 512 \tag{3.35}$$

$$\text{Weight decay: } \lambda = 1 \times 10^{-4} \tag{3.36}$$

$$\tag{3.37}$$

BC training differs from RL, as it samples from a fixed dataset rather than sampling from environment interaction.

Because of this, BC training requires validation strategies to detect overfitting, particularly given the temporal correlation of data in expert trajectories. We encountered significant challenges with standard validation approaches, and we think future work could expand on this topic and formulate a more robust way to select validation techniques.

We initially flattened all observations and actions, and applied a random 80-20 split between training and validation data. This failed to achieve meaningful validation, as consecutive time steps within trajectories are intrinsically correlated, and a random split leads to nearly identical data distribution in both training and validation sets, causing the validation signal value to be very close if not identical to the value of the training loss. We visually noticed clear overfitting signs during the deployment of the policy, while validation values remained relatively low.

We subsequently implemented trajectory-wise splitting, reserving 20 trajectories for validation purposes. However, this approach also provided insufficient validation capabilities, achieving the same performance as the random 80-20 split. The high similarity between trajectories means that validation remains artificially similar to training performance, denoting similar data distributions like the first method. Policies again exhibited overfitting behaviour in deployment, but we were unable to accurately identify it from validation data.

These challenges highlight an issue in using traditional machine learning validation techniques for robotics behavioural cloning methods, which are inadequate for temporally correlated demonstration data. Future work should explore alternative strategies for validation, such as task-specific validation or domain shift deployment.

Despite these limitations, regularization techniques such as weight decay proved to be helpful for mitigating overfitting while maintaining convergence. Model selection was mainly based on loss function convergence and success rate evaluation. Task success was defined as lifting the cube above 4cm for Task 1, and transporting the cube to within the target area for Task 2 (above the second box). The metric chosen for Task 2 disregards the cube release phase, but nonetheless is a good indicator of a successful policy.

3.4 Hybrid Learning Approaches

3.4.1 Motivation and Overview

Reinforcement learning and behavioural cloning both have limitations when independently applied to solve dexterous manipulation tasks. RL requires complex and lengthy hand-designed reward engineering, which increases in complexity with the task, especially given longer horizons and multiple phases with different objectives. BC suffers from compounding errors due to covariate shift and poor generalisation outside of states covered by expert trajectories [9]. Hybrid approaches leverage the

strengths of both frameworks by combining them within the same algorithm, and they try to mitigate the individual weaknesses mentioned above.

We first explore inverse reinforcement learning theory, analysing its general formulation and the concept of cost function learning. We then formulate the imitation learning task in terms of a min-max problem between a generator and a discriminator network with Generative Adversarial Imitation Learning (GAIL) [7], and we implement improvements on such setup, strengthening its mathematical formulation by leveraging Wasserstein GANs and 1-Lipschitz gradient constraint on the policy network [11, 10].

Finally, we propose a novel implementation within NVIDIA Isaac Lab of a maximum likelihood estimation behavioural cloning algorithm, combined with proximal policy optimization. We define a mixed optimization objective optimised end-to-end by sampling both from expert demonstrations and environment interactions. The algorithm requires a simplified reward function and performs neural network updates for both its components within the same loop. Previous works implemented a similar setup for the autonomous driving with object avoidance task, using soft actor-critic [13] instead of proximal policy optimization.

Our exploration of hybrid imitation learning methods allows us to develop practical solutions for high-dimensional manipulation tasks, having state-of-the-art baselines for comparison, and more complex approaches.

The core insight of hybrid methods is that expert demonstrations can be used to provide guidance through a complex task, while RL can provide exploration and mitigate expert coverage deviation, incentivising task success and fine-tuning behaviour.

3.4.2 Inverse Reinforcement Learning Background

Inverse reinforcement learning aims to recover a cost function form expert demonstrations. Reinforcement learning can then be used with this cost function to generate a policy, effectively leveraging demonstrated trajectories for guiding the robot's behaviour.

IRL is based on the assumption that expert behaviour is optimal with respect to the unknown cost function that the algorithm seeks to recover from observed trajectories. The IRL objective can thus be formulated as a min-max optimisation problem as follows [7]:

$$\max_{c \in \mathcal{C}} \left(\min_{\pi \in \Pi} -H(\pi) + \mathbb{E}_{\pi}[c(s,a)] \right) - \mathbb{E}_{\pi^E}[c(s,a)] \tag{3.38}$$

where an inner minimisation finds the best policy π^*, yielding the lowest value under the cost function $c(s,a)$. The outer maximisation finds the cost function $c \in \mathcal{C}$ which provides the largest difference between the generated policy and the

expert. This is done because the goal is finding the cost for which the expert represents optimality, following the algorithm's initial assumption. The intuition behind this is that the expert's expected cost must be lower than (or equal to) the best possible policy under $c(s, a)$, and therefore optimal behaviour must be given by imitating expert demonstrations.

Once IRL has recovered a cost function, reinforcement learning is applied to optimise a policy under this learned cost. The entire process can be formulated as the minimisation of a convex function ψ^* as follows [7]:

$$\text{RL} \circ \text{IRL}_\psi(\pi_E) = \arg\min_{\pi \in \Pi} -H(\pi) + \psi^*(\rho_\pi - \rho_{\pi_E}) \tag{3.39}$$

where ρ_π and ρ_{π_E} are the occupancy measures (state-action visitation distributions described in section 2.2.3) of the learned policy and expert policy respectively. RL minimizes the difference between the occupancy measures under the recovered cost function ψ^*, and $H(\pi)$ is an entropy term which maintains a level of exploration for the agent.

Theoretical guarantees are provided for this process, but running RL in an inner loop is computationally expensive and impractical for high-dimensional and continuous control tasks [7], such as robotic manipulation with a multi-finger hand. This motivates the search for robust algorithms which can leverage expert trajectories without the recovery of an underlying cost function, directly measuring the distance between expert and agent's action distributions.

3.4.3 Generative Adversarial Imitation Learning Implementation

Generative Adversarial Imitation Learning (GAIL) [7] is a commonly used state-of-the-art imitation learning framework that aims to overcome the weaknesses just described for IRL.

GAIL Architecture and Training

GAIL is an alternative to IRL that avoids recovering an explicit reward function. This algorithm instead formulates imitation learning as a distribution matching problem, where a generator network G_θ outputs a policy which aims to be as close as possible to the one of the expert. A discriminator network D_ϕ compares the trajectories of the generated policies with the ones from the expert, and provides a feedback signal to the generator. This structure is entirely similar to Generative Adversarial Networks (GANs) [21], hence the algorithm name.

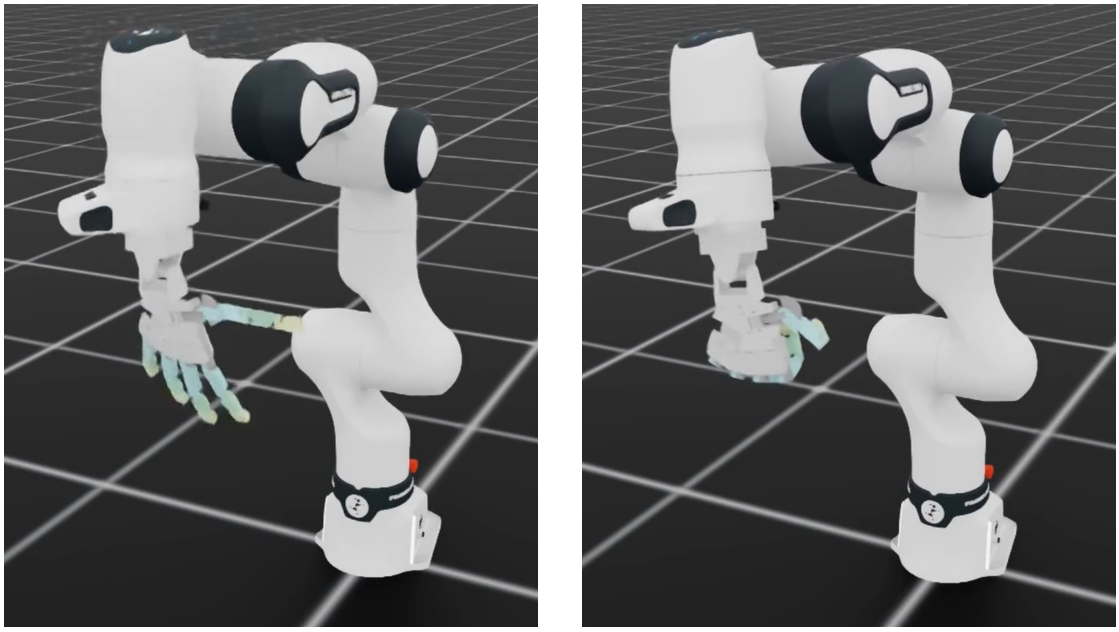
The generator employs PPO to maximise the following reward signal given by the discriminator:

$$r(s, a) = \log(D(s, a)) - \log(1 - D(s, a)) \tag{3.40}$$

where there is a balance to be maintained by the two networks. The discriminator acts as a classifier for state-action pairs.

The network architecture for the discriminator is a three layer MLP with hidden dimensions $256 \rightarrow 128 \rightarrow 64$, and for the generator it matches the PPO implementation of section 3.2. The most relevant hyperparameter for this framework is the ratio between generator and discriminator updates within the same neural network update cycle: we found that a proportion of 5 to 1 in favor of the generator produced the best results. If the discriminator converges much faster than the generator, which is the case with an equal ration of updates, its feedback becomes progressively less meaningful, leading to less accurate policy updates for the generator and eventual collapse.

Wasserstein GAIL Modification



(a) WAIL algorithm: open hand (b) WAIL algorithm: task completion

Figure 3.7: Wasserstein Generative Adversarial Imitation Learning applied on a hand motion task

Standard GAIL training suffers from issues typical to adversarial network approaches, such as instability and policy collapse. This motivated the implementation of Wasserstein GAIL [10], which is an algorithmic variation that replaces the Jensen-Shannon divergence in standard GAIL (derived from the GAN formulation) with the Wasserstein distance, which provides more stable gradients and better convergence

properties. This particularly focuses on the meaningfulness of the discriminator feedback when expert and policy distributions are distant.

The Wasserstein formulation modifies the discriminator objective by removing the logarithmic terms and enforcing a 1-Lipschitz constraint through gradient penalty, and its objective function can be written as follows:

$$\mathcal{L}^{WGAIL} = \mathbb{E}_{(s,a)\sim\pi_\theta}[D(s,a)] - \mathbb{E}_{(s,a)\sim\pi^E}[D(s,a)] \tag{3.41}$$

$$+ \lambda_{GP}\mathbb{E}_{\hat{x}}[(\|\nabla_{\hat{x}} D(\hat{x})\|_2 - 1)^2] \tag{3.42}$$

where $\hat{x}$ represents interpolated samples between expert and generated data, and $\lambda_{GP} = 10$ is the gradient penalty coefficient. This overall results in a simpler (by removing the logarithmic discriminator term) and more stable implementation of this imitation learning framework.

Despite the shown theoretical advantages, our Wasserstein GAIL implementation failed to achieve satisfactory performance on robotic manipulation tasks, and instead only managed to complete hand configuration tasks, like the one shown in 3.7, where the agent needs to replicate the expert's hand motion. This environment still involved the motion of 15 separately actuated joints, but we were not able to achieve arm-hand coordination, as training remained unstable with frequent mode collapse.

Our Wasserstein GAIL task has an observation tensor of dimension 15, containing all the joint angles of the Adapt Hand and its wrist, and an action space of equal dimension, as lower-level PD controllers within Isaac Lab implement position control given joint angle targets. Although this is a relatively high-dimensional task, the horizon is of about 2 seconds, and the complexity of the task itself isn't high.

3.4.4 Behavioral Cloning with Proximal Policy Optimization

Algorithm Overview

This hybrid algorithm combines behavioural cloning with proximal policy optimization by optimising a combined objective function within the same training loop. The algorithm's base is RSL-RL's PPO implementation [18], which uses both Generalized Advantage Estimation [5] and the classic surrogate objective to obtain a robust and efficient learning framework. A maximum likelihood estimation behavior cloning algorithm is manually implemented within PPO, and its objective function is manually combined.

This approach aims to address the weaknesses of both base algorithms by leveraging their strengths [9]: the main policy behaviour is driven by BC, avoiding the need for a complex hand-engineered reward function, while exploration and

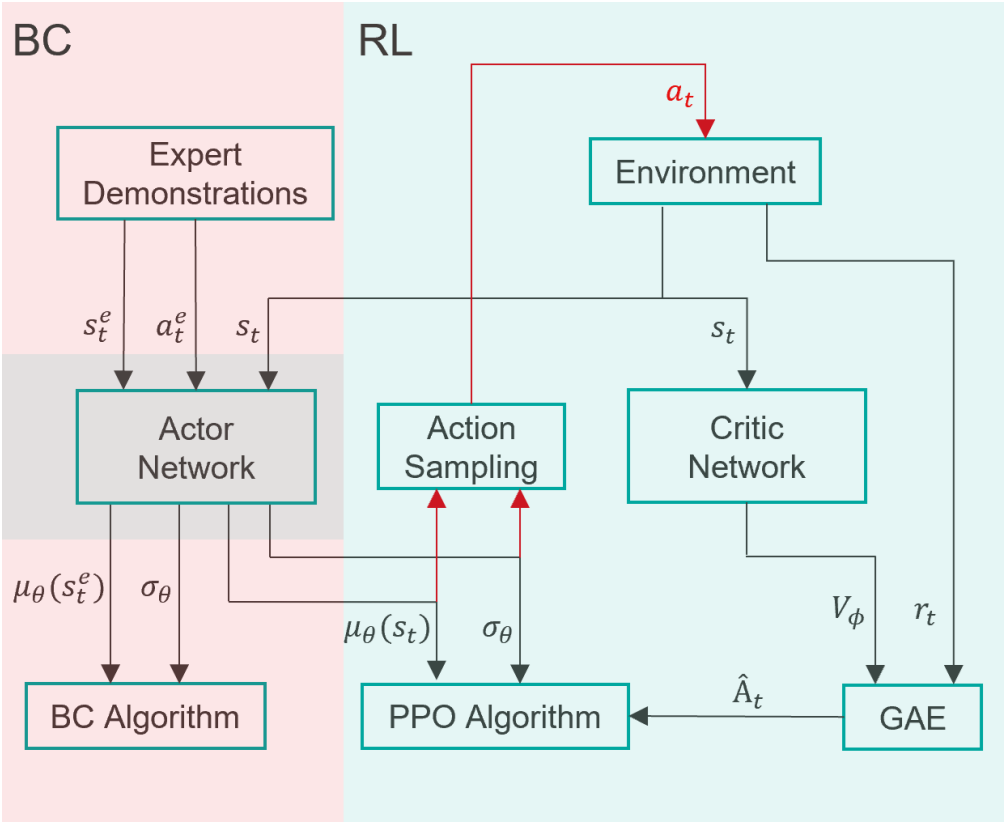


Figure 3.8: Block diagram of the BC+PPO algorithm.

out-of-expert-distribution states are handled by PPO. PPO also fine-tunes the policy and increases success and task completion with the use of high-valued sparse reward signals.

The algorithm maintains a single policy network that now simultaneously learns from expert demonstrations through supervised learning, and from environment interaction through reinforcement learning. Unlike previously described adversarial approaches, this direct combination of objectives avoids training instability typical of GANs, while providing a simpler implementation strategy and hyperparameter tuning process.

Figure 3.8 shows the complete structure of the algorithm: the Actor Network on the left is responsible of generating the policy and handling both expert demonstrations and environment interactions in the form of expert state-action pairs and environment states, while the Critic Network on the right is responsible of estimating the value function used for Generalised Advantage Estimation [5]. The action sampling is performed from the current policy distribution, using RSL-RL's PPO sampling, and actions are then executed on the environment to progress to

the next state.

Mathematical Formulation

The combined objective function integrates PPO and BC losses with a weighting parameter λ_{BC}:

$$\mathcal{L}_{total} = \mathcal{L}_{PPO} + \lambda_{BC} \cdot \mathcal{L}_{BC} \tag{3.43}$$

where $\mathcal{L}_{PPO}$ is the standard PPO clipped surrogate objective combined with value and entropy terms:

$$\mathcal{L}_{PPO} = \mathcal{L}_{surrogate} + c_1\mathcal{L}_{value} - c_2\mathcal{L}_{entropy} \tag{3.44}$$

and $\mathcal{L}_{BC}$ employs maximum likelihood estimation for better integration with the stochastic policy given by PPO:

$$\mathcal{L}_{BC} = -\mathbb{E}_{(s,a)\sim\mathcal{D}}[\log \pi_\theta(a|s)] \tag{3.45}$$

The weighting parameter λ_{BC} controls the relative importance of policy guidance using expert demonstrations. In our implementation, we set $\lambda_{BC} = 0.1$, as the log-likelihood BC estimation has initial values one order of magnitude higher than the reinforcement learning loss component, rendering it not influential if λ_{BC} were to be too high. In later stages of training, RL aims to fine-tune the policy and improve success rate, as the BC loss function decreases.

The combined loss is weighted by the hyperparameter λ_{BC}, which balances the two objectives. We set $\lambda_{BC} = 0.1$ in our experiments because the BC loss magnitude given by the negative log probability is significantly larger (at least one order of magnitude) than the PPO loss magnitude initially, when the policy is far from expert behavior. A value of $\lambda_{BC} = 1.0$ would cause the BC gradient to dominate completely, preventing the RL component from giving meaningful contributions to the learning signal. This weighting allows both components to be effective for each policy update.

Implementation Details

A custom demonstration buffer class manages expert trajectories with 80-20 train-validation split and efficient sampling. During each training iteration, the algorithm samples batches from both the environment rollout buffer (replicating the standard RSL-RL PPO implementation) and the demonstration buffer. Specifically, PPO performs 24 environment steps on each of the 4096 parallel environments, collecting a total of 98,304 transitions, which are divided into 4 mini-batches of 24,576

transitions each and processed over 5 epochs, for a total of 20 updates. Each of those 20 updates samples 512 additional transitions from expert trajectories.

The BC component samples expert state-action pairs, it updates the policy's action distribution using expert states, and it computes the negative log probability of expert actions under the current policy update. This gradient signal encourages the policy to follow the expert by increasing the likelihood of sampling an action close to the expert's choice. On the other hand, exploratory behaviour is given by the state distribution's entropy computation, and PPO provides task completion incentives.

BC+PPO algorithm implementation parameters:

$$\text{BC batch size:} \quad = 512 \tag{3.46}$$

$$\text{BC weighting: } \lambda_{BC} = 0.1 \tag{3.47}$$

$$\text{Weight decay:} \quad = 1 \times 10^{-4} \tag{3.48}$$

$$\text{Demonstration split:} \quad = 80\% \text{ train, } 20\% \text{ validation} \tag{3.49}$$

Network architecture maintains consistency with our BC-only implementation using $512 \rightarrow 512$ hidden units to handle the increased learning complexity from dual objectives. The shared policy network outputs stochastic policy distributions for both the BC loss and the PPO loss, while the Critic Network is used for advantage computation and PPO gradient direction.

We do not use dropout regularization in our approaches involving PPO. Standard dropout introduces unpredictability in policy gradient algorithms: during environment rollout, the policy is evaluated with one random dropout mask to collect trajectories (state-action pairs), while during training updates the same state-action pairs are evaluated with different dropout masks to compute probability ratios. This leads PPO's ratio $r_t(\theta) = \pi_\theta(a_t|s_t)/\pi_{\theta_{old}}(a_t|s_t)$ to compare different network architectures entirely, due to the different masks applied, rather than different parameters, essentially breaking the learning process and causing the policy to diverge [22]. Consistent dropout variants exist that maintain the same mask throughout trajectories, but we opt for the simpler approach of using only weight decay ($\lambda = 1 \times 10^{-4}$) for regularization, which does not affect the computation of the forward pass during transition collection in rollout.

Training Procedure

The training runs for a maximum of 1500 iterations as specified in the algorithm configuration. Model checkpoints are saved every 50 iterations for later evaluation. All BC-PPO experiments use the same base hyperparameters as described above, with only task-specific elements (observation/action dimensions, demonstration paths) varying between tasks.

Chapter 4

Experimental Results

This chapter presents the experimental evaluation of reinforcement learning, behavioural cloning, and hybrid approaches such as Wasserstain Generative Adversarial Imitation Learning (Wd-GAIL) and BC-PPO for dexterous robotic manipulation. Performance is compared using different metrics like convergence time, training time, and task success rate. Section 4.1 defines the metrics used in this chapter, while sections 4.2, 4.3, and 4.4 present results for RL, BC, and hybrid approaches respectively. A comparative analysis is performed at the end between BC and the proposed hybrid method, and advantages and limitations are considered throughout.

4.1 Evaluation Metrics

To evaluate the performance of the different learning approaches presented, we use the following metrics.

4.1.1 Success Rate

This is a task-specific metric, and is defined as follows:

- **Reach with Hand Pose Task**: The end-effector reaches within 0.01 m of the target position and 0.15 radians of the target orientation, and the distance between thumb and index fingertips is below 0.005m.
- **Lift Task**: The target object is raised above the minimum height threshold of 4cm and maintained for at least 50 time steps, equivalent to 0.5 seconds.
- **Pick-and-Place Task**: The target object is transported to within the area of the second large box, centred at (0.65, -0.2, 0.125) with a side of 0.25 m. Therefore, the object's (x,y) coordinates must satisfy: $x \in [0.525, 0.775]$ and $y \in [-0.325, -0.075]$.

Success rate is computed as the percentage of successful episodes over 30 evaluation episodes for all experiments. This metric can be arbitrarily precise depending on the task considered: the pick and place goal, for example, can be further complicated by defining the it to be a specific point on the surface of the second large box, or even at a specific height above another object. We chose a simplified success condition for pick and place because the teleoperated trajectories do not position the target cube always in the same location, and the metric we chose is indicative enough of the performance of our algorithm in comparison with the baselines.

4.1.2 Convergence time

Convergence time is measured as the number of environment time steps required to achieve a stable success rate. The reason for this choice is that all tasks stabilise at a different success rate percentage. In particular, the ones employing the Adapt Hand do not achieve success rate much higher than 60% due to the complexity of the state and action spaces, thus we aim to compare all approaches in a fair way.

For reinforcement learning-based methods (PPO, Wd-GAIL, BC-PPO), we calculate time steps as: iterations to convergence times 98,304 steps per iteration. For pure behavioural cloning, which employs supervised learning without environment interaction, we instead report the number of expert demonstrations used for the experiment, as the method does not collect from the environment during training, and each iteration involves passing through the entire expert dataset once.

4.1.3 Training Time

Training time is reported as wall-clock time in minutes, measured on our workstation, which has an NVIDIA RTX 4090 GPU. This includes both data collection and training time for all methods. Teleoperated data collection time is estimated based on multiple recording batches, as that represents the way it was actually acquired.

4.2 Reinforcement Learning Results

This section presents results for pure reinforcement learning using RSL-RL's PPO implementation [18], which serves as a baseline of this learning approach, and displays the main strengths and limitations of reward engineering.

4.2.1 Franka Lift Task

The PPO algorithm successfully learns the Franka Lift task, achieving a final success rate of 100% after 1500 training iterations. Figure 4.1 shows the training progression over 1500 epochs (approximately 147.5 million environment steps, as computed

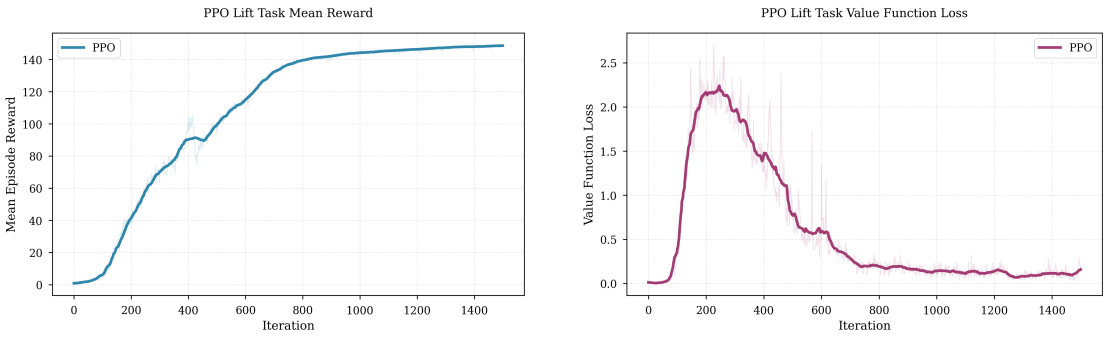


Figure 4.1: PPO training metrics on Franka Lift task over 1500 iterations.

in section 3.2.4. The left plot shows steady reward improvement, computed as the mean over the 4096 parallel environments, with the policy reaching a plateau around iteration 800. The mean episode reward converges to approximately 150, indicating consistent and fast task completion, as none of the episodes trigger a failure condition and the cube is always lifted above the minimum height threshold. Training shows variance in early iterations, which is typical of RL, as the policy explores the available state space before stabilising on a specific strategy. The exploration is mainly given by the entropy term in PPO's objective function, computed using the standard deviation of the policy distribution, which gradually decreases as the agent becomes more confident.

The right plot in Figure 4.1 shows the value function loss decreasing from an initial peak to values very close to zero by iteration 1000, indicating that the critic network successfully learns to predict expected returns. The initial spike in value loss is expected, as the critic has to adjusts to early policy improvements, which although mitigated by PPO's clipping factor ($\epsilon = 0.2$), they still cause an initial increase in value for this loss component. After convergence, the value loss remains stable, and the timing coincides with the reward signal reaching its plateau.

Training finished in 536.81 seconds on an NVIDIA RTX 4090 GPU with 4096 parallel environments. The parallelized simulation enabled the collection of 98,304 environment transitions per iteration, which greatly speeds up learning and updates.

Figure 4.2 demonstrates the learned policy deployed in simulation. The policy shows smooth motion, fine-tuned by the curriculum learning components of the reward function described in section 3.2.2, which further penalise large actions and joint velocity. From left to right the figure shows three frames, corresponding respectively to the beginning of the task, the grasping of the cube, and task completion with the lifting of the cube. The learned policy is able to generalise well with the initial randomised positions of the cube, although the randomisation area remains relatively small (square area with a 6 cm side).

This task establishes a good baseline for comparison with other more complex

algorithms.

- Convergence time: 286 seconds / 800 iterations
- Final success rate: 100%
- Reward engineering process: 4 components function with curriculum learning
- Action space dimension: 8

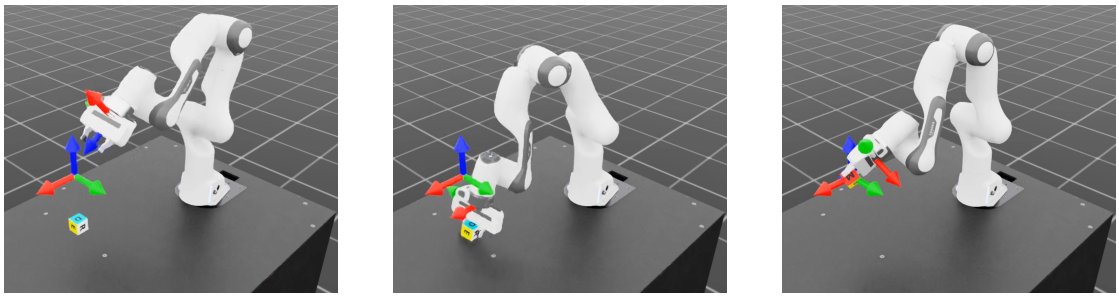


Figure 4.2: Progression of Franka Lift task.

4.2.2 Franka Reach with Hand Pose Task

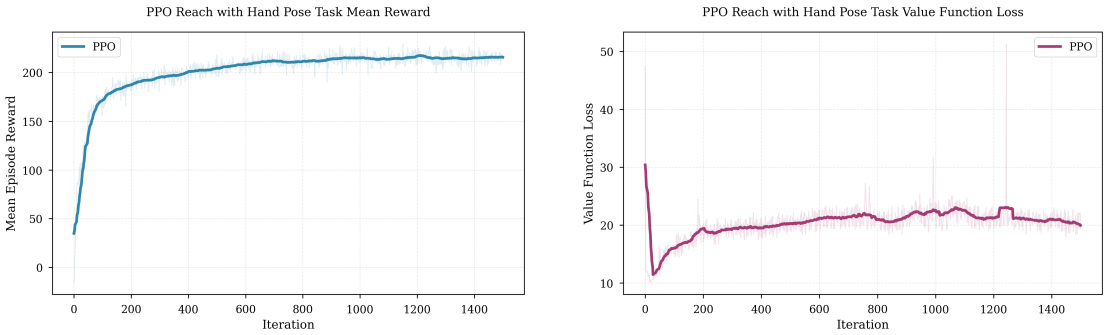


Figure 4.3: PPO training metrics on Franka Lift task over 1500 iterations.

The PPO algorithm successfully learns the Franka Reach with Hand Pose task, achieving a final success rate of 100% after 1500 training iterations. This task combines the Franka Reach task, for which the arm moves using Isaac Lab's operational space controller with null space projection, described in section 3.1.2, with a custom hand pose task, where all finger joints are controlled in position and all couplings are handled with code. This task has significantly larger observation

and action spaces (see section 3.2.3), but in contrast to Franka Lift, it does not require interaction with objects.

Figure 4.3 shows the training progression over 1500 epochs. The left plot demonstrates steady reward improvement, with the policy reaching a plateau after around 600 iterations. This shows that although the control problem is higher dimensional compared to the previous task, object interaction remains a bigger challenge for learning approaches. The mean episode reward converges to approximately 210, indicating consistent completion and speed in reaching the target pose, considering the maximum length of 300 time steps, and the penalty terms of the reward function based on target distance (see section 3.2.3 - Reward Function Design).

The right plot in Figure 4.3 shows value function loss stabilizing at approximately 21, which is notably higher than the lift task but remains within acceptable bounds given the task configuration: the critic predicts returns that depend on reaching the target wrist pose and fingertip distance minimisation, where the success threshold is of 0.005m. The value loss magnitude of 21 corresponds to a prediction error of approximately $\sqrt{21} \approx 4.6$ reward units per state, which corresponds roughly to 2% error relatively to the mean episode reward. This accuracy is sufficient for advantage computation and for effective policy updates, as shown by the high rewards and task success rate.

Training finished in 1232.27 seconds on an NVIDIA RTX 4090 GPU, which is an averaged value over 4096 parallel environments. This result is approximately twice the duration of the lift task, which can be attributed to the higher dimensional action space (22 for arm, wrist and hand, 8 for arm and standard gripper) and the computations performed by the operational space controller each time step. Despite the increased computation complexity, learning is still quite fast and performed with the same settings as the simpler Franka Lift task.

Figure 4.4 displays the learned policy deployed, which exhibits coordinated motion between arm and hand, maintaining the pinch gesture after reaching the target end-effector pose. The learned behaviour generalizes well on the randomised wrist targets, which are sampled from a cylindrical task space distribution described in section 3.2.3.

This task establishes the capability of pure RL to solve complex dual-objective manipulation problems, albeit with increased reward engineering complexity compared to simpler tasks.

- Convergence time: 492.9 seconds / 600 iterations
- Final success rate: 100%
- Reward engineering process: 6 component function without curriculum learning
- Action space dimension: 22

RL is able to solve high-dimensional tasks involving the Adapt Hand, but the reward function is already relatively complicated, and would become a major limitation for tasks with higher complexity than this one.

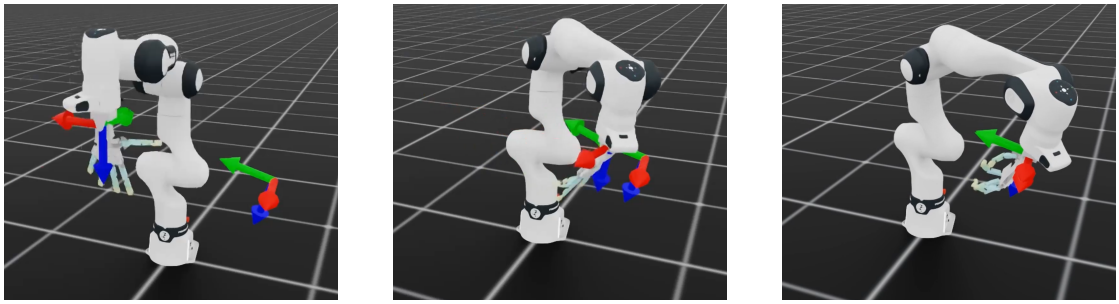


Figure 4.4: Progression of Franka Reach and Adapt Hand Pose task.

4.3 Behavioural Cloning Results

This section presents results for maximum likelihood estimation behavioural cloning, using demonstrations collected from trained RL policies for tasks that employ the standard 1-DoF gripper, and teleoperated demonstrations for tasks that employ the 15-DoF Adapt Hand. Unlike online RL, BC training does not require interaction with Isaac Sim, and it is implemented as supervised learning from expert state-action pairs.

4.3.1 BC with Synthetic Demonstrations - Franka Lift

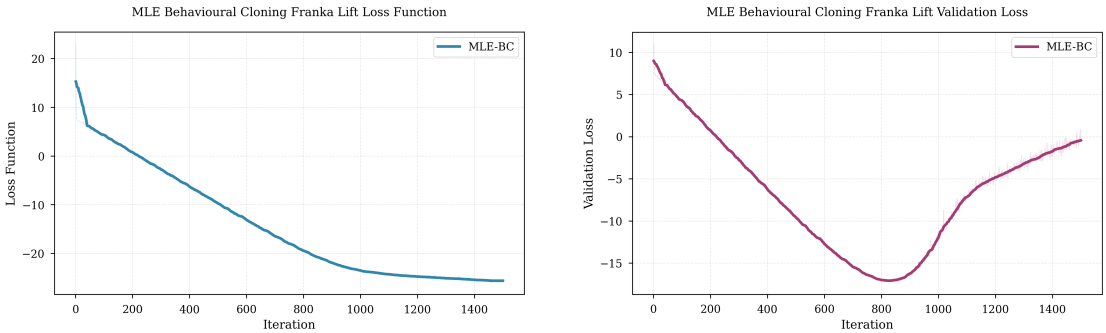


Figure 4.5: MLE Behavioural Cloning on Franka Lift task over 1500 iterations.

Using 100 demonstrations collected from the trained PPO policy described in Section 4.2.1, we train a BC policy using a maximum likelihood estimation

approach. The demonstrations were generated by deploying the converged RL policy (checkpoint corresponding to iteration 1500) in an environment with randomised cube positions, collecting state-action pair trajectories. No data augmentation is used for any of the described tasks, as the amount of demonstrations is observed to be sufficient to achieve satisfactory behaviour. Moreover, synthetically collecting demonstrations is a rapid process.

The BC policy is implemented as a three-layer MLP with 256, 128 and 64 hidden units per layer respectively, replicating the network architecture used for RL experiments described in the previous section. A Gaussian distribution over actions is parametrised, and the policy outputs action means directly, with a learned log standard deviation parameter shared across all demonstration data. Training aims to minimise the negative log likelihood of the sampled demonstrated actions under the learned policy distribution, as described in section 3.3.5, thus performing supervised regression in the probability distribution space, leading to a stochastic definition of the policy, which is the fundamental difference with mean squared error BC approaches.

Figure 4.5 shows the training progression over 1500 epochs. The left plot demonstrates a monotonically decreasing loss function, with training loss (negative log likelihood) decreasing from approximately 15 to -23 within the first 900 epochs, which represents the point of convergence, after which training slows down considerably, reaching a plateau as the network memorises the expert dataset. The negative value of the converged loss indicates that the network predicts actions with high confidence, as the learned standard deviation becomes progressively smaller as the policy distribution fits the demonstrations distribution.

The right plot displays significant overfitting behaviour, and this is the only instance out of all our experiments for which this is clearly identifiable from training loss-validation comparison. In fact, as described in the BC Training Configuration paragraph of section 3.3.5, future experiments in more complex environments will only show overfitting behaviour when visually checking the deployed policies, whereas loss and validation function plots will mostly be monothonically decreasing and will have very similar values in the end. The validation loss is initially similar to the training loss, reaching a minimum of approximately -17.2 around iteration 850. However, beyond this point, the validation loss value increases becoming less negative, while the training loss value continues decreasing. This creates a considerable gap between training and validation. This overfitting behaviour can also be noticed by visually analysing the policy, where the gripper often misses the cube and the arm's movement diverges from desired behaviour, and it's due to the fact that the network architecture has sufficient capacity to fully memorise the given dataset despite weight decay regularisation.

Training completed in 207.90 seconds on an NVIDIA RTX 4090 GPU, with convergence occurring at around 110 seconds (800 epochs). This is approximately

2.6 times faster than RL convergence (286 seconds), and it's largely because BC does not interact with the Isaac Sim simulator to collect environment interactions, which creates high computational overhead. It's interesting to notice that the required epochs to converge are the same between RL and BC approaches.

- Convergence time: 110 seconds / 800 iterations
- Final success rate: 93.3%
- Number of demonstrations: 100 trajectories
- Action space dimension: 8

Compared to the RL baseline (Section 4.2.1), BC offers faster training but requires recorded expert demonstrations. The success rate is slightly lower for BC likely due to the lack of complete coverage of the available state space by the expert demonstrations, and the limited generalisation capabilities of BC, although use of a stochastic policy.

BC with Teleoperation Data - UR5 + Adapt Hand Lift

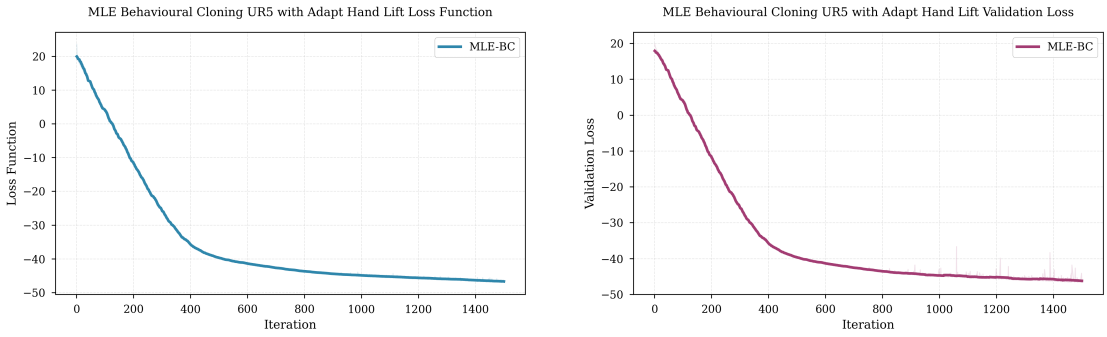


Figure 4.6: MLE Behavioral Cloning on UR5 + Adapt Hand Lift task over 1500 iterations.

From this point forward, experiments transition from the Franka Research arm to the Universal Robots UR5 manipulator. This change is done because of practical deployment plans for future work, for which the UR5 would be more readily available in our laboratory. The standard gripepr is also replaced by the Adapt Hand for this and successive tasks.

For this task, demonstrations are collected via human teleoperation and not through synthetic collection using a trained RL policy. The teleoperation setup described in section 3.3.3 allows a person to control both the arm and individual

finger joints to perform the lift task, introducing variability in the execution of the task. The increase in action space dimension (21 here compared to 8 with the standard gripper) together with the use of teleoperated expert trajectories contribute to the challenge of the control problem.

Figure 4.6 shows training progression over 1500 epochs. Unlike the Franka lift task, this experiment does not display clear overfitting in the loss curves. In fact, both training and validation losses decrease monotonically and remain close throughout training. The best model is therefore chosen by considering success rate metrics and visual analysis on the deployed policy, which resulted in the choice of checkpoint 400, a relatively early model. Nonetheless, this model achieves the best results that we observed.

The absence of train-validation divergence in the loss plots indicates that validation in this BC approach fails to predict deployment performance and help with model selection, hinting at the need in future works for better validation strategies.

Training completed in 380 seconds, with the optimal checkpoint reached at approximately 101 seconds (iteration 400). Training time remains comparable to the Franka Lift task despite the increased complexity, but overfitting seems to occur twice as fast.

- Convergence time: 101 seconds / 400 iterations
- Final success rate: 86.7%
- Number of demonstrations: 100 trajectories
- Action space dimension: 21 (6 arm + 15 hand joints)

The success rate of 86.7% is a decrease compared to the Franka BC baseline (93.3%). This performance degradation is attributable to the increased action space dimension, and the variability in the teleoperated trajectories. One of the main challenges in employing imitation learning for dexterous manipulation is in fact the quality of expert demonstrations, which have a major impact on the possibility of convergence and the final success rate of the task. Task completion is highly dependent on the provided coverage of the available task space, which scales in complexity with the dimensionality of the control problem.

Figure 4.7 demonstrates successful task execution with the UR5 and Adapt Hand system. The policy exhibits coordinated motion between arm and hand, with the fingers correctly timing closure around the cube and the arm lifting above the target height threshold of 4 cm. However, the learned behaviour is less consistent than the Franka Lift baseline, occasionally exhibiting failure modes typical of BC's poor generalisation and covariate shift problem.

Figure 4.8 displays two example BC failure cases that represent the majority of the 16.7% failure rate. The left image shows a spatial error: the arm undershoots

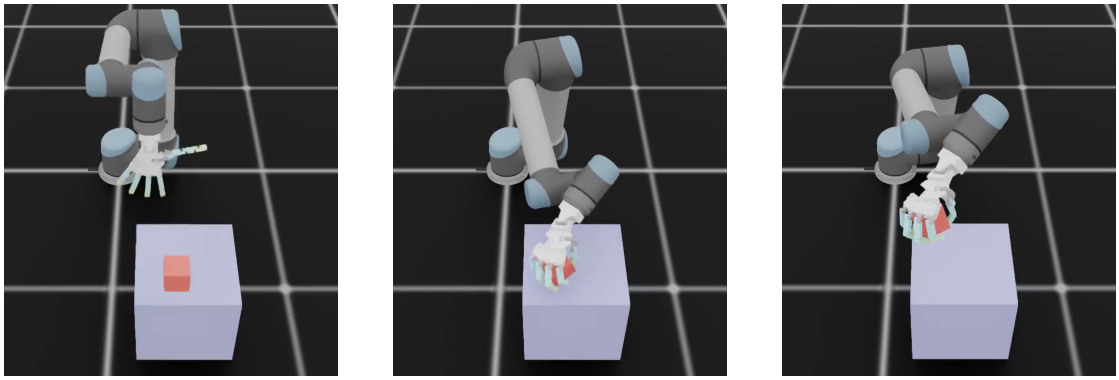


Figure 4.7: Successful execution of UR5 + Adapt Hand lift task.

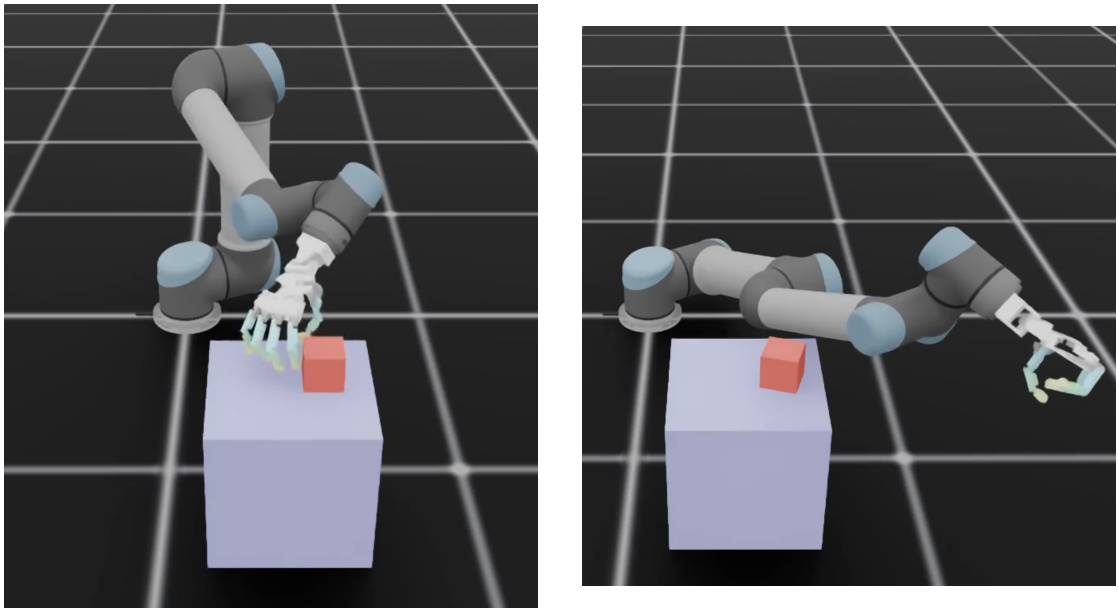


Figure 4.8: Example failure modes in BC deployment.

the real position of its target, and the hand closes its fingers without having an object to grasp, and subsequently it is lifted up without its target. This mainly occurs when the cube's initial position is sampled from a region poorly covered by the expert demonstrations, and the agent is unable to generalise and achieve task completion. The right image shows a more severe case where due to compounding errors the arm exhibits complete behavioural divergence and deviates from its target. This is likely due to the policy encountering a state sufficiently far from the expert distribution, leading to arbitrary action predictions which continue over time, moving further out-of-distribution.

These failure modes highlight BC's fundamental limitation: without online interaction with the environment and feedback during training (in the form of a reward signal), the policy has no mean to recover from errors or explore the task space which received poor coverage from the expert. hybrid approaches aim to address these limitations and learn additionally from experience.

BC with Teleoperation Data - UR5 + Adapt Hand Pick and Place

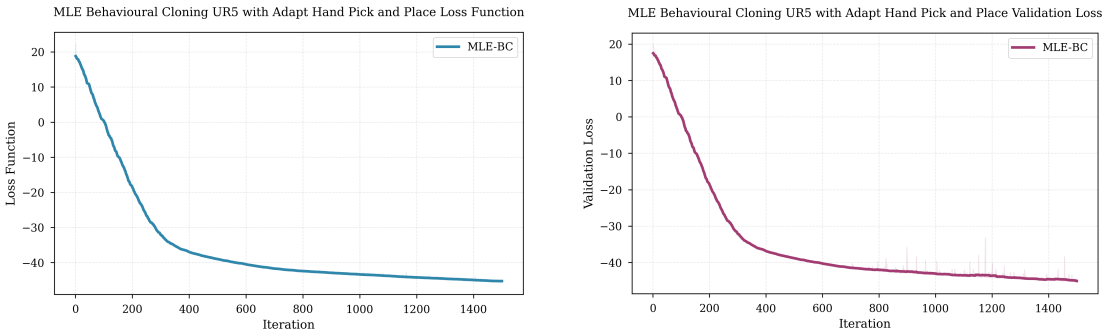


Figure 4.9: MLE Behavioral Cloning training on UR5 with Adapt Hand Pick and Place task over 1500 epochs.

The pick and place task represents an increase in complexity compared to lifting the target cube. This task requires the coordination between the arm and the hand to grasp the cube, and also the release of the cube at the end of the task. This particular feature makes standard reward engineering for RL more complex than previous tasks, as a reward term for the minimisation of the distance between cube and hand would then become detrimental when releasing the cube is required. Maintaining the cube stable in the hand during transportation also proved to be a significant challenge, as without dedicated sensors it is hard for the agent to establish how to keep the cube from slipping.

Figure 4.9 shows training progression over 1500 epochs. The loss curves exhibit very similar characteristics to the lift task, with training and validation losses both decreasing monotonically and remaining closely aligned throughout training. Convergence occurs at around iteration 500, where we selected the best performing checkpoint from success rate and visual analysis of the deployed policy. The absence of visible train-validation gap in the loss plots mirrors the lift task behavior, demonstrating the incapability of this validation strategy to predict the best model.

Training completed in 508 seconds, with the optimal model achieved at approximately 169 seconds (iteration 500). The extended training duration compared to the lift task (101 seconds) reflects the increase in complexity of pick and place, mainly due to the longer horizon of the task, which averages at around 6.9 seconds

compared to the 4.2 seconds of the lift expert trajectories.

- Convergence time: 169 seconds / 500 iterations
- Final success rate: 70%
- Number of demonstrations: 100 trajectories
- Action space dimension: 21 (6 arm + 15 hand joints)

The success rate of 70% represents a 13.3 percentage point decrease from the lift task (83.3%), which further implies the increase in difficulty of the task for the reasons mentioned above.

Figure 4.10 demonstrates successful execution of the complete manipulation sequence. The top-left image shows the initial approach towards the target cube, the top-right image shows the hand grasping the cube successfully, bottom-left displays the transportation of the cube over the gap between the two boxes, and finally on the bottom-right the completed task where the hand deposited the cube and moves away from it is displayed. The policy exhibits coordinated behaviour across the task's trajectory. However, the 33.3% failure rate indicates that this is not consistently achieved across many randomised initial positions of the target cube, highlighting poor generalisation.

Figure 4.11 illustrates three example failure modes accounting for most of the 33.3% of task failures. The left image shows the arm avoiding the cube entirely in the initial phase of the task, resulting in empty-handed transportation to the goal position. The central image displays the result of grasp instability during transport, which results in the cube either falling through the fingers, or in some cases hitting the edge of the second box and falling. Finally, the right image shows the hand actually transporting the cube to its destination, but failing to release its grip. This is the previously mentioned compounding error problem, which likely caused the arm to reach an unknown, out-of-distribution state towards the end of the task. BC's inability to recover from errors and generalise well to randomised initial target positions demonstrates to be a bigger problem for longer horizon tasks like pick and place.

The following section investigates whether hybrid BC-PPO approaches can address these limitations by using online policy fine-tuning with PPO and possible better error recovery.

4.4 Hybrid Approach Results

This section presents results for hybrid approaches that combine imitation learning with reinforcement learning: Wasserstein Generative Adversarial Imitation Learning (Wd-GAIL) and our proposed BC-PPO algorithm.

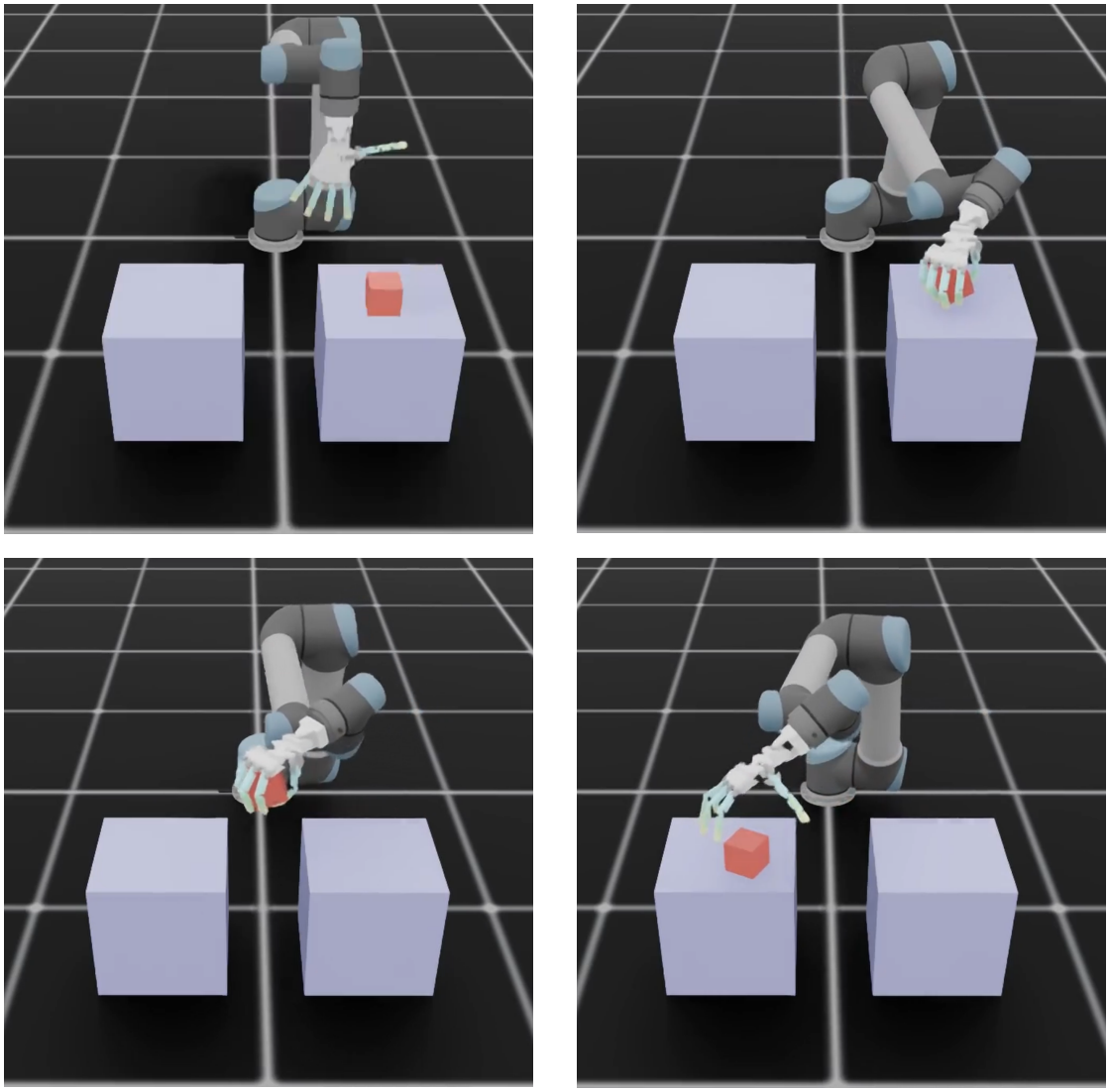


Figure 4.10: Sequential progression of pick and place task succesful execution.

4.4.1 Wasserstein GAIL Performance

Wasserstein Generative Adversarial Imitation Learning (Wd-GAIL) represents an alternative approach to combining demonstrations with reinforcement learning, using adversarial training to learn a reward function from the given optimal expert demonstrations rather than combining objectives. This section evaluates Wd-GAIL on a hand configuration task, requiring the movement of wrist and the hand's fingers.

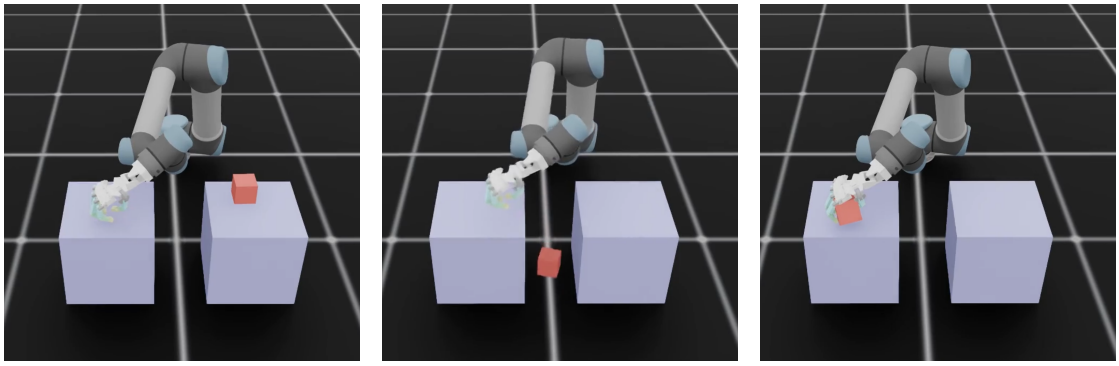


Figure 4.11: Characteristic failure modes in pick and place task.

Hand Configuration Task

Wd-GAIL was evaluated on a simplified hand configuration task where the agent must replicate target hand poses using the 15-DoF Adapt Hand. This task does not require arm motion and coordination, or interaction with objects.

The algorithm achieved 100% success rate with 1500 training iterations, requiring 736 seconds of total training time. Convergence occurred at approximately iteration 300, corresponding to 147.2 seconds. This represents faster convergence than BC-PPO presented in the next section (550 seconds for Franka lift), but this task is also significantly simpler.

- Convergence time: 147.2 seconds / 300 iterations
- Final success rate: 100%
- Number of demonstrations: 100 trajectories
- Training duration: 736 seconds / 1500 iterations
- Action space dimension: 15 (hand joints only, no arm control)

However, training exhibited instability which is characteristic of adversarial methods. The discriminator and generator accuracies failed to converge to the desired 50-50 equilibrium that is theoretically desired. Instead, they oscillated between mostly generator domination at first, and relative balance or discriminator domination towards the later stages of training. Despite this instability, this task is successfully completed

Limitations on Complex Manipulation Tasks

Attempts to apply Wd-GAIL to the full manipulation tasks (UR5 + Adapt Hand lift and pick-and-place) had training divergence and task failures. The policy

showed undesired behaviours when deployed and was unable to achieve correct grasping of objects.

GAIL's stability depends on the discriminator providing meaningful feedback to the generator, which becomes more difficult as the task complexity increases, due to an increase in the available task space and action spaces.

Given these results, subsequent experiments focus on BC-PPO as the primary hybrid approach, as we were unable to apply Wd-GAIL to obtain a reasonable comparison baseline.

4.4.2 BC-PPO Hybrid Approach

The BC-PPO hybrid algorithm combines behavioural cloning with proximal policy optimization by using a combined objective function, as described in section 3.3.6. This sections evaluates whether BC can be improved by having access to online environment interactions.

Franka Lift Task

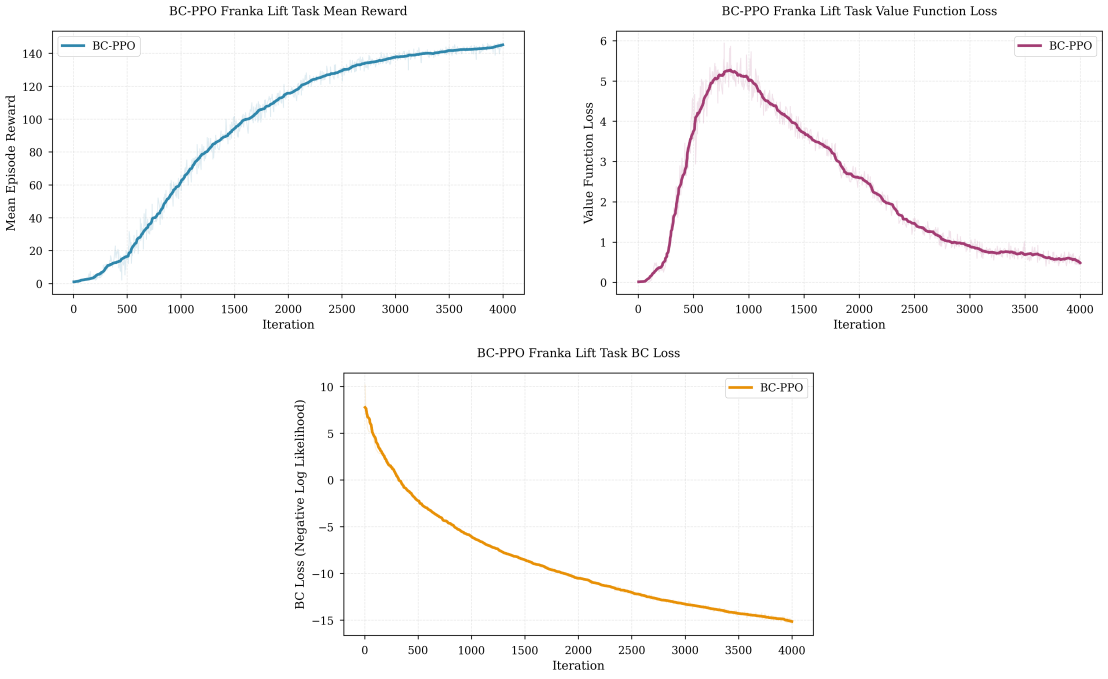


Figure 4.12: BC-PPO training on Franka Lift task over 4000 epochs.

Using the same 100 demonstrations employed in the pure BC experiments, BC-PPO training was conducted with a weighting coefficient $\lambda_{BC} = 0.1$ for the

BC loss component. This low weight is chosen because the negative log likelihood computation tends to have values one order of magnitude higher than the surrogate PPO objective throughout training, and therefore BC would dominate the optimisation of the objective.

Figure 4.12 shows training progression over 4000 iterations. The reason for longer training is that this algorithm was slower in reaching a plateau. Convergence to 100% success rate happened around 1400, but the reward signal is lower in magnitude compared to pure RL training. A visual analysis on the deployed policy displays that the reason 100% success rate is still achieved while the reward is lower compared to pure RL is that BC-PPO tends to initially converge with a slower execution of the task, thus obtaining the lift reward component of the function at a later time step.

The delayed convergence is further attributed to the fact that the combined objective must satisfy both BC and RL components, limiting the initial aggressive exploration typical of PPO. Reward then steadily increases to the value of approximately 150 without modifying the task success rate.

The top-left image of Figure 4.12 shows monotonically increasing mean reward for the task, up to values close to 150 after 4000 iterations. The top-right image shows that the value function loss has its typical shape, with a peak at around 800 iterations, followed by a decrease in value throughout training. At the bottom, the BC log likelihood loss starts from the relatively high value of 7.5, and ends at around -15, indicating good coverage of the expert dataset.

- Convergence time: 550 seconds / 1400 iterations
- Final success rate: 100%
- Number of demonstrations: 100 trajectories
- BC weighting: $\lambda_{BC} = 0.1$
- Training duration: 1570.34 seconds / 4000 iterations
- Action space dimension: 21 (6 arm + 15 hand joints)

BC-PPO successfully addresses pure BC's overfitting problem, as shown by the success rate remaining at 100% even after 4000 iterations, and the visual feedback upon deployment confirming the performance of the agent. However, there is no improvement upon pure PPO, as substantially longer training is required to reach convergence. This is likely caused by the optimisation of the double objective, and the lower sample efficiency, as the algorithm also needs to acquire expert state-action pairs within the RL update loop.

This negative result suggests that BC-PPO's value proposition depends on task complexity and the quality and quantity of provided demonstrations. For the next

tasks we analyse, which make use of the Adapt Hand, a solution with pure RL would be much more complicated to achieve and BC-PPO is instead able to solve the tasks with minimal reward engineering required.

UR5 + Adapt Hand Lift Task

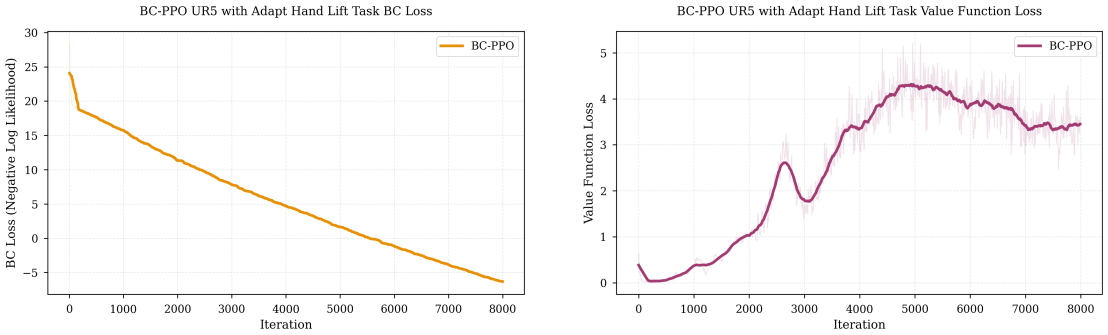


Figure 4.13: BC-PPO training on UR5 with Adapt Hand Lift task over 8000 epochs

For the UR5 + Adapt Hand lift task we use a deliberately sparse reward function to evaluate the algorithm's capability to solve the task by using expert demonstrations for guidance, and Rl for fine-tuning. This task provides reward feedback only upon successful task completion. This choice is based on the motivation for hybrid approaches to reduce the reward engineering effort, while maintaining competitive deployment performance. This should theoretically enable training with PPO on tasks where reward design would be prohibitively difficult.

Given the sparse reward structure, mean episode reward curves provide limited insight into learning progression, as rewards are essentially a binary signal for each of the 4096 parallel environments (success or failure), rather than less sparse rewards based for example on object-target distance like in Franka lift. Consequently, Figure **??** displays value function loss on the left and BC loss on the right. The value loss plot shows different behaviour from the previous tasks and struggles to converge towards zero, but nonetheless remains stable at relatively low values after an initial peak. The BC loss plot instead shows monotonically decreasing magnitude and converge to very negative values due to its log-likelihood nature. This indicates that the expert dataset correctly guides the policy towards completion. Training was conducted over 8000 iterations to try and reach a plateau, although this is not yet reached in the plots. The converging success rate is obtained at around 4000 iterations after approximately 1449 seconds of training, and remained stable throughout. The best model's success rate is 83.3%, which is a competitive result with the BC baseline, and it's achieved after the mentioned 4000 iterations.

This is a substantial increase in training time compared to the baseline, and it's due to the online interaction with the Isaac Sim simulator, and it highlights the challenge in learning from sparse rewards. The problem of overfitting does not manifest even after 8000 iterations, but nonetheless the performance is slightly worse than the baseline, showing the same failure modes (grasping target missed, divergent arm movement). Trajectories in under-represented regions of the task space are still problematic for the agent, as they constitute the main cause of failure.

- Convergence time: 1449 seconds / 4000 iterations
- Final success rate: 83.3%
- Number of demonstrations: 100 trajectories
- BC weighting: $\lambda_{BC} = 0.1$
- Training duration: 2898 seconds / 8000 iterations
- Action space dimension: 21 (6 arm + 15 hand joints)
- Reward structure: Sparse (success-only feedback)

BC-PPO achieves a success rate which is a 3.3 percentage point decrease over pure BC's 86.7% performance on the same task, and significantly, this performance remains stable even after 8000 iterations of training, demonstrating the potential to overcome the overfitting problem that characterises pure BC. RL's exploratory behaviour seems to prevent strict memorisation of the expert demonstration dataset.

The performance gap between BC-PPO (83.3%) and the pure BC baseline (93.3% on the Franka lift with dense rewards) can be explained by considering the simplified environment of the Franka Lift task, characterised by a lower-dimension action space, as well as the difference in density and reward complexity employed for the task.

The addition of PPO provides value regarding the preservation of performance throughout training, but further work is needed to overcome the baseline's success rate on complex tasks.

Pick-and-Place Task Results

The pick and place task represents the most challenging manipulation problem evaluated in this work, combining accurate grasping of the target object, transportation and release. Training was conducted with a reward function incentivising the hand to be close to the cube until the transportation phase reached the target area of the leftmost large box, switching then to a task success-based reward.

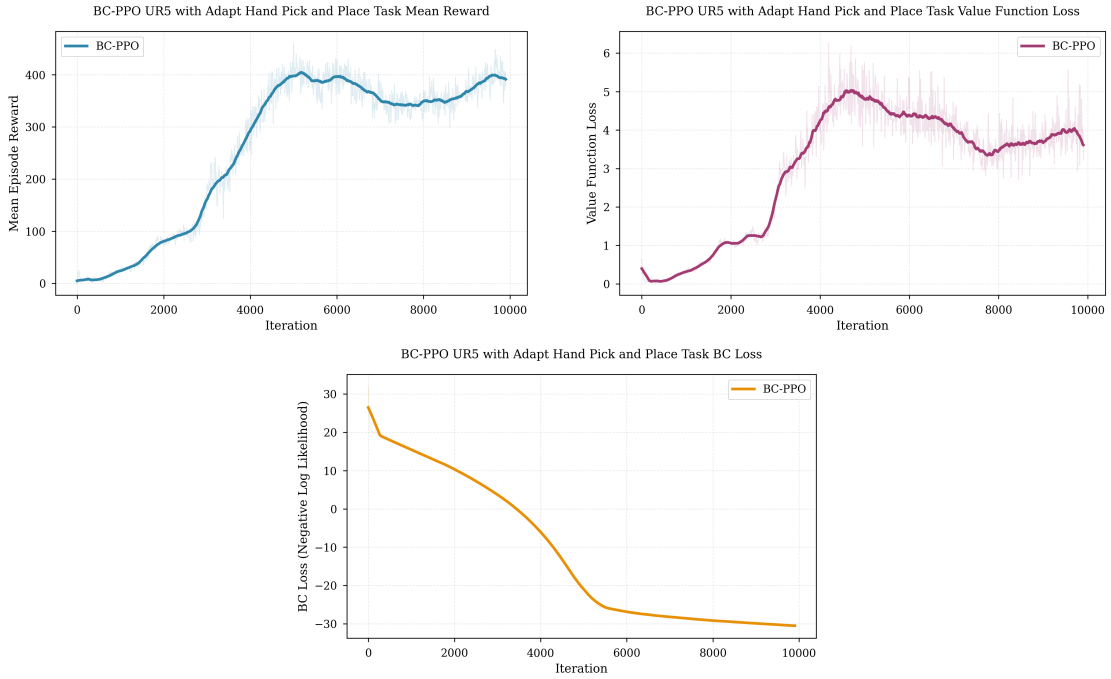


Figure 4.14: BC-PPO training on UR5 with Adapt Hand Pick and Place task over 10,000 epochs.

BC-PPO training was conducted over 10,000 iterations, requiring 3994 seconds total. Convergence to the final success rate occurred at iteration 4400, corresponding to 1597.6 seconds of training time.

Figure **??** shows the dynamics throughout training: the top-left plot shows an initially monotonically increasing mean reward until around iteration 4000, where the best model is achieved, with a subsequent small drop and partial recovery towards the end of training. The top-right plot shows a very similar behaviour of the value function loss as the Adapt Hand Lift task, where the value after an initial peak does not converge towards zero, but instead maintains a relatively stable low value. This indicates slight instability and lack of complete convergence. The plot at the bottom shows a monotonically decreasing BC loss function, which converges to very negative values (around -30) at the end of training, after a plateau around iteration 4400.

- Convergence time: 1597.6 seconds / 4400 iterations
- Final success rate: 63.3%
- Number of demonstrations: 100 trajectories
- BC weighting: $\lambda_{BC} = 0.1$

- Training duration: 3994 seconds / 10,000 iterations
- Action space dimension: 21 (6 arm + 15 hand joints)
- Reward structure: Target distance term and sparse success feedback

The final success rate of 63.3% represents a performance degradation compared to pure BC's 70% on the same task. This negative result reveals a fundamental limitation in the current BC-PPO implementation, which is the demonstration sampling strategy. While pure BC performs complete passes through the entire expert demonstration dataset each training epoch, essentially ensuring that the policy observes all expert state-action pairs in their entirety every epoch, BC-PPO samples random batches of size 512 from the dataset at the same rate as PPO updates. BC-PPO observes each demonstration trajectory far less frequently than pure BC in its current state.

This sampling disparity creates a tradeoff between overfitting and agent performance: as the policy never fully memorises the fixed expert dataset, undesired behaviours at later stages of training are entirely removed. However, more complex long-horizon tasks like pick and place, compared to Franka lift, pose a significant problem for this algorithm.

The performance degradation on pick and place highlights a critically needed improvement for BC-PPO: modifying the demonstration sampling strategy. A potential approach would implement increased expert trajectory exposure throughout training, combined seamlessly within PPO update cycles.

Table 4.1: Comparison of PPO, BC and BC-PPO performance across manipulation tasks. Convergence time measured to final success rate plateau.

Task	**Method**	**Success**	**Conv. Time**	**Total Time**
Franka Lift (8-DoF)	Pure PPO	**100%**	286s	537s
	Pure BC	93.3%	**110s**	**208s**
	BC-PPO	**100%**	550s	1570s
UR5+AH Lift (21-DoF)	Pure BC	**86.7%**	**101s**	**380s**
	BC-PPO	83.3%	1449s	2898s
UR5+AH Pick-Place (21-DoF)	Pure BC	**70.0%**	**169s**	**508s**
	BC-PPO	63.3%	1598s	3994s

Table 4.1 shows that BC-PPO's performance is very task-dependent, and it struggles to overcome the pure BC baseline. Nonetheless it achieves better results on the Franka Lift task, hinting that further algorithmic refinement could potentially

increase performance with the Adapt Hand also, and it completely removes the overfitting problem from all tasks.

Chapter 5

Conclusion and Discussion

This chapter interprets the experimental results presented in Chapter 4, focusing on the characteristics and limitations of reinforcement learning, behavioural cloning and hybrid approaches. Particular focus is placed on the analysis of the limitations of the BC-PPO algorithm and the possible improvements it can provide to robotic manipulation control problems.

5.1 Methods Comparison

5.1.1 Training Time and Efficiency

Experimental results show that pure behavioural cloning achieves the fastest training times (within 170 seconds for convergence on all tasks), requiring no environment interaction and therefore no need to load the computationally heavy Isaac Sim simulator platform. However, the algorithm's performance is highly dependent on the quality and quantity of expert demonstrations, and it suffers from poor generalisation and distribution shift during deployment because of this. We encountered particular difficulty initially when transitioning from synthetic expert trajectories, collected from a trained RL policy, to teleoperated trajectories. The RL policy is able to always reproduce the same movements when in the same state, yielding very consistent and accurate trajectories. On the other hand, it is complex to achieve this through teleoperation and small variations in the human response to different initial conditions are present in the final result, creating a more complex learning problem for the agent. More data would likely help with this issue, but teleoperated data cannot be collected as fast and as precisely as synthetic data: RL can generate hundreds of usable demonstrations in minutes, while the same process would take several hours for a human.

Pure PPO represents an extreme opposite to BC, as it requires no demonstrations

to learn from, but instead it interacts with NVIDIA's Physix system to collect state-action pairs, achieving a convergence time 3 to 5 times higher than BC on equivalent or simpler tasks.

BC-PPO attempts to overcome algorithmic limitations by combining objectives, but obtains the worst performance in terms of convergence time (even reaching 26 minutes for 63.3% success rate on the pick and place task), due to having to run essentially both the previous algorithms within the same loop. BC's overfitting problem is entirely missing in BC-PPO, as much less expert sampling is performed each iteration, and higher performance is achieved in the Franka Lift task. However, tasks involving the Adapt Hand perform worse and hint at the need for algorithmic improvements.

5.1.2 Task Complexity and Performance

Task complexity emerges as the critical factor determining which approach performs best. On the Franka Lift task, featuring 8-DoF control, a dense reward function which includes dynamic penalties with curriculum learning, and short horizon, all methods achieve satisfactory performance. Pure PPO reaches 100% success with the reward function implemented by NVIDIA Isaac Lab described in Section 3.2.2. BC achieves 93.3% with minimal training time. BC-PPO matches PPO's perfect performance but requires exactly 5 times longer training compared to maximum likelihood BC.

As task complexity increases with the switch to the UR5 and the use of the Adapt Hand (15-DoF end-effector compared to 1-DoF parallel-jaw gripper), the best performing algorithm is pure MLE behavioural cloning, achieving 86.7% success rate on the lift task compared to BC-PPO's 83.3%, and 70% success rate on the pick and place task compared to BC-PPO's 63.3%. Convergence time is also highly in favour of BC, with BC-PPO taking around 10 times longer to converge to stable performance. As for the simpler task, overfitting is again heavily present in pure behavioural cloning, but absent in BC-PPO, indicating that if performance improvement can be achieved in future work, this algorithm has a clear way to determine the best model, as performance is either improved or maintained until the end of training, whereas we faced significant challenges at determining best performing BC models, often leading to manual comparison of success rates without an automated pipeline. Section 3.3.5 highlights our validation process and focuses on the challenges of implementing validation strategies when expert trajectory state distributions are very similar to each other.

The results demonstrate a pattern where pure RL excels on simple tasks with tractable reward engineering, pure BC solves medium-complexity tasks for which good task space coverage is provided by the expert, and our hybrid approach mainly eliminates overfitting while maintaining competitive performance with BC, rather

than surpassing it.

5.1.3 The Demonstration Sampling Problem

The performance degradation of BC-PPO on tasks using the 15-DoF Adapt Hand, and specifically Pick-and-Place reveals an implementation limitation with the demonstration sampling strategy. Pure BC performs complete passes through the expert dataset each epoch, effectively using every trajectory state-action pair for each network update. BC-PPO on the other hand samples random minibatches of 512 transitions at the PPO update rate, meaning each demonstration is observed far less frequently over the course of training, and lack of coverage can occur with higher probability.

With a dataset of approximately 70,000 transitions and batches of size 512, BC-PPO observes each demonstration transition roughly once every 136 sampling function calls. This is the cause of the following tradeoff: reduced demonstration exposure prevents overfitting entirely, as the algorithm was able to maintain performance even after 10,000 iterations, but it also prevents sufficient behavioural guidance to achieve good success rate, to the point that RL can only be used for fine-tuning and not as the main driving force of the learning process.

This suggests a clear direction for BC-PPO's improvement, which is in an implementation of variable expert sampling frequency within the RSL-RL PPO framework. This aims to find a balance between the amount of trajectories needed to achieve good task space coverage, while maintaining this amount constrained to prevent overfitting. PPO would remain integral for fine-tuning and additional success incentive.

5.1.4 Failure Mode Analysis and Generalisation

The failure modes observed in BC and BC-PPO are primarily represented by lack of task space coverage and compounding errors due to distribution shift in imitation learning. The most frequent case is failure to grasp the target cube either by undershooting the necessary wrist position, or due to divergent arm behaviour. Other problems arise with Pick and Place, where the cube occasionally falls during transportation, or does not get released by the hand once the destination is reached.

Despite online RL experience, BC-PPO exhibits similar failure modes as pure BC, suggesting that the sparse reward framework does not provide enough generalisation. Corrective behaviour is not shown by the agents as it is not present in the provided demonstrations, and the reward signal does not generally contain terms which would generate such action choices. Overall, PPO prevents overfitting and adds exploration, as generalisation was observed to be slightly better specifically in the Franka Lift task, achieving 100% success rate even before the convergence to an

optimal reward magnitude of 150, as shown in Figure 4.12 of Section 4.4.2.

BC-PPO exhibits similar failure modes despite online RL experience, suggesting that the sparse reward structure provides insufficient corrective signal when the policy ventures into out-of-distribution states. The PPO component successfully prevents overfitting to the training set, but it does not enable the policy to recover from errors or explore successful strategies in poorly-covered regions of state space.

We observed that longer horizon also is a strong cause of failure, as the failed release of the cube is primarily the effect of out-of-distribution states reached after the transportation phase, which is a mode that does not occur in shorter tasks.

5.1.5 Wd-GAIL's Instability

Wasserstein GAIL's success on the hand configuration task (100% success, 147 seconds training) demonstrates theoretically that the algorithm can work with a 15-DoF end-effector. On the other hand, training instability on more complex tasks and the discriminator's inability to provide meaningful feedback to the generator, leading to divergent arm and hand behaviour highlights the complexity in the deployment of such algorithms, and the careful hyperparameter tuning needed to reach the required balance typical of adversarial network frameworks.

5.2 Limitations

5.2.1 Demonstration Quantity and Quality

All experiments in this work used exactly 100 expert demonstrations. This number is chosen as a compromise between human data collection effort and potential learning performance. Synthetically collected data is easy to obtain in great quantity, but teleoperation requires the availability of the hardware and is substantially more time consuming to perform. Considering that many trajectories are discarded while collecting, even with perfect execution Pick and Place demonstrations (of 7.5 seconds average length) would require 12.5 minutes. Setup preparation and downtime between collection often quadruples the duration of the procedure.

Furthermore, the teleoperated demonstrations for UR5 + Adapt Hand tasks exhibit human variability in execution strategy and quality, compared to synthetic data collected from well-trained RL policies. While this variability potentially improves robustness by showing the policy different successful approaches to the task, it may also hinder learning, as BC-PPO samples considerably less than standard BC implementations. This is particularly reflected in Pick and Place, which has even less dense task space coverage due to its length.

Future work should vary demonstration quantity to characterise the robustness of the proposed algorithms to this variation.

5.2.2 Hyperparameter sensitivity and Scalability

The BC-PPO experiments used a fixed BC weighting coefficient $\lambda_{BC} = 0.1$ for all tasks, chosen to balance the difference in magnitude between the log likelihood behavioural cloning estimation with PPO's surrogate loss function. Reinforcement learning's loss component is consistently one order of magnitude lower than the BC counterpart on our experiments, motivating our choice for the weighting parameter. Nonetheless, λ_{BC} is task-dependent and should be carefully chosen by analysing the numerical values of each loss component. This represents a potential downside of this hybrid approach, which currently can't be deployed as a *black box* method avoiding any manual tuning, although our choice appears to be a safe starting point for our framework's deployment.

BC-PPO's learning performance is highly sensitive to changes in λ_{BC}, where a high value might accelerate convergence, as imitation learning would be the main guiding force for the policy, but a complete domination results in the addition of RL being meaningless and instead actively disruptive when it starts affecting updates. Likewise, very low values would introduce highly exploratory behaviour relying on RL rewards, which by choice are sparse and should not be used as the principal guiding component for the robot's behaviour, but instead only for fine-tuning and success rate incentive.

The most complex system evaluated (UR5 + Adapt Hand) has a 21-DoF action space: 6 arm joints and 15 hand joints. While this represents a significant increase over standard gripper-based systems (8 DoF), it proved to be very challenging and lead to a degradation in performance for our proposed algorithm. Since computational costs scale up with action dimensionality, future work could also be in the direction of dimensionality reduction and human hand synergies for complex manipulation [23].

Considering that the Franka lift is the best performing task, where BC-PPO surpasses pure BC, a reduction in the action space could favour the use of this algorithm which simplifies the model selection by eliminating overfitting behaviour.

5.3 Simulation to Reality Gap

All experiments presented in this thesis were conducted in NVIDIA Isaac Lab's physics simulation environment. While simulation provides considerable advantages for rapid prototyping, and parallel training is a very useful asset in learning-based control with the use of neural networks, the transition to real-world deployment introduces challenges that cannot be entirely represented in a simulation environment.

The UR5 manipulator represents a well-tested setup for simulation-to-reality transfer. Position control applied to industrial manipulators like the UR5 is widely

deployed in real-world applications and often presents very little complications. The arm's joint-level position control can be directly implemented on the physical hardware using Universal Robots' standard control interface, and a custom mapping between joint position commands from the learned policy would be the only external interaction needed with the arm's low-level controller.

The effects of friction and unmodelled dynamics are are typically manageable for position-controlled arms, particularly when operating at relatively moderate speeds. Therefore, the arm component of each of our learned policies should transfer to real applications with small amounts of fine-tuning required.

The Adapt Hand presents a more complex sim-to-real scenario. As discussed in Section 3.3.3, during the teleoperation data collection process we verified that the simulated Adapt hand is able to accurately replicate the real kinematics and range of movement of the hand, and that a custom linear mapping of joint position signals would be sufficient to replicate the exact movement of each simulated finger joint on the real hardware. Our tests were successful and the real hand shows good reproducibility of simulated movements at Isaac Sim's frequency of 100hz.

However, important material differences exist between the simulated and physical hands that may impact manipulation performance. The simulation model is essentially a rigid body approximation of the Adapt Hand, without contact compliance modelling. This feature is nevertheless one of the principal advantages of the Adapt Hand [16]. The physical hardware includes a thin silicon skin covering the palm and fingers, which was not modelled in simulation. This layer provides increased friction for grasping objects, potentially preventing slipping (which is one of the key failure modes in our Pick and Place task), and introduces mechanical compliance similar to the real human hand.

The silicon skin is both a model mismatch that could cause problems with the learned behaviour, and a potential improvements on interaction characteristics on the real hardware. Future work will have the aim to breach the sim-to-real gap and test whether hand compliance can be helpful for learning-based dexterous manipulation. The more substantial challenge in real-world deployment remains in the end the coordinated movement between arm and hand, and precise grasping timing.

5.4 Closing Remarks

This thesis shows an analysis of state-of-the-art reinforcement learning and imitation learning methods, and proposed hybrid learning approaches combining behavioral cloning with reinforcement learning for dexterous manipulation. The experimental results demonstrate that our hybrid method successfully eliminates the overfitting problem and maintains competitive performance on tasks of moderate complexity.

However, the findings also reveal that simply combining BC and RL objectives does not automatically produce superior results to the baseline.

The most significant contribution of this work lies in characterizing when hybrid methods provide value, and potential ways in which these methods could overcome current state-of-the-art performance in the future.

.1 Appendix A: Equivalence of MLE and MSE for Gaussian Policies

For a Gaussian policy $\pi_\theta(a|s) = \mathcal{N}(\mu_\theta(s), \sigma^2)$, the MLE objective can be expanded as follows:

$$\mathcal{L}^{BC}(s_e, a_e, \theta) = \log \pi_\theta(a_e|s_e) \tag{1}$$

$$= \log \left[\frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(a_e - \mu_\theta(s_e))^2}{2\sigma^2} \right) \right] \tag{2}$$

$$= -\log(\sigma) - \frac{1}{2}\log(2\pi) - \frac{(a_e - \mu_\theta(s_e))^2}{2\sigma^2} \tag{3}$$

The pairs $[s_e, a_e]$ are sampled from expert trajectories either individually (independent identically distributed assumption) or as small trajectory mini-batches.

When maximizing the log-likelihood (minimizing negative log-likelihood), the constant terms can be ignored, leaving:

$$\mathcal{L}^{MLE} \propto \frac{(a_e - \mu_\theta(s_e))^2}{2\sigma^2} \tag{4}$$

This shows that MLE is proportional to MSE scaled by $\frac{1}{2\sigma^2}$. For the multivariate case with diagonal covariance $\Sigma = \text{diag}(\sigma_i^2)$:

$$\mathcal{L}^{MLE} = \sum_{i=1}^{d} \left[\frac{(a_{e,i} - \mu_i(s_e))^2}{2\sigma_i^2} - \log(\sigma_i) \right] \tag{5}$$

When σ is fixed, MLE reduces exactly to MSE. When σ is learned, MLE automatically weights each dimension by its uncertainty using the standard deviation σ_i, providing better gradient properties and moving towards a more deterministic policy as the uncertainty in the action selection decreases.

.2 Appendix B: BC+PPO Algorithm

This appendix provides the complete mathematical formulation of the BC+PPO hybrid algorithm, detailing each component of the combined objective function.

1) Policy Distribution and Value Function

$$\pi_\theta(a|s) = \mathcal{N}(\mu_\theta(s), \sigma_\theta^2) \tag{6}$$

$$V_\phi(s) = \text{critic}_\phi(s) \tag{7}$$

The policy outputs a Gaussian distribution over actions, while the critic estimates state values for advantage computation.

2) Generalized Advantage Estimation (GAE)

$$\hat{A}_t = \sum_{i=0}^{\infty} (\gamma\lambda)^i \delta_{t+i} \tag{8}$$

$$\delta_t = r_t + \gamma V_\phi(s_{t+1}) - V_\phi(s_t) \tag{9}$$

GAE provides a bias-variance trade-off for advantage estimation, with λ controlling the trade-off between temporal difference residuals.

3) PPO Surrogate Objective with Clipping

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} = \exp\left(\log \pi_\theta(a_t|s_t) - \log \pi_{\theta_{\text{old}}}(a_t|s_t)\right) \tag{10}$$

$$\mathcal{L}^{\text{CLIP}}(\theta) = \mathbb{E}\left[\max\left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\varepsilon, 1+\varepsilon)\hat{A}_t\right)\right] \tag{11}$$

The importance sampling ratio $r_t(\theta)$ measures policy change, with clipping preventing large policy updates that could destabilize training.

4) Value Function Loss (with optional clipping)

$$\mathcal{L}^{\text{VF}}(\phi) = \mathbb{E}\left[\max\left((V_\phi(s) - V^{\text{target}})^2, (\text{clip}(V_\phi(s)) - V^{\text{target}})^2\right)\right] \tag{12}$$

$$V^{\text{target}} = \hat{A}_t + V_{\phi_{\text{old}}}(s) \tag{13}$$

Value function clipping prevents large updates to the critic, maintaining training stability similar to policy clipping.

5) Policy Entropy Regularization

$$H(\pi_\theta) = \mathbb{E}[-\log \pi_\theta(a|s)] = \sum_{i=1}^{k}\left[\frac{1}{2} + \frac{1}{2}\log(2\pi) + \log(\sigma_i)\right] \tag{14}$$

Entropy regularization encourages exploration by penalizing overly deterministic policies during training.

6) Behavioral Cloning Maximum Likelihood Objective

$$\mathcal{L}^{\text{BC}}(\theta) = -\mathbb{E}_{(s_e,a_e)\sim\mathcal{D}}[\log \pi_\theta(a_e|s_e)] \tag{15}$$

The BC component maximizes the likelihood of expert actions, providing supervised learning guidance from demonstrations.

7) Combined BC+PPO Objective

$$\mathcal{L}^{\text{BC+PPO}}(\theta,\phi) = \mathcal{L}^{\text{CLIP}}(\theta) + c_1\mathcal{L}^{\text{VF}}(\phi) + \lambda_{BC}\mathcal{L}^{\text{BC}}(\theta) - c_2 H(\pi_\theta) \tag{16}$$

The unified objective balances policy improvement from environment interaction (PPO terms) with imitation of expert behavior (BC term), weighted by hyperparameters c_1, c_2, and λ_{BC}.

Bibliography

[1] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Second. Cambridge, MA: MIT Press, 2018 (cit. on pp. 3–5, 7).

[2] Pierre Sermanet, Kelvin Xu, and Sergey Levine. «Unsupervised Perceptual Rewards for Imitation Learning». In: *arXiv preprint arXiv:1612.06699* (2016) (cit. on p. 4).

[3] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. «Proximal policy optimization algorithms». In: *arXiv preprint arXiv:1707.06347* (2017) (cit. on pp. 4, 5, 16).

[4] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. «Trust region policy optimization». In: *International conference on machine learning*. PMLR. 2015, pp. 1889–1897 (cit. on p. 4).

[5] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. «High-dimensional continuous control using generalized advantage estimation». In: *arXiv preprint arXiv:1506.02438* (2015) (cit. on pp. 5, 17, 31, 32).

[6] Ahmed Hussein, Mohamed Medhat Gaber, Eyad Elyan, and Chrisina Jayne. «Imitation learning: A survey of learning methods». In: *ACM Computing Surveys* 50.2 (2017), pp. 1–35 (cit. on p. 5).

[7] Jonathan Ho and Stefano Ermon. «Generative Adversarial Imitation Learning». In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 29. 2016 (cit. on pp. 5–7, 28, 29).

[8] Dean A Pomerleau. «Efficient training of artificial neural networks for autonomous navigation». In: *Neural computation* 3.1 (1991), pp. 88–97 (cit. on p. 6).

[9] Yiren Lu et al. «Imitation is not enough: Robustifying imitation with reinforcement learning for challenging driving scenarios». In: *arXiv preprint arXiv:2212.11419* (2022) (cit. on pp. 6, 8, 27, 31).

[10] Huang Xiao, Michael Herman, Jörg Wagner, Sebastian Ziesche, Jalal Etesami, and Thai Hong Linh. «Wasserstein Adversarial Imitation Learning». In: *arXiv preprint* arXiv:1906.08113 (2019). URL: https://arxiv.org/abs/1906.08113 (cit. on pp. 7, 28, 30).

[11] Martin Arjovsky, Soumith Chintala, and Léon Bottou. *Wasserstein GAN*. 2017. arXiv: 1701.07875 [stat.ML]. URL: https://arxiv.org/abs/1701.07875 (cit. on pp. 7, 28).

[12] Ikechukwu Uchendu et al. «Jump-Start Reinforcement Learning». In: *Proceedings of the 40th International Conference on Machine Learning (ICML)*. 2023. URL: https://arxiv.org/abs/2204.02372 (cit. on p. 8).

[13] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. «Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor». In: *Proceedings of the 35th International Conference on Machine Learning (ICML)*. 2018, pp. 1856–1865. DOI: 10.5555/3327757.3327902. URL: https://arxiv.org/abs/1801.01290 (cit. on pp. 8, 28).

[14] Mayank Mittal et al. «Orbit: A Unified Simulation Framework for Interactive Robot Learning Environments». In: *IEEE Robotics and Automation Letters* 8.6 (2023), pp. 3740–3747. DOI: 10.1109/LRA.2023.3270034 (cit. on p. 9).

[15] NVIDIA Corporation. *Isaac Lab: A Unified Framework for Robot Learning*. Accessed: 2024. 2024. URL: https://isaac-sim.github.io/IsaacLab/main/index.html (cit. on p. 9).

[16] Cheng Pan, Kai Junge, and Josie Hughes. «Vision-Language-Action Model and Diffusion Policy Switching Enables Dexterous Control of an Anthropomorphic Hand». In: *arXiv preprint arXiv:2410.14022* (2024). URL: https://arxiv.org/abs/2410.14022 (cit. on pp. 9, 22, 61).

[17] Bruno Siciliano, Lorenzo Sciavicco, Luigi Villani, and Giuseppe Oriolo. *Robotics: Modelling, Planning and Control*. London: Springer Science & Business Media, 2009 (cit. on p. 11).

[18] Clemens Schwarke, Mayank Mittal, Nikita Rudin, David Hoeller, and Marco Hutter. «RSL-RL: A Learning Library for Robotics Research». In: *arXiv preprint arXiv:2509.10771* (2025) (cit. on pp. 16, 31, 36).

[19] Younghyo Park. *Teleoperation System using Apple Vision Pro*. Version 0.1.0. 2024. URL: https://github.com/Improbable-AI/VisionProTeleop (cit. on p. 19).

[20] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. «Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World». In: *arXiv preprint arXiv:1703.06907* (2017) (cit. on p. 20).

[21] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. «Generative Adversarial Networks». In: *Advances in Neural Information Processing Systems*. Vol. 27. Curran Associates, Inc., 2014, pp. 2672–2680 (cit. on p. 29).

[22] Matthew Hausknecht and Nolan Wagener. *Consistent Dropout for Policy Gradient Reinforcement Learning*. arXiv:2202.11818. 2022. URL: https://arxiv.org/abs/2202.11818 (cit. on p. 34).

[23] Marco Santello, Martha Flanders, and John F. Soechting. «Postural hand synergies for tool use». In: *Journal of Neuroscience* 18.23 (1998), pp. 10105–10115. DOI: 10.1523/JNEUROSCI.18-23-10105.1998 (cit. on p. 60).

List of Figures

3.1 Robotic arm bases used in experiments. 10
3.2 Cylindrical workspace for Task 2 shown from multiple perspectives. 15
3.3 AVP teleoperation system and Adapt Hand sim2real setup for recording 21
3.4 AVP teleoperation system and Adapt Hand sim2real diagram and signal flow 22
3.5 Behavioural cloning lift task environments: parallel-jaw gripper and Adapt Hand 24
3.6 Behavioural cloning pick and place task environment: beginning and end of task frames 25
3.7 Wasserstein Generative Adversarial Imitation Learning applied on a hand motion task 30
3.8 Block diagram of the BC+PPO algorithm. 32

4.1 PPO training metrics on Franka Lift task over 1500 iterations. . . . 37
4.2 Progression of Franka Lift task. 38
4.3 PPO training metrics on Franka Lift task over 1500 iterations. . . . 38
4.4 Progression of Franka Reach and Adapt Hand Pose task. 40
4.5 MLE Behavioural Cloning on Franka Lift task over 1500 iterations. 40
4.6 MLE Behavioral Cloning on UR5 + Adapt Hand Lift task over 1500 iterations. 42
4.7 Successful execution of UR5 + Adapt Hand lift task. 44
4.8 Example failure modes in BC deployment. 44
4.9 MLE Behavioral Cloning training on UR5 with Adapt Hand Pick and Place task over 1500 epochs. 45
4.10 Sequential progression of pick and place task succesful execution. . 47
4.11 Characteristic failure modes in pick and place task. 48
4.12 BC-PPO training on Franka Lift task over 4000 epochs. 49
4.13 BC-PPO training on UR5 with Adapt Hand Lift task over 8000 epochs 51
4.14 BC-PPO training on UR5 with Adapt Hand Pick and Place task over 10,000 epochs. 53

List of Tables

4.1 Comparison of PPO, BC and BC-PPO performance across manipulation tasks. Convergence time measured to final success rate plateau. 54