A Modular Framework for Generic Imitation Learning using Graph-based Spatio-Temporal Representation of Demonstrations : application to Robotic Learning

Yassine El Manyari

COLLEGE ED MATHEMATIQUES
DOCTORAL STIC
PAYS DE LA LOIRE

Nantes Université

THÈSE DE DOCTORAT

NANTES UNIVERSITÉ

ÉCOLE DOCTORALE N° 641
Mathématiques et Sciences et Technologies du numérique, de l'Information et de la Communication
Spécialité : *Informatique*

Par

Yassine EL MANYARI

A Modular Framework for Generic Imitation Learning using Graph-based Spatio-Temporal Representation of Demonstrations : Application to Robotic Learning

Thèse présentée et soutenue à Polytech Nantes, le 22 Juin 2023
Unité de recherche : umr6004 – LS2N

Rapporteurs avant soutenance :

Olivier BUFFET Chargé de recherche - Inria Nancy Grand-Est
Alain DUTECH Chargé de recherche - Inria Nancy Grand-Est

Composition du Jury :

Président :	David FILLIAT	Professeur ENSTA - ENSTA Paris
Examinateurs :	Mehdi KHAMASSI	Directeur de recherche - CNRS
	Sylvain LAMPRIER	Professeur des Universités - Université d'Angers
	Silvia ROSSI	Associate professor - University of Naples "Federico II"
	Olivier BUFFET	Chargé de recherche - Inria Nancy Grand-Est
	Alain DUTECH	Chargé de recherche - Inria Nancy Grand-Est
Dir. de thèse :	Patrick LE CALLET	Professeur des Universités - Nantes Université
Co-dir. de thèse :	Laurent DOLLE	Ingénieur Chercheur - CEA

"Robots may never replace the human touch, but they sure can learn from it."

–

Résumé

Cette thèse porte sur le développement et la mise en œuvre d'une approche méthodologique et générique pour l'apprentissage par démonstration, visant spécifiquement à enseigner aux robots comment résoudre des tâches de manipulation en observant un expert humain. Les démonstrations fournissent un aperçu et des indications sur la façon de résoudre une tâche donnée. L'apprentissage de représentation est utilisé pour extraire les caractéristiques pertinentes de la tâche à partir des démonstrations et, en utilisant l'apprentissage par renforcement, l'agent-imitateur peut explorer l'environnement et acquérir des connaissances par essais et erreurs, ce qui conduit à un comportement plus efficace.

Notre recherche a commencé par une étude visant à explorer l'impact d'entrées bruitées sur l'entraînement des algorithmes d'apprentissage par renforcement et la capacité des politiques résultantes à être transférées de la simulation à un environnement réel en utilisant un bras robotique industriel UR10e. Ces entrées bruitées se produisent pendant le processus de collecte des données sensorielles dont le robot a besoin pour prendre des décisions et interagir avec l'environnement.

Le reste de notre recherche était centré sur l'apprentissage par imitation, comprenant une revue bibliographique des approches d'apprentissage par démonstration, suivie de la mise en œuvre de la methode de l'état de l'art "Generative Adversarial Imitation Learning" (GAIL), une méthode de pointe, pour résoudre une tâche de manipulation en utilisant des démonstrations au lieu d'une fonction de récompense prédéfinie et spécifique à la tâche. Nos résultats ont mis en évidence les biais de récompense inhérents qui entravent l'apprentissage, comme noté dans des recherches antérieures, et ont proposé un ajustement pour résoudre ce problème et en améliorer la convergence.

Enfin, nous avons développé une approche nouvelle, modulaire et générique qui résout le problème de l'apprentissage par imitation en deux phases. La première phase consiste à apprendre un modèle de représentation qui capture les caractéristiques spatiales et temporelles des démonstrations observées, et la deuxième phase utilise un algorithme RL prêt à l'emploi avec une fonction de récompense générique prédéfinie pour apprendre la politique d'imitation. Le modèle de représentation utilise des graphes et des réseaux

de neurones "GNN" pour l'apprentissage de représentations spatiales et la modélisation séquentielle (Seq2Seq) pour l'apprentissage de caractéristiques temporelles. Les résultats montrent que notre méthode est plus performante que les méthodes de l'état de l'art dans la résolution d'une tâche de manipulation et qu'elle s'applique efficacement au monde réel. En outre, notre approche présente des capacités de généralisation prometteuses pour diverses tâches de manipulation, surpassant les méthodes génératives dans la plupart des cas.

ABSTRACT

The focus of this thesis is on the development and implementation of a methodological and generic framework for learning from demonstrations (LfD), specifically aimed at teaching robots to solve manipulation tasks by observing a human expert. Demonstrations provide insight and guidance on how to solve a given task. Representation Learning is used to extract the task-relevant features from demonstrations. And by employing Reinforcement Learning (RL), the imitator-agent can explore the environment and acquire knowledge through trial and error, leading to more effective behaviour.

Our research began by conducting a study to explore the impact of noisy inputs on the RL training and how well the resulting policies transfer from simulation to a real-world setup using a UR10e industrial robotic arm. These noisy inputs occur during the process of gathering the required sensory data for the robot to make decisions and interact with the environment.

The remainder of our research was centred on imitation learning, including a literature review of Learning from Demonstration approaches, followed by the implementation of Generative Adversarial Imitation Learning (GAIL), a state-of-the-art method, to solve our use case task using demonstrations instead of a predefined and task-specific reward function. Our findings highlighted the inherent reward biases that hinder learning, as noted in previous research, and proposed an adjustment to address this issue and improve convergence.

Finally, we developed a novel, modular, and generic framework that solves the imitation learning problem in two phases. The first phase involves learning a representation model that captures the spatial and temporal features of the observed demonstrations, and the second phase uses an off-the-shelf RL algorithm with a predefined task-agnostic reward function to learn the imitation policy. The representation model utilises graphs and Graph Neural Networks for learning spatial embeddings and Sequence-to-Sequence modelling for learning temporal features. The results show that our method outperforms state-of-the-art methods in solving a pushing task and effectively transfers to the real-world. Moreover, our approach exhibits promising generalisation abilities in various manipulation tasks, surpassing generative methods in most cases.

Acknowledgement

I am grateful to many people who have contributed to the completion of this thesis. First and foremost, I would like to express my deepest gratitude to my two thesis advisors Laurent DOLLE and Patrick LE CALLET, for their invaluable guidance, encouragement, and patience throughout this research journey. Without their continuing support, insightful comments, and constructive criticisms, this thesis would not have been possible. Their insightful feedback and expertise were invaluable in shaping my ideas and improving the quality of my work. I would also like to thank the members of my thesis follow-up committee, Mehdi KHAMASSI and Sebastien LE LOCH, for their valuable feedback and suggestions, which helped me to refine my research questions and methodology.

My research journey has been greatly enriched by the exchange of ideas with my colleagues from the "Robotics & AI" team at CEA Nantes and IPI team at Polytech Nantes. Their constructive feedback, insightful discussions, and helpful suggestions have played a crucial role in shaping my research and assisting me in overcoming various obstacles. I would also like to extend a special thanks to Mathieu Riand, with whom I have had the privilege of collaborating extensively on the same project. Furthermore, I am thankful to the IT team of CEA for their technical support and for providing me with necessary resources such as computing clusters, as well as helping me resolve any technical issues that arose during my research. Additionally, I am thankful to "Region Pays de la Loire" for funding my PhD thesis work without which my research would not have been possible.

Finally, I would like to extend my heartfelt thanks to my family and friends, who have been a constant source of love, motivation, and inspiration throughout this journey. Their unwavering support, understanding, and encouragement have helped me overcome the challenges and stay focused on my goals.

Table of Contents

List of Figures

List of Tables

Glossary

BC Behaviour Cloning

BCO Behaviour Cloning from Observation

DAgger Dataset Aggregation

DDPG Deep Deterministic Policy Gradient

DDPGfD Deep Deterministic Policy Gradient from Demonstration

DQN Deep Q-Network

DQNfD Deep Q-Network from Demonstration

DRL Deep Reinforcement Learning

GAIL Generative Adversarial Imitation Learning

GAILfO Generative Adversarial Imitation Learning from Observation

GANs Generative Adversarial Networks

GATs Graph Attention Networks

GCN Graph Convolution Network

GNN Graph Neural Network

IL Imitation Learning

ILO Imitation Learning from Observation

ILPO Imitating Latent Policies from Observation

IRL Inverse Reinforcement Learning

IfO Imitation from Observation

JSRL Jump-Start Reinforcement Learning

KL Kullback-Leibler

LSTM Long Short-Term Memory

LfD Learning from Demonstration

MAE Mean Absolute Error

MDP Markov Decision Process

ML Machine Learning

MLP Multi-Layer Perception

MoCap Motion Capture

MuJoCo Multi-Joint Dynamics with Contact

PPO Proximal Policy Optimisation

RL Reinforcement Learning

RLT Remaining Life Time

RMSE Root Mean Square Error

RNN Recurrent Neural Network

ReLu Rectified Linear Unit

SAC Soft Actor Critic

SDK Software Development Kit

Seq2Seq Sequence to Sequence

TCN Time Contrastive network

TD3 Twin Delayed Deep Deterministic Policy Gradient

TRPO Trust Region Policy Optimisation

INTRODUCTION

Contents

1.1 Motivation

Robotics has become an essential part of modern industry, with a wide range of applications in various sectors such as healthcare, agriculture, and manufacturing. One of the most significant advantages of employing robots is the automation of repetitive, monotonous, and unsafe tasks that workers are often required to perform. This not only increases efficiency, but also minimises labour costs. Additionally, robots are more reliable and accurate than humans, as they do not experience distractions or fatigue.

The supply of affordable and efficient robots is driving the expansion of the worldwide robotics market. The market demand for the advancement of robotics has grown exponentially in recent years, with a projected market size of $189.36 billion by 2027 [1]. As robotics becomes more prevalent across various industries, the need to enhance robotics controllers' sophistication and flexibility is imperative to ensure they meet the diverse needs of these applications.

The hardware side of robots has been developed to reach sophisticated capabilities enabling them to perform complex tasks with high precision. Additionally, the recent breakthroughs in computer vision have been instrumental in empowering robots with perception systems that enable autonomous navigation and interaction with their surroundings. Despite these advances, flexibility in programming robots is still a challenge. Robots can perform highly precise and repeatable tasks, but they are limited in their

1. https://www.alliedmarketresearch.com/robotics-technology-market

ability to adapt to new situations or tasks that require human-like dexterity. As a result, the development of more flexible and adaptable robotics controllers remains crucial.

Machine Learning (ML) has brought about significant advances in building intelligent systems. By leveraging ML for robotics, it is now possible to create intelligent controllers that can adapt to various tasks and environments, reducing the manual effort required to program robots. This presents a tremendous opportunity for businesses and organisations, particularly those that are low to medium in size, to automate their processes and increase efficiency while reducing programming costs.

Traditionally, programming a robot requires a programmer to write explicit instructions for the robot to follow. This involves manually specifying every movement, action, and decision that the robot should make. However, with ML, it is possible to teach a robot how to perform tasks on its own, without explicit instructions from a programmer. This is achieved by using ML algorithms to train robots to process and extract patterns in their surroundings to take decisions towards solving a given task. Two primary approaches to this are Reinforcement Learning (RL) and Imitation Learning (IL).

Reinforcement Learning enables robots to acquire behavioural skills from their own experience, similar to how humans learn through trial and error. The robot receives a feedback from a pre-defined reward function, which evaluates its actions with respect to the task at hand. The robot's behaviour is adjusted towards maximising the rewards it receives. Imitation Learning, on the other hand, involves learning from an expert behaviour. The expert provides demonstrations of how the task is solved, then, a control policy is trained to solve the task in similar situations present in the demonstrations.

Reinforcement Learning and Imitation Learning each come with their own set of challenges. RL requires significant effort and time in designing a reward function that accurately represents the task, while IL struggles with generalising to new situations that were not present in the initial demonstrations dataset, and needs access to state-action pairs for good performance. Combining both approaches can help overcome these challenges and provide a more robust learning system for robots by leveraging their complementary strengths. In this thesis we study both approaches and combine them in a generic and modular framework that can be used for learning control policies using state-only demonstrations to guide the learning process while allowing the robot to interact and explore the environment by using Reinforcement Learning.

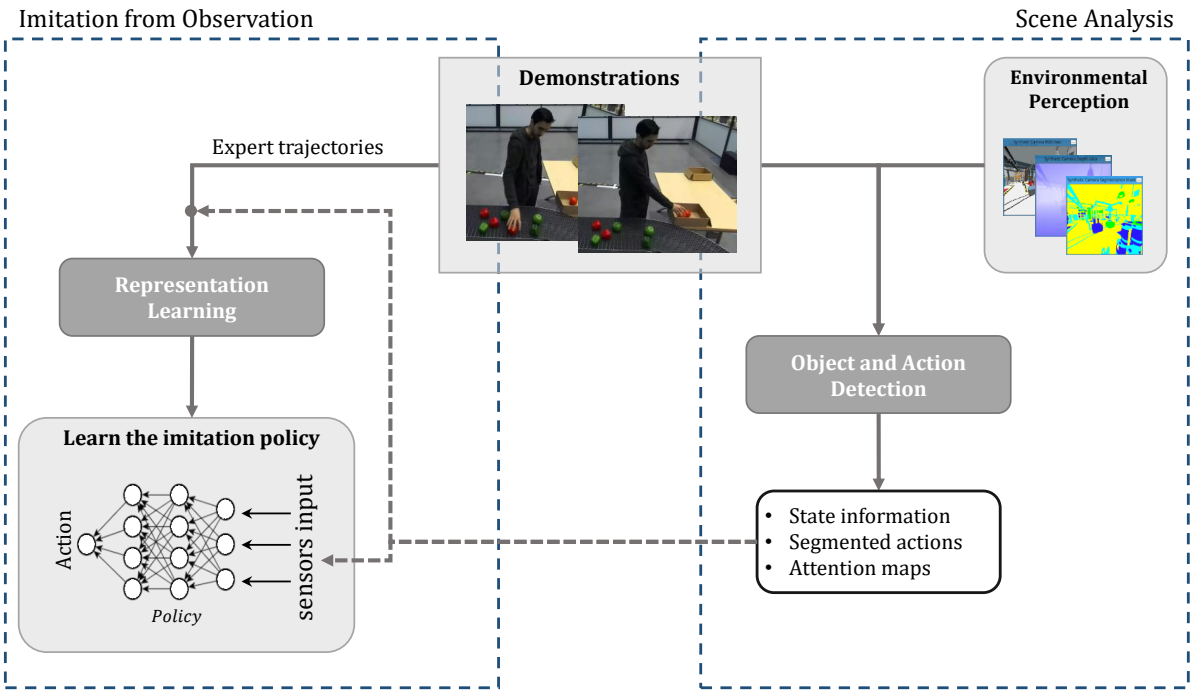


Figure 1.1 – Illustration of the complementary relationship between our thesis and the scene analysis thesis in developing a framework for visual imitation learning.

1.2 Context and Objectives

The CEA [2], in collaboration with the LS2N laboratory [3], has embarked on a research project to create and develop a general and flexible framework for autonomous robots to learn manipulation tasks through observing demonstrations provided by a human expert. The goal is to develop a solution that is applicable to a wide range of robotic tasks and can learn from visual demonstrations. This solution should be adaptable to fit into a robotic platform that includes an industrial robot equipped with a vision system to observe the expert's demonstrations and navigate the environment.

To achieve this goal, the research work has been divided into two PhD theses, as depicted in Fig. 1.1, with each thesis focusing on different aspects of analysing and learning from demonstrations. The first thesis is centred on utilising computer vision and deep learning to analyse the surroundings of a human expert demonstrating a given task and identify the relevant objects in the scene and the observed actions.

The second thesis, which is the focus of this research, aims to develop an approach for

2. https://www.cea.fr/english
3. https://www.ls2n.fr/?lang=en

learning control policies for autonomous agents from state-only demonstrations provided by an expert. A control policy is a decision-making process used to determine an agent's actions in response to the inputs coming from its environment. State-only demonstrations refer to the case where demonstrations are a sequence of states or observations, without explicit knowledge of the actions that led to those states/observations. Learning from Demonstrations is combined with Reinforcement learning to exploit the strengths of both for learning behavioural skills. Specifically, the thesis aims to create a modular and generic approach for this purpose.

Modularity is a design concept that breaks down an approach into interchangeable components, or modules, that can be assembled in various configurations to solve different tasks. This adaptability allows the approach to be tailored to a variety of scenarios without the need for extensive redevelopment. Generality, in this context, refers to the approach being flexible enough to be effective across a wide range of tasks.

1.3 Outline of the Dissertation

The chapters in this thesis are interconnected, as depicted in Fig. 1.2.

Chapter 2 is dedicated to presenting the fundamental concepts that are necessary for understanding Reinforcement Learning problem and algorithms. It also highlights the state-of-the-art methods which will be used in the subsequent chapters.

In Chapter 3, we explore the use of simulation-based training of Reinforcement Learning policies for real-world applications. The research investigates the influence of noisy inputs on the performance of different RL algorithms in solving a pushing task. Additionally, we deploy the obtained policies in a real-world setting with the UR10e robot, using a Motion Capture (MoCap) system and vision sensor to localise objects in the scene. The results suggest that transferring policies from simulation to real world scenarios can be effective with the introduction of optimal levels of input noise during training.

Chapter 4 provides a taxonomy of various Imitation Learning methods, with an emphasis on Imitation from Observation where actions are not observable in demonstrations. Among the approaches discussed, the Generative Adversarial Imitation Learning method was selected for its efficiency in using fewer demonstrations and leveraging Reinforcement Learning to improve generalisation.

Chapter 5 takes a deep dive into the workings of Generative Adversarial approaches, analysing the effects of the discriminator reward choice on training performance and pro-

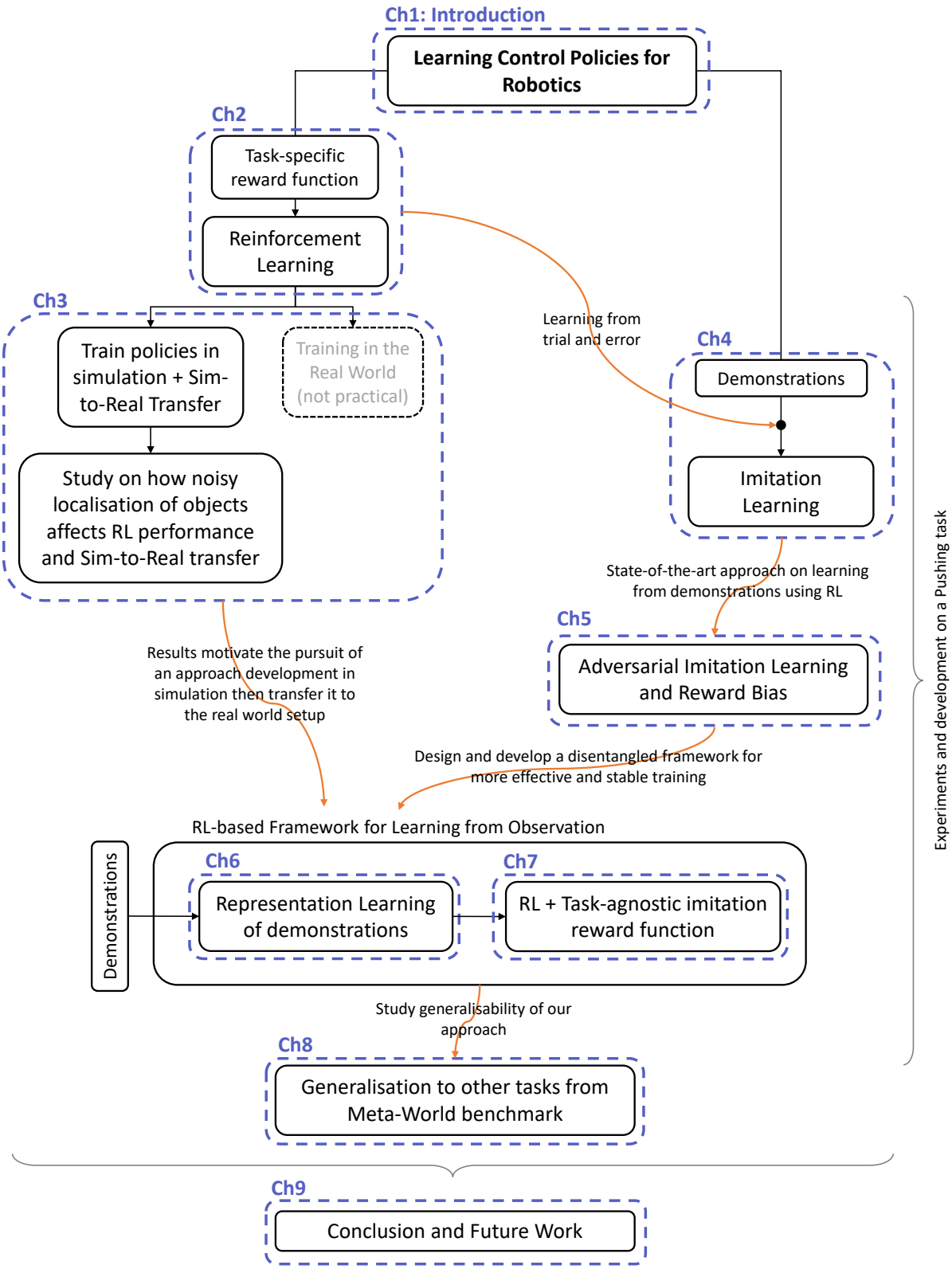


Figure 1.2 – Overview of the composition of the chapters forming the manuscript.

viding a remedy to the reward bias issue. In order to address the stability of training and improve performance, rather than learning directly from demonstrations as in generative methods, we propose an alternative solution by advocating for a two-stage process in which a representation model is first trained to capture spatial and temporal patterns in demonstrations, followed by leveraging this model to train the control policy for a behaviour with similar patterns.

In Chapter 6, we introduce a representation pipeline that extracts both spatial and temporal features from demonstrations. To accomplish this, we use graphs to model the scene and Graph Neural Networks to extract spatial features. Additionally, we utilise LSTM and Transformer networks to capture temporal features that relate to how the graphs evolve over time within demonstrations. The representation model is trained to predict the trajectory of the expert based on previous observations.

Chapter 7 introduces an approach that employs the predictive model trained on demonstrations to learn the imitation policy by trial and error using a task-agnostic reward function and an out-of-the-shelf Reinforcement Learning algorithm. Proof of concept experiments were carried out on a pushing task with results comparing favourably to the state-of-the-art methods, including Generative methods.

Chapter 8 evaluates the generalisation potential of our Imitation Learning from Observation framework on various robotic manipulation tasks, utilising a different robot and simulator than those used in the pushing task. The findings suggest that our approach has promising generalisation capabilities, indicating potential practical uses. Finally, in Chapter 9, we present a conclusion and recommendations for future research.

2

Reinforcement Learning

Contents

2.1 Introduction

This chapter presents the fundamentals of Reinforcement Learning and provides a comprehensive overview of the different classifications of different algorithms. Popular Reinforcement Learning algorithms utilised in challenging domains such as robotics, including PPO, DDPG, TD3 and SAC, are introduced and discussed. These algorithms will be incorporated in our study in the subsequent chapter.

2.2 Preliminaries

Reinforcement Learning (RL) is a class of machine learning that is geared towards sequential decision making problems [1] [2]. A behavioural policy is learnt by trial and error through the interaction with the environment. RL enables an agent to make a series of decisions in order to maximise a reward. At each time-step t, a decision-maker called the agent receives the state s_t of the environment and selects an action a_t based on its current policy. Upon execution of the chosen action, the environment transitions to a new state s_{t+1} and returns a reward r_{t+1}. The agent's policy is improved by utilising the knowledge of the state transitions, in the form of $(s_t, a_t, s_{t+1}, r_{t+1})$ tuple, and maximising the cumulative rewards received throughout each episode. This process is illustrated in Fig.2.1.

More formally, RL problems are defined using Markov Decision Process (MDP) formulation.

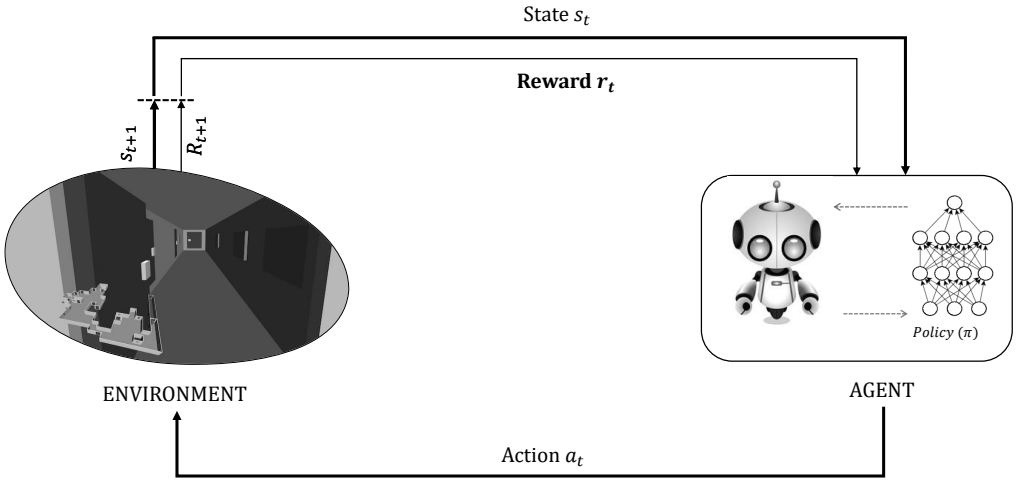


Figure 2.1 – The agent-environment interaction in Reinforcement Learning. The agent can be an animal, a game player, a control system etc., interacting with an environment and able to improve its behaviour

2.2.1 Markov Decision Process

A Markov Decision Process (MDP) consists of defining :

— The state $s \in S \subset \mathbb{R}^{d_s}$ as all the information that the agent has about the environment at a given time-step. S is the state space and d_s is its dimensionality.

— The action $a \in A \subset \mathbb{R}^{d_a}$ encodes how the agent can interact with the environment and it defines the possible moves that can be made. A is the action space and d_a is its dimensionality.
— The reward $r \in \mathbb{R}$ is a feedback evaluating the immediate action of the agent. Typically, the function generating the rewards is defined by an expert.
— The transition probability $\mathcal{P}(s'|s,a) = Pr(s_{t+1} = s'|s_t = s, a_t = a)$ which is the probability that the action a taken in the state s at time t will lead to the state s'.

The goal of the agent is to maximise the sum of rewards received from the environment in each episode of length T, namely the return $R = \sum_{k=0}^{T} r_k$.

To model a system using an MDP, the Markov property should be verified.

The Markov property refers to the consideration that the current state and the taken action are sufficient to determine the future state regardless of the past. For example, in the board game Diplomacy [1], the Markovian property is not valid, as the players have to consider not only the current placement of their pieces on the board, but also past alliances and conflicts with other players.

2.2.2 Terminology

— **State and observation** Typically, the state encompasses the information needed by the agent to undertake an action, while observation comprises the raw information available to the agent about the environment. The state is often obtained by pre-processing the observation to eliminate irrelevant features.
— **Action space** The action space is the ensemble of the possible actions that can be executed in the environment of the agent. The action space can be continuous like commanding the torque or the speed of a robot's joint, or discrete like choosing to move one step right or left in the Breakout game from Atari (see Fig.1 in [3]).
— **Policy** The policy governs the behaviour of the agent. In Deep Reinforcement Learning, the policy is typically a mapping from states to actions which is represented as a parametrised function. The policy can be trained either to output, for a given state, a deterministic action (deterministic policy) or a probability distribution over the possible actions (stochastic policy). RL algorithms can be either on-policy or off-policy. on-policy methods use the experiences generated by the

1. `https://en.wikipedia.org/wiki/Diplomacy_(game)`

current policy to optimise the objective function while off-policy methods exploit all the history of experiences to update the policy.

— **Trajectory** A trajectory $\tau = (s_0, a_0, s_1, a_1, s_2, a_2, ...)$, also called episode or roll-out, is the succession of states and the decisions taken by the policy. The initial state s_0 is randomly sampled from a distribution ρ_0.

— **Return** The return $R(\tau)$ is the cumulative of all rewards received by the agent over a trajectory τ.

$$R(\tau) = \sum_{t=0}^{T} \gamma^t r_k \quad \gamma \in]0, 1] \tag{2.1}$$

The discount factor γ is to guarantee the convergence of the sum if the horizon T is not finite (non-episodic tasks).

— **Value function** The value function of a policy π computes, for each state s, the expected cumulative reward starting from the current state s and following the current policy in future time-steps. It is defined as follows:

$$V_\pi(s) = \mathbb{E}_{\tau\sim\pi}(R(\tau) \mid s_0 = s) \tag{2.2}$$

— **Action-value function** The action-value function of a policy π computes, for each state s, the expected cumulative reward starting from the current state s and taking an action a (not necessarily according to the policy), and then operating on the current policy in future time-steps. It is defined as follows:

$$Q_\pi(s, a) = \mathbb{E}_{\tau\sim\pi}(R(\tau) \mid s_0 = s, a_0 = a) \tag{2.3}$$

The value function and the action-value function are related by the following equation:

$$V_\pi(s) = \mathbb{E}_{a\sim\pi}(Q_\pi(s, a)) \tag{2.4}$$

— **Advantage function** The advantage function provides an estimate of the relative advantage of choosing an action a in a given state s compared to other actions that the agent could take in the state s. It is defined as the difference between the action-value function and the value function:

$$A_\pi(s, a) = Q_\pi(s, a) - V_\pi(s) \tag{2.5}$$

The optimal value function and the optimal action-value function are obtained by finding the policy that maximises the expected reward:

$$V^*(s) = \max_{\pi} V_\pi(s) \tag{2.6}$$

$$Q^*(s,a) = \max_{\pi} Q_\pi(s,a) \tag{2.7}$$

$V^*(s)$ is the optimal value for a given state s and for any action a. $Q^*(s,a)$ is the optimal value for a given pair of state s and action a. Hence, for a given state s, computing the max of Q^* over all possible actions results in $V^*(s)$:

$$V^*(s) = \max_{a} Q^*(s,a) \tag{2.8}$$

2.2.3 Bellman Equations

The Bellman equations express the correlation between the value of the current state and that of potential future states.

The Bellman expectation equations are expressed as:

$$V_\pi(s) = \mathbb{E}_{a\sim\pi, s'\sim P}\left[r(s,a,s') + \gamma V_\pi(s')\right] \tag{2.9}$$

$$Q_\pi(s,a) = \mathbb{E}_{s'\sim P}\left[r(s,a,s') + \gamma \mathbb{E}_{a'\sim\pi}\left[Q_\pi(s',a')\right]\right] \tag{2.10}$$

And the optimality functions are expressed as:

$$V^*(s) = \max_{a\sim\pi} \mathbb{E}_{s'\sim P}\left[r(s,a,s') + \gamma V^*(s')\right] \tag{2.11}$$

$$Q^*(s,a) = \mathbb{E}_{s'\sim P}\left[r(s,a,s') + \gamma \max_{a'\sim\pi} Q_\pi(s',a')\right] \tag{2.12}$$

2.3 Model-based vs Model-free

Model-based Reinforcement Learning involves finding an approximate model to the environment's dynamics [4–6]. The environment model is then used as a simulator to predict the outcome of actions. The agent evaluates different actions and plans ahead to

choose the best sequence of actions. This provides the advantage of learning with less interactions with the environment than the model-free approach, however, constructing and maintaining an accurate enough environment's model demands more computing resources [4].

Model-free Reinforcement Learning, on the other hand, learns a policy directly by trial and error without prior knowledge about the underlying transition dynamics model of the environment [7]. The model-free approach has a simpler computation process and can converge to a high asymptotic performance than the model-based approach, yet it requires substantial amount of data to converge, which makes it sample inefficient [4].

2.4 Exploration vs Exploitation

Exploration in Reinforcement Learning entails collecting new (i.e. previously unseen) information about the environment in order to gain a broader knowledge of the possible outcomes of different actions and the associated rewards. Insufficient exploration could result in the agent being trapped in a suboptimal behaviour. Exploitation, in contrast, uses information already accumulated from previous experiences in the environment to select the best policy that produces the action with the highest expected cumulative reward.

Finding the right trade-off between exploration and exploitation is a fundamental challenge in reinforcement learning. At each time-step of interacting with the environment, the agent has to decide whether to explore new possibilities or exploit the current best policy that maximises known rewards. In order to find the optimal policy, the agent must reconcile these two opposing objectives by finding a balance between them. The following are some of the exploration strategies proposed in the literature:

— *Epsilon-greedy:* The RL algorithm chooses a random action with probability ϵ and selects the action with the highest expected reward with probability $1 - \epsilon$ [1]. The value of ϵ can be maintained fixed, or gradually decreased throughout the training.

— *Intrinsic motivation:* The agent is internally rewarded for exploring novel and unusual regions of the environment [8].

— *Noise-based Exploration:* A random noise is added to the process of actions selection. This helps the policy to escape local optima [9].

— *Maximise the entropy of the policy:* The agent is encouraged to explore actions with high uncertainty and therefore gain more information about the environment [10].

2.5 Value-based vs. Policy-search

2.5.1 Value-based

Value-based methods, also referred to as critic-only methods, seek to estimate and optimise the state value function or the state-action value function. Then, the policy is derived by following the actions with the highest estimated values.

One popular value-based method is Deep Q-Learning [3]. It uses a deep neural network to approximate the Q-function (Eq.2.3), which allows handling large finite environments and continuous state spaces.

The neural network takes as input the state of the environment and outputs the Q-values for all possible actions. The key properties of the Deep Q-Learning algorithm are:

— Previous experiences are stored as tuples of the form (s_t, a_t, r_t, s_{t+1}) in a replay buffer. The algorithm then samples batches from the buffer to update the Q neural network. This allows to mitigate the problem of correlation between consecutive transitions in trajectories.

— A separate network (called the target network) is used besides the main network that represents the Q value, to calculate the target $y = r(s, a, s') + \gamma \max_{a' \sim \pi} Q_\pi(s', a')$ in the Bellman equation Eq. 2.12. The target network has the same architecture as the main network and its weights are updated less frequently. This make the Q-values more stable.

2.5.2 Policy-search

Policy-search methods, also referred to as actor-only methods, seek to learn a policy directly instead of learning the value or the value-action function. The policy π_θ is parametrised with θ and trained to maximise the expected return $J(\theta)$ following π_θ:

$$J(\theta) = V_{\pi_\theta}(s_0) = \mathbb{E}_{\tau \sim \pi}(R(\tau)) \tag{2.13}$$

The policy is evaluated by performing rollouts following the current policy and calculating rewards. The parameters of the policy are then updated in the direction of increasing $J(\theta)$:

$$\theta_{k+1} = \theta_k + \alpha \nabla_\theta J \tag{2.14}$$

The way the gradient is formulated and evaluated gives rise to a wide variety of policy

search approaches [11], most of which are based on the vanilla REINFORCE algorithm [12].

Policy-search methods have better convergence and stability properties than value-based methods [13]. Because the policy is typically represented as a probability distribution over actions, it can adjust the parameters smoothly and thus result in more stability. Furthermore, policy-search methods are compatible with continuous action spaces without the need to be discretised as in value-based methods. The main drawback of policy-search methods is their sample-inefficiency compared to value-based methods.

2.5.3 Actor-Critic

Actor-Critic methods combine both value-based and policy-search approaches to find the optimal policy. The agent follows an actor (i.e. the policy) that decides which actions to take, while the value function serves as the critic that evaluates those actions. The goal of the actor-critic approach is to update both the critic and the actor so that the actions taken by the agent yield the maximum expected reward.

Integrating the value function from value-based approaches and policy optimisation from policy-search approaches leads to faster convergence, enhanced stability in learning, and better handling of high-dimensional and continuous action spaces compared to pure value-based or policy-search methods. This has made this class of methods perform exceptionally well in complex and high-dimensional domains such as video games [14], autonomous driving [15] and robotics [16]. The next section delves into the specifics of the most frequently used actor-critic algorithms in the literature, which have demonstrated remarkable performance. These algorithms are used in our study in the next chapter.

2.6 Algorithms

2.6.1 Proximal Policy Optimisation (PPO)

PPO [17] is an on-policy method that follows the actor-critic architecture. The critic is a value function $V_w(s_t)$ that outputs the mean expected reward in state s_t, used to compute the advantage function $A_t(s, a)$:

$$A_t(s, a) = r_t + \gamma r_{t+1} + ... + \gamma^{T-t+1} r_{T-1} + \gamma^{T-t} V_w(s_T) - V_w(s_t) \tag{2.15}$$

The actor learns a stochastic policy $\pi_\theta(a_t|s_t)$ that maps states with Gaussian distributions over actions. PPO is a simplified version of the trust-region policy optimisation (TRPO) algorithm [18]. The intuition behind trust-region based algorithms is that, policy updates must keep the output distribution close to the current distribution. PPO follows the same logic and optimises the policy to improve performance while ensuring that it does not deviate excessively from the preceding policy. This is achieved by optimising a surrogate objective that is defined as the ratio of the updated policy to the previous policy, multiplied by the advantage function.

$$L(\theta) = \mathbb{E}_{s,a\sim\pi_{\theta_{old}}}\left[\frac{\pi_\theta(a|s)}{\pi_{\theta_{old}}(a|s)}A^{\pi_{\theta old}}(s,a)\right] \tag{2.16}$$

In the literature the ratio between the updated and previous policy is denoted as: $r_t = \frac{\pi_\theta(a|s)}{\pi_{\theta_{old}}(a|s)}$. During policy updates, the direction of the policy gradient is determined by the advantage. If the advantage is positive, the policy gradient is reinforced in that direction, indicating that the action taken is superior to the average. Conversely, if the advantage is negative, the gradient is pushed in the opposite direction.

Unrestrained optimisation of the loss function $L(\theta)$ could result in substantial changes to the policy during each training step. To prevent this, two versions of PPO – PPO-Penalty and PPO-Clip – can be utilised to ensure that the updated policy remains close to the previous policy.

— **PPO-Penalty:** maximises the following unconstrained objective with penalising the Kullback-Leibler (KL) divergence between the old and updated policies:

$$L^{KL}(\theta) = \mathbb{E}_{s,a\sim\pi_{\theta_{old}}}\Big[r_t(\theta)A^{\pi_{\theta old}}(s,a) - \alpha_k D_{KL}[\pi_\theta||\pi_{\theta_{old}}]\Big] \tag{2.17}$$

Where α_k is an adaptive KL penalty coefficient that ensures π_θ remains close to $\pi_{\theta_{old}}$.

— **PPO-Clip:** utilises a "clipping" process to prevent the policy optimisation from drifting too far from the preceding policy:

$$L^{Clip}(\theta) = \mathbb{E}_{s,a\sim\pi_{\theta_{old}}}\Big[\min\Big(r_t(\theta)A^{\pi_{\theta old}}(s,a),\ \ clip\Big(r_t(\theta),1-\epsilon,1+\epsilon\Big)A^{\pi_{\theta old}}(s,a)\Big)\Big] \tag{2.18}$$

Where ϵ is a hyperparameter that determines the range of changes that the policy is allowed to make, and $clip()$ is a function that reduces the value of $r_t(\theta)$ to the range $[1-\epsilon, 1+\epsilon]$

2.6.2 Deep Deterministic Policy Gradient (DDPG)

DDPG [9] is an off-policy and actor-critic method. The critic is represented by the action-value function $Q(s, a)$ and is trained as in deep Q-network (DQN) [3]. The policy (actor) is trained with the guidance of the critic. As in DQN, the *experience replay* is used to break the temporal correlation of data points while training, and *target networks* are used to ensure stability in the training.

— **Critic update:** The critic is parametrised by a neural network with parameters θ and is updated based on the Bellman equation Eq. 2.12 by minimising the following loss function:

$$L(\theta) = \mathbb{E}\left[\Big(y_t - Q_\theta(s_t, a_t)\Big)^2\right] \tag{2.19}$$

With,

$$y_t = r(s_t, a_t) + \gamma Q_{\theta'}\Big(s_{t+1}, \pi_{\phi'}(s_{t+1})\Big)$$

Where $Q_{\theta'}$ and $\pi_{\phi'}$ refer to the target networks of the critic and the policy with weights θ' and ϕ' respectively.

— **Actor update:** The actor (i.e. policy) is updated in the direction that improves $Q_\theta(s, a)$ the most by solving the following equation:

$$J(\phi) = \max_\phi \mathbb{E}\Big[Q_\theta\Big(s, \pi_\phi(s)\Big)\Big] \tag{2.20}$$

This is achieved by taking a step of gradient ascent:

$$\begin{aligned} \nabla J(\phi) &= \mathbb{E}\Big[\nabla_a Q_\theta(s, \pi_\phi(s))\Big] \\ &= \mathbb{E}\left[\nabla_a Q_\theta(s, a)|_{s=s_t, a=\pi(s_t)} \nabla_\phi \pi_\phi(s)|_{s=s_t}\right] \end{aligned} \tag{2.21}$$

Since the policy is deterministic, noise is added to actions to enhance exploration. Gaussian noise and Ornstein-Uhlenbeck noise are commonly used in the literature [9]. Another option is to introduce noise into the parameters of the neural network [19].

2.6.3 Twin delayed DDPG (TD3)

While DDPG has shown great performance in many applications, it suffers sensitivity of the performance with regard to the hyperparameters [20]. A common failure mode is caused by the Q-values being overestimated, which results in high variance in the policy gradient estimates and hinders policy convergence. TD3 [20] improves upon DDPG by

incorporating the following key properties:

— Two separate Q-networks are used, Q_1 and Q_2, to reduce the overestimation of the Q-values. The minimum output of the two networks is used to compute the target in the Bellman loss functions:

$$y_t = r(s_t, a_t) + \gamma \min\Big(Q_{\theta'_1}\big(s_{t+1}, \pi_{\phi'}(s_{t+1})\big), Q_{\theta'_2}\big(s_{t+1}, \pi_{\phi'}(s_{t+1})\big)\Big) \tag{2.22}$$

And then the two Q-networks are updates using the same target y_t:

$$L(\theta_1) = \mathbb{E}\left[\big(y_t - Q_{\theta_1}(s_t, a_t)\big)^2\right] \tag{2.23}$$

$$L(\theta_2) = \mathbb{E}\left[\big(y_t - Q_{\theta_2}(s_t, a_t)\big)^2\right] \tag{2.24}$$

— The target policy is smoothened by adding a small random noise, ϵ, to the target action values, which acts as a regularizer and enhances exploration and stability. The target action is calculated as:

$$a'(s) = clip\big(\pi'(s') + \epsilon, a_{low}, a_{high}\big) \tag{2.25}$$

Where a_{low} and a_{high} define the valid range for action values.

— Similar to DDPG, the policy network, π, is updated to maximise Q (it can be either Q_1 or Q_2):

$$\max_{\phi} \mathbb{E}\Big[Q_\theta\big(s, \pi_\phi(s)\big)\Big] \tag{2.26}$$

Unlike DDPG, the policy updates occur less frequently than the updates to the Q-network, which enhances the training stability. The update for the policy network is performed every k update steps of the Q-network, where k is a hyperparameter specified as $k = 2$ in the original paper.

2.6.4 Soft Actor Critic (SAC)

SAC [10] is an actor-critic algorithm that incorporates entropy-regularisation in the training process. It is a method that uses a stochastic policy to generate actions, rather than a deterministic policy as in DDPG and TD3. The objective function to maximise is composed of the expected cumulative reward and the entropy of the policy. The entropy of

Algorithm	Architecture	Policy update	Supported action space	Exploration strategy
DQN [3]	Value-based	Off-Policy	Discrete	$\epsilon - greedy$
REINFORCE [12]	Policy-search	On-Policy	Discrete & Continuous	Train a stochastic policy
PPO [17]	Actor-Critic	On-Policy	Discrete & Continuous	Train a stochastic policy
DDPG [9]	Actor-Critic	Off-Policy	Continuous	Add noise to actions
TD3 [20]	Actor-Critic	Off-Policy	Continuous	Add noise to actions
SAC [21]	Actor-Critic	Off-Policy	Continuous	Maximise the entropy of the policy

Table 2.1 – Summary of the characteristics of RL algorithms mentioned in this chapter.

the policy reflects its level of randomness, and by promoting its maximisation, the agent is incentivised to explore a wide region of the environment. The policy objective function is given as:

$$J(\pi) = \max_{\pi} \mathbb{E}\left[\sum_{t=0}^{T}\Big(r_t + \alpha\mathcal{H}\Big(\pi_\phi(.|s_t)\Big)\Big)\right] \tag{2.27}$$

Where α is a hyperparameter that controls the importance of the entropy of the policy with regard to the reward and ϕ are the policy's parameters. The action-value function is modified as well to include the entropy term as follows:

$$Q_\pi(s,a) = \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{T}\gamma^t r_t + \alpha\sum_{t=1}^{T}\gamma^t\mathcal{H}\Big(\pi_\phi(.|s_t)\Big) \mid s_0 = s, a_0 = a\right] \tag{2.28}$$

SAC learns two Q-functions Q_{θ_1} and Q_{θ_2} similarly to TD3 along with the policy π_ϕ.

2.7 Summary

Reinforcement Learning was the focus of this chapter, which presents its basic concepts, classifications and the state-of-the-art algorithms. RL trains agents by trial and error through taking actions, observing the outcome and adjusting their behaviours. The chapter provided an overview of the three major types of RL methods, namely value-based, policy-search, and actor-critic. The chapter also introduced state of the art RL methods, summarised in Table 2.1, including PPO, TD3, and SAC algorithms that will be tested for their robustness to noisy inputs and their effectiveness in transferring from simulation to real-world environments.

3

SIM-TO-REAL TRANSFER

Contents

3.1 Introduction

Recent advancements in deep reinforcement learning (DRL) have yielded promising results in robotics, enabling the creation of high-performance controllers for complex tasks. Deep RL has been deployed to solve various tasks including manipulation tasks [22–26], locomotion [9] [27], navigation [28–31] and autonomous driving [15] [32, 33]. Yet, it is still challenging to train policies for deployment on real robots. Real-world applications are fraught with challenges associated with various sources of uncertainty. It is therefore challenging to achieve the same level of performance as in simulation. One source of uncertainty is the noise coming from sensors. In robotics, manipulation tasks in particular require knowledge of objects location. Camera sensors are commonly used to detect,

recognise and determine the location of objects in the 3D workspace [34, 35]. Hence, the accuracy of such an estimation is tied to the processing pipeline from pixels to the 3D location.

A typical vision system consists of two steps: the acquisition of image frames and performing analysis and recognition. The first step is characterised by numerous noise sources and are extensively investigated in the literature [36–38]. In this chapter, we are particularly interested in the noise arising from the transformation from image coordinates to 3D coordinates.

Mapping the 3D world coordinates of an object to 2D image pixels space is a common issue in computer vision [39]. When the object is located in the field of view of the camera sensor, it appears in the image frame as a block of pixels. The correspondence between the two spaces is obtained using the camera projection matrix. The projection matrix is formed from both intrinsic parameters which refer to the geometric and optical characteristics of the camera, and extrinsic parameters that determine the 3D location of the camera including the position and orientation with respect to the world coordinate system. The mapping from 3D world coordinates to the image plane is achieved, on one hand, by transforming the 3D coordinates into camera coordinates using the extrinsic parameters, and on the other hand, the intrinsic parameters transform the camera coordinates into the 2D image space.

The uncertainty in localisation estimation stems from different sources of noise. First, there is the quantization induced by the resolution of the camera, and whether the target object is in or out of focus. For a given camera resolution, the representation of an object in the pixel space depends on the distance between the camera and the object [39]. The more distant the object is from the camera lens, the fewer pixels are assigned to it in the image plane, and the more likely it is that the pixels belonging to the object will be erroneously estimated, and thus information about the object will be lost. Defocusing, on the other hand, causes the object to span a wider range of pixels, propagating noise to neighbouring pixels. When the object is out of focus, the intensity of pixels blurs, so we lose information about the shape and edges of the object, which makes it quite difficult to correctly estimate its position. External factors such as scene lighting conditions and occlusion can also be a source of noise in object position estimation.

In this chapter, we conduct an experimental study examining how noisy object location estimation affects Reinforcement Learning training and we examine whether the policies trained with a noisy estimation of object locations help overcome the issue of sensors noise

when simulation policies are transferred to the real-world.

3.2 Sim-to-Real Gap

Simulation environments have been the most effective solution for gathering substantial amount of data needed for training deep reinforcement learning models. This yields a rich source of data and also saves real robots from being exposed to unsafe exploration policies. Yet, deploying policies from simulation to the real world has been challenging due to the mismatch between the two domains. Mismatches arise from sensing, actuation, and environment dynamics [40].

The most popular sim-to-real approach is domain randomisation [41]. It narrows the reality gap by exploiting enriched variations of simulation settings during training. Instead of building a simulation environment with parameters close to the realistic environment, the parameters of the simulation are randomised to cover the data distribution of the real world. This technique has been used for visual domain randomisation and dynamics randomisation [41–43]. The former is utilised in training vision-based models to deliver sufficient variability in simulation to generalise to real visual inputs when deployed to the real-world. The latter is used to learn controllers robust to dynamics uncertainty. It involves varying physical parameters such as robot link masses, damping of robot joints, surface friction coefficients, etc.

In this chapter, we are particularly interested in the gap between simulation and reality that is induced by the noise inherent to physical localisation systems that is generally not taken into account in most simulators. In simulation, object localisation is straightforward and accurate, whereas object localisation in the real world is affected by various sources of noise, making it difficult to estimate object positions as accurately as in simulation.

3.3 Problem Formulation

We formulate a pushing task as a sequential decision-making problem and solve it using RL. The objective is to figure out the optimal sequence of actions to push an object to a random target location. The decision process is modelled using Markov Decision Process (MDP). At each time-step t, the controller receives the state s_t from the environment and executes an action based on its current policy. After the action is executed, the environment performs a one-step transition, delivering the next state s_{t+1} along with a

reward r_t. In this work we focus on engineering the following signals:

- **State (s_t):** The state encloses all the information the robot needs from the environment at a given time-step.
- **Action (a_t):** An action encodes how the agent can interact with the environment and it defines possible moves that can be taken on the environment.
- **Reward (r_t):** The reward is a feedback evaluating the immediate action of the agent.

The RL algorithm learns a policy that maps states to actions. Algorithms can be either on-policy or off-policy. On-policy methods use the experiences generated by the current policy to optimise the objective function while off-policy methods exploit all the history of experiences to update the policy.

In this work, we use three state of the art RL algorithms, PPO [17], TD3 [20] and SAC [10] which were introduced in Chapter 2, Section 2.6. The three methods are used to solve the pushing task with noisy estimation of the location of the object to push. The following is a brief summary of the key properties of each algorithm.

- PPO, Proximal Policy Optimization [17], is an on-policy method. It trains stochastic policies by making the greatest step possible towards improving performance while satisfying a Kullback-Leibler (KL) divergence-based constraint on the closeness between current and updated policies.
- TD3, Twin Delayed Deep Deterministic Policy Gradient [20], is an off-policy and actor-critic method. It learns a deterministic policy, and noise is added to the actions for exploration. Two Q-functions are learned and the smaller of the two Q-values is used to train the policy.
- SAC, Soft Actor Critic [10], learns a stochastic policy in an off-policy fashion. It relies on entropy regularization, where the policy optimises simultaneously the expected cumulative reward and the entropy, a function that provides an indicator of the randomness of the policy.

We select the three aforementioned RL methods as they recently have grown increasingly popular for their high performance in solving a variety of robotic tasks in simulation and real-world [21]. Our choice for these three algorithms is also motivated by the fact that they differ in the way they perform exploration. PPO initialises a stochastic policy with high entropy and gradually becomes less random. TD3 injects noise into the action space. And SAC optimises the entropy of the policy in the objective function.

In this paper we seek to verify the following hypotheses:

— ($\mathcal{H}_1$): We expect the training of the aforementioned algorithms to be negatively affected by noisy inputs to an extent that depends on the magnitude of the noise.
— ($\mathcal{H}_2$): The success rate of the resulting policies is expected to be higher when noise is minimal compared to when it is substantial.
— ($\mathcal{H}_3$): Models trained with noise are more likely to perform well when deployed on the real-world setup.

3.4 Experimental Setup

We evaluate the performance of RL models trained using two different types of noise and at multiple noise scales, and we examine the success rate of the obtained policies after 5M time-steps of training. The architecture of the neural networks as well as the hyperparameters of the RL model are maintained fixed throughout all the experiments. We use the implementation and default hyperparameters of the Stable Baselines3 library [44].

The task we intend to solve requires the robot to push an object towards a target location. We simulate the noisy inputs by adding uniform or Gaussian noise to the inputs that are supposed to be estimated using a vision system. In our experiments, the noise is added to the 2D location of the centroid of the object.

3.4.1 Simulation Environment

We use PyBullet, a python module based on Physics Bullet SDK. Within our scene shown in Fig. 3.1, we include a target in red, an object to manipulate, the UR10e, a 6 degree of freedom robotic arm that would perform the task, and a board where the object and target rest.

The elements of the scene are defined by their physics parameters including mass, friction coefficients, etc. There is no inherent noise in the simulation (the physics simulation in PyBullet is deterministic). At each physics time-step, the environment state is recomputed and any collisions are recorded.

We want the robot to learn to push an object from a fixed initial position to a random target area in a minimum number of iterations, relying only on the centroid locations of objects as input. We conduct experiments by varying only the type and magnitude of noise added to the inputs that are supposed to be estimated using a vision system.

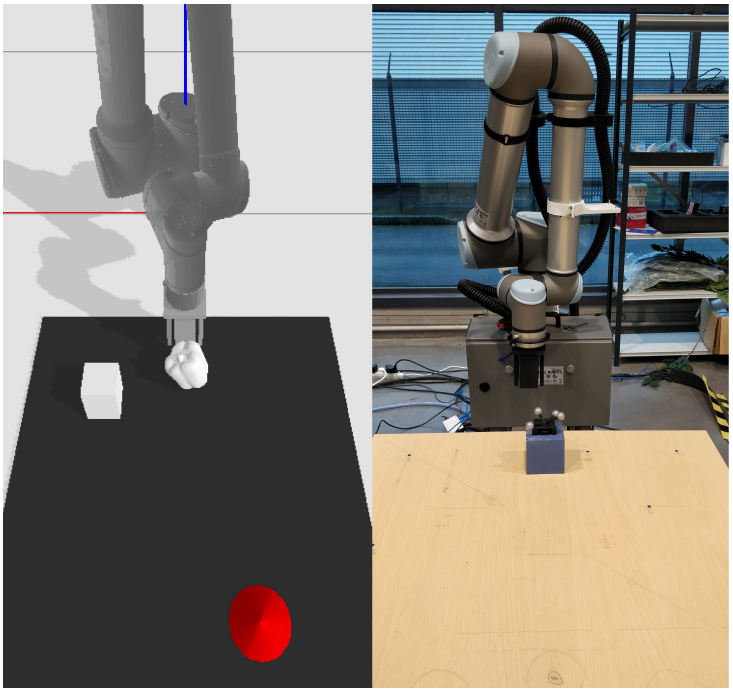


Figure 3.1 – The experimental setup on PyBullet simulator (left side) and using a real robot (right side). All models are trained exclusively in simulation using one of the displayed objects, a pepper object or a cube object.

3.4.2 Training

The learning process was performed during a fixed maximal number of time-steps. At each time-step, the policy generates an action, specifically an elementary displacement along the x and y axes. An inverse kinematics solver is used to generate the necessary movements in the joints space of the robot. After executing the action, the new state of the environment is calculated and a reward signal is given, assessing the optimality of the action performed by the robotic arm. The task to be learned is episodic. Therefore in addition to the state and reward, the environment also provides a binary signal $done_t = done(s_t)$ informing the robot whether the state s_t is a terminal state. Upon reaching a terminal state s_T at the time-step T, the interaction between the agent and the environment is concluded and the environment is reset to the initial state. We initialise each episode by placing the target area at random on the board in front of the object and the robotic arm. The object is either a cube object or a pepper object (see Fig 3.1), and it is replaced in front of the gripper of the robot at the beginning of each episode. The same object is used throughout the entire training phase.

3.4.3 RL Design Choices

The state consists of a 6 dimensional vector: $s = [p_g, p_o, p_t] \in \mathbb{R}^6$. p_g is the gripper's 2D-position, p_o is the object's centroid 2D-position, and p_t is the 2D position of the target area. All the coordinates are expressed in the absolute Cartesian coordinate system. This codification of the state is low dimensional and allows for faster training.

The action space consists of a 2 dimensional vector: $a = (dx, dy) \in [-5cm, 5cm]^2$. At each time-step the policy gets the state defined above as input and produces displacements in the 2D Cartesian plane with a maximal move of 5 centimetres in each direction.

The learning process is episodic. The robot has a maximal number T of time-steps to complete the task. The episode maximal length is $T = 150$ time-steps. A discrete time-step is terminal if one of the following conditions is verified:

— The object or gripper is out of the board.

— The maximal length of an episode is reached ($t = T$).

— The object has reached the target position.

The reward function is defined as follows: following a transition from s_t to s_{t+1}, we ascertain if the object has been moved during this transition. If so, we add a positive reward if the object has been pushed towards the target and a negative reward if it has been pushed far away from the target. In addition to this, we extend the reward function with penalties to prevent undesired behaviours. The function that generates rewards is defined in Eq. 3.1.

$$r_t = \begin{cases} +a \times b^{d(t)} & \text{if the object is pushed towards target} \\ -a \times b^{d(t-1)} & \text{if the object is pushed away from the target} \\ +1 \times RLT & \text{if the object has reached the target} \\ -0.1 \times RLT & \text{if the object is out of the board} \\ -0.1 \times RLT & \text{if the gripper is out of the board} \\ -0.1 & \text{if none of the above is verified} \end{cases} \tag{3.1}$$

Where:

— $d(t)$ is the distance between the object and the target at time-step t.

— a and b are parameters that define the magnitude of the recompense given to the robot if the object is moved during the transition. We selected $a = 10$ and $b = 0.03$ so the maximum value is 10, and it converges exponentially to 0 as the object

moves far away from the target.

— *RLT* stands for Remaining Life Time and corresponds to the number of time-steps the robot still has left until the episode is terminated. This term is used in the reward function to encourage the robot to reach the target in the shortest possible time and avoid stepping out of the board early in the episode.

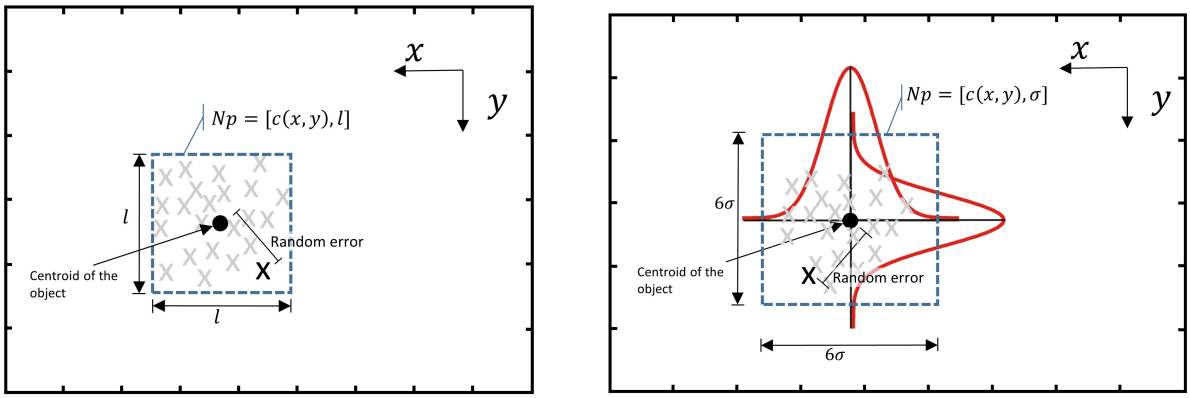
3.4.4 Noise model

We aim to introduce uncertainty to the object's position estimation. Position estimations are presumed to be within a symmetric range. The localisation belief is spread over a centred square around the ground-truth position. We represent objects by their centroid. Other properties such as orientation, size, and shape are not given to the training model. In our experiments, the noise is added to the location of the object while training the policy. Fig.3.2 depicts how the error in estimating the position in the Cartesian space is sampled. We randomly sample a scalar that lies within a square centred on the centroid of the object. Let $Np = [c(x, y), noise_{level}]$ be a square that limits the noise added to a given object with a centroid $c(x, y)$ that has as coordinates x and y. $noise_{level}$ is a parameter that determines the magnitude of noise. The uniform noise is defined as $Np = [c(x, y), l]$ with l is the length of the square. The Gaussian noise is defined as $Np = [c(x, y), \sigma]$ with σ is the standard deviation of the normal distribution, and the length of the Np square is 6σ. All sampled values that are out of the Np range are rounded to the limit values.

Depending on the pose of the camera with regard to the scene, a more realistic scenario would involve a noise magnitude that changes with regard to the distance between the camera lens and the object, as well as the focus of the camera. In this work, we assume a location of the camera that maintains the same uncertainty in sensor readings regardless of the working distance of the camera sensor. Such a setting would entail that the camera orientation is approximately orthogonal to the plane on which the object is located. We assume that the noise is temporally and spatially independent.

3.5 Results and Discussion

In section 3.5.1, we independently train three algorithms to solve the pushing task with different scales of noise added to the position of the object and evaluate how it affects the performance of RL training. All models were trained using four different randomisation



(a) Uniform noise

(b) Gaussian noise

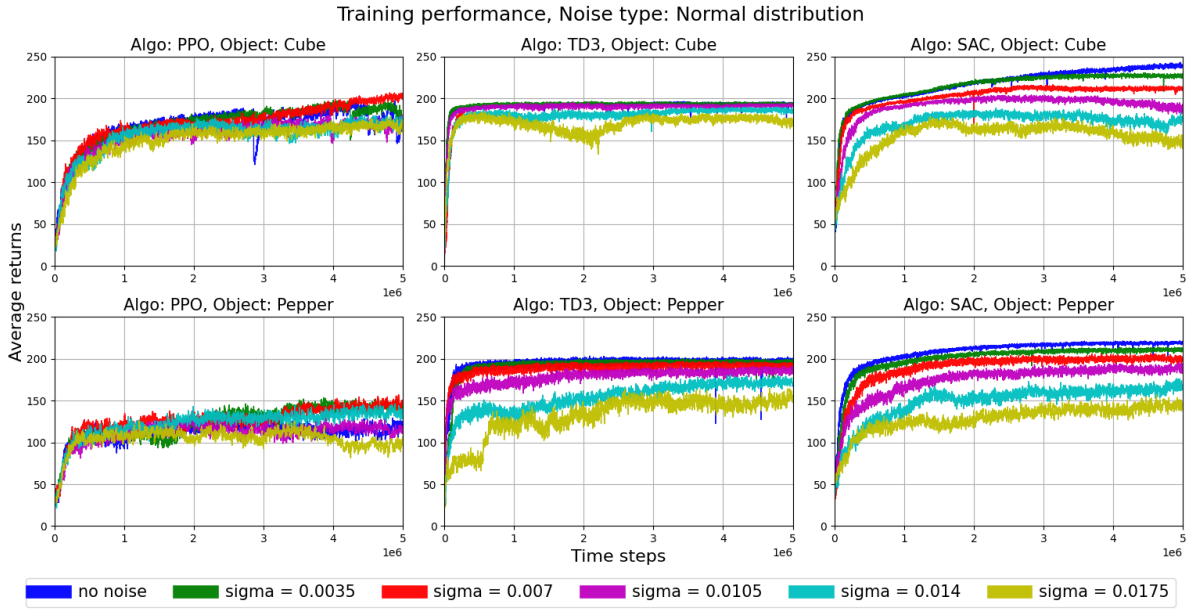
Figure 3.2 – Illustrative diagram showing how the estimation error of the object's centroid is sampled for each time-step. The uniform noise is defined with the length l of a square centred on the centroid of the object. The Gaussian noise is defined with a mean $\mu = 0$ and a standard deviation σ.

seeds. In section 3.5.2, we evaluate the performance of each individual obtained policy in the same environment and average results across seeds. In section 3.5.3, we deploy the policies with high performance on a real robot and evaluate their performance.

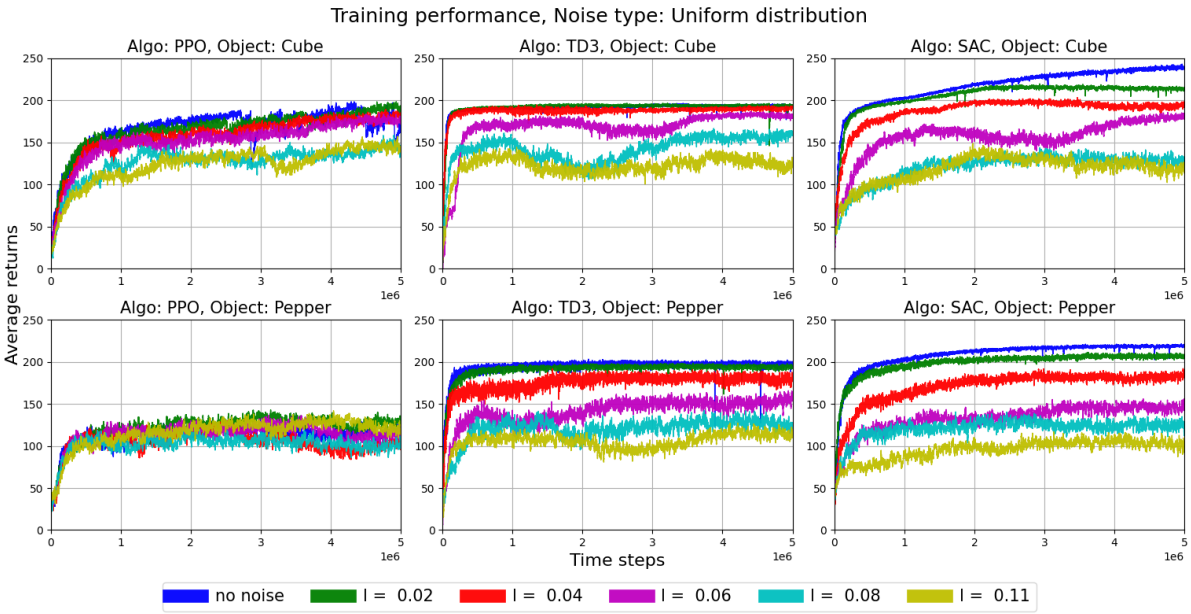
3.5.1 Training performance

In section 3.3, we plot the average cumulative rewards during the training process of every 100 consecutive trials. Our experiments were conducted using two different objects, a simple cube object, and a pepper object that has a relatively more complex shape. We train models for 5 Million time-steps to solve the pushing task using three different algorithms, only varying the scale of noise added to the object position. The difficulty of the task is characterised by two factors, the size of the target and the shape of the object. The size of the target was kept fixed during all experiments.

It is clear from Fig 3.3 that the noise added to the position estimation of the object impacts the stability and convergence of the training. TD3 and SAC converged to the highest average returns for training without noise. The higher the scale of noise the more disruption it implies in the learning process. However, they are not equally impacted by noise added to the location of the object. SAC gets strictly worse results as the scale of noise is augmented, while TD3 can handle low level of noise up to $\sigma = 0.014$ and $l = 0.04$ for the cube object and up to $\sigma = 0.0105$ and $l = 0.02$ for pepper object. PPO on the



(a) Training performance, Noise type: Normal distribution



(b) Training performance, Noise type: Uniform distribution

Figure 3.3 – Performance vs. training time-steps. The learning curves for PPO, TD3, and SAC algorithms with different scales of noise for two different object shapes. Cube and pepper object. We smooth episode rewards to their running average using the moving average metric with a window of 100 consecutive episodes. We plot the average of the curves for all four seeds except the models that failed to solve the task with an extreme behaviour. The unit of the noise scale is meters.

other hand performed comparably for all Gaussian noise scales and for a uniform low noise level.

Solving the task using a pepper object adds uncertainty to the learning process. The pepper object shape makes it hard to predict where the object will be directed if pushed in a particular direction. This depends on the contact between the gripper of the robot and the object, as well as the contact between the object and the board. The same action can result in different outcomes depending on the orientation of the object, which increases uncertainty in the environment dynamics and thus, leads to challenging training compared to the training with a cube object. The cube object, in contrast, has surface-to-surface contact with the board which makes the pushing task easier to solve.

The performance of TD3 on the pepper object is comparable to the performance on the cube object for models trained without noise to low magnitudes of noise. SAC obtains slightly worse performance on the pepper object compared to the cube object. For PPO, the shape of the object impacts considerably the training performance.

It is important to note that our inputs do not provide insight about the shape of the object to the model. Therefore, it must be learned implicitly through interaction with the object. From the aforementioned observations, we suspect that the exploration strategies of the three algorithms are not equally efficient, with TD3 being the most effective policy to handle noisy inputs. We believe this is due to the noise added to the actions for exploration, which appears to be a useful strategy for dealing with uncertainty arising from the shape of the object.

The hypothesis $\mathcal{H}_1$ is supported by the results of SAC and TD3 algorithms, except for TD3 models that were trained with a low level of noise, which showed an ability to maintain the same convergence performance as models that were trained without noise. The PPO results do not fulfil $\mathcal{H}_1$ hypothesis as they do not show a considerable difference between the performance of the models across various noise parameters.

3.5.2 Obtained policies evaluation in simulation

We evaluate the obtained policies on the same environment in simulation. We run 100 trials of each trained model and calculate the success rate of when the trained policy is able to push the object to the desired target. Fig 3.4 depicts the results for each trained model, averaged across the four seeds.

TD3 and SAC achieved the highest success rates with noiseless training. TD3 models can maintain similar or slightly better performance using a low scale of noise on both

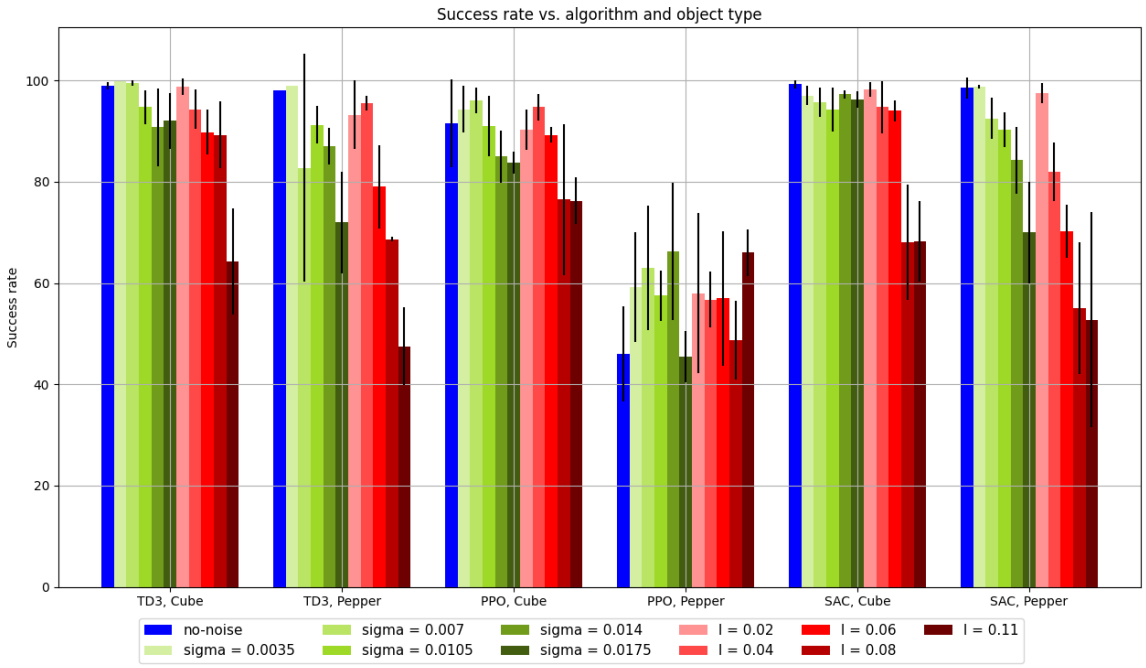


Figure 3.4 – Success rate and standard deviation across seeds for each method and object type across 100 trials. We roll out trained policies on the same environment with the same level of noise used in training.

the cube and pepper object. SAC achieved similar success rates for all noise parameters except high magnitudes of uniform noise (l=0.08 and 0.11). On the pepper object, similar performance is achieved using no noise or a very low scale of noise ($\sigma = 0.0035$ and $l = 0.02$) while adding noise beyond this scale deteriorated the performance considerably.

The success rates obtained by PPO did not vary too much as a function of the noise parameters for both objects. Yet, surprisingly, high noise levels increased the success values for the pepper object but did not result in sufficiently high success rates (below 70%).

The hypothesis $\mathcal{H}_2$ holds for TD3 and SAC for both objects but not for PPO. The latter achieved high success rates when trained on higher levels of noise for solving the task with the pepper object. This can be explained by the policy stochasticity not being sufficient to drive the pepper object and adding noise to the object location slightly helps the policy to achieve more efficient policies.

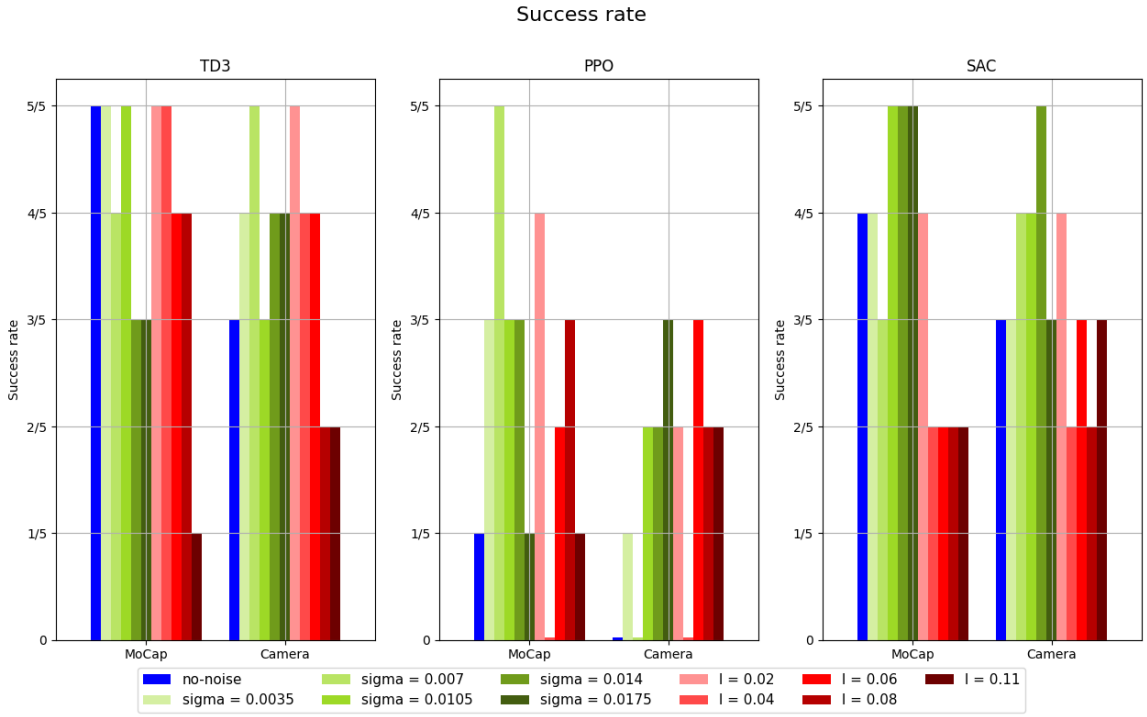


Figure 3.5 – Evaluation results on real robot using a cube object. We report the success rate of achieving 5 different targets. We ran the experiments using either a Motion Capture (MoCap) system, or an object detection by colour algorithm using an RGBD camera.

3.5.3 Sim to Real evaluation

We transfer policies trained in simulation to a real robot without further training or fine-tuning. Two systems for localising objects are employed. We utilise a Motion Capture System and object detection by color using an RGBD camera. The former solution allows for a high degree of accuracy calculation of positions while the latter inherits various sources of noise.

— **Motion Capture System (MoCap):** We use six motion infrared tracking cameras from ART [45]. The location of the object is calculated using a passive retro-reflective target that is installed on the top of the object (see Fig 3.1).

— **Detection by color:** We use the ZED2 camera from Stereolabs [46]. The object is detected by its color and the location is calculated using the depth image and the extrinsic matrices obtained from the extrinsic calibration. The detection and positioning of objects in the 3D space using a camera is challenging due to the presence of camera sensor noise and occlusion issues. The camera was placed in an orthogonal position to have a top-down view.

Fig 3.5 summarises the results. Using the MoCap system, both TD3 and SAC achieved a success rate of at least 4/5 for models trained without noise. While when deployed with a camera for object localisation the success rate didn't exceed 3/5. TD3 models trained with a low level of noise ($\sigma = 0.0035$ and $l = 0.02$) maintain the same success rate of 5/5 using either MoCap or Camera. SAC models trained with a high level of Gaussian noise ($\sigma = 0.0105$ to 0.0175) or uniform low scale of noise ($l = 0.02$) maintain success rates of at least 4/5 using both MoCap and camera. The results of PPO using the camera are not promising while using the MoCap reaches acceptable success values for some specific noise parameters.

The hypothesis $\mathcal{H}_3$ holds for the three algorithms and especially in the camera setting. The results look promising and suggest that for every algorithm there is a set of noise parameters that transfer better to the real-world.

Occlusion in the camera setting

Occlusion is a typical issue in computer vision that arises when a part of an object is hidden from view. When this occurs, it becomes difficult for computer vision systems to identify and detect objects in the scene. Occlusions may occur due to various causes including shadows and overlapping objects.

Our application requires the position of only one object, detected by its colour. To obtain the 2D position of the object, we use the camera to detect the orange-painted top surface of the object and calculate its center. The object detection is visualised by colouring the detected pixels in blue (see Fig.3.6). As seen in Fig.3.6, the object can sometimes become partially hidden by the robotic arm (e.g. Trajectory 2, time-step 46 in Fig.3.6). In such cases, the calculated position only accounts for a portion of the object and yields erroneous estimations of its position. In the event of full occlusion from the camera view (e.g. Trajectory 2, time-step 60 Fig.3.6), the previous position of the object serves as input to the policy.

To reduce the impact of occlusion, various methods can be employed, including utilising multiple viewpoints of the scene or tracking and predicting the objects motion. This can help to provide a better estimation of the position of objects when they are occluded.

Figure 3.6 – Illustration of samples from five different trajectories executed by the real robot. Each trajectory corresponds to solving the pushing task for a different target position. The camera is placed to capture a top-down view. The pixels depicting the object are recoloured in blue.

3.6 Summary

Reinforcement learning algorithms typically require a large amount of data to converge on optimal policies. This data is often generated in simulation and then the policies are transferred to the real world. The discrepancy between simulation and real-world environments in sensory data, nonetheless, necessitates that more consideration should be given to simulation training before transferring policies to the real world.

Our investigation in this chapter delved into how different algorithms respond to noisy estimates of an object in a pushing task. Specifically, we evaluated the performance of TD3, PPO, and SAC when subjected to two different noise models, Gaussian and uniform. Our analysis revealed that training the models with noise in simulation resulted in better real-world performance. However, we could not identify an ideal noise level that was applicable to all three algorithms.

Further research should be pursued to build a more thorough noise model by analysing every step of the computer vision pipeline used for objects localisation. Such a model would also take into account external factors, such as the distance between the camera lens and the object, the resolution of the camera, and whether the object is partially occluded. The findings of our study in this chapter have been published in [47].

4

Learning from Demonstrations

Contents

4.1 Introduction

Despite the remarkable success of reinforcement learning in various domains, its successful application is heavily dependent on the engineering of a task-specific reward function that indicates the desirable behaviour and guides the agent to the goal of the task. This can be a critical challenge, especially for complex tasks, as the reward function must be designed to accurately reflect the desired behaviour, taking into consideration the boundary cases and avoiding reward hacking [48].

To address this challenge, Learning from Demonstrations (LfD), also referred to as Imitation Learning (IL), provides an alternative approach to training autonomous agents. In LfD, the task to be solved or the skills to be acquired are illustrated to the agent by examples demonstrated by an expert. Unlike RL, where the learning process is driven by a reward signal, LfD relies on the expert demonstrations. This allows LfD to circumvent the explicit design of a task-specific reward function by learning from the expert's experience,

making it particularly useful in scenarios where the desired behaviour is complex to define by a reward function.

The application of LfD has yielded positive results in a variety of fields, such as robotics [49] [50], games [51], and autonomous driving [52], among others. This approach of learning control policies has the potential to greatly decrease the time and effort required to create and develop autonomous agents.

This chapter delves into the concept of Learning from Demonstrations, including its different variations and how it can be applied to control problems.

4.2 Learning from Demonstration approaches

Demonstrations are instances that showcase how a given task can be solved, and they are usually supplied by a human or an expert agent. These demonstrations often take the form of input-output pairs, where the input represents the state of the environment and the output indicates the action taken by the expert in that state. The goal of LfD is to train a control policy that can perform the demonstrated task with expert-like proficiency or greater [53–55]. Fig. 4.1 provides an overview of the various LfD approaches that we explore in the following sections of this chapter.

4.2.1 Behaviour Cloning

A commonly used approach for LfD is to treat it as a regular supervised learning problem. Assuming that a set of expert demonstrations composed of state-action pairs $\mathcal{D}^e = \{(s, a)\}$ is given, the aim of Behaviour Cloning (BC) is to learn a mapping π_θ from states to actions so that $\pi_\theta(s) = a$ [56]. Depending on whether the action space is discrete or continuous, a classifier or regression model is used to maximise the expected likelihood:

$$\mathcal{L} = \mathbb{E}_{(s,a)\sim D^e}\Big[\log \pi_\theta(a|s)\Big] \tag{4.1}$$

Compared to other LfD approaches, BC is simple to set up and can be trained offline without the need for agent-environment interaction.

The major issue with BC is that it is often considerably impacted by the distributional shift (a.k.a covariate shift) problem, characterised by a discrepancy between the distribution of the input states encountered during deployment and the distribution of the input states in the training dataset. This can arise if the environment changes or the given

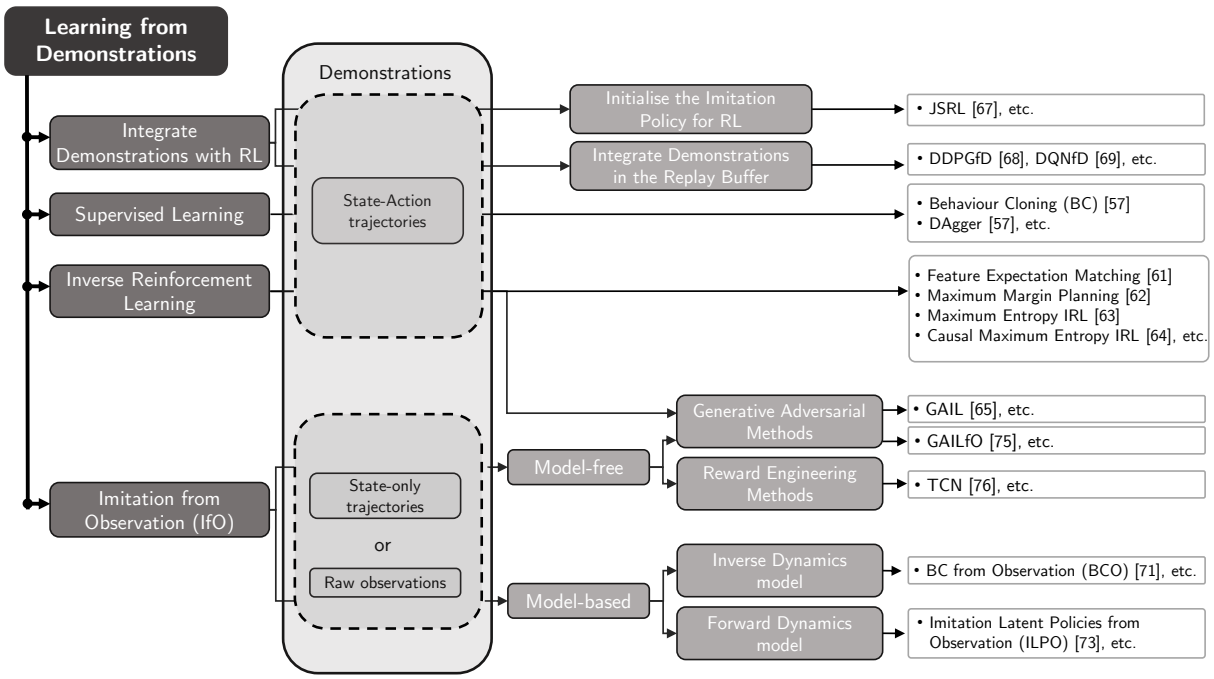


Figure 4.1 – Overview of the different Learning from Demonstration (LfD) methods covered in Chapter 4.

demonstrations are not fully representative of all the possible situations that the agent may experience. Insufficient exposure of the agent to representative examples of the new distribution of input states during training is likely to result in inaccurate or incorrect predictions and cause the agent to fail during deployment.

BC, being a supervised learning method, assumes that the state-action pairs are independent and identically distributed (i.i.d). This assumption is not met as the state-action pairs from demonstrations are temporally dependent. When the agent interacts with the environment during deployment after being trained on expert demonstrations, any deviation from the target actions results in an error that accumulates over time, causing the agent to increasingly diverge from the expert's behaviour (see Fig 4.2 for an illustration). This is known in the literature as the *compounding error* problem.

Dataset Aggregation

Dataset Aggregation (DAgger) [56] is an algorithm that addresses the distributional shift problem in Behaviour Cloning by leveraging the expert feedback on what actions to take in encountered states that were not seen in the initial demonstrations dataset.

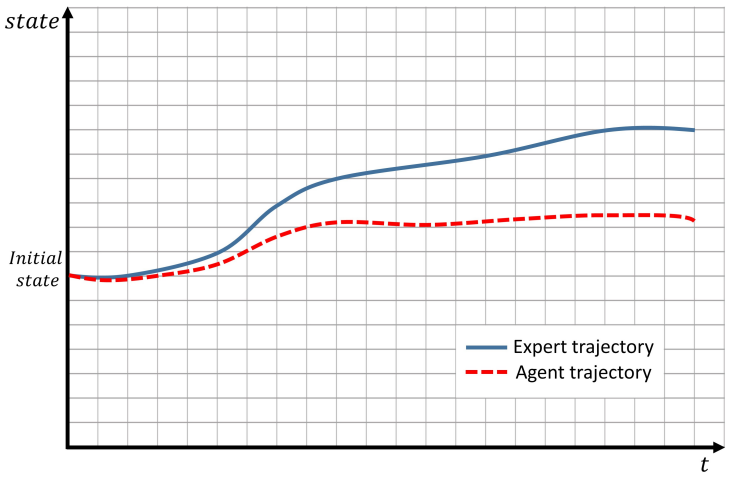


Figure 4.2 – Illustration of the compounding error in Behaviour Cloning. Small errors in the first few time-steps can steer the agent towards unfamiliar states that greatly deviate from the expert's demonstrated behaviour.

DAgger improves the imitation policy iteratively as follows:

1. Train an initial policy $\pi(a|s)$ on the original demonstration dataset $\mathcal{D}$.
2. Execute the current policy in the environment to collect agent trajectories.
3. Rectify the agent's incorrect actions with the expert intervention.
4. Include the new state-action pairs with their expert-assigned actions to the demonstration dataset.
5. Improve the policy $\pi(a|s)$ by training on the new demonstration dataset.
6. Iterate steps 2 to 5 until the agent's behaviour approaches the expert's.

Step 2 allows the agent to exploit the trained policy to interact with the environment. Through steps 3 to 5, the agent is given customised feedback from the expert, which helps it improve $\pi(a|s)$ towards mirroring the expert's policy. This results in policies that outperform classical Behaviour Cloning. One instance of DAgger applications to real-world problems is shown in [57], where a quadrotor is trained to autonomously navigate cluttered environments while avoiding obstacles. While DAgger has proven effective in practice, the iterative nature of the learning process and the need for the expert feedback make it computationally expensive and time-consuming, especially for complex tasks.

4.2.2 Inverse Reinforcement Learning

Inverse Reinforcement Leaning (IRL) learns from demonstrations by attempting to infer the expert's objective [58, 59]. Unlike behaviour cloning, where the goal is to mimic the expert's actions, IRL seeks to recover the implicit reward function that is driving the expert's behaviour. The recovered reward function can be then used with RL to train a policy that solves the task demonstrated by the expert.

Solving the IL problem using IRL assumes that the expert behaviour can be formulated using a Markov Decision Process (MDP) by defining the tuple $(\mathcal{S}, \mathcal{A}, R, \mathcal{P}, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathcal{R}$ is the reward function, $\mathcal{P}$ is the transition function, and γ is the discount factor. This formulation is identical to the RL formulation introduced in Chapter 2, Section 2.2, except that the reward function is assumed to be unknown in IRL.

IRL assumes that the provided demonstrations $\mathcal{D} = \{\tau_i^e\}_{i=1}^{i=N}$ are generated by a policy π_e followed by the expert, and attempts to find an estimate $\hat{r}_e$ of the reward that best describes the underlying motivation of the expert. The purpose of IRL is to find an estimate of the expert reward function such that the expected reward is maximised when the expert's policy is followed, i.e,

$$\mathbb{E}_{\tau\sim\pi_e}\left[\sum_{t=0}^{T}\gamma^t\hat{r}(s_t,a_t)\right] \geq \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{T}\gamma^t\hat{r}(s_t,a_t)\right] \quad \forall\pi \neq \pi_e$$

Conventionally, IRL methods iteratively alternate between adjusting the reward function and optimising it using RL.

Feature Expectation Matching

Feature Expectation Matching [60] consists in shaping the estimated expert reward function as a linear combination of predefined features:

$$\begin{aligned}\hat{r}_w(s,a) &= \sum_{i=1}^{i=n} w_i\phi_i(s,a) \\ &= w^\top\phi(s,a)\end{aligned} \tag{4.2}$$

where $w = (w_1, w_2, ..., w_n)$ is a vector of weights that determine the importance of each feature in $\phi(s,a) = \Big(\phi_1(s,a), \phi_2(s,a), ..., \phi_n(s,a)\Big)$. For instance, to drive a car, the features $\phi_i(s,a)$ might be speed, acceleration, steering angle, fuel efficiency, etc. The goal

is to determine the values of w_i that yield the best possible match between the reward function and the expert's actions. This match is evaluated by comparing the expected cumulative discounted rewards of the expert's policy with those of the imitator's policy, and measuring the difference between them. The expected cumulative discounted reward for a policy π is given as:

$$\begin{aligned}\mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{T}\gamma^t\hat{r}_w(s_t,a_t)\right] &= \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{T}\gamma^t w^\top\phi(s,a)\right] \\ &= w^\top\mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{T}\gamma^t\phi_i(s_t,a_t)\right] \\ &= w^\top\mu(\pi)\end{aligned} \tag{4.3}$$

where $\mu(\pi) = \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{T}\gamma^t\phi_i(s_t,a_t)\right]$ is the feature expectation of the policy π. The IRL problem is then solved by seeking a policy π_θ that has a feature expectation $\mu(\pi_\theta)$ close to the feature expectation of the expert's policy $\mu(\pi_e)$. Nonetheless, the issue with this approach is that the expected feature count is not a unique indicator and several policies may yield the same count. In the following, we briefly introduce three common methods in the literature that guarantee a unique solution for the IRL problem.

Maximum Margin Planning

Maximum Margin Planning (MMP) is proposed in [61]. The unique solution is ensured by using the maximum margin criterion to estimate the reward function. The reward function parameters are determined to maximise the difference between the behaviour of the expert's policy and any other given policy. MMP searches for the reward function that yields higher cumulative rewards for the expert-demonstrated trajectories in comparison to any other trajectories by a certain margin.

Maximum Entropy IRL

The insight behind Maximum Entropy IRL [62] is to seek the policy π_θ that, besides matching the expert's feature expectations, results in a distribution over trajectories $p(\tau)$ where $\tau \sim \pi_\theta$ that maximises the entropy of the distribution $H(p)$. This is formulated as:

$$\underset{\theta}{\operatorname{argmax}}\, H(p) = \underset{\theta}{\operatorname{argmax}} \sum_{\tau\sim\pi_\theta} p(\tau)\ln\left(\frac{1}{p(\tau)}\right) \tag{4.4}$$

subject to

$$\|\mu(\pi_e) - \mu(\pi_\theta)\| < \epsilon \tag{4.5}$$

$$\sum_{\tau \sim \pi_\theta} p(\tau) = 1, \quad p(\tau) > 0 \ \forall \tau \tag{4.6}$$

Where Eq. 4.5 determines how close the agent policy should match the expert policy, defined by ϵ.

While Maximum Entropy IRL can be applied only to deterministic environments with known transition function [62], Causal Maximum Entropy IRL extends to problems with unknown transition function and stochastic environments. Further details can be found in [63].

Generative Adversarial Imitation Learning

Ho and Ermon [64] show that the IRL problem can be framed as an occupancy measure matching instead of feature expectation matching. The occupancy measure $\rho_\pi : \mathcal{S} \times \mathcal{A} \longrightarrow \mathbb{R}$ of a given policy π is expressed as:

$$\rho_\pi = \pi(a|s) \sum_{t=0}^{\infty} \gamma^t \mathcal{P}(s_t = s|\pi) \tag{4.7}$$

The optimisation problem in GAIL is formulated to learn a policy that minimises the occupancy measure gap between the agent and the expert, while considering a regularizing term for the causal entropy:

$$\underset{\pi}{\text{minimise}} \quad d\Big(\rho_\pi(s,a), \rho_{\pi_e}(s,a)\Big) - H(\pi) \tag{4.8}$$

where $d(.)$ is a function that measures the difference between occupancy measure of the agent and the expert and $H(\pi)$ is the causal entropy of the policy π. Further details about how GAIL functions are given in the next chapter.

4.2.3 Combining demonstrations with RL

Combining RL with demonstrations can help improve the agent training performance through the RL's trial-and-error exploration and demonstration's indicative information about how the task should be solved. Demonstrations provide guidance and limit the state

space exploration. They can be incorporated into the initialisation of policies for RL, or included in the replay buffer of off-policy RL methods.

Initialise the Imitation Policy for RL

The agent's policy can be initialised using any IL method, with BC being the most commonly used due to its simplicity to train. For instance, in [65], BC is used to pretrain a policy that efficiently guides a policy-based algorithm exploration, eliminating the need for reward shaping. However, this strategy doesn't work well with value-based and actor-critic methods, as simply copying the pretrained policy's weights to the RL initial policy often leads to poor performance due to the randomly initialised critic network. This often results in a sharp decline in the actor performance from the first RL update, as the untrained critic provides misleading signals to the actor. To solve this problem, Uchendu et al. [66] suggest an approach named Jump-Start Reinforcement Learning (JSRL) that uses a separate policy that can be trained on demonstrations called guide-policy along with the RL policy, called the exploration-policy. The exploration-policy is trained online while the guide-policy is trained offline prior to training the exploration policy. At the beginning of training, the guide-policy is used for a number of time-steps in each episode to bring the agent closer to the goal state, after which the explore-policy takes over and generates actions towards the goal state.

Integrate Demonstrations in the Replay Buffer

One other approach is to include demonstrations directly into the replay buffer of an off-policy RL method. The replay buffer is where exploration interactions are stored. DDPG from Demonstrations (DDPGfD) [67] and Deep Q-Network from Demonstrations (DQNfD) [68] are two LfD methods that are designed to solve problems with continuous and discrete action spaces respectively. Both methods use prioritised experience replay [69] to balance between sampling from the exploration interactions and demonstrations.

4.3 Imitation from Observation

The imitation learning approaches outlined above have been designed for the cases where both states and actions are available for the imitator. The methods' reliance on action observation renders them unsuitable for learning from demonstrations without ex-

plicit access to actions, such as those obtained by using a camera sensor to watch a human expert complete a given task, where demonstrations are simply a sequence of images without any labels of the corresponding actions. Recent works have attempted to overcome this limitation, paving the way for a more specific problem, which is imitation from observation (IfO), in which imitation learning is solved using state-only demonstrations or raw observations only, where the actions are not observable. In the following, we classify the IfO methods to model-based and model-free methods

4.3.1 Model-based

A potential approach to address the IfO issue involves allowing the agent to explore its action space and acquire a model of the environment dynamics, which can be used to deduce the absent actions from the demonstrations dataset.

Inverse Dynamics Model

An inverse dynamics model is a mapping from state transitions to the corresponding actions. Once it has been learnt, it can be used to predict the actions that would result in the state transitions in the demonstrations dataset. Behavioural Cloning from Observation (BCO) [70] is one of the methods that follow this process. It consists of three phases: 1- learn a task-independent inverse dynamics model in a pre-demonstration, exploratory phase. The state space is partitioned to agent-specific features and task-specific features. In this phase, only the agent-specific part of the states are stored along with the corresponding actions and a model is learnt to predict the actions that would explain the transitions. 2- Upon state-only demonstrations, BCO utilises the learnt model to infer the missing actions. 3- Then BCO uses the state-action pairs to learn a policy via BC. Another work by Nair et al. [71] proposes a method for manipulating deformable objects (a rope in this paper), by first, learning a predictive model of rope behaviour using generated data from robot-rope interactions. Based on its assessment of the current and target state of the rope, the model is trained to predict the action that can be performed by the robot to achieve the target configuration of the rope from the current state. Then a human user provides step-by-step images that show intermediate goal configurations of the rope and the learnt model is used to determine the actions that would lead to the desired configuration of the rope.

Forward Dynamics Model

In the context of imitation learning, a forward dynamics model is a function that takes a state-action pair and returns the next state. For instance, Imitating Latent Policies from Observation (ILPO) [72] follows a two-phase process to learn the imitation policy. In the first phase, the agent learns a forward dynamics model along with a latent policy $\pi(z|s_t)$, which estimates the likelihood of the underlying latent action that accounts for a transition given the current state s_t. Then, the agent learns a latent-real action remapping network to match latent actions to the actions in the agent's action space. Another approach is proposed by Wu et al. [73] where first a forward dynamics model is learnt. The model takes as input the current image and an action to predict the next realistic image. Then, the model is used with a trained function that evaluates how similar generated state-action pairs are to the expert state-action pairs. The optimal action to take is selected by examining several potential action choices.

4.3.2 Model-free

The main limitation of model-based methods is that learning a dynamics model generally requires a considerable number of interactions with the environment which scales with the complexity of the task to learn. Model-free methods learn policies without requiring any model of the dynamics.

Generative Adversarial Methods

Inspired by the work of Ho and Ermon [64],Generative Adversarial Imitation from Observation GAILfO [74] algorithm was introduced as an approach to learn from state-only demonstrations. The core idea is to feed state transitions to the discriminator instead of state-action pairs, and the goal is to bring the imitator's state transitions distribution closer to that of the expert. Despite its success with low dimensional manually defined state representation, GAIfO failed to scale to the case where high-dimensional observations (e.g. raw pixel images) are used as inputs. This limitation could be addressed by learning a state representation from images that captures task-relevant features while discarding the irrelevant aspects of the observation [3].

Reward Engineering Methods

Another approach to solve IfO problem is to incorporate RL with a hand-engineered reward function that measures the distance between the observations in the demonstrations dataset and those observed by the imitator policy. One such method is presented by Liu et al. [49]. It aims to learn a model that can convert demonstrations from the context of the human expert to the robot's context. Context differences might include changes in viewpoint, background etc. Once this translation model is obtained, a reward function is defined as a squared Euclidean distance that measures the similarity between observations of the imitator and the expert's translated observations. Similarly, time-contrastive networks (TCN) [75] address imitation from observation by following two steps. First, an embedding is learnt from diverse passive observations with different objects and backgrounds. A triplet loss function is used to learn a representation that brings frames coming from the same time but different viewpoints closer to each other in the embedding space, while visually similar frames close to each other in time are pushed apart in the embedding space. This encoding is used thereafter as input to a RL algorithm to optimise a reward function that measures the distance between the encoding of a single expert video demonstration and the encoding of the robot's observations. Both approaches require multiple demonstrations of the task to be time-aligned, which is generally not a reasonable assumption for an intuitive learning from human demonstrations.

4.4 Summary

In this chapter, we introduced the paradigm of learning from demonstrations as a solution to the manual design of the reward function in RL, including approaches such as supervised Behaviour Cloning and Inverse Reinforcement Learning. We then focused on imitation from observation where actions are not available in the demonstrations dataset. With its ability to achieve comparable results with fewer demonstrations and directly recover the imitation policy, Generative Adversarial Imitation Learning (GAIL) has recently emerged as a powerful method for learning from demonstrations. Additionally, this method has an extension called GAIL from Observation (GAILfO), which can handle state-only demonstrations. In the next chapter we dive into the implementation details of GAIL and GAILfO algorithms, and apply it to our use case task.

5

Adversarial Imitation Learning

Contents

5.1 Introduction

Generative Adversarial Networks (GANs) [76] proposed by Goodfellow et al. in 2014 have proven to be a remarkably powerful approach to solve problems in various domains, ranging from computer vision [77] to natural language processing [78]. The ability of GANs to generate realistic data samples has led to a series of exciting and creative applications, such as music and speech synthesis [79], and realistic image generation [80].

Recently, adversarial learning has been extended to the field of imitation learning, where the objective is to learn a policy to imitate the behaviour of an expert. Generative Adversarial Imitation Learning (GAIL) is an approach that uses GANs to learn an imitation policy that can generate trajectories resembling those of the expert [81–84]. This

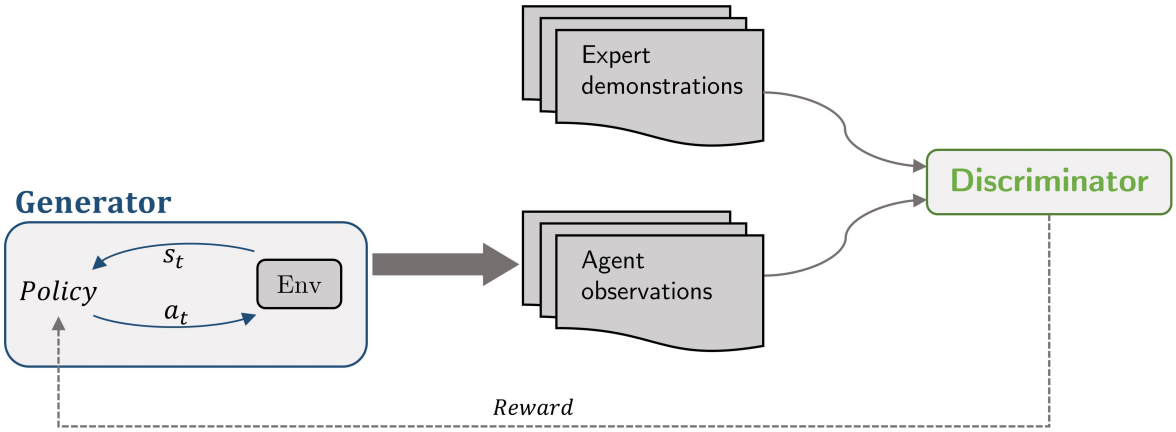


Figure 5.1 – Generative Adversarial Imitation Learning (GAIL) architecture. In imitation learning, adversarial methods use the discriminator to construct a reward function that measures the similarity between the trajectories generated by the agent and those provided by the expert. The policy gets as input the current state of the environment s_t and executes an action a_t, and trained using the reward provided by the discriminator. The reward can be seen as an indication of how the imitator is performing compared to the expert.

approach has been applied to various real-world problems in robotics [85] [86]. GAILfO [74] is an extended version of GAIL designed to solve imitation from state-only demonstrations.

In the remainder of this chapter, we delve deeper into the inner architecture of GAIL and GAILfO, exploring their key components and their performance for solving a robotic manipulation task.

5.2 Adversarial Imitation Learning

5.2.1 Generative Adversarial Imitation Learning

The GAIL method [64] solves the imitation learning problem by training two entities in an adversarial fashion (see Fig 5.1). Inspired by the work of [76], the generator G is represented by a stochastic policy $\pi_\theta(a|s)$ that takes as input the state of the environment and outputs an action to execute by the imitator. The discriminator is trained to separate inputs sourced from the expert and inputs generated by the imitator's policy.

In the original paper [64], the discriminator produces a value within the range $[0, 1]$ representing the probability that a given state-action pair originates from the expert policy. The imitator policy (the generator) is updated using a policy gradient method

with a reward function that uses the output of the discriminator.

To derive the objective function to optimise by the generator and the discriminator, Ho and Ermon [64] suggest to learn the imitation policy by minimising the distance between the occupancy measures of the agent and the expert while maximising an entropy-regularisation term. The optimisation problem is defined as follows:

$$\underset{\pi}{\text{minimise}} \quad d\Big(\rho_\pi(s,a), \rho_{\pi_e}(s,a)\Big) - H(\pi) \tag{5.1}$$

The occupancy measure $\rho_\pi : \mathcal{S} \times \mathcal{A} \longrightarrow \mathbb{R}$ of a policy $\pi_\theta(a|s)$ can be understood as the distribution of the state-action pairs $(s,a) \in \mathcal{S} \times \mathcal{A}$ induced by executing the actions produced by the policy π_θ and is defined as follows:

$$\rho_\pi(s,a) = \pi_\theta(a|s) \sum_{t=0}^{\infty} \gamma^t P(s_t = s|\pi_\theta) \tag{5.2}$$

The distance function $d(.)$ is calculated between the occupancy measure of the agent and the expert. In the original paper [64], d is chosen to be the Jensen-Shannon divergence between the two measures. This results in the objective function defined in Eq. 5.3, that both the policy and the discriminator optimise to find a saddle point (π, D).

$$\mathbb{E}_\pi\Big[\log(D(s,a))\Big] + \mathbb{E}_{\pi_e}\Big[\log\Big(1 - D(s,a)\Big)\Big] - \lambda H(\pi) \tag{5.3}$$

Further details of the derivation of the formulation above are given in [64]. The objective function is optimised by training a parametrised policy π_θ which plays the role of the generator, and a discriminator network D_w with the parameters w. The training is done by alternating between updating w and updating θ. The weights w are updated to maximise Eq. 5.3 with respect to w using supervised learning while the weights θ are updated using reinforcement learning to decrease Eq. 5.3 with respect to θ. The reward function is constructed using the discriminator's output, which denoted as $R(s,a) = \log\Big(D(s,a)\Big)$ in the original paper. Alg. 1 provides a condensed version of the key steps of the GAIL algorithm..

5.2.2 Generative Adversarial Imitation Learning from Observation

Generative Adversarial Imitation Learning from Observation (GAILfO) [74] is an extension of GAIL that can handle state-only demonstrations, which refer to cases where

Algorithm 1: Generative Adversarial Imitation Learning (GAIL)

Input: Expert trajectories $\tau_e = \{(s, a)\} \sim \pi_e$,
Initialise the parameters of G and D with θ_0 and w_0 respectively ;
for *i=1,2,...* **do**
  Sample agent trajectories $\tau_i = \{(s, a)\} \sim \pi_{\theta_i}$;
  Update D_w using the loss

$$\mathcal{L}_D = -\Big[\mathbb{E}_{\tau_i}\Big[\log(D_w(s,a))\Big] + \mathbb{E}_{\tau_e}\Big[\log(1 - D_w(s,a))\Big]\Big] \ ;$$

  Update π_θ by performing a TRPO update using the reward function

$$R(s,a) = \log\Big(D_w(s,a)\Big) \ ;$$

end

the expert provides state information without the corresponding actions. This can prove helpful in scenarios where it is difficult or costly to provide expert demonstrations with actions.

To adapt GAIL to handle state-only demonstrations, GAIL was modified to learn from state transitions, rather than state-action pairs. During training, the algorithm seeks to minimise the distance between the state transitions distribution generated by the learned policy and the distribution of those observed in the demonstration data. The GAILfO algorithm is given in Alg. 2.

Algorithm 2: Generative Adversarial Imitation Learning from Observation (GAILfO)

Input: Expert trajectories $\tau_E = \{(s, s')\} \sim \pi_E$,
Initialise the parameters of G and D with θ_0 and w_0 respectively ;
for *i=1,2,...* **do**
  Sample agent trajectories $\tau_i = \{(s, s')\} \sim \pi_{\theta_i}$;
  Update D_w using the loss

$$\mathcal{L}_D = -\Big[\mathbb{E}_{\tau_i}\Big[\log(D_w(s,s'))\Big] + \mathbb{E}_{\tau_E}\Big[\log(1 - D_w(s,s'))\Big]\Big] \ ;$$

  Update π_θ by performing a TRPO update using the reward function

$$R(s,s') = \log\Big(D_w(s,s')\Big) \ ;$$

end

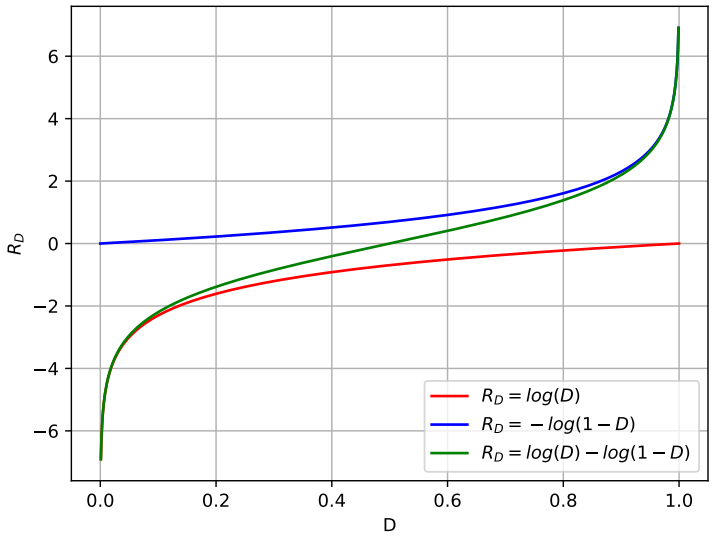


Figure 5.2 – Plot of the reward functions defined in Eq.5.4, Eq.5.5 and Eq.5.6.

5.3 Discriminator Reward Function

The imitator is trained to maximise the reward function computed using the discriminator's output. Let $D(s,a) \in [0,1]$ be the output of the discriminator and high values classify the input (s,a) as coming from the expert. The state-action pair input (s_t, a_t) can be replaced by the state transition (s_t, s_{t+1}), or single state s_t. The reward function to be used by the imitation policy (i.e. the generator) can be chosen from the following:

$$R_D = \log\Big(D(s,a)\Big) \tag{5.4}$$

$$R_D = -\log\Big(1 - D(s,a)\Big) \tag{5.5}$$

The reward defined in Eq. 5.4 is always negative and that defined in Eq. 5.5 is always positive for $D \in [0,1]$. The two rewards are likely to induce bias in the learning process leading to behaviours that diverge from that of the expert [87] [88].

— **Termination bias:** Using the reward function defined in Eq.5.4 exposes the agent to a termination bias. In this case, the agent seeks to complete the episode in the shortest possible time to reduce the number of penalties it receives. This is desirable for goal-based tasks where the agent should find the shortest path to achieve the goal state. Nonetheless, if another option is available to the agent to complete the episode faster, it will be favoured by the agent's policy even if it does not solve the

intended task.

— **Survival bias:** If the agent is positively rewarded in every time-step, as in the case of using Eq.5.5, the agent may prioritise collecting additional rewards over completing the task, which may potentially lead to looping through the environment until all available time-steps are exhausted. This behaviour may occur because the agent is not being motivated to complete the task, as it is already receiving a positive reward for its actions at each time-step.

Another option is to use a neutral reward function defined as follows:

$$R_D = \log(D(s,a)) - \log(1 - D(s,a)) \tag{5.6}$$

Where $R_D \in]-\infty, +\infty[$ for $D \in [0,1]$ (see Fig. 5.2). It is demonstrated in [89] that this reward can help overcome the two aforementioned reward biases. Since the Eq.5.6 supplies both positive and negative rewards, the survival bias can be overcome by penalising the agent (i.e. returning negative rewards) for taking actions that are considerably different from those of the expert, and the termination bias can be overcome by positively rewarding the agent for taking actions that are similar to those of the expert and avoid a premature termination of training episodes.

5.4 Experiments and Results

5.4.1 Experimental setup

The task to solve is the pushing task where a robot has to push an object to a target location (see Fig. 5.3). We use the best trained RL model from Chapter 3 as the expert policy to generate demonstrations. The expert policy was trained on the shaped reward function defined in Chapter 3, Eq.3.1. We refer to this reward function as the ground truth reward in the rest of this chapter.

A total of $N = 100$ trajectories of successful completion of the task were generated using the expert policy. The success of a trajectory is determined by the object being fully pushed to the target destination. Demonstrations are collected by executing the expert policy in the environment to solve the pushing task for random target locations. At each time-step, the state and associated action are recorded. Actions are required for GAIL algorithm, whereas GAILfO algorithm relies solely on state transitions.

The imitator policy is designed to have the same state space and action space of the

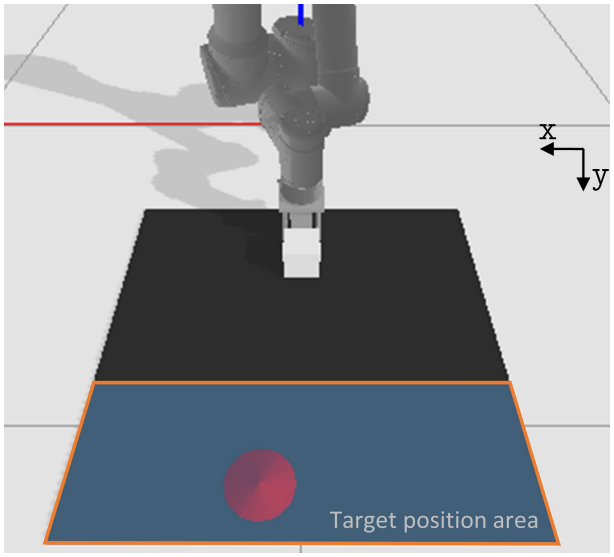


Figure 5.3 – Snapshot of the pushing task environment in simulation. The object should be pushed to the target position represented in red. The target position is sampled randomly from the target area.

expert. It receives as input the $2D$ coordinates of the robot's end-effector, the object and the target. At each time-step, the imitator policy outputs two continuous values dx and dy, the two displacements to make along the Cartesian axes x and y. An episode of training concludes when one of the following conditions is satisfied:

— The object or the end-effector of the robot is out of the board.

— The maximal length of an episode has been reached (150 time-steps in our case).

— The object has reached the target position.

In the next section, we train an agent to learn from the provided demonstrations using the GAIL algorithm. We use the implementation of GAIL from [90], which utilises Stable-Baselines3 library [44]. Their implementation features a discriminator that assigns high values to the samples produced by the generator, which is opposite to the choice made in the original paper where high values are assigned to the expert samples [64]. Accordingly the discriminator reward function can be chosen from the following:

$$R_D = \log\Big(1 - D(s,a)\Big) \tag{5.7}$$

$$R_D = -\log\Big(D(s,a)\Big) \tag{5.8}$$

$$R_D = \log\Big(1 - D(s,a)\Big) - \log\Big(D(s,a)\Big) \tag{5.9}$$

Description	Detail
Environment	
Episode maximum length	150
seed	17
Generator (policy)	
RL algorithm	PPO
NN hidden layers	(64, 64)
Activation function	Tanh
Learning rate	3e-4
Discriminator	
NN hidden layers	(32, 32)
Activation function	ReLu

Table 5.1 – GAIL and GAILfO training Hyperparameters.

Where Eq.5.7 corresponds to Eq.5.4, Eq.5.8 corresponds to Eq. 5.5 and Eq. 5.9 corresponds to Eq. 5.6. The original GAIL implementation was modified to support the three options for choosing the discriminator reward function.

5.4.2 Train GAIL with positive, negative and neutral discriminator reward functions

We train three models using the same policy and discriminator architectures outlined in Table 5.1. Each model is trained for 5M time-steps using one of the discriminator reward functions defined in Eq. 5.7, Eq. 5.8 and Eq. 5.9. The results are shown in Fig. 5.4. The RL-expert was trained on the ground truth reward function, while the GAIL models are trained on the discriminator reward function. Each one of the discriminator reward functions results in a different behaviour. The reward $R_D = \log(1 - D)$ is negative for $D \in [0, 1]$, which induces the *termination bias* in the training process and interferes with the robot's ability to learn the task demonstrated by the expert. This causes the robot to complete episodes as fast as possible to minimise the penalties incurred from the discriminator rewards, which prevents it from efficiently exploring the environment and acquiring the skills needed to perform the desired task. Using the positive reward function $R_D = -\log(D)$ results in better performance compared to the negative reward function. However, the robot suffers from the *survival bias* issue, where it is sufficient to remain in the environment for as long as possible to get cumulative positive rewards and the robot, also in this case, is diverted from learning the demonstrated task. Using the

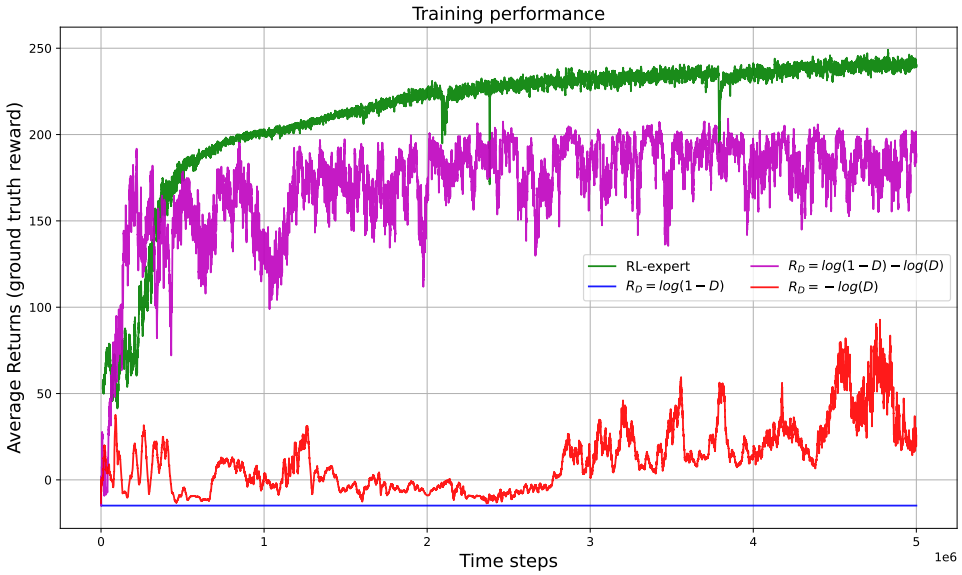


Figure 5.4 – Training performance vs. training time-steps, showing the impact of the discriminator reward function on the performance of GAIL trained models compared to the expert's performance.

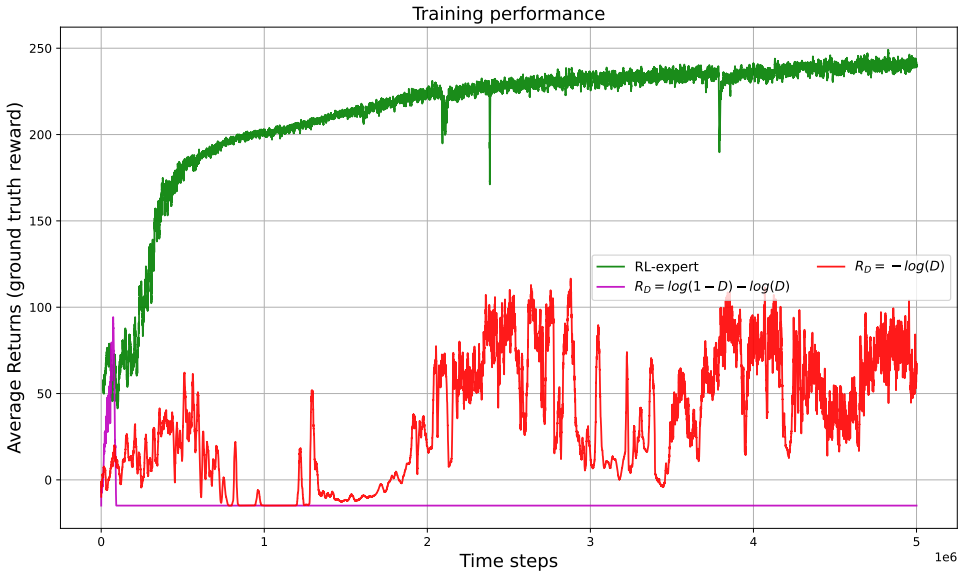
reward function $R_D = \log\Big(1 - D(s,a)\Big) - \log\Big(D(s,a)\Big)$ proposed in [89], which addresses both biases by providing both negative and positive rewards, the trained model converges to a saddle point where the average ground truth reward fluctuates between 150 and 200. The policy with the highest value got a success rate of 99%.

5.4.3 Train GAILfO with positive and neutral discriminator reward functions

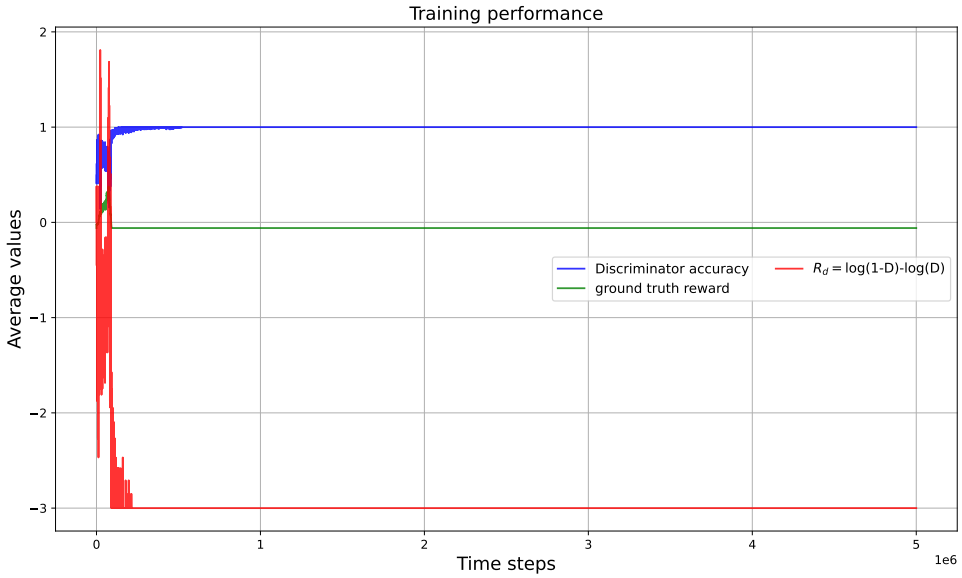
While in the previous section, state-action pairs were used for training the imitator, in this section we employ GAILfO which makes use only of the state transitions. We slightly modified the implementation from [90] to handle state-state pairs as inputs for the discriminator. The results are shown in Fig. 5.4-a. Surprisingly, none of the three rewards, including the neutral reward function that worked well with GAIL, resulted in the agent learning an acceptable behaviour.

Fig. 5.5-b reveals a correlation between the agent's cumulative ground truth rewards, the discriminator cumulative rewards, and the discriminator accuracy. The following is a breakdown of each element:

— The average of the ground truth rewards indicates the agent's performance in



(a) Training performance vs. training time-steps.



(b) Training performance vs. training time-steps, exhibiting the progression of the discriminator accuracy, the average cumulative rewards, and the cumulative discriminator rewards received by the agent during training. The cumulative ground truth reward is divided by the expert's highest score of 250 for normalisation.

Figure 5.5 – Training performance of GAILfO

solving the task demonstrated by the expert. The higher the cumulative reward, the more the agent is on track to accomplishing the task.

— The discriminator rewards R_d are used for training the imitation policy and give

an insight into the sign of the rewards received by the agent.

— The discriminator accuracy indicates how comfortable the discriminator is with distinguishing the expert rollouts from those of the imitator. The accuracy score is computed by dividing the number of correct predictions by the total number of predictions.

At the start of training, the GAILfO exhibits signs of improvement, as evidenced by an increase in the average cumulative ground truth rewards (see Fig 5.5-b). However, the model's performance takes a sudden downturn, and the cumulative ground truth rewards converge to a negative value. Concurrently, the discriminator's reward also becomes negative, and its accuracy converges to one.

There are two potential scenarios that may cause the sudden performance drop:

— Discriminator overfiting, which is experienced when the discriminator can easily distinguish between the samples coming from the expert and those coming from the imitator. This may occur if, for instance, the agent explores areas of the state space that are considerably distinct from the states visited by the expert in the demonstration dataset. As a result, the accuracy of the discriminator converges towards one, causing the imitator rollouts to be correctly recognised as non-expert and assigned values that are close to zero:

$$D(s, s') \approx 0 \quad \text{with } (s, s') \sim \pi_\theta$$
$$\implies R_D \to -\infty$$

In response to receiving only negative rewards, the imitator chooses to quickly terminate episodes to minimise the cumulative negative rewards.

— If the agent encounters terminal states other than the goal state during exploration, the discriminator can easily distinguish the agent's trajectories from those from the expert as demonstrations don't cover those terminal states, leading to an increase in the discriminator accuracy. This, in turn, leads to generating only negative rewards, causing the agent to prioritise reaching terminal states over imitating the expert.

5.4.4 Addressing the reward bias in GAILfO

The reward function suggested by Jena et al. [89] successfully eliminates the discriminator reward bias when GAIL is trained with state-action pairs, as demonstrated in the

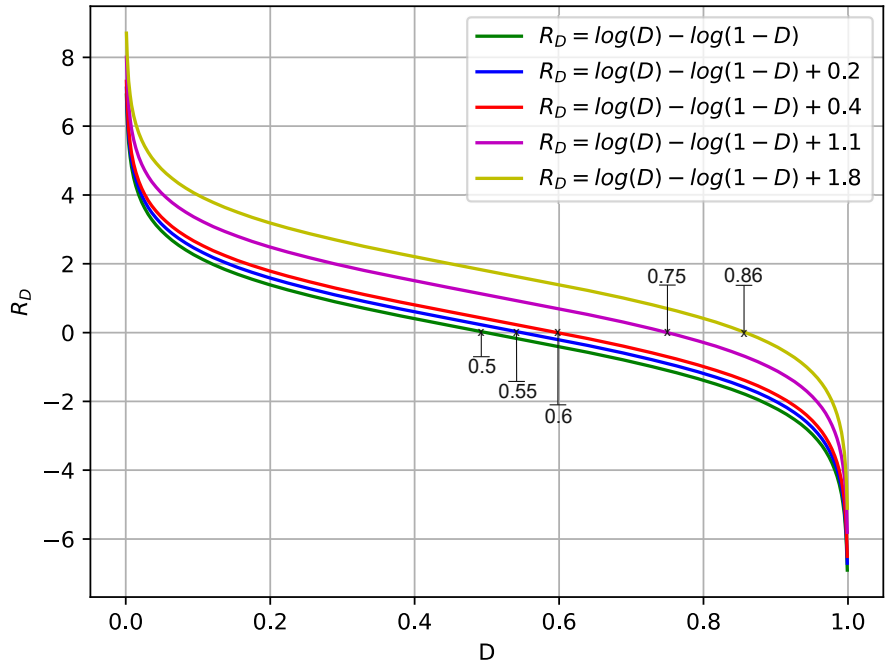


Figure 5.6 – Plot of the discriminator reward function defined in Eq. 5.10 with different values of R_D offset.

previous two sections. However, this approach fails to work when GAILfO is trained on state transitions. Although the discriminator rewards can be both positive and negative, they tend to converge to negative values when state-state pairs are involved.

This section delves into modifying the discriminator reward function defined in Eq. 5.9 to balance positive and negative rewards by introducing an offset so the discriminator reward becomes:

$$R_D = \log\Big(1 - D(s,a)\Big) - \log\Big(D(s,a)\Big) + R_d \,\text{offset} \tag{5.10}$$

Where R_d offset $\in \mathbb{R}^+$. We investigate the impact of varying R_d offset on the training performance. We conduct experiments using four different values of R_d offset, namely $\{0.2, 0.4, 1.1, 1.8\}$. The resulting discriminator reward function curves are shown in Fig. 5.6. In the case where $R_D = \log\Big(1 - D(s,a)\Big) - \log\Big(D(s,a)\Big)$, the discriminator output threshold that decides whether the reward is going to be positive or negative is 0.5, i.e., rewards are positive for $D \in [0, 0.5]$ and negative for $D \in [0.5, 1]$. The higher the offset added to the discriminator reward the larger the range that gets positive rewards. For instance, for R_d offset $= 1.1$, rewards are positive for $D \in [0, 0.75]$ and negative for $D \in [0.75, 1]$.

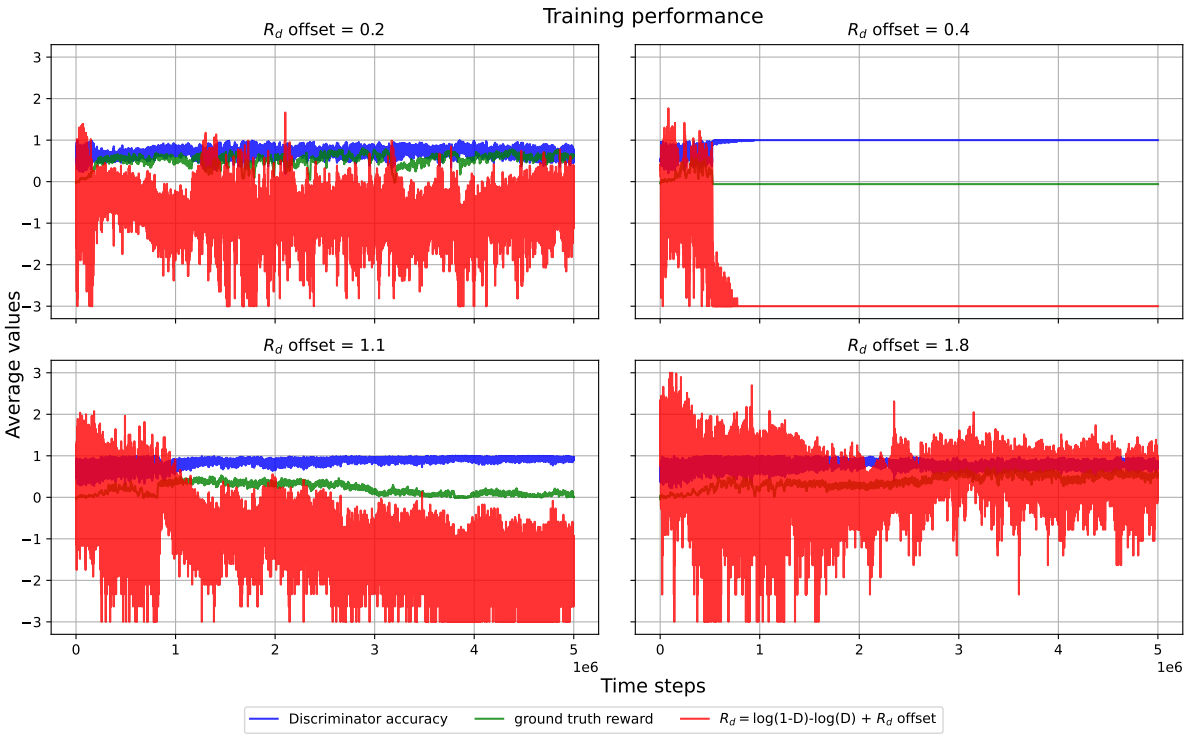


Figure 5.7 – Training performance vs. training time-steps, showing the impact of adding an offset to the discriminator reward function defined in Eq.5.9 on the performance of GAILfO trained models. The cumulative ground truth reward is divided by the expert's highest score of 250 for normalisation.

Four models were trained with the generator and discriminator architecture from the previous section (see Table 5.1), each with a different R_d offset. The results are depicted in Fig. 5.7. The models with R_d offsets of 0.2 and 1.8 converged to a saddle point, while those with R_d offsets of 0.4 and 1.1 experienced a sudden drop in performance. The successful models maintained a balance between positive and negative rewards, while the unsuccessful ones had a higher proportion of negative rewards.

The findings indicate that, simply incorporating an offset to the discriminator rewards can assist in controlling the balance of positive and negative rewards. This can significantly aid in overcoming reward bias in adversarial learning.

5.5 Summary

This chapter focused on Generative Adversarial Imitation Learning GAIL and its application to learning from state-only demonstrations, known as GAILfO. The use of

a neutral reward function that generates both negative and positive rewards was found to be effective in facilitating GAIL's learning from state-action pairs, but not GAILfO, which learns from state-state pairs. In order to counter the issue of unbalanced rewards that results in *termination* and *survival bias*, we suggested incorporating an offset into the discriminator's reward. This proved to be a helpful adjustment for achieving convergence.

Even in the case where an adversarial model successfully converges to a saddle point, the instability of the training still causes fluctuations in the behaviour of the imitator due to the noisy rewards provided by the discriminator. In the next chapters, we propose a novel framework that incorporates a stable, task-agnostic reward function with an off-the-shelf RL method to learn the imitation policy. Our experimental findings demonstrate that our method offers better performance and transferability to the real-world than Generative Adversarial Learning.

6

Representation Learning

Contents

6.1 Introduction

We consider tasks in which a hand or an effector manipulates objects in a certain way to achieve a specific goal. The agent should grasp the essence of the task in question by observing an expert performing it, then perform the task by itself. Part of solving the imitation problem is to determine how demonstrations are construed and how they should

be reasoned about. We follow the approach of disentangling the "how" from the "what" to imitate. The "what" to imitate are the essential aspects of the demonstrations that can guide the imitation learning algorithm to an optimal policy that successfully solves the task. The "how" to imitate is how the task is performed by the demonstrator which depends on the physical embodiment and capabilities of the agent performing the task. For example, if a robot is to imitate a human expert, the embodiment mismatch between humans and robots makes it infeasible to directly transfer the "how" from observing the human expert to the robot's actuator commands. The hierarchical approach of extracting the "what" to imitate followed by deciding the "how" to imitate is by no means a new idea. Numerous visual imitation learning methods have been developed following the same approach (e.g. [91]). Psychology refers to this as *observational learning* [92].

Depending on how the demonstrations are captured, part of solving the imitation problem is to engineer the way the demonstrations are delivered to the agent. In imitation from observation, representing demonstrations in a way that is informative for the imitation learning process is an important challenge. Demonstrations should be leveraged in a convenient format that is conducive to optimal learning.

Raw demonstrations are generally high dimensional and include redundant, irrelevant and superfluous information. Learning the imitation policy directly from raw data necessitates the use of large and deep machine learning architectures with a large number of trainable parameters. This implies the need for a large amount of data to learn useful features and discard those that are irrelevant. This is particularly associated with visual demonstrations where an expert is observed using visual sensors. The data produced are high-dimensional recordings in the format of pixels per frame. Most pixels contain information that are extraneous to the task being demonstrated, representing mainly the background and the surroundings of the expert. It is hence essential to extract relevant features from raw data for effective learning.

Feature extraction can be done automatically or designed manually. Expert knowledge of the task can be used to hand-engineer functions to extract the features that are considered relevant to solving the task at hand [74]. Machine Learning, on the other hand, can be used to automatically extract features from the data based on a predefined objective function [22].

In this chapter, we propose a representation pipeline that comprises both manually designed and automated feature extractors. We first design how each observation in the demonstrations is represented, and then learn a spatio-temporal embedding to capture

the essential elements of the scene as well as their change over time. More precisely, we suggest modelling the observations as graphs of connected nodes. Graph Neural Networks (GNNs) are used to process the graphs and learn a spatial embedding for each graph. Sequence modelling is used to capture temporal features about how the graphs evolve in time within demonstrations.

In what follows, we first present a brief review of the literature on GNN applications and their functioning in section 6.2. In section 6.3, we discuss how sequence modelling is used to learn temporal features. Then, we present our pipeline that couples Graphs, GNNs, and sequence modelling to learn a spatio-temporal representation model of demonstrations in section 6.4.

6.2 Graph Neural Networks

Graph Neural Networks (GNNs) [93] are deep learning methods tailored for graph processing. A graph is a structured representation of linked entities. The entities are represented by nodes that encapsulate characteristics specific to a given entity and links are represented by edges that encode relationships between connected nodes. Different data modalities can be represented by graphs. From text and images to social networks and molecules. Once the graphs are designed, GNNs can be used to process them by aggregating node and edge features into a representation that preserves both the graph topological structure and the content of the graph. The core insight is to train mapping models that embed the nodes and edges to continuous embedding spaces. The downstream task defines what features should be focused on in graphs and which to discard in order to successfully solve the task at hand.

Graph representation learning has recently received a considerable amount of attention after its successful application to solve various problems in different domains such as social networks, natural language processing, traffic, transportation networks, smart grids, etc [94]. GNNs have also gained traction in robotics, action recognition and autonomous agent applications [95–98]. Most of these applications use computer vision as a backbone to perceive the environment. Instead of using raw images, these can be processed into *scene graphs*: networks of interconnected objects that make the representation of the scene more efficient and low dimensional.

For instance, GNNs are used to solve the motion planning problem where many objects ought to be taken into consideration by the planner. Having to deal with many objects

puts a constraint on the time needed for calculation. GNNs were used in [96] to increase the planning calculation efficiency. This is achieved by learning a GNN-based prediction model to predict subsets of relevant objects for solving the planning problem. The pertinence of objects is calculated by reasoning about the attributes of the objects and their relations. Similarly, in [97], Khan et al. employ GNNs to address planning problems involving a large number of objects with only a reduced subset that should be considered for a given purpose. Graphs represent a collection of points in the planning space interconnected by edges that embed the cost to transit between the connected nodes. GNNs are used to process the constructed graphs and identify critical nodes. Identifying only relevant nodes allows optimising the time for graph search for the feasible path that has the lowest cost. In [95], manipulation tasks are regarded as a series of spatial constraints and relations between the manipulated objects. Demonstrations from an expert are used to train a GNN policy for controlling a robot to perform the shown task. The environment is represented in terms of a graph with nodes representing the objects and goals relevant to the task to be performed. The graph is fed to a graph neural network that decides both the most relevant object to select from the scene and the corresponding goal for the chosen object to attain. Then an action that achieves the transition is selected from a predefined set of movement primitives and executed.

GNNs are also used for action recognition. Either for body-action recognition or manipulation-action recognition. By body-action recognition, we refer to the recognition of actions that only require the observation of the movement of the human body. For this class of problems, using skeleton detection has shown great success in action recognition [99–101]. First, human skeleton is detected from videos and then actions are recognised based on the movements of the skeleton. The skeleton can be represented as a graph and processed by GNNs [102] [103]. Manipulation-action recognition involves identifying actions that require a human to interact with and manipulate objects in a given scene to achieve a specific goal. This process typically begins with the detection and categorisation of objects in the scene, using attributes such as size, class, 3D position, and bounding box. For instance, Dreher et al. [98] model the scene as a graph of detected objects linked by symbolic relations and train a GNN model for action recognition and frame-wise segmentation.

In the rest of this section, we first introduce a formal definition of a graph. Then, we classify the methods into three categories: node-level, edge-level and graph-level tasks.

6.2.1 Graph Definition

A graph $G = (\mathcal{V}, \mathcal{E}, A)$ is defined by a set of nodes $\mathcal{V}$, a set of edges $\mathcal{E}$, and the adjacency matrix A. We denote a node in a graph G by $v \in \mathcal{V}$ and an edge between two nodes v and u as $e_{u,v} = (u, v) \in \mathcal{E}$. The adjacency matrix is constructed as follows: $A_{u,v} = 0$ if $e_{u,v} \notin \mathcal{E}$ and $A_{u,v} > 0$ if $e_{u,v} \in \mathcal{E}$. Node attributes X can be assigned to a graph, where $X \in R^{N \times D}$ is a matrix of node features in which $X_v \in R^D$ represents the feature vector of node v.

6.2.2 Training GNNs

GNNs can be used to solve tasks at the node, edge or graph level.

Node-level tasks aim to predict a label or continuous vector for each node $u \in V$. Label prediction can be used either for node classification, to sort nodes into distinct classes, or for node clustering (i.e. splitting the nodes into separate groups, where related nodes are to be in the same group).

Edge-level tasks are primarily concerned with link prediction (also called relational inference or graph completion). Link prediction is useful when a set of nodes is given with a partial list of the edges connecting the nodes. Each missing edge between two nodes u and v is predicted by applying a neural network to the hidden representation of the two nodes.

Graph-level tasks train a representation for the whole graph. Compact representations of graphs are obtained by applying pooling and readout functions to the node and edge features. The graph embedding can then be used either for graph classification, graph clustering or graph regression.

In our work, we use GNNs to process scene graphs and generate a *graph-level* embedding that encodes their structural features. In the following, we formally describe how a GNN processes graphs to project them in a low-dimensional space.

As shown in Fig. 6.1, the GNN model takes as input a graph and learns an embedding for all nodes in the graph by aggregating each node's features with its neighbours' features. Let $G = (\mathcal{V}, \mathcal{E}, A)$ be a graph and $h_v^0 = X_v$ the initial feature vector of a given node $v \in \mathcal{V}$ before any training iteration. The initial representation of the nodes is drawn from their hand-designed labels and features. Every node $v \in \mathcal{V}$ is passed through L *message-passing*

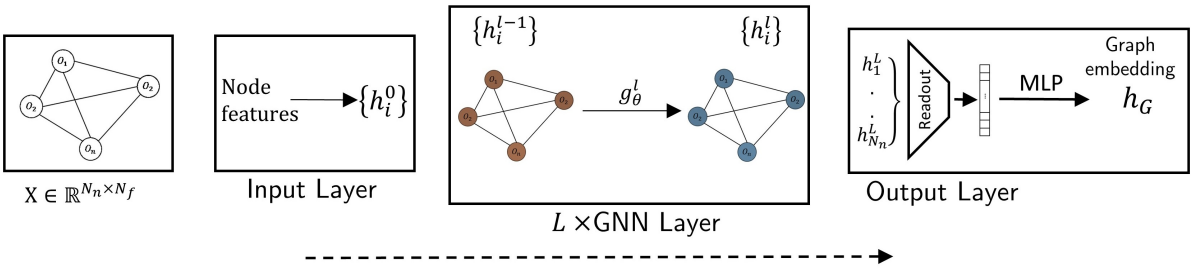


Figure 6.1 – Illustration of the GNN model.

layers. At each layer l the embedding h_v^l of the node v is updated according to Eq. 6.1:

$$h_v^l = g_\theta^l \left(h_v^{l-1}, h_{N(v)}^l \right) \tag{6.1}$$

where h_v^{l-1} is the features of node v from the previous iteration; $N(v)$ is the set of nodes that are connected to the node v by an edge: $N(v) = \{u \in \mathcal{V} \mid (u, v) \in \mathcal{E}\}$; and $h_{N(v)}$ is an aggregation of all neighbouring nodes:

$$h_{N(v)}^l = AGG_l \left(\left\{ h_u^{l-1} : \forall u \in N(v) \right\} \right) \tag{6.2}$$

where AGG_l is a permutation invariant function such as the mean, max pooling [104] or summation [105]. g_θ is a parametric function for which weights θ are trained by gradient descent. Different GNN methods differ in the conception of the g_θ function. All nodes are processed with the same function g with the parameters θ. Thus, once the training is completed, the obtained GNN model can be deployed on new graphs with any given number of nodes.

The graph-level embedding h_g is computed by applying a permutation invariant *Readout* function to all nodes embedding from the last layer L:

$$h_g \;=\; Readout(h_v^L | v \in \mathcal{V}) \tag{6.3}$$

Usually, the output of the Readout function is passed through a Multi-Layer Perceptron (MLP):

$$h_g \;=\; MLP(Readout(h_v^L | v \in \mathcal{V})) \tag{6.4}$$

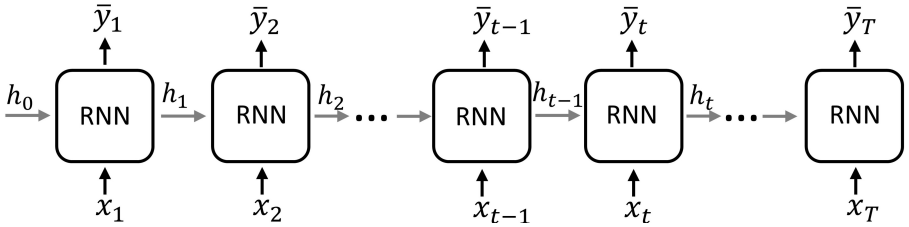


Figure 6.2 – Illustration of an RNN.

6.3 Seq2Seq Modelling

Sequence to Sequence (that we abbreviate to Seq2Seq) learning refers to training models to capture temporal patterns of a source input sequence to predict a target output sequence. It has been extensively used in natural language processing applications such as machine translation [106], text summarisation [107], chatbots [108], etc. The prevalent architecture used as the backbone to design Seq2Seq models is the Encoder-Decoder architecture. The encoder processes an input sequence and encodes it into a fixed-size continuous vector representation. The decoder takes the encoded vector and outputs a target sequence. Different models have been proposed in the literature to build efficient encoders and decoders and improve their performance [109].

6.3.1 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are a distinct type of neural networks, designed to process sequential data. Hence, they are more suited for input modalities such as audio, video and text. Unlike feed-forward neural networks, where all inputs are independent of each other, RNNs compute outputs as a function of previous inputs. When an input sequence is passed to an RNN, each element of the sequence is processed at a time along with a history vector that contains information about previously processed elements in the sequence. The history vector—also called the hidden state—retains information from the past elements that are relevant for predicting the current output.

Fig 6.2 depicts the basic structure of a standard RNN. Let $(x_1, x_2, ..., x_T)$ be an input sequence of length T, and $(y_1, y_2, ..., y_T)$ the desired output sequence. Each element from the output sequence y_t is calculated as a function of the hidden state h_t:

$$y_t = W_{yh} \times h_t \tag{6.5}$$

where h_t is the hidden state of the RNN. The vector h_t is updated as a function of the current input x_t and the previous hidden state h_{t-1}:

$$h_t = \sigma(W_{hx}x_t + W_{hh}h_{t-1} + b_h) \tag{6.6}$$

where σ is an activation function, $\sigma = \tanh$ is used in the vanilla RNN. W_{hx} and W_{hh} are the weight matrices.

Training RNNs involves optimising a loss function representing the gap between the predicted outputs and the targets. The gradient of the loss function is backpropagated in time to learn the temporal dependencies. Standard RNNs are known to have difficulties in processing long sequence inputs. This is because during backpropagation, the derivative of the gradient with respect to previous hidden layers tends to decay or explode exponentially at each time-step, giving rise to the *vanishing/exploding gradient* problem [110]. Advanced versions of RNNs have been proposed in the literature to remedy this problem, notably the Long Short-Term Memory (LSTM) architecture which is introduced next.

6.3.2 LSTMs

Long Short Term Memory (LSTM) has emerged as one of the most popular alternatives for RNNs to solve the *vanishing/exploding gradient* problem. The LSTM architecture is shown in Fig.6.3. The LSTM network controls the flow of information using three trainable gates. All gates receive as input the pair (x_t, h_{t-1}) at time t, and calculate their outputs as follows:

$$i_t = \sigma(U_i x_t + W_i h_{t-1}) \tag{6.7}$$

$$o_t = \sigma(U_o x_t + W_o h_{t-1}) \tag{6.8}$$

$$f_t = \sigma(U_f x_t + W_f h_{t-1}) \tag{6.9}$$

where i_t, o_t and f_t are the input gate, the output gate, and the forget gate respectively; U and W are trainable weight matrices; and σ is the logistic sigmoid function that provides an output between zero and one, where zero implies that the whole input to the sigmoid is discarded while a value of one implies the whole input will pass through. First, the cell state C_t is calculated as follows:

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t \tag{6.10}$$

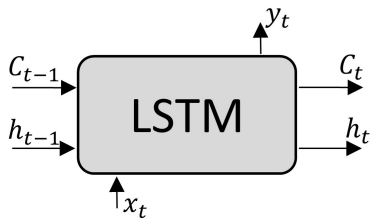


Figure 6.3 – Illustration of an LSTM.

where:

$$\tilde{C}_t = \tanh(U_c x_t + W_c h_{t-1})$$

The input gate determines the information to be stored in the memory cell vector C_t, and the forget gate determines which information from the previous memory cell C_{t-1} is irrelevant to the current prediction and omits it.

The hidden state is calculated using the output gate o_t which controls how much of the memory cell vector C_t is retained:

$$h_t = o_t \odot \tanh(C_t) \tag{6.11}$$

Then, the output y_t is calculated as follows:

$$y_t = \phi(W_y h_t + b_y) \tag{6.12}$$

where ϕ is an activation function; W_y is a trainable weight matrix; b_y is a bias; and $\odot$ is the element-wise product.

6.3.3 Transformers

Transformers are a type of deep learning architectures that were introduced in [111]. They are primarily used for natural language processing tasks, such as text summarisation [112], machine translation [113], and text classification [114], but has also been applied to other domains such as computer vision [115].

Transformers, unlike LSTMs, use attention mechanisms instead of a "memory" component, allowing Transformers to better handle long-range dependencies. Additionally, Transformers can be trained more efficiently on large datasets as they are more easily parallelised than LSTMs. A brief overview of the functioning of Transformers is provided below.

As shown in Fig.6.4 the input is first mapped to a high-dimensional space and a positional encoder is used to preserve the order of the input sequence. The embedded input is then passed in parallel through a stack of attention layers. Each attention layer is composed of a multi-headed self-attention mechanism and a fully connected layer.

Given an input sequence $(x_1, x_2, ..., x_n)$, the multi-head attention module calculates for each current position i in the input sequence a weighted sum of the input elements where each weight determines the relevance to the current position i. This is done by first computing for each element in the sequence a query vector q, a key vector k and a value vector v. The weights α_j for the weighted sum are computed using the dot product between the query vector of the current position and the key vector k_j of each element in the sequence.

$$\alpha_j = softmax(\frac{q_i * k_j}{\sqrt{(d_k)}}) \quad | \quad j \in \{1, 2, ..., n\} \tag{6.13}$$

Where d_k is the dimensionality of the key vector. The output of the self-attention module for each element i is then computed as follows:

$$o_i = \sum_{j=1}^{j=n} \alpha_j \times v_j \tag{6.14}$$

This output is then non-linearly transformed by a fully connected layer. The two outputs from the self-attention mechanism and the feed forward layer are passed through a series of residual connections and a normalisation layer. It is common to use a stack of $N >= 2$ attention layers, where the output of a given layer k forms the input to the next layer $k + 1$, $k \in \{1, ..., N - 1\}$.

6.4 Spatio-Temporal Representation of Demonstrations

The demonstrations provided by an expert are a succession of observations where the state of the objects is manipulated through time to accomplish a specific task. We propose to shape the observations as graphs of connected objects. For instance, if demonstrations are provided in the form of videos, the objects can first be extracted using an object detector [116] and then mapped into a graph of connected objects. We suggest a representation pipeline that can capture both spatial features, including the state of objects

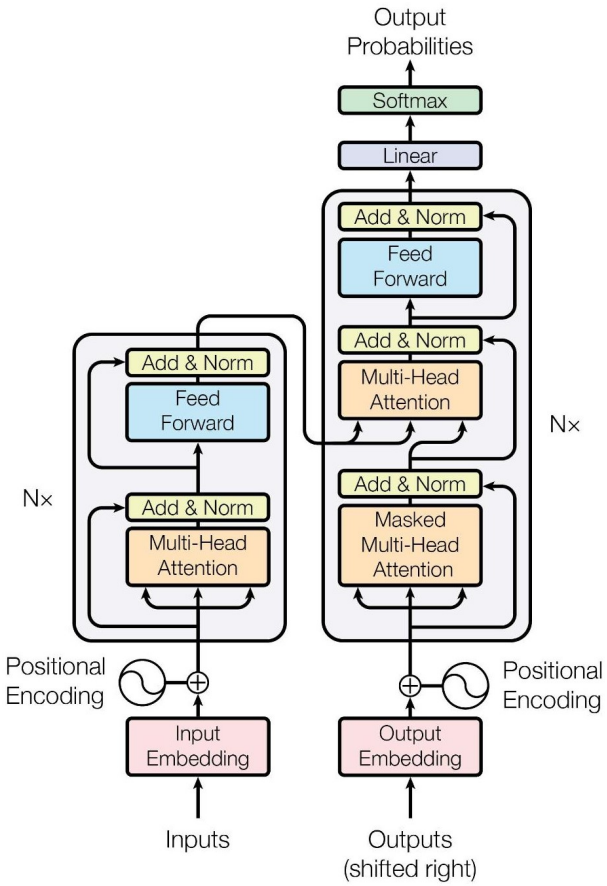


Figure 6.4 – Illustration of a Transformer taken from [111].

and the relations between objects, and temporal features reflecting the evolution of the spatial features over time.

Our representation model depicted in Fig. 6.5, consumes input sequences of observations and attempts to predict a sequence of the next observations. We consider a manipulation task where an end-effector interacts with a set of objects to solve a given task. The end-effector can be the hand of a human operator or a gripper of an industrial robot.

First, each observation o_t from the demonstrations is represented as a graph G_t where each node corresponds to an object in the scene and embeds various features that describe the object such as size, colour, class, etc. Second, each constructed graph is passed through a GNN to learn a graph-level representation h_g^t that we term a spatial embedding for each observation in the input sequence. Finally, the sequence of the spatial embedding vectors is fed to a Seq2Seq model that attempts to learn a temporal representation $\Phi(o_{t-l}, \ldots, o_{t-1})$ that allows for predicting next time-steps.

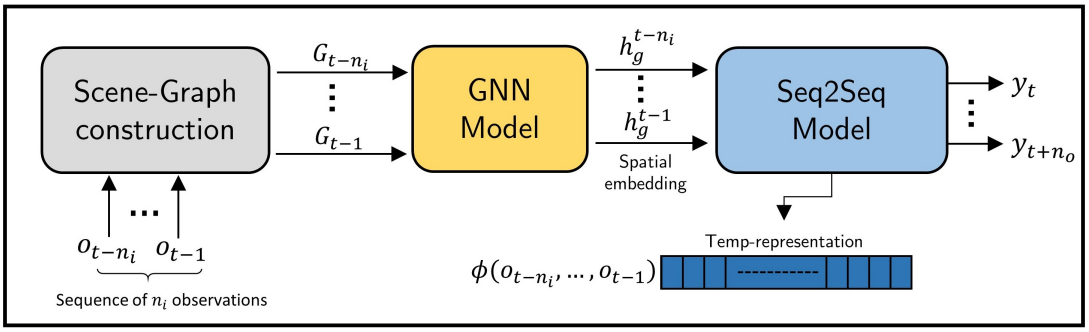


Figure 6.5 – The architecture of our representation model. It is composed primarily of three modules: Scene-Graph Construction module that builds a graph from each observation in the input sequence, the GNN module that processes the built graphs to provide a spatial representation for each graph, and the Seq2Seq module to encode the spatial-embedding-sequences and predict future time-steps.

6.4.1 Scene-Graph Construction

The scene-graph construction module transforms the observation of the environment into an architecture of connected nodes. An intuitive option for designing such a representation for manipulation tasks would be to encapsulate object features in nodes and relations between objects into edges. This approach for representing the scene as a graph has been used in various robotic applications [95, 98]. For a manipulation task that involves an end-effector (e.g. human hand, robotic gripper) manipulating objects, a generic approach to constructing the scene graph is to link the end-effector to the manipulable objects and the objects to targets.

6.4.2 GNN Module

Once the scene graphs are constructed, they are passed through a GNN model that processes them one by one to learn a scene representation h_G following the training mechanism described in Section 6.2.2. We experiment with three types of GNNs: GCN, GraphConv, and SAGEConv, which we briefly introduce below. In the following, h_v^l is the updated embedding of a node v at the layer l; Θ, Θ_1, and Θ_2 are trainable weight matrices; $e_{u,v}$ refers to the edge weight from source node u to target node v; and σ is an activation function such as $ReLu$.

Graph Convolution Network (GCN): GCN [117] layer is defined as follows:

$$h_v^l = \sigma\left(\Theta \times \sum_{u \in N(v) \cup \{v\}} \frac{e_{v,u}}{\sqrt{\hat{d}_v \hat{d}_u}} h_u^{l-1}\right)$$

Where $\hat{d}_v = 1 + \sum_{u \in N(v)} e_{u,v}$ is a normalisation factor.

GraphConv: Unlike GCN, GraphConv [118] layer learns a separate weight matrix for the self-features:

$$h_v^l = \sigma\left(\Theta_1 \times h_v^{l-1} + \Theta_2 \times \sum_{u \in N(v)} e_{u,v}.h_u^{l-1}\right)$$

GraphSage (SAGEConv): SAGEConv [104] updates a node v by taking the mean of the neighbour nodes instead of the sum:

$$h_v^l = \sigma\left(\Theta_1 \times h_v^{l-1} + \Theta_2 \times \underset{u \in N(v)}{mean} h_u^{l-1}\right)$$

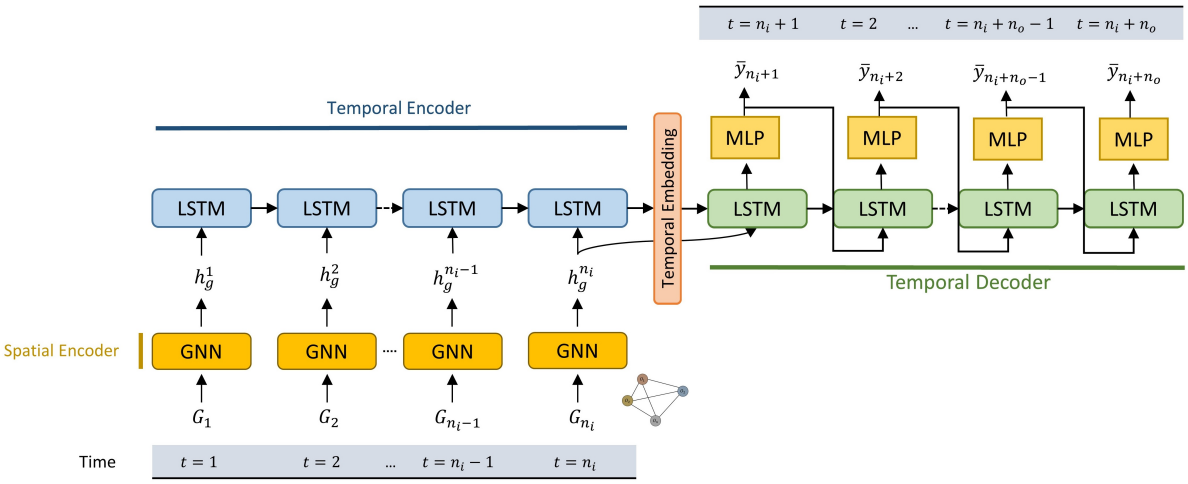
We employ PyTorch Geometric [119] for all implementations of GNN layers.

6.4.3 Seq2Seq Module

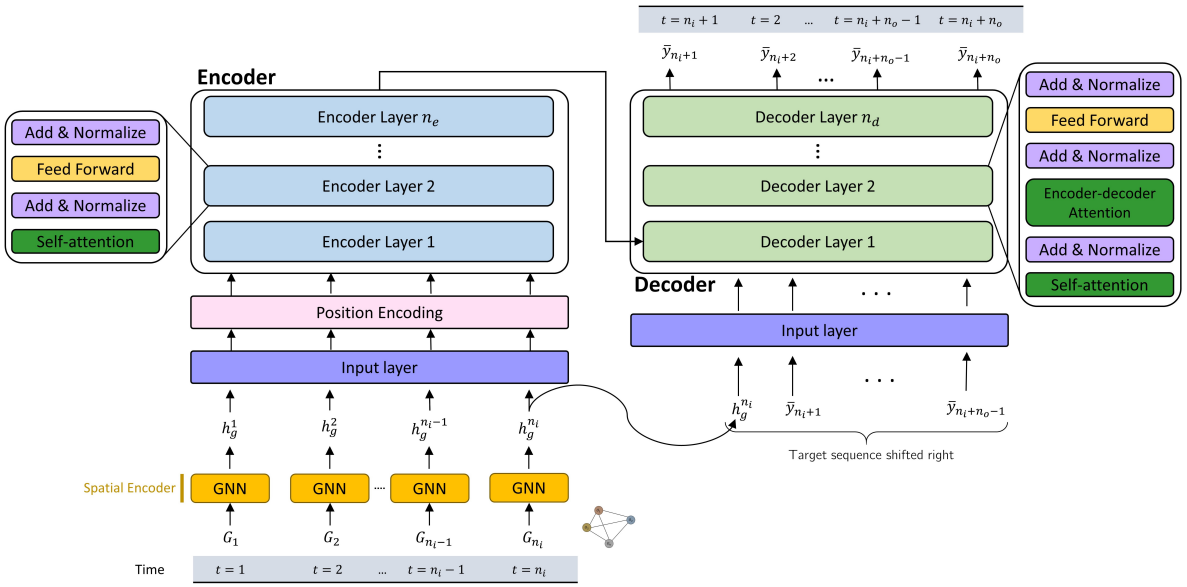
We use an Encoder-Decoder architecture to process sequences of time-steps. The sequence of scene graph embeddings is fed to the encoder that learns a low dimensional vector which summarises the temporal patterns in the input sequence. The temporal representation of the input sequence is then passed to a decoder to predict the next time-steps. We experiment with both LSTMs, introduced in Section 6.3.2 (shown in Fig. 6.6-a), and Transformers, introduced in Section 6.3.3 (Fig. 6.6-b).

6.5 Experimental Setup

Environment: The task to be solved is for the robot to push an object to a target destination (see Fig 6.7-a). The robot gets as input the 2D coordinates of the object, target, and the end-effector. At each time-step, the policy outputs two values dx and dy, the two displacements to make along the Cartesian axes x and y.

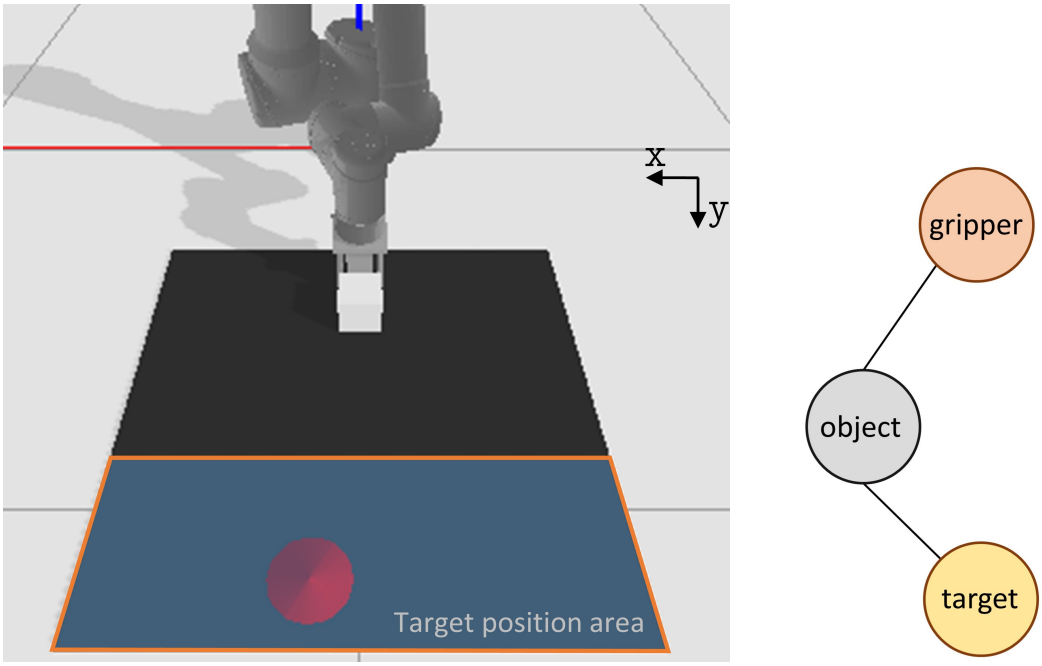


(a) GNN-LSTM



(b) GNN-Transformer

Figure 6.6 – GNN-Seq2Seq architecture using LSTMs and Transformers. The Seq2Seq model takes as input the computed embedding of the input sequence and predicts a future target sequence.



(a) Snapshot of the pushing task in simulation (b) Scene graph

Figure 6.7 – Snapshot of the pushing task in simulation and the corresponding scene graph

Scene-Graph design: We encode the scene observation into a graph, in which the nodes represent the objects in the scene and the gripper of the robot (see Fig. 6.7-b). To establish connections between objects, we follow the approach highlighted in section 6.4.1. We link the end-effector to the manipulable object and the object to the target. The node features consist of the 2D position of the object and a one-hot encoding vector to identify each object in the scene.

6.6 Experiments and Results

We conduct a set of experiments to provide analysis of how several design choices affect the performance of the representation model. We evaluate the accuracy of predicting the augmented observation y_{t+1} from the previous sequence of observations $(o_{t-l+1}, \ldots, o_t)$ of length l. y_{t+1} is a concatenation of the observation o_{t+1} and a vector enclosing distances between the connected objects in the scene graph. The additional distances are intended to encourage the GNN model to also encode the relationships between the connected nodes.

Description	**Detail**
Graph design	
Nodes	Each node represents an object
Edges	Drawn according to Section 6.5
GNN Hyperparameters	
GNN-Type	GCN
Hidden channels size	16
Number of layers	1
Aggregation function	Average pooling
Dropout	20%
Seq2Seq Hyperparameters	
Autoencoder	LSTM
Input sequence size n_i	3
Output sequence size n_o	1
Hidden state	8
General Hyperparameters	
Learning rate	1e-3
Loss function	RMSE

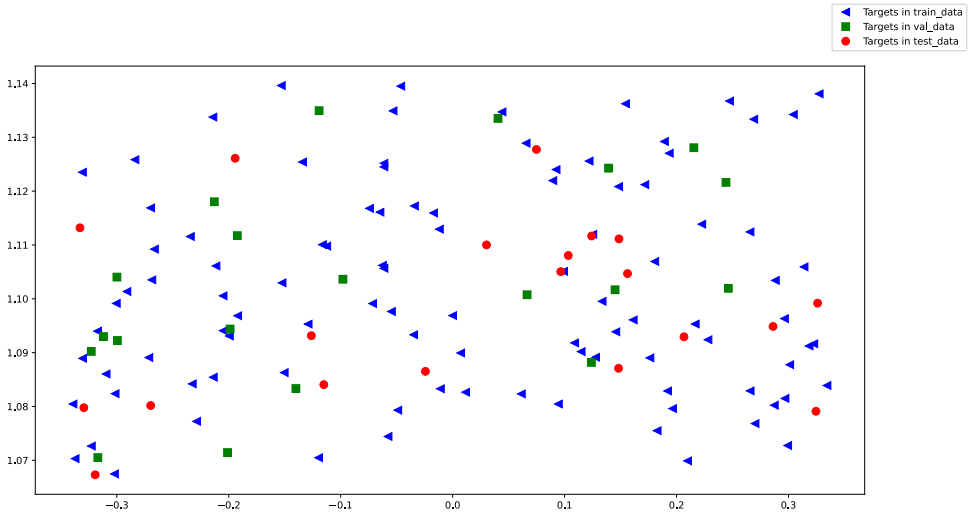
Table 6.1 – The representation model training hyperparameters.

6.6.1 Generating Demonstrations

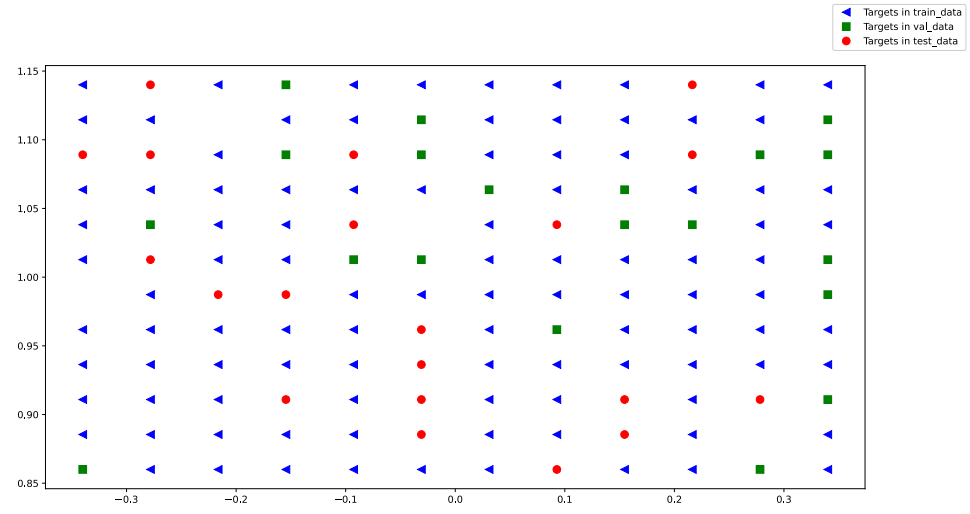
We generate $N = 140$ trajectories of successful executions for solving the task using an artificial RL-expert. A trajectory is considered successful if the object is pushed all the way to the target. The expert was trained using the SAC algorithm [10] and a shaped reward driven by the distance between the object and the target. Demonstrations are collected by executing the expert policy in the environment while recording the observation and the associated action for each time-step. Our method does not require access to actions, but some of the baselines to which we compare our method do.

In all the following experiments, the scene graph is constructed as described in the previous section and all models are trained to predict one time-step ahead. We use 100 trajectories of demonstrations (2362 observations) for training, 20 trajectories (478 observations) for validation, and 20 trajectories (473 observations) for testing. We train our model end-to-end, and we measure performance by calculating the Root Mean Squared Error (RMSE) between the predicted and the ground-truth value on the test dataset.

We consider two ways of generating demonstrations: in one we randomise the target position using a uniform distribution, while in the other, the target position area (see Fig. 6.7-a) is first discretised into a grid of evenly distributed positions; then, each time



(a) Random generation of targets



(b) Grid-based generation of targets

Figure 6.8 – Generation of target positions using two methods: (a) Targets are randomly generated using a uniform distribution, and (b) Target positions are chosen from a predefined grid.

the expert policy is invoked to generate a demonstration, one position from the finite generated positions is selected. In both scenarios, the position of both the gripper and the object is kept fixed. Fig 6.8 shows the target distribution obtained for the training, validation and test datasets using both methods.

We train two identical GNN-LSTM representation models where the details are shown in Table 6.1. Two different sets of demonstrations are used, produced using the two aforementioned methods for generating targets. After the training is completed, we compute the RMSE on the test dataset. We also visualise in Fig. 6.9 four original trajectories from the test dataset against their prediction using the obtained representation models.

The model that was trained with equitably distributed targets outperformed the model that was trained using randomly generated demonstrations with an RMSE=0.014 against an RMSE=0.066. This considerable margin of improvement is reflected in predicted trajectories that are closer to the original trajectories as can be seen in Fig. 6.9.

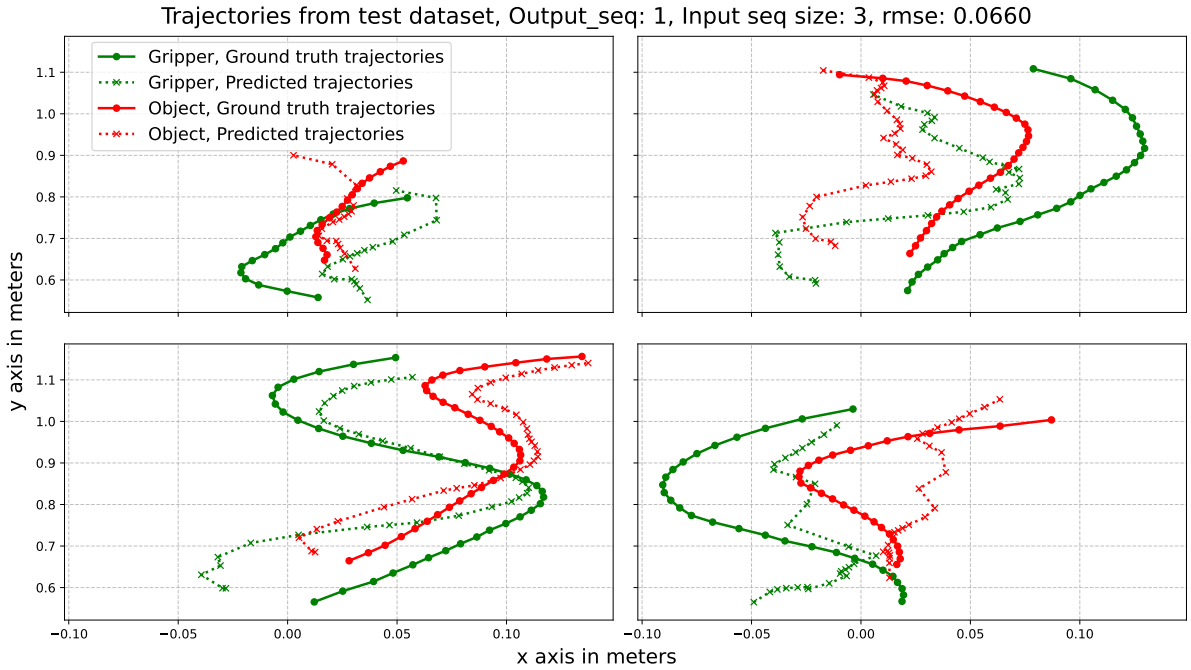
This shows that it is essential to carefully choose the trajectories of the demonstrations to be used for training the representation model in order to achieve optimal performance. Random generation of demonstrations can lead to data redundancy and underrepresentation of the state space. This is clearly illustrated in Fig.6.8-a, where there is an overlap between the generated points in the same data split as well as between the three splits. In what follows, we exclusively use grid-based generated demonstrations.

6.6.2 GNN module ablation

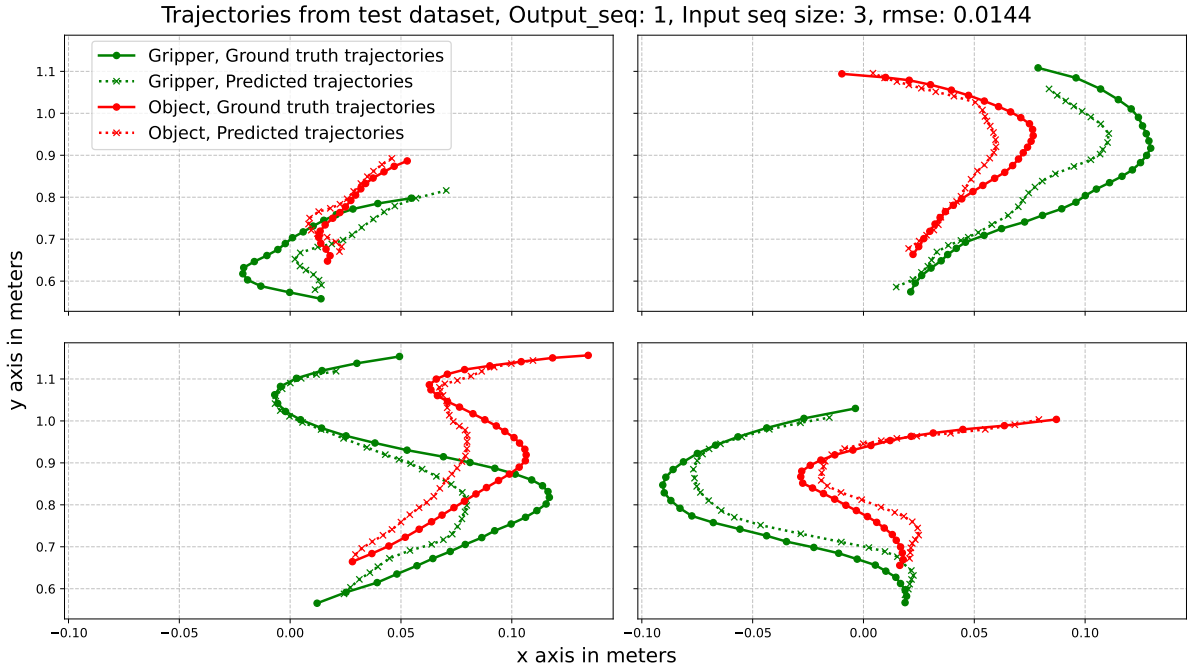
We first investigate the ability of the GNN module to learn a spatial representation of the scene observation that can provide the Seq2Seq module useful features for predicting the next time-step. We use the architecture shown in Fig. 6.6-a. The LSTM seq2seq model hyperparameters are held constant with an input sequence size of three consecutive observations. We focus on the effect of varying GNN' number of layers and hidden channels on the performance of the trained representation. We experiment with the three variants of GNNs: GCN, GraphConv, and SAGEConv, previously introduced in section 6.4.2.

By varying the number of GNN layers, we evaluate how deep we have to reach into the graph to learn an efficient embedding for each node while varying the number of the GNN hidden channels evaluates the information capacity needed for each node embedding to adequately predict y_{t+1}.

The results are shown in Fig 6.10-a. GraphConv and SAGEConv outperformed GCN with the best RMSE values of 0.0057 and 0.0048 respectively. This can be explained by

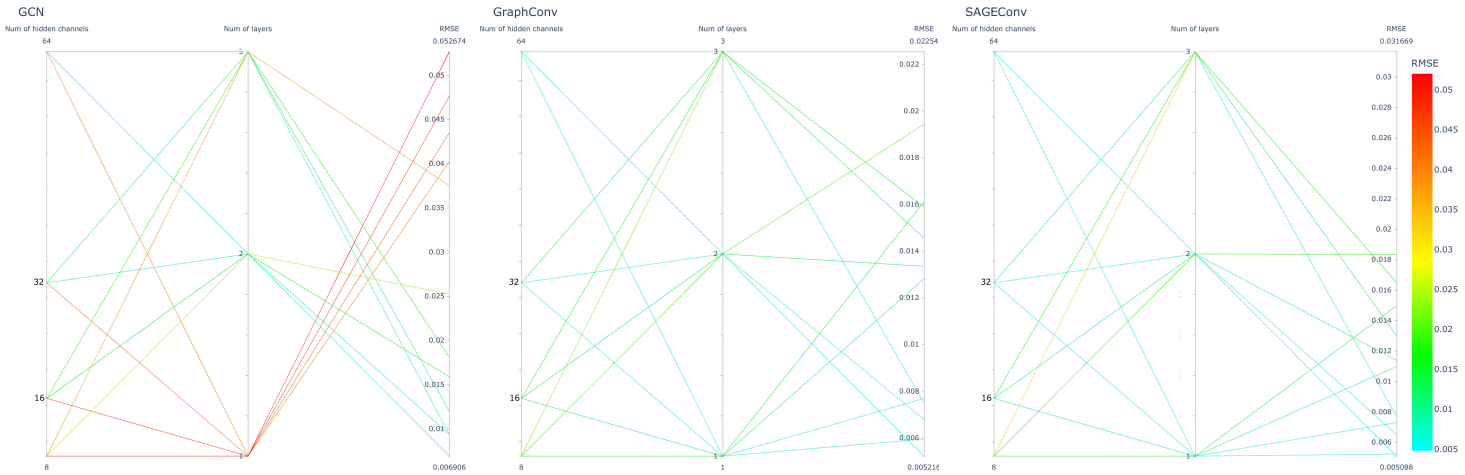


(a) Results for the representation model trained on demonstrations using randomly generated targets

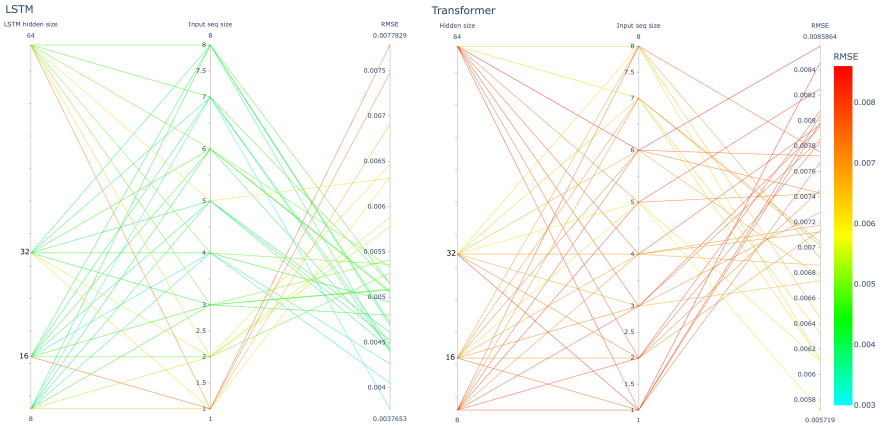


(b) Results for the representation model trained on demonstrations using grid-generated targets

Figure 6.9 – A plot of original trajectories vs. predicted trajectories of the robot gripper position (represented in red) and the object position (represented in green). The solid lines represent trajectories from original demonstrations, and the dotted lines represent predicted trajectories using the trained representation model.



(a) GNN module ablation



(b) Seq2Seq module ablation

Figure 6.10 – Ablation study results plotted on the parallel coordinates chart. The lines connecting data points are drawn by joining the values of each variable represented by the parallel axes. On the right side of each plot is the *RMSE* calculated on the test dataset and the other axes present the hyperparameters for each variant of the representation model. An enlarged view of each parallel coordinates chart is given in Appendix A

the efficiency of the update functions of GraphConv and SAGEConv which have more flexibility in combining the node features and the neighbouring nodes as opposed to GCN which learns one weight matrix for both. For the three types of GNN the higher the number of hidden channels the better the performance. The best RMSE values were obtained using a number of layers equal to two. Two is also the diameter [1] of our graphs, which means that it allows reaching all nodes in the graph from a given node. It can also be seen that using more layers than the diameter of the graph does not improve performance, which is consistent with the findings in [94] and with the results in Table 4 in [120]. Using a single layer can considerably degrade performance, which is more evident with the GCN layer.

6.6.3 Seq2Seq module ablation

We set the GNN model to the optimal configuration from the previous experiments and we train the representation model using two different architectures, GNN-LSTM and GNN-Transformer shown in Fig. 6.6-a and Fig. 6.6-b respectively. In addition to the hyperparameters that are specific to each seq2seq model, we experiment with different values of the input sequence size $l \in \{1, 2, ..., 8\}$. This will provide an indication of how many time-steps to retain from the past in order to reliably predict the next time-step.

For the GNN-LSTM, we train different models by exhausting every combination of the input sequence size values and the LSTM hidden size $\in \{8, 16, 32, 64\}$. The latter represents the size of the memory that the seq2seq model keeps over the previously processed time-steps. For the GNN-Transformer, we train a model for each combination of input sequence size values and the following set of parameter values: Transformer hidden size $\in \{8, 16, 32, 64\}$, and the number of attention heads $\in \{1, 2, 4\}$. For the sake of clarity, we plot only for a number of heads of four that achieved the best performance.

The results are shown in Fig 6.10-b. For both architectures, we observe that the more time-steps we include in the input sequence the better the performance and the representation performance is not remarkably sensitive to the variation of the hidden size in both the LSTM and the Transformer. Using the Transformer the performance is poorer compared to LSTMs. We believe that this is tied to the high complexity of the Transformer model. In our architecture, the number of trainable parameters in the Transformer encoder-decoder is 280136 vs. 11016 trainable parameters for the LSTM encoder-decoder. Since the demonstration data is usually limited in imitation learning, it

1. The diameter of a graph is the shortest path between the most distanced pair of nodes in the graph.

Table 6.2 – Results summary. Optimal values are depicted in bold.

	RMSE	MAE	Max-error
GNN-LSTM	0.0048	0.00348	0.02454
GNN-Transformer	0.0065	0.0048	0.02936
LSTM-only	**0.00166**	**0.00127**	**0.00395**
Transformer-only	0.00174	**0.00127**	0.00919

is most desirable and practical to train simpler architectures with as few parameters as possible.

We note that the task of predicting the subsequent time-step based on the preceding sequence of observations can be performed in numerous ways and GNNs are not necessarily required. We also trained models using Seq2Seq-only architectures. We feed directly the sequence of observations to the Seq2Seq model and attempt to predict the next augmented observation. The training is done similarly to the previous experiments using the optimal parameters found in the last results. Table 6.2 summarises the best-obtained results for each architecture variant for an input sequence length $l = 3$. We can see that LSTM-only and Transformer-only outperform the GNN variants. This implies that the GNN model can be further improved, either by modifying the way the scene graph is constructed or by using more complex GNN layers such as [105]. Although Seq2Seq-only models might be the most efficient, they are limited to fixed-sized vector inputs, which requires knowing and setting the number of objects in advance. GNNs on the other hand provide the benefit of handling a variable number of objects [121].

6.7 Summary

This chapter outlines a novel representation pipeline for modelling expert demonstrations. A general framework is proposed to learn a representation model that simultaneously captures the spatial and temporal features of demonstration trajectories. Observations are modelled as scene graphs of connected objects and processed using GNNs to derive spatial patterns while the temporal patterns are tracked using Sequence-to-Sequence modelling. The representation pipeline is trained to predict future time-steps based on previous observations. In the next chapter, we present an Imitation Learning from Observation framework, which exploits the predictive model introduced in this chapter to a

RL-based learning setup.

7

RL-based Framework for Learning from Observation

Contents

7.1 Introduction

The imitation policy can be either trained directly from demonstrations or involve the agent to learn by trial and error. Direct imitation learning is advantageous on two counts: firstly, it permits offline training, and secondly, it requires no interaction between the agent and the environment to learn the imitation policy. Nevertheless, relying solely on demonstrations is insufficient to train robust policies [122]. The typical difficulty associated with direct imitation is that the training dataset is not entirely representative of the demonstrated task. The gathered demonstrations only cover a selection of possible

situations that the imitator might encounter during the deployment phase. The agent is prone to divert from the optimal path and land in a state that is neither recognised nor represented by the imitation policy. The agent may drift from the desired path either by compounding errors along a trajectory or as a consequence of external factors such as a change in the environment dynamics. This raises a major generalisation challenge and motivates learning from experience.

Learning from experience provides the agent with the ability to explore the environment and learn about what to do in unseen situations. Reinforcement Learning is a common choice for learning behavioural skills by trial and error. The task to be solved must be modelled as a Markov Decision Process and a reward function needs to be designed. Engineering a task-specific reward function is one of the major challenges faced by RL users. The reward function can be either shaped (a.k.a. dense reward) or sparse. A shaped reward involves providing the agent with intermediate rewards that guide it throughout the exploration of the environment towards the goal of the task [123]. Although shaping rewards has been shown to be effective in assisting the training to converge to optimal policies for solving various tasks, it is nonetheless difficult to design and requires careful engineering by an expert as it can lead to sub-optimal and undesirable behaviours if it is not adjusted properly [48]. A more straightforward alternative is to use a sparse reward that provides a recompense only for the last action leading to achieving the goal state and solving the task. The shortcoming of this solution is that it is quite difficult to converge to the optimal policy if the goal state is not visited during the exploration of the environment. This is much more liable to be experienced if the state space is continuous and of high dimension [68].

By combining demonstrations and experience, the shortcomings of each approach could be alleviated. While demonstrations provide guidance to the imitator to converge on the expert's behaviour, learning by trial and error gives the agent more flexibility on the way the task is performed while exploring a great deal of the environment and optimising the objective function.

This chapter presents a modular framework for learning from observation. To learn the imitation policy, demonstrations are leveraged through the representation model introduced and discussed in the previous chapter. Reinforcement Learning is employed to allow the agent to generalise to a wider area of the environment state space. The reward function is task-agnostic and drives the agent to a policy that is close to that of the expert.

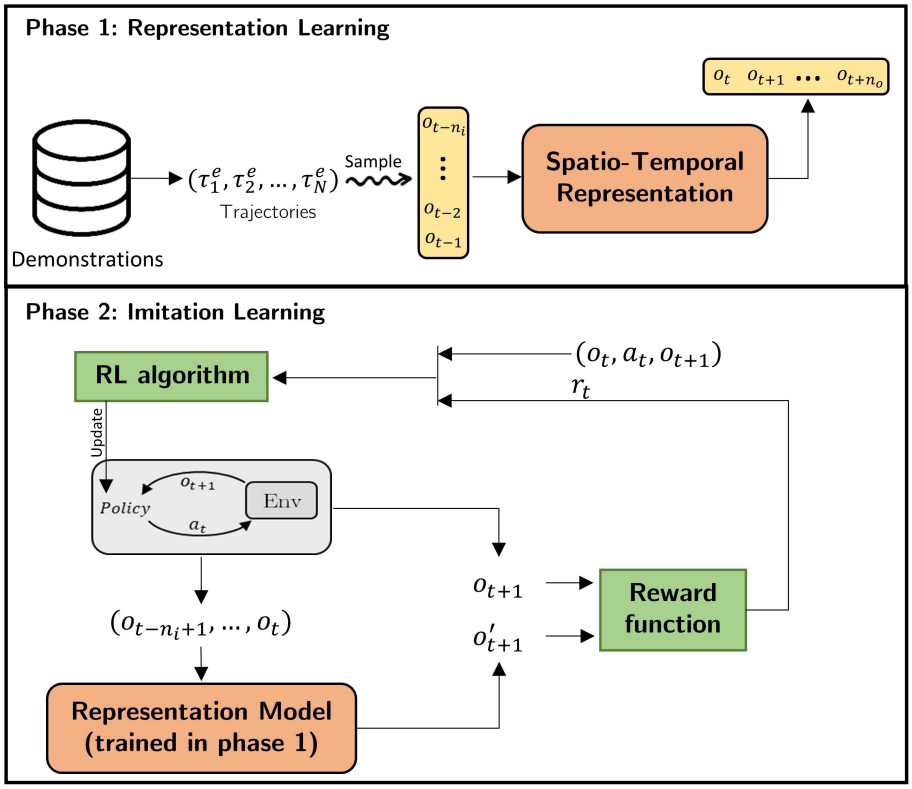


Figure 7.1 – Our framework for solving ILO overview. It consists of two phases: (1) A spatio-temporal representation is trained to capture both spatial features for each observation and temporal aspects of sequences of observations in the demonstrations dataset. (2) Train the imitation policy using RL with a task-independent reward function that builds on the representation trained in phase 1.

7.2 Proposed Method

We propose a flexible and general framework to teach robots to imitate manipulation tasks demonstrated by an expert. The architecture of our method is depicted in Fig. 7.1. It is mainly composed of two phases that can be optimised and refined separately: The first phase consists in learning a representation that embeds the essential aspects of the observed demonstrations (detailed in the previous chapter), and the second phase uses an out-of-the-shelf RL algorithm with a predefined task-agnostic reward function to learn the imitation policy.

Our goal is to lead the imitator to solve the object manipulation task demonstrated by the expert. We accomplish this by using RL jointly with the representation model introduced in the preceding chapter. The representation model guides the policy search

towards a behaviour whose representation matches the expert as closely as possible. To this end, we design a task-independent reward function that we term *imitation reward function* and define in the following section.

7.3 Task-agnostic Imitation Reward Function

The imitation reward function is defined in Eq 7.1 and illustrated in Fig. 7.2. It measures the gap between the state predicted by the representation model given the history of the last visited states and the actual state reached by the agent.

$$
\begin{aligned}
R\left(s_t\right) &= \begin{cases} 0 & \text{if } t \in [0, l-1] \\ R\left(s_t \mid s_{t-1}, \ldots, s_{t-l}\right) = f(d) & \text{if } t \geq l \end{cases} \\
d &= \left\| s_t - s'_t \right\|_2 \\
f(d) &= \begin{cases} 0 & \text{if } d \geq \varepsilon \\ r_{\max} \times \left(1 - \frac{1}{\varepsilon} d\right) & \text{if } d < \varepsilon \end{cases}
\end{aligned}
\tag{7.1}
$$

More concretely, consider a representation model that has been trained on the expert data to predict the future time-step using the last l observations. While learning the imitation policy, the agent starts the episode by executing actions of the current policy and receives no reward until it piles up l observations. Then the representation model is invoked and applied to the input sequence to predict the next time-step. The predicted observation is compared to the actual observation of the agent after executing the action for the given state. If the distance $d = \left\| s_t - s'_t \right\|_2$ between the actual and the predicted observation is greater than ε, the agent is unrewarded. Otherwise (if $d < \varepsilon$), it receives a positive reward to induce the agent to reproduce actions that are predictable by the representation model. The maximum reward is r_{max} which can be obtained if the imitator performs an action that leads to a state coinciding with the forecasted one. ε and r_{max} are hyperparameters in our framework that we set empirically. We have opted for the reward to be linearly proportional to the distance between the predicted state and the actual state, but one can experiment with any decreasing function defined in the interval $[0, \varepsilon]$.

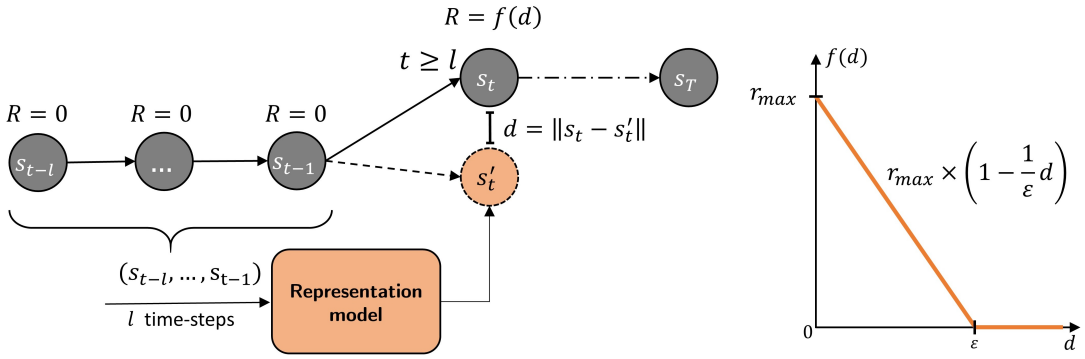


Figure 7.2 – Illustration of the imitation reward. The representation model commences predicting the future time step once the required l observations have been stacked and the agent begins to receive rewards according to its proximity to the predicted observation. Prior to this, the agent receives neutral rewards of zero. The reward is calculated as a function f of the distance $d = \|s_t - s'_t\|_2$ between the actual state s_t attained by the imitator and the state predicted by the representation model s'_t. We also define a threshold ε from which the agent starts receiving positive rewards.

7.4 Experiments and Results

7.4.1 Experimental setup

We use the set of demonstrations generated in Chapter 6, Section 6.6. We adopt the same setup as the expert. The robot learns the pushing task by trial and error. At every time-step, the policy decides which move to make along the x and y axes with a maximal displacement of $5cm$ in each direction. The maximum episode length is set to 85 time-steps and it is terminated in the following cases: if the robot exits the working space, if the object is pushed off the table, or if the target state is achieved.

7.4.2 Training the imitation policy

Any off-the-shelf RL algorithm can be used to optimise the imitation reward defined in Section 7.3 to learn the imitation policy. We use Soft Actor-Critic (SAC)—a state-of-the-art RL algorithm introduced in Chapter 2. We utilise three trained GNN-Seq2Seq models from the experiments conducted in Chapter 6, Section 6.6 to compute the reward function as described in the last section. The training is performed for 5M time-steps.

For the same input sequence size $l = 3$, we experiment with two representation models, one being the optimal model and the other being the poorest model obtained in the rep-

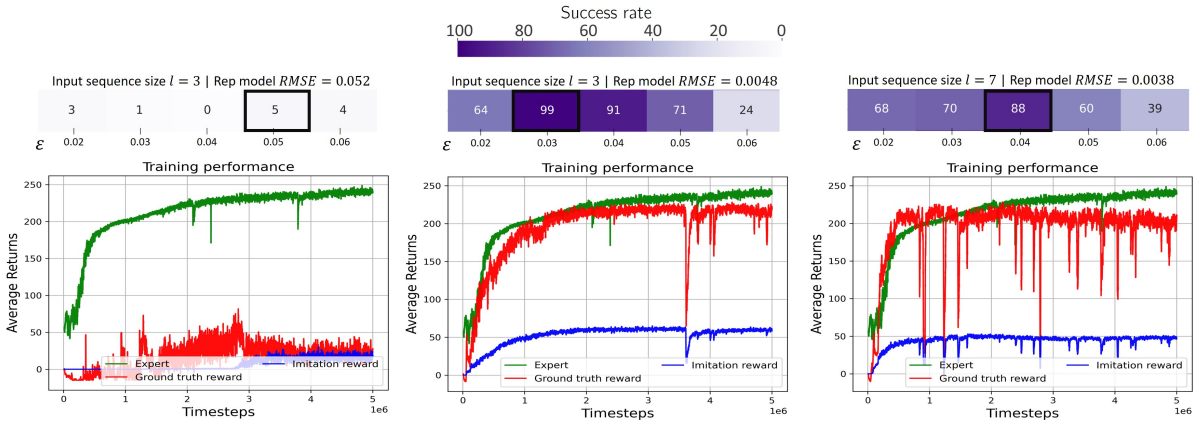


Figure 7.3 – Imitation learning results. For each of the three representation models, we show at the top the success rate obtained for each value of ε and plot the training curves for the highest success rate.

resentation learning experiments with $RMSE = 0.0048$ and $RMSE = 0.052$ respectively. We also experiment with a representation model trained on a larger sequence of $l = 7$ that obtained an $RMSE = 0.0038$. For every representation model, we train different imitation policies using a set of ε values $\in \{0.02, 0.03, 0.04, 0.05, 0.06\}$. ε is an important parameter in our framework as it denotes the threshold at which the agent begins to receive a non-zero reward.

The results are shown in Fig. 7.3. For each representation model, we plot the training performance curve that corresponds to the highest success rate. It can be seen that using an under-performing representation model does not allow the robot to gain any expertise from the expert demonstrations. Good results are obtained for the other two representation models, and the performance varies with the value of ε. The best success rate scores were achieved for intermediate values of $\varepsilon \in \{0.03, 0.04\}$. Low values of ε cause the reward to be sparse and the agent may never discover regions yielding non-zero rewards. Setting ε to a high value makes the policy prone to the reward bias issue: Once the policy begins generating actions leading to values of d inferior to ε, the policy is susceptible to divert its attention from approaching a behaviour recognisable by the representation model and simply settle for accumulating small rewards. In this case, the agent attempts to stay the longest possible time in the scene and exhaust all available time-steps. This can be observed from the mean episode length which we found to be quite close to the maximum time-steps allowed for the robot to solve the task. Increasing the size of the input sequence results in faster convergence but the drop in performance occurs multiple times and the

final performance is slightly worse than when using only a sequence of size three.

The training performance curves show that the imitation reward function is positively correlated with the expert's optimised ground-truth reward and increases to converge in a monotonic pattern. This suggests that solving the imitation problem in our framework corresponds to optimising the objective function of the expert. The correlation between the imitation reward and the expert reward has the advantage of allowing the user to save the last policy with the highest imitation reward, knowing that it is the optimal policy for maximising the expert's objective function for a given training configuration. This cannot be done, for instance, with adversarial methods since the discriminator reward function is non-stationary and oscillates around a saddle point [64].

7.4.3 The impact of the ε threshold

Here, we study the impact of varying the threshold ε on the training performance. We fix the maximum number of time-steps that the agent has to solve the task in each episode to 150 and experiment with different values of ε. The threshold ε is a hyperparameter in our framework that determines the proximity to the demonstrations from which the agent starts receiving a positive reward. The results are displayed in Fig 7.4. The training performance depends on the value of ε. For very high or very low values, the performance is poor, even if the imitation reward is converging. Low values of ε lead to a scarcity of reward returns and make it difficult for the agent to obtain positive rewards to be guided towards the optimal policy. On the other hand, the higher the value of ε, the more likely it is to induce the survival bias that we detail below.

During training, the imitation policy is optimised using RL and the reward function defined in Eq 7.1. The cumulative reward for a trajectory of length T is as follows (we omit the discounting factor γ for simplicity):

$$\begin{aligned} &\sum_{t=0}^{t=T} R_t = \sum_{t=0}^{t=l-1} R_t + \sum_{t=l}^{t=T} R_t \quad \text{with} \sum_{t=0}^{t=l-1} R_t = 0 \\ \implies &\sum_{t=0}^{t=T} R_t = \sum_{t=l}^{t=T} R_t \end{aligned} \tag{7.2}$$

For simplicity, let us consider that $d_t = \|s_t - s'_t\|_2 = \varepsilon_1$ for $t \in \{l, ..., T\}$, where

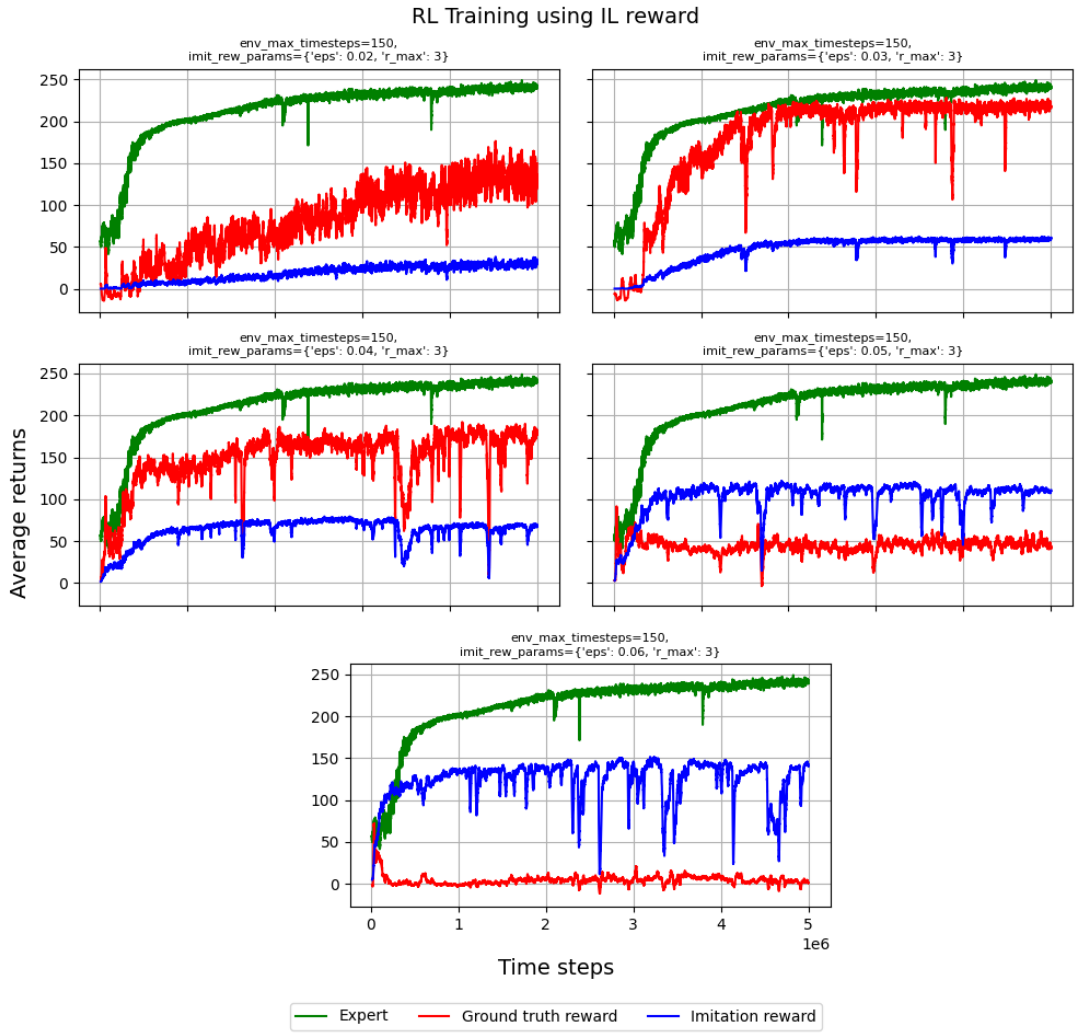


Figure 7.4 – Imitation Learning performance using the imitation reward function defined in Eq 7.1 with different values of ε.

$f(\varepsilon_1) = 1$. Hence, the total reward is:

$$R_{\varepsilon_1} = \sum_{t=0}^{t=T} R_t = \sum_{t=l}^{t=T} f(d_t) = T - l + 1 \tag{7.3}$$

If the task is not solved yet, the agent will consume all the available time-steps in an episode interacting with the environment (i.e. $T = T_{max}$):

$$R_{\varepsilon_1} = T_{max} - l + 1 \tag{7.4}$$

Now consider the case where the agent follows a trajectory with a representation closer to the representation followed by the expert. Let's consider that $d_t = \varepsilon_{1+\alpha}$ for $t \in \{l, ..., T\}$, where $f(\varepsilon_{1+\alpha}) = 1 + \alpha$. The total reward of the trajectory is:

$$R_{\varepsilon_{1+\alpha}} = \sum_{t=0}^{t=T} R_t = \sum_{t=l}^{t=T} f(d_t) = (T - l + 1) \times (1 + \alpha) \tag{7.5}$$

For the agent to prefer this trajectory to the former, $R_{\varepsilon_{1+\alpha}}$ should be greater than R_{ε_1}, which implies the following:

$$\begin{aligned} (T - l + 1) \times (1 + \alpha) &> T_{max} - l + 1 \\ \implies T &> \frac{T_{max} + \alpha(l - 1)}{1 + \alpha} \end{aligned} \tag{7.6}$$

In our experiments we used $l = 3$ and $T_{max} = 150$:

$$\implies T > \frac{150 - 2\alpha}{1 + \alpha} \tag{7.7}$$

For an improvement of $\alpha = 0.05$:

$$\implies T \geq 143 \tag{7.8}$$

This implies that all trajectories of a length less than 143 are going to be regarded as less favourable, even if a higher reward $(1 + \alpha)$ is given for every time-step of the path and with a representation closer to the expert's ($\varepsilon_{1+\alpha} < \varepsilon_1$).

7.4.4 Comparison to baselines

We evaluate the performance of our method against the following baselines: GAIL [64], BC [56], GAILfO [74], and GAILfO-s [124]. We note that GAIL and BC require access to actions while GAILfO, GAILfO-s, and our method make use solely of observations to learn the imitation policy.

GAIL, GAILfO, and GAILfO-s are trained using the discriminator reward function defined in Eq. 7.9, Eq. 7.10, and Eq. 7.11 respectively.

$$R(s, a) = \log D_\theta(s, a) - \log(1 - D_\theta(s, a)) \tag{7.9}$$

$$R(s, s') = \log D_\theta(s, s') - \log(1 - D_\theta(s, s')) \tag{7.10}$$

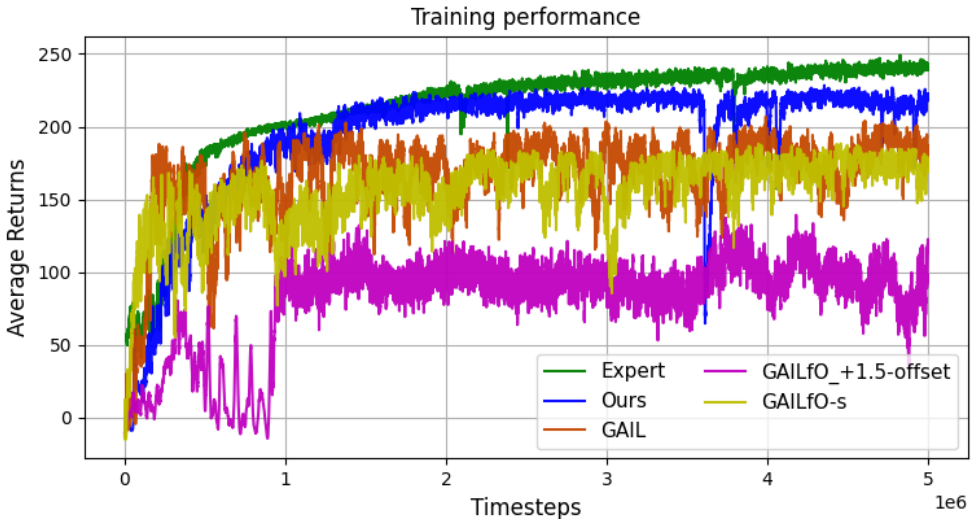


Figure 7.5 – Learning curves for our approach and three other baselines: GAIL, GAILfO, and GAILfO-s, compared to the expert. All curves refer to the average returns of the ground-truth reward that is optimised by the expert.

$$R(s) = \log D_\theta(s) - \log(1 - D_\theta(s)) \tag{7.11}$$

While the use of this reward proved effective for training GAIL and GAILfO-s, it was not the case for GAILfO, which suffered from the reward bias problem [89] that led the agent to converge on the behaviour of exiting the environment at the soonest opportunity to avoid receiving negative rewards. The solution introduced in Chapter 5, Section 5.4.4, involves adding an offset to the reward function, which effectively mitigates the problem and leads to policy convergence.

The training curves are depicted in Fig. 7.5, and Table 7.1 reports the final performance. Our approach surpasses all other methods by a considerable margin, converging close to the expert's asymptote. The final performance of our method is comparable to GAIL's with a success rate of 99%, outperforming all other methods.

	GAIL	GAILfO	GAILfO-s	BC	Ours
Success rate	99%	75%	96%	93%	99%

Table 7.1 – The final performance of different IL methods. The success rate is calculated over 100 rollouts in simulation.

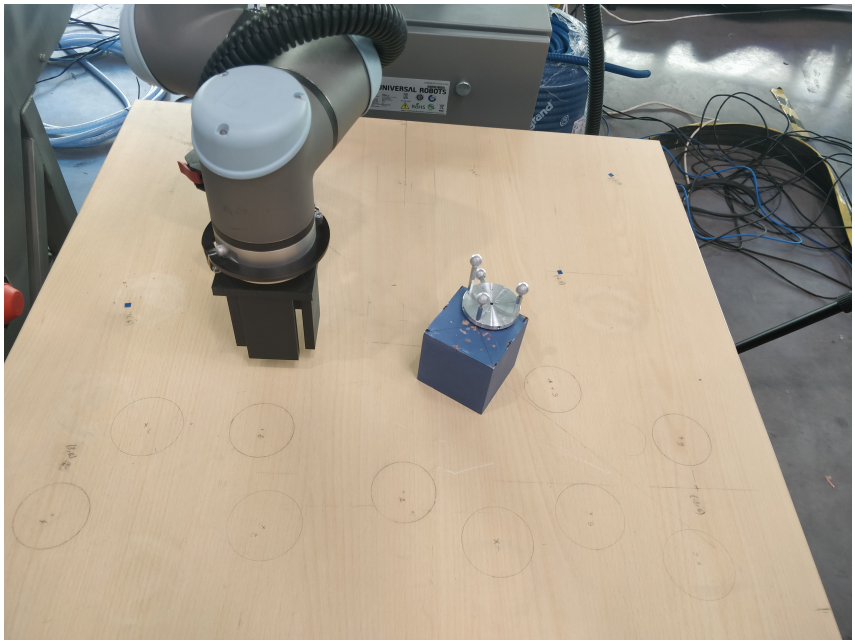


Figure 7.6 – The real-world setup for the pushing task. The robotic arm (UR10e) is required to push the blue object to a target position.

	GAIL	GAILfO	GAILfO-s	BC	Ours	Expert
Success rate	6/10	1/10	6/10	8/10	10/10	10/10

Table 7.2 – Sim-to-Real results. We report the success rate of solving the pushing task for 10 different targets.

7.4.5 Deployment on a real robot

We run the obtained policies on a real robot without fine-tuning or additional training. We utilise a Motion Capture System (MoCap) to locate the object in the scene. A retro-reflective target is installed on the top of the object (see Fig. 7.6) and six motion infrared tracking cameras covering the scene are used to compute the object's position using DTrack2 software from ART [45].

The results are reported in Table 7.2. Our method outperforms all other methods and succeeds in reaching 10 out of 10 goals. GAIL and GAILfO-s, which achieved 99% and 96% respectively in simulation, scored only a 6/10 success rate on the real robot. The BC method surprisingly performed better than adversarial methods on the real robot, given that the BC method is known to suffer from the covariate shift problem [56].

7.5 Adaptive Imitation Reward Function

The imitation reward defined in Eq 7.1 is sensitive to the parameter ε. The findings in Section 7.4.3 indicate a correlation between the selection of ε and the imitation training performance.

On one hand, the higher ε, the more margin the agent has to move away from trajectories whose representation is similar to the expert's while still obtaining a positive reward. This results in learning a policy not close enough to the expert's behaviour. On the other hand, low values of ε make the reward feedback very sparse, which makes the demonstrated task difficult and time-consuming to solve. The difficulty of convergence varies with the complexity of the task and the scale of the state and action spaces.

A hyperparameter search is necessary to locate the optimal value for ε, which lies between the two extremes stated above. In the following, we address this limitation and propose a strategy that adaptively changes the reward function throughout the training process.

7.5.1 Adaptive Reward Definition

We propose the adaptive reward function outlined in Eq. 7.12. It modifies the imitation reward function defined in Eq. 7.1 to adaptively change the value of ε as shown in Fig 7.7. The idea is to start with an initial value of ε and gradually decrease it during the learning process as the imitator gains experience. This would allow the imitator to be guided smoothly towards the expert policy.

$$
\begin{aligned}
R\left(s_t\right) &= \begin{cases} 0 & \text{if } t \in [0, l-1] \\ R\left(s_t \mid s_{t-1}, \ldots, s_{t-l}\right) = f(d) & \text{if } t \geq l \end{cases} \\
d &= \left\| s_t - s'_t \right\|_2 \\
f(d) &= \begin{cases} 0 & \text{if } d \geq \varepsilon \\ r_{\max} \times \left(1 - \frac{1}{\varepsilon_{max}} d\right) & \text{if } d < \varepsilon \end{cases} \\
\varepsilon &\in [\varepsilon_{min},\ \varepsilon_{max}]
\end{aligned}
\tag{7.12}
$$

We propose two versions of the adaptive reward function, V2 defined in Algo 3, and V3 defined in Algo 4. The next section presents a detailed overview of the two rewards, along with the results of experimenting with different hyperparameters.

Algorithm 3: Adaptive Imitation Reward Function V2.2

Input: ε_{min}, ε_{max}, ε_{step}, $ratio$
$\varepsilon \leftarrow \varepsilon_{max}$;
while *Training is not done* **do**
  while *Episode is not done* **do**
    $a \leftarrow \pi(s_t)$;
    $(s_{t+1}, r_t, done) \leftarrow$ *execute the action* a;
    (r_t `is calculated according to Eq 7.12 using the current` ε)
  end
  begin Update ε **for Reward 2.2**
    Stack episodes in a queue Q of size $n = 20$;
    if *Q is full* ***and*** *the number of non-zero rewards in Q* $\geq ratio \times length(Episodes \in Q)$ **then**
      $\varepsilon \leftarrow \varepsilon - \varepsilon_{step}$;
      Fully clear Q ;
    end
  end
end

Algorithm 4: Adaptive Imitation Reward Function V3.2

Input: ε_{max}, $ratio$, hw_{size}
$\varepsilon \leftarrow \varepsilon_{max}$;
while *Training is not done* **do**
  while *Episode is not done* **do**
    $a \leftarrow \pi(s_t)$;
    $(s_{t+1}, r_t, done) \leftarrow$ *execute the action* a;
    (r_t `is calculated according to Eq 7.12 using the current` ε)
    begin Update ε **for Reward 3.2**
      Stack $d = \|s_t - s'_t\|_2$ in a queue Q of size $N = hw_{size}$;
      if *Q is full* ***and*** $size(\{d,\ d \in Q\ and\ d \leq \varepsilon\}) \geq ratio \times hw_{size}$ **then**
        $\varepsilon \leftarrow median(\{d, d \in Q\ and\ d \geq \varepsilon\})$;
        Fully clear Q ;
      end
    end
  end
end

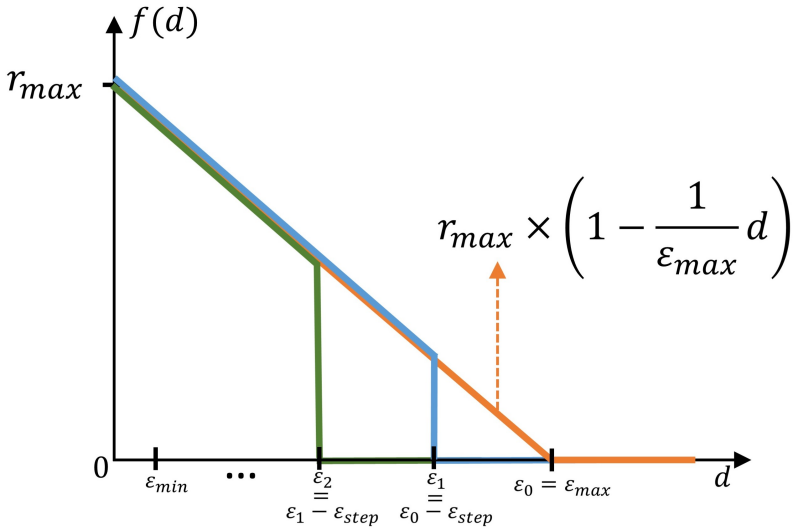


Figure 7.7 – The figure illustrates the adjustment of the ε value in Eq.7.12, starting at ε_{max}, decreasing by ε_{step} and bounded by ε_{min}.

7.5.2 Experiments and Results

Adaptive Reward Function V2

The value of ε is updated according to how many non-zero rewards are received for each episode. For each update, ε is decreased by ε_{step}. The value of ε is updated if the average of non-zero rewards over a queue of $n = 20$ recent episodes is greater or equal to $ratio \times length(Episodes\,in\,the\,queue)$. Each time ε is updated, the whole queue is cleared. We also experiment with two other variants V2.0 and V2.1 in Appendix B. V2.1 removes only the oldest item in the queue. V2.0 updates ε based only on a single episode.

The *ratio* parameter represents the fraction of the number of state predictions with positive rewards over the total time-steps of the episode.

We experiment using different values of ε_{step} and *ratio* and report results in Fig 7.8. The lower the value of ε_{step} (i.e. slightly modifying ε), the slower the imitation policy is to converge. Whereas using a high value of ε_{step} may result in reaching a very low ε in record time before the agent acquires any expertise using intermediate ε and thus make it impossible to converge to an optimal policy since positive values will be scarce for very low values of ε. This is clearly visible in the results of Reward V2.0 in Appendix B, Fig B.1 for $(ratio = 0.5, \varepsilon = 1e-03)$ and $(ratio = 0.7, \varepsilon = 1e-03)$.

High values of *ratio* make it difficult for the agent to update the value of ε and therefore it remains at high values, which prevents convergence.

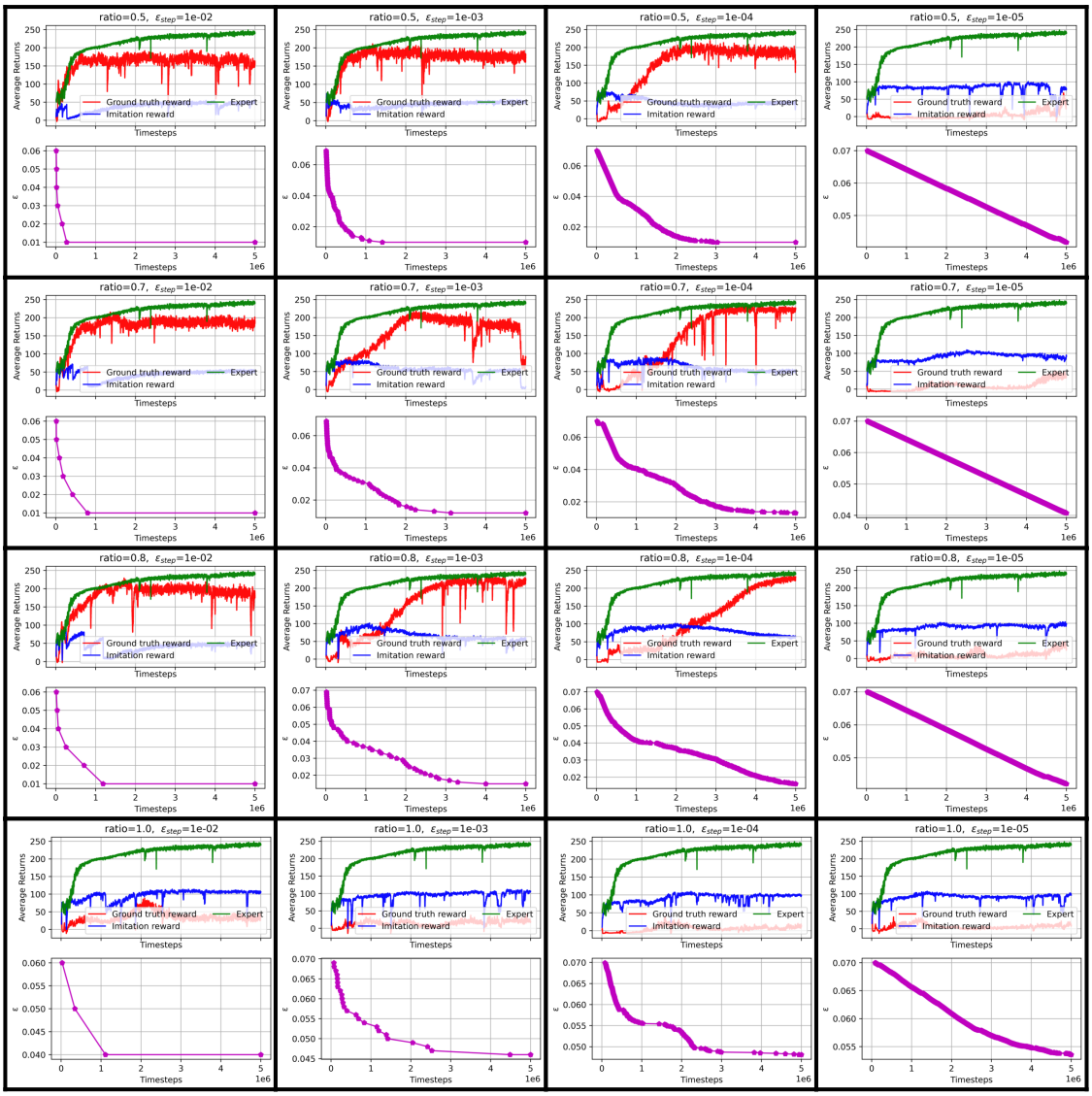


Figure 7.8 – Results for Reward "V2.2". Each cell in the grid depicts the training performance and the evolution of ε for different values of ε_{step} and *ratio*. $\varepsilon_{step} \in \{1e-2, 1e-3, 1e-4, 1e-5\}$ and $ratio \in \{0.5, 0.7, 0.8, 1\}$.

In summary, results for Reward V2, suggest that an imitation policy that converges to a policy with performance close to the expert is possible for a *ratio* around 0.7 and 0.8 with a careful tuning of ε_{size}. Version V3 of the adaptive reward function employs an automatic update process of ε, rather than using a predefined ε_{size}.

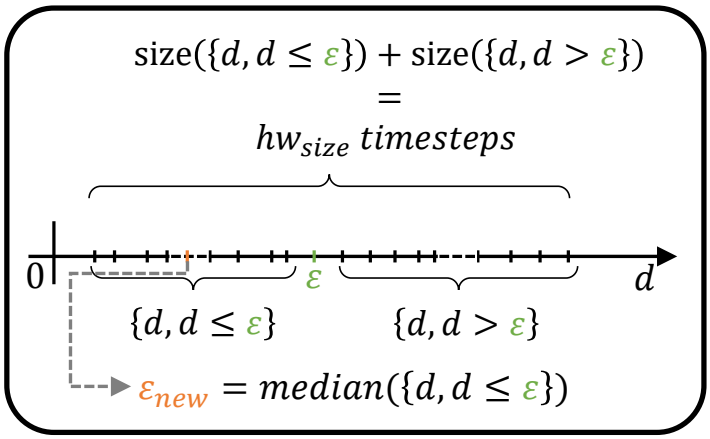


Figure 7.9 – Illustration of the updating rule for the Reward "V3.2". The updated value ε_{new} is computed by taking the median of distances $d = d(s, s')$ that are less than the current ε from the hw_{size} most recent time-steps.

Adaptive Reward Function V3

The value of ε is updated according to how many time-steps the distance between the predicted state and the actual state $d(s, s')$ is less than ε (see Fig 7.9 for an illustration). This evaluation is done over a window of the hw_{size} recent time-steps. Instead of decreasing ε by ε_{step}, we adaptively update ε depending on the scale of how close actual states are from predicted states. ε is updated only if the number of time-steps with $d \leq \varepsilon$ is higher than a *ratio* multiplied by the history window size hw_{size}. ε is updated to the median of the distance values that are less than ε. For instance, suppose that $hw_{size} = 10$ and the current value of $\varepsilon = 0.07$ and $d(s, s')$ values measure $\{0.02, 0.03, 0.04, 0.05, 0.06, 0.09, 0.1, 0.2, 0.3, 0.4\}$. The next value of ε will be set to the $median(d, d < \varepsilon) = median(0.02, 0.03, 0.04, 0.05, 0.06) = 0.04$. Updating ε to the median ensures that the current policy can still get positive rewards for at least $\frac{size(\{d, d \leq \varepsilon\})}{2}$ time-steps. Another option for updating ε is to use the first quartile or the third quartile instead of the median. Using the first quartile will make it more challenging for the agent to progress to the next level of difficulty, as it results in an important decrease in ε and a reduction in the number of points that receive positive rewards under the current policy. Using the third quartile has the opposite effect, as it will make it easier for the agent to progress to the next level of difficulty with a slight decrease in ε and more points that still can receive positive rewards under the current policy.

The parameter *ratio* is the fraction of the number of state predictions that satisfy

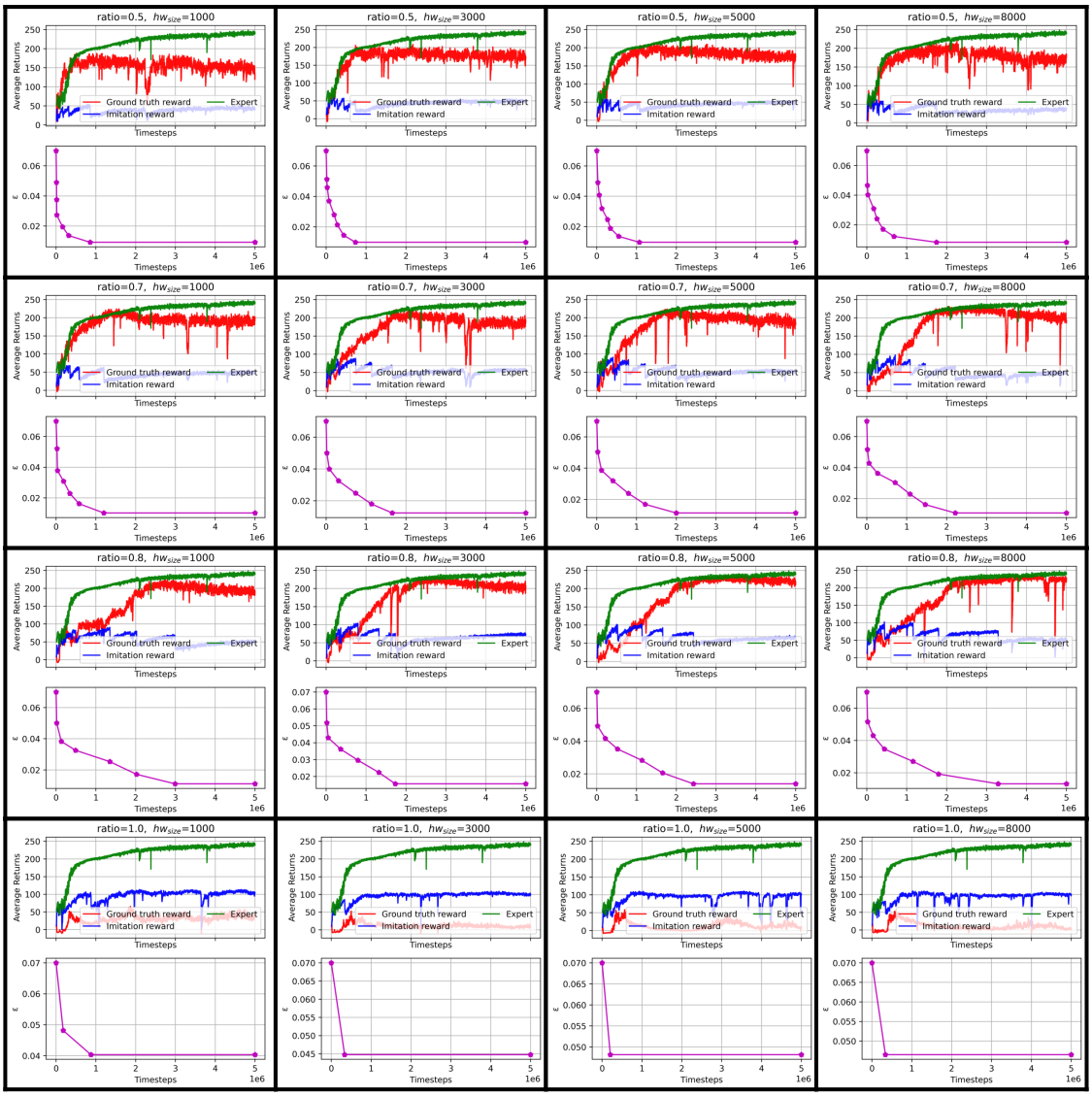


Figure 7.10 – Results for Reward "V3.2". Each cell in the grid depicts the training performance and the evolution of ε for different values of hw_{size} and $ratio$. $\varepsilon_{step} \in \{1e3, 3e3, 5e3, 8e3\}$ and $ratio \in \{0.5, 0.7, 0.8, 1\}$.

$\|s_t - s'_t\|_2 \leq \varepsilon$ over the total number of time-steps in the history window hw. When ε is updated, all of the hw_{size} time-steps are removed. We also experiment with removing only the oldest time-step in Appendix B. The results are reported in Fig 7.10 using different values of hw_{size} and $ratio$.

The imitation policy converges for all values of $ratio \in [0.5, 0.9]$ and all values of hw_{size}. The advantage of Reward V3 over Reward V2 is that we do not need to manually define the rate of change of ε.

The adaptive imitation reward function can be seen as a curriculum generator [125]. The imitation policy is learnt by minimising the distance between the actual state of the imitator and the predicted state by the representation model. The margin from which the agent starts receiving positive rewards is ε. The lower ε the harder it is for the agent to find trajectories with positive rewards and converge. By setting ε to a high value and decreasing it gradually during training, the agent is gradually guided towards the expert's behaviour. Once one level of difficulty has been solved, the next value of ε sets a slightly harder level of difficulty, and the skills acquired in the last level receive zero-rewards to induce the agent to further improve its performance. Our adaptive imitation reward function allows for automatic curriculum learning that is not required to be stipulated by the experimenter.

7.6 Summary

In this chapter, we presented a general framework for imitation from observation utilising RL with an agnostic reward function. The RL model does not have direct access to demonstrations such as in DDPGfD [67] or GAIL [64]. Our method learns implicitly from the imitation reward function, which employs the representation model trained on demonstrations. The representation model is trained to extract spatial and temporal features from the expert demonstrations and the imitation learning process encourages the imitator to adopt a behaviour that has similar spatial and temporal patterns as the expert.

The findings, published in [126], demonstrate promising results and compare favourably to state of the art methods. Additionally, the trained policies obtained using our method transfer better from simulation to real-world compared to BC and GAIL.

The next chapter will focus on evaluating the generalisability of our method to a variety of tasks. We apply our method to a set of manipulation tasks from the Meta-World benchmark [127].

8

Generalisation to Other Tasks

Contents

8.1 Introduction

This chapter aims to explore the applicability of our approach introduced in Chapter 7 to more complex tasks from the Meta-World robotic environments. While our previous work has used the pushing task for preliminary experiments, the Meta-World tasks pose particular challenges due to their high-dimensional state and action spaces and significant differences in the simulator, robot, and the environment settings. This affords us the opportunity to assess the generalisability of our approach across diverse environments. The experimental methodology is outlined as follows.

— Train expert agents using RL with task-specific and predefined reward functions.
— Generate demonstrations using the successfully trained RL models
— Train representation models of demonstrations
— Train an imitation policy for each task

— Compare to GAIL and GAILfO baselines

8.2 Metaworld Benchmark

Meta-World [127] is an open-source benchmark purpose-built for Meta-Learning and Multi-task learning. A total of 50 manipulation tasks are designed using the MuJoCo simulator [128] and the OpenAI Gym interface [129]. The tasks range from simple point reaching in the 3D space to more complex tasks such as grasping objects and opening/closing a door. A Sawyer robot is used to solve the manipulation tasks by either manipulating an object to reach a target. All the tasks share a common structure for the state and action space. The state of the environment is defined as follows:

$$s_t = \begin{cases} ob_{t-1} \\ ob_t \\ g_t \end{cases} \tag{8.1}$$

Where ob_t includes the measurements at time t of the 3D Cartesian position of the end-effector, a scalar value measuring the opening of the gripper, the 3D Cartesian position and the quaternion of the first and second object. And g_t is the 3D Cartesian position of the goal. If no second object is present, the corresponding values are set to zero.

The robot acts on the environment by executing the following output of the policy $[\Delta x, \Delta y, \Delta z, gr]$. The first three values $(\Delta x, \Delta y, \Delta z) \in [-0.1, 0.1]^3$ indicate the change of the absolute position of the end-effector in the 3D Cartesian space and the last value, gr, commands the gripper opening. Table 8.1 summarises the comparison between the state and action spaces of our Pushing task and the Meta-World tasks.

The description of all Meta-World tasks are listed in Table C.1 in the Appendix. C. Each task involves either the manipulation of one object with a varying goal position or the manipulation of two objects with a stationary goal position (see Fig. 8.1 for an example of each). The success metric, $\|pos_{object} - pos_{goal}\|_2$, is defined based on the euclidean distance calculated between the object to manipulate and the goal position.

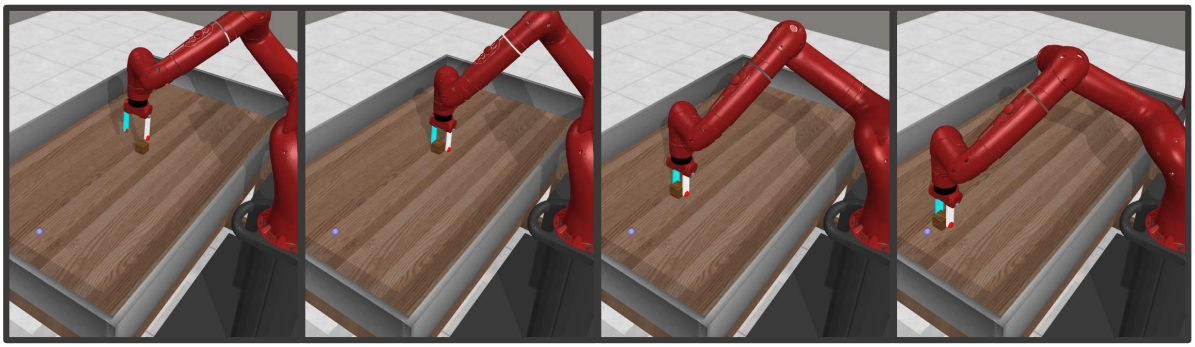
	State Space			Action Space	
	Description		**Dim**	**Description**	**Dim**
Our Pushing Task	- 2D position of the gripper, object and the target - Only the current state is used		$\mathbb{R}^6$	- 2D control - $(dx, dy) \in [\![-0.5, 0.5]\!]^2$	$\mathbb{R}^2$
Meta-World Tasks	- 3D position of the gripper + Opening of the gripper - 3D position and the orientation of each object (one or two objects) - 3D position of the target	Use only the current state	$\mathbb{R}^{21}$	- 3D + gripper control - $(dx, dy, dz) \in [\![-0.1, 0.1]\!]^3$	$\mathbb{R}^4$
		Use the current state+ previous state	$\mathbb{R}^{39}$	- 3D + gripper control - $(dx, dy, dz) \in [\![-0.1, 0.1]\!]^3$	$\mathbb{R}^4$

Table 8.1 – Comparison between the state and action spaces of our Pushing task and Meta-World tasks.

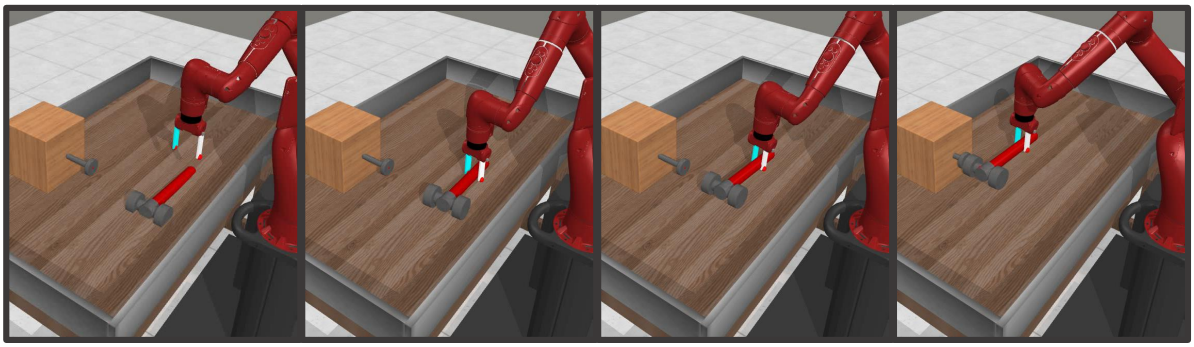
8.3 Generating Demonstrations

We first train RL models on the predefined reward functions for each task of the benchmark. The reward details can be found in Appendix E1 in [127]. We use SAC method with a fixed neural network architecture and the hyperparameters depicted in Table C.2. We experimented with both an input that consists of only the current state and an input that concatenates the current and previous time-steps. The latter proved to be more effective. The corresponding success rate scores are shown in Fig. C.4 and the training curves are shown in Fig. C.2 and Fig. C.3.

The successfully trained models were used to generate demonstrations for 33 tasks. For each task we generated 140 successful trajectories where the objective of the task is reached. For this, we rolled out the RL-obtained policies on the environment to solve tasks for different initial states. The random space from which the position of objects are samples is first discretised into a grid of evenly distributed positions. Then, for each demonstration, the initial state is randomly chosen from the possible values from the discrete space of initial states. The statistic details of the generated demonstrations are given in Table C.3.



(a) sweep-v2 task



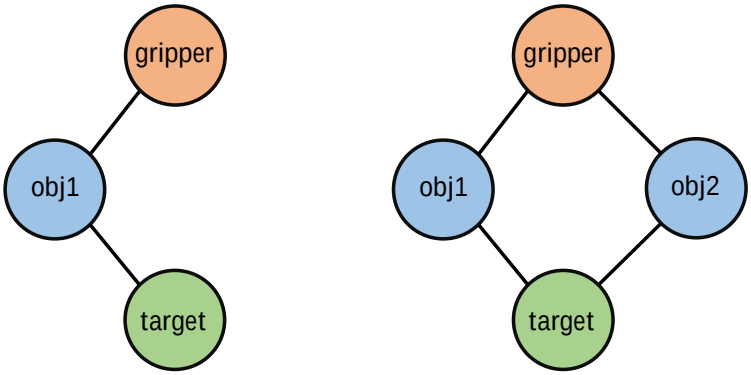
(b) hammer-v2 task

Figure 8.1 – Two examples of tasks in the Meta-World benchmark. The sweep-v2 task requires the robot to sweep a puck towards a goal location. In the hammer-v2 task the robot needs to pickup the hammer and use it to hammer a screw into the wall.

8.4 Representation Learning of Demonstrations

8.4.1 Scene-graph Construction

The state of the environment for a given task is represented as a graph where each node corresponds to a physical element in the scene, such as the gripper, target, and the manipulable objects. The edges between nodes indicate the potential interactions between them. In our case the gripper is connected to all the manipulable objects which are in their turn connected to the target (see Fig.8.2). Each node is characterised by a set of features that include the 3D position in the Cartesian space and a one-hot encoding vector that identifies which element is being represented.



(a) Scene-graph of tasks involving one manipulable object

(b) Scene-graph of tasks involving two manipulable objects

Figure 8.2 – Illustration of scene-graphs.

8.4.2 Representation Model Training

We use the representation model pipeline introduced in Chapter 6. The model takes as input a sequence of l observations $(o_{t-l+1}, ..., o_t)$ sampled from the demonstration dataset and attempts to predict information about the subsequent time-step. The predicted information is a vector y_{t+1} that concatenates the 3D position of all the objects in the scene and the distance between the objects that are connected by an edge in the scene graph. The hyperparameters that yielded the best results in the experiments conducted in Chapter 6 have been selected for training the representation models of demonstrations, and they are listed in Table 8.2. The representation models are trained end-to-end using 100 demonstration trajectories, while using 20 each for validation and testing. The evaluation performance results on the test split are shown in Table C.4.

8.5 Training the Imitation Policy

8.5.1 Experimental Setup

The robot attempts to learn a given task by trial and error using the predictive representation model trained on demonstrations and doesn't have direct access to demonstrations. At each time-step, the robot decides which move to make in the 3D space along the x, y, and z axes with a maximum displacement of $1cm$ in each direction. The maximum episode length, T_{max}, is set to be slightly higher than the length of the longest trajectory

Description	Detail
GNN Hyperparameters	
GNN-Type	SAGEConv
Hidden channels size	64
Number of layers	2
Aggregation function	Average pooling
Dropout	20%
Seq2Seq Hyperparameters	
Autoencoder	LSTM
Input sequence size	3 or 7
Hidden state size	32
General Hyperparameters	
Learning rate	1e-3
Loss function	RMSE

Table 8.2 – The representation model training hyperparameters.

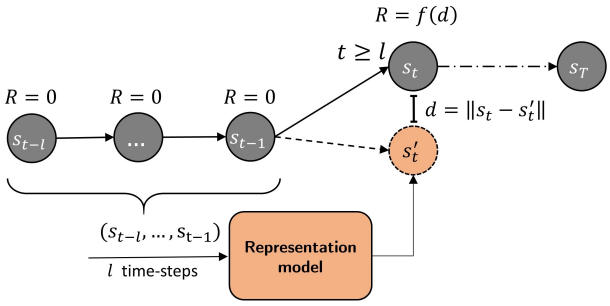
in the demonstration dataset. The episode is concluded if the goal state is reached or when the time-steps have been exhausted.

8.5.2 Training the Imitation Policy

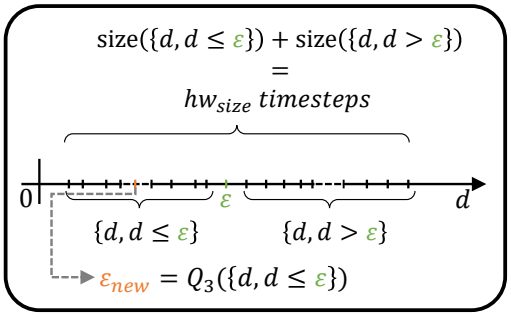
We use the adaptive reward function introduced in Chapter 7 and we train the imitation policies by following Alg.4. The imitator is trained using Reinforcement Learning and rewarded based on the difference between the current state of the environment (s) and the one predicted by the representation model (s'), as computed according to Eq. 8.2. This reward is positive only when the distance between the two states, $d(s, s')$, does not exceed a threshold ε. The value of ε is pre-initialised at the beginning of the training and gradually decreased as illustrated in Fig. 8.3-b and detailed in Alg.4.

$$
\begin{aligned}
R\left(s_t\right) &= \begin{cases} 0 & \text{if } t \in [0, l-1] \\ R\left(s_t \mid s_{t-1}, \ldots, s_{t-l}\right) = f(d) & \text{if } t \geq l \end{cases} \\
d &= \left\| s_t - s'_t \right\|_2 \\
f(d) &= \begin{cases} 0 & \text{if } d \geq \varepsilon \\ r_{\max} \times \left(1 - \frac{1}{\varepsilon_{max}} d\right) & \text{if } d < \varepsilon \end{cases}
\end{aligned}
\tag{8.2}
$$

The initial value of ε is set to $\varepsilon_{max} = 100 \times RMSE$ where the $RMSE$ is the Root Mean Squared Error of the representation model calculated on the test split of the demonstration



(a) Illustration of the imitation reward given to the agent at every time-step. The f function is defined in Eq.8.2



(b) Illustration of the updating rule of ε in the adaptive reward function given in Eq. 8.2. The third quartile Q_3 is used for a smoother adjustment of ε.

Figure 8.3 – Illustration of the calculation of the imitation reward along with the updating process of its hyperparameter ε.

dataset. It is then adjusted based on the number of time-steps with $d \leq \varepsilon$ in the recent hw_{size} time-steps, provided that this number meets a certain *ratio* threshold. In our experiments, we set hw_{size} to forty times the maximum length of training episodes $hw_{size} = 40 \times T_{max}$ and *ratio* to 0.6.

The SAC algorithm is used to optimise the imitation reward function. The hyperparameters details can be found in Table C.5. The training curves are shown in Fig. 8.4, where different training progress patterns are displayed, while Fig. 8.5 illustrates the progress of ε, the success rate, and the cumulative imitation reward for 6 tasks representing these patterns.

The models trained to solve tasks such as door-close-v2 and door-lock-v2 exhibit rapid convergence to 100% success rate, with ε swiftly reaching a threshold that enables the imitator to closely match the expert's behaviour and solve the task. On the other hand, the models trained for tasks such as lever-pull-v2 and stick-push-v2 take a considerable time of trial-and-error before the success rate rises and eventually solve the tasks with 100% success rate. For example, in the lever-pull-v2 task, it takes about 71 million training time-steps before the success rate rises at $\varepsilon = 8mm$ to reach 100% success rate at $\varepsilon = 4mm$.

The sudden drop in performance observed, for instance, in window-close-v2 and button-press-topdown-wall-v2 is associated with a decline in the cumulative imitation reward, which suggests that it is caused by a deterioration in the RL optimisation process, rather than being caused by misleading signals from the imitation reward function.

It is worth noting that the imitation policies achieve their best performance when the

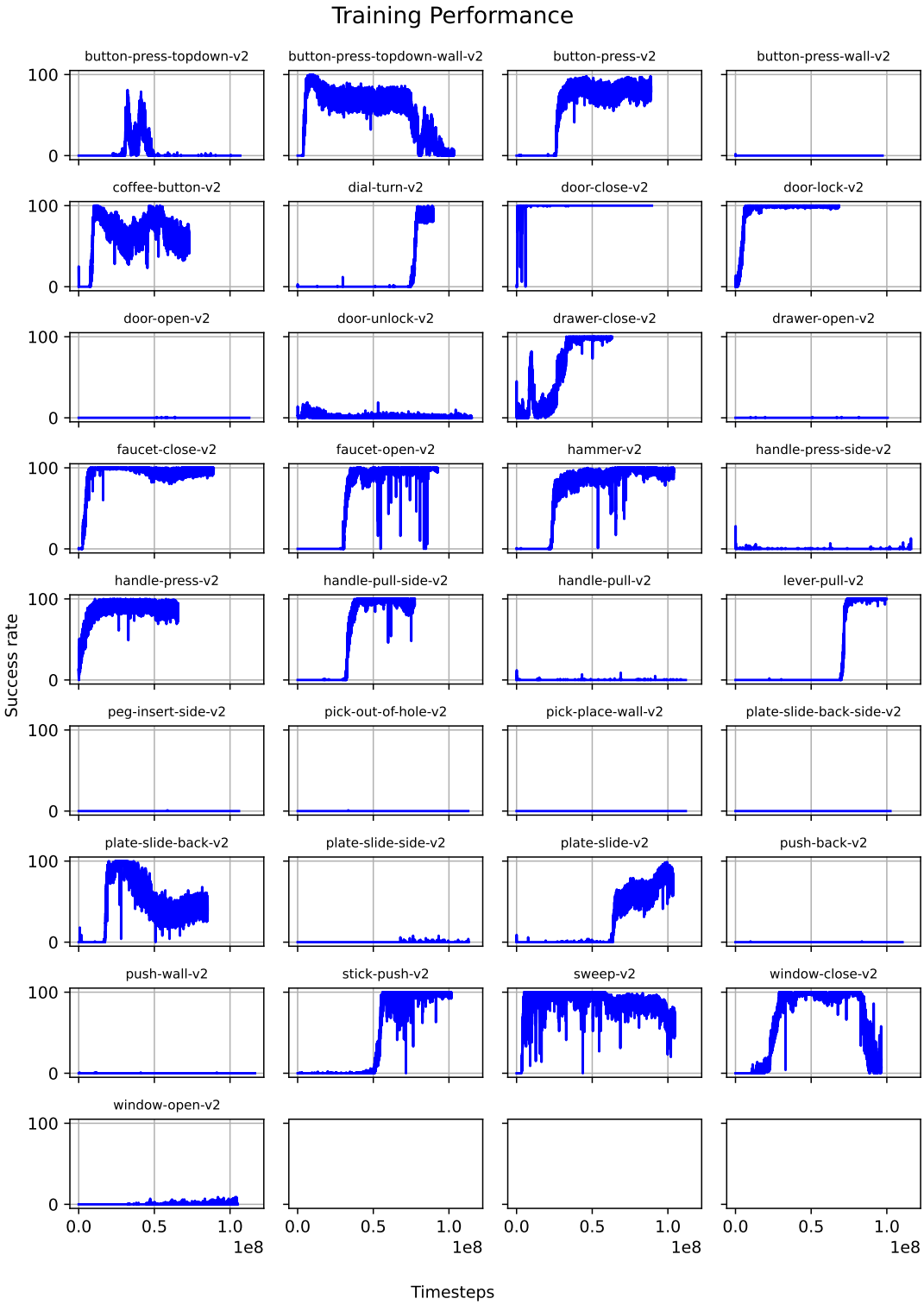


Figure 8.4 – Plots depicting the progression of the success rate during the imitation policy training. Only the imitation reward function is used for training.

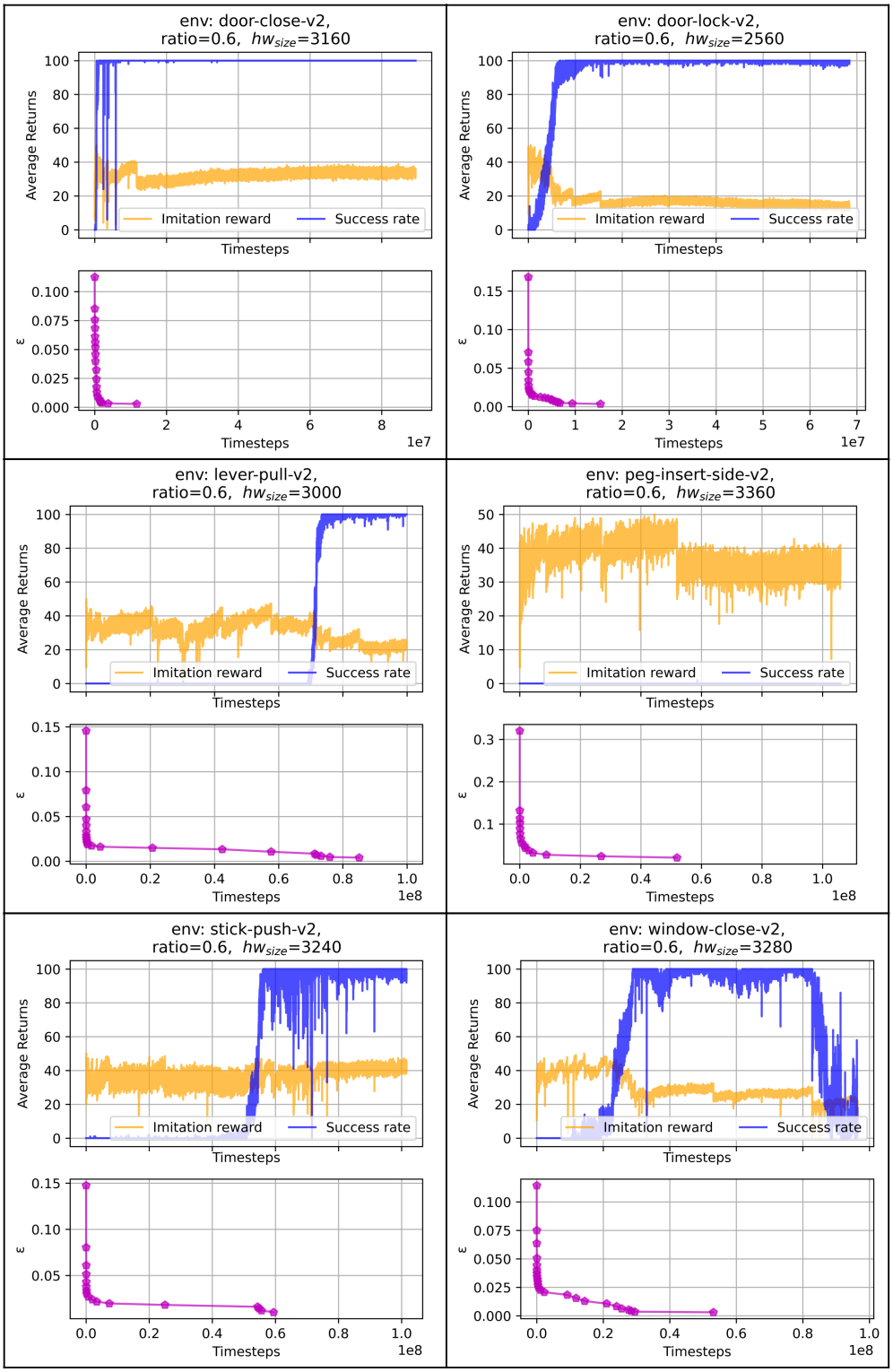


Figure 8.5 – Training performance along with the progress of ε.

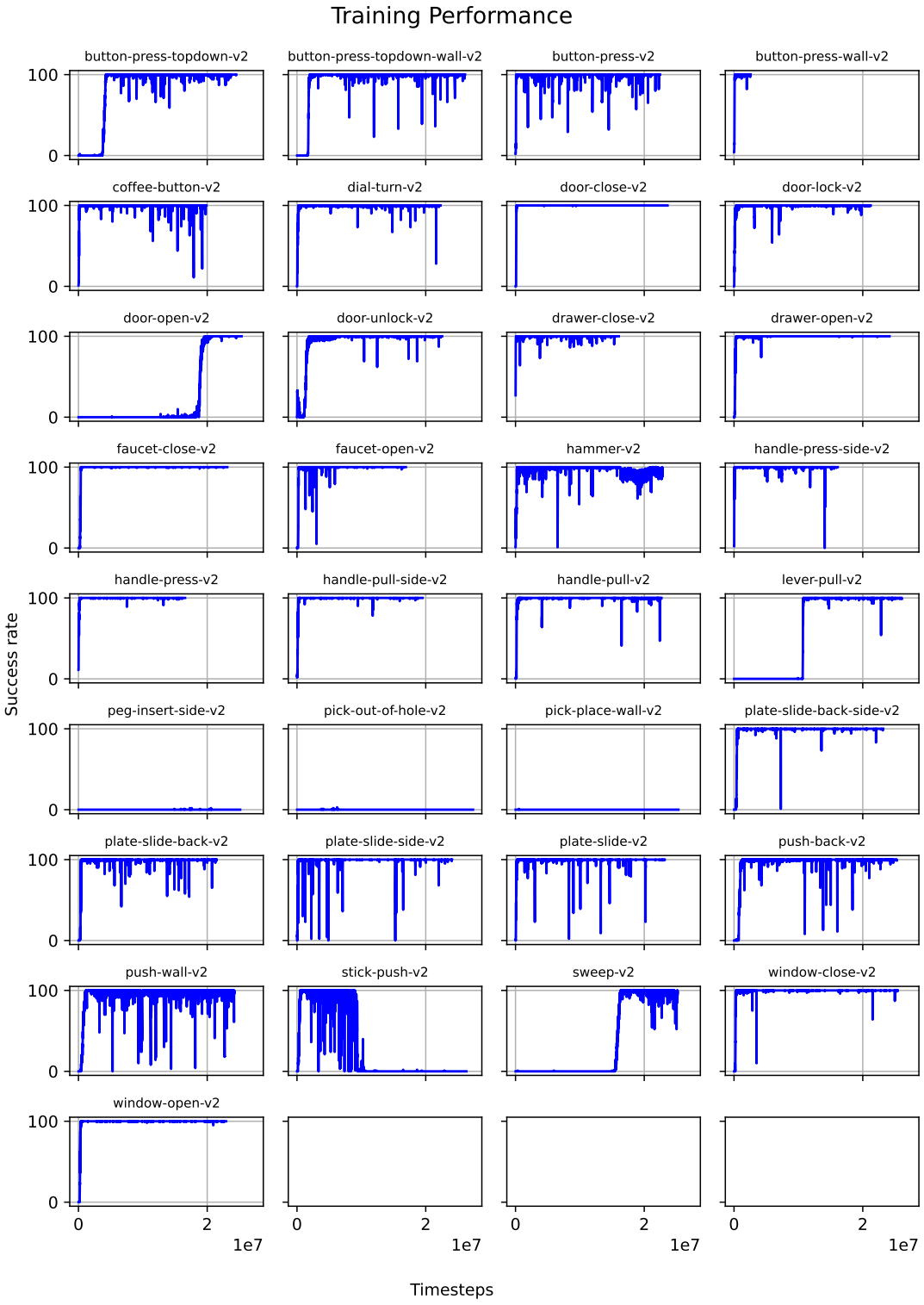


Figure 8.6 – Plots depicting the progression of the success rate during the imitation policy training. The imitation reward and an additional reward for reaching the goal state are utilised together for training.

cumulative imitation rewards are at their peak during the last ε update. This characteristic enables us to retain the optimal policy that best imitates the expert during training, even if the goal state is indeterminate.

Tasks such as door-open-v2 and peg-insert-side-v2 were not solved. The training failure may be attributed to insufficient exploration of the environment by the agent and the need for additional training time-steps for convergence. Additionally, the pre-trained representation models may be inaccurate and require further improvement.

Adding an extra reward for the goal state

The models trained to solve tasks such as door-unlock-v2, handle-pull-v2 and window-open-v2 exhibit evidence of learning, where the goal state is achieved during training but the behaviour that leads to achieving the goal state is not reinforced. This is probably because the imitation reward function is not sufficiently precise (i.e. ε is not sufficiently low) for the RL optimisation to result in an imitation policy that matches the expert's behaviour. In this section we retrain the same model architecture with the same imitation reward function from the last section plus an extra reward of $R = +100$ that the agent receives if the goal state is achieved. The training curves are shown in Fig. 8.6.

All the tasks were solved except peg-insert-side-v2, pick-out-of-hole-v2, and pick-place-wall. In the early stages of training, the imitation reward helps guiding the exploration and bring the imitator closer to the expert's behaviour. Upon achieving the goal state, the additional reward reinforces the behaviour that solves the task. For the unsolved tasks, the representation model should be improved by searching for more optimal hyperparameters for a more precise representation model.

8.5.3 Comparison to Baselines

We evaluate the performance of our method against GAIL and GAILfO. The discriminator in GAILfO takes as input the state s_t and the previous state s_{t-1}. The discriminator in GAIL takes the same input as GAILfO in addition to the action a_t taken in the state s_t. Both GAIL and GAILfO are trained using PPO algorithm and take the same input as our method, which consists of the concatenation of the current and the previous state. The discriminator reward function used in GAIL is defined in Eq. 8.3 and the one used in GAILfO is defined in Eq. 8.4.

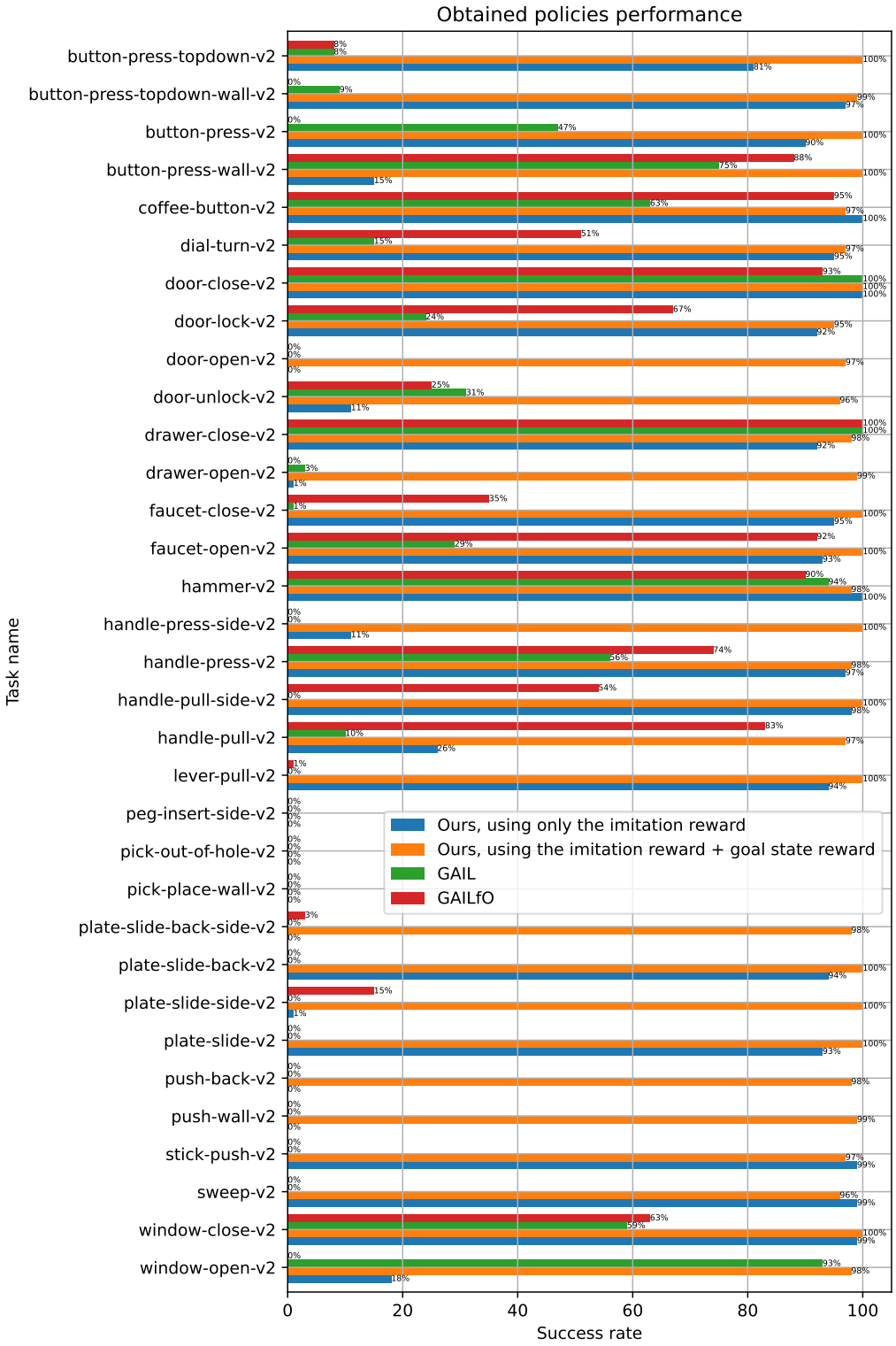


Figure 8.7 – The success rate of models trained using GAIL, GAILfO and our method, evaluated on 100 episodes using the most successful policy from the training phase, which yielded the highest success rate.

$$R(s_{t-1}, s_t, a) = \log\Big(D(s_{t-1}, s_t, a)\Big) - \log\Big(1 - D(s_{t-1}, s_t, a)\Big) \tag{8.3}$$

$$R(s_{t-1}, s_t) = \log\Big(D(s_{t-1}, s_t)\Big) - \log\Big(1 - D(s_{t-1}, s_t)\Big) \tag{8.4}$$

During training, we save the optimal policy that achieved the highest success rate. We then evaluate the obtained policies by running them to solve the concerned tasks for 100 trials and record the success rate. Our findings, reported in Fig. 8.7, show that our method outperforms GAIL and GAILfO in solving the majority of the tasks, and adding an extra reward to the goal state in our method helps getting higher success rate in most of the tasks.

Our method differs from the generative adversarial approach in two respects. Firstly, we utilise a stationary reward function that yields consistent feedback for a given value of ε, as opposed to the fluctuating rewards that arise in adversarial learning due to the continuous updates of the discriminator during training. Second, instead of having direct access to demonstrations, we use a representation model that is pre-trained on demonstrations prior to the imitation policy training. Together, these two properties help to stabilise the training process and achieve superior results.

The stationary reward function ensures that the agent is rewarded consistently for taking actions that yields trajectories predictable by the representation model. This reduces the noise in the reward signal and helps the agent to learn more effectively. By using the representation model that captures the essential aspects of demonstrations, the imitation training process can save effort that would otherwise be spent on extracting features while learning the control policy.

8.6 Summary

In this chapter, we investigated the generalisation capability of the imitation learning approach that was introduced in Chapter 7, where it was successfully applied to solve a pushing task in a 2D action space. The ability of a method to generalise to different tasks is crucial for its practical application. Therefore, we examined how well the proposed approach performed when faced with tasks that differed from the original one it was developed on.

To assess the generalisation ability of our approach, we evaluated its performance on more complex tasks from the Meta-World robotic environments that required different

skills and actions. The results demonstrate that our approach has promising generalisation capabilities, demonstrating its potential for practical applications beyond the original pushing task.

9

Conclusion and Future Work

Contents

9.1 Summary

The primary goal of the thesis was to devise and develop an approach for learning from demonstrations that only involve observations without access to actions, which is also known as Imitation Learning from Observation. We subdivided this problem into two sub-problems, namely learning a representation of demonstrations and learning the imitation policy via Reinforcement Learning.

Our proposed solution for addressing these two sub-problems involves a disentangled framework that follows a two-phase training approach. In the first phase, a representation model is trained on the demonstrations to extract spatial and temporal features. This process includes converting the observations in the demonstrations into graphs. Then, spatial features are extracted using GNNs, while sequence modelling captures the temporal patterns of spatial feature evolution over time. The representation model is trained to predict future time-steps in the expert's trajectory based on the observation history. In the second phase, the trained representation model is incorporated in the design of a task-agnostic reward function that can be used with an out-of-the-shelf RL algorithm to learn the imitation policy. The imitator's training process involves comparing the expected

outcome generated by the representation model with the actual outcome of its actions, and adjusting the imitation policy towards narrowing the discrepancy between the two.

Our approach showcased superior learning ability in solving a simple 2D pushing task, surpassing Generative Adversarial Learning [64, 74] in transferring the policies to the real-world setting using state-only demonstrations. Furthermore, the approach was applied to various robotic tasks in the Meta-World benchmark [127], demonstrating its adaptability and effectiveness in various tasks. The results were promising, and our approach's benefits are threefold. Firstly, its stability is ensured as the reward signal doesn't change constantly over time as in Adversarial Learning. Secondly, it provides a modular solution where different components can be enhanced individually. Finally, the graph representation of the scene and GNNs allow for handling a variable number of objects.

Our study in Chapter 3 showed that training policies using RL in simulation and then transferring them to the real world can be effective, provided the training process considers the uncertainties inherent in the real world. The study focused on sensor noise, but depending on the task setup, noise can be also introduced, for instance, in the form of environmental disturbances and deliberate random actions taken by the agent. The goal is to expose the agent to a broad range of situations and uncertainties that it is likely to encounter in the real world, enabling it to adapt to diverse conditions.

9.2 Limitations and Insights for Potential Improvement

The successful implementation and application of the proposed LfD methodological framework to different tasks demonstrate its practicality and potential as a solution for learning control policies. However, certain tasks from Meta-World benchmark took a considerable amount of time to converge and three remained unsolved. It should be noted that the same representation model and policy learning hyperparameters were used for all the tasks, and customised hyperparameter tuning may lead to better convergence. Additionally by virtue of its modular design, our framework provides flexibility in enhancing various aspects of the learning process, including the representation model and the imitation policy learning. Each module of the framework is responsible for a specific purpose and contributes to the overall functioning of our solution. Improving each module, therefore, leads to an overall improvement in the framework's performance.

Sections 9.2.1 and 9.2.2 highlight potential avenues for enhancing the representation

model and the imitation policy training phase respectively. Moreover, Section 9.2.3 brings to light how our framework can be adapted for learning from human visual demonstrations. Additionally, we propose a method to integrate our representation model with GAIL for learning from visual demonstrations in Section 9.2.4.

9.2.1 Representation Learning

The representation learning phase precedes the imitation policy training, during which the expert's behavioural features are extracted from the demonstrations data. Enhancing the representation model will result in a more precise characterisation of the expert's behaviour, and thus a better performance of the entire framework in solving the intended task.

Scene graphs are used for describing the structure of the scene, including the objects involved, their attributes, and their relationships to one another. Our approach involves an object-centric scene graph, where every node represents an object in the scene. Node attributes primarily consisted of object positions in our experiments; however, attributes such as shape, dimensions, colour, etc., can be incorporated for tasks requiring such characteristics. The edges are binary with only two possible states, which represent either the presence or absence of a connection between two nodes. Further research can explore the outcome of incorporating additional information into the edges to provide more context and meaning to the connections they represent. For instance, in [98], high-level spatial relationships such as "getting close," "moving together," and "moving apart" were utilised to establish symbolic relations between objects for action recognition. Although the use of symbolic relations was effective in the experiments described in [98], the design of a separate module was required to recognise these relationships. Such a module might be intricate to design and may require considerable training data to learn the complex relationships, which can be time-consuming and costly.

Scene graphs can quickly become large and complex in the presence of multiple objects in the scene, making it difficult to identify the relevant information for taking the next action. Attention mechanisms can be incorporated into the representation model to allow it to selectively focus on specific parts of the scene over the course of the sequential decision making. For instance, our framework can replace the GNN module with Graph Attention Networks (GATs) [105] which utilise attention mechanisms to focus on particular nodes and edges. This can improve the model's ability to efficiently understand complex scenes and capture important details.

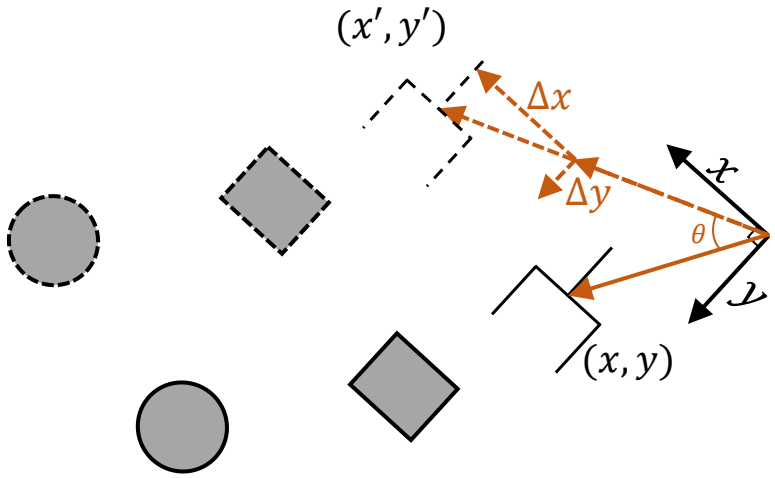


Figure 9.1 – Illustration of how demonstrations can be augmented using translation and rotation in the 2D space. The position of the objects in the augmented trajectories can be calculated using Eq.9.1.

Another research direction to improve the proposed representation model is to focus on the demonstrations dataset. In order to improve generalisation, data augmentation can be used to increase the size of the training dataset by generating new trajectories from the provided demonstrations. For example, translating the entire trajectory in 3D space or rotating it around a specific 3D point produces a new and distinct trajectory while still preserving the same essential behavioural information. An illustration for creating new trajectories in 2D is given in Fig.9.1. The position (x', y') of objects in the new trajectory can be calculated as a function of the original positions (x, y):

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix} \tag{9.1}$$

This augmentation approach can be mainly applied to manipulation tasks involving relocating objects in the 3D space. It enables a wider coverage of the 3D space. By providing the model with a more diverse range of training instances, it can acquire better generalisation skills and perform more effectively across a broader range of scenarios. For example, if we are training a robot to pick up objects from a conveyor and move them into a bin, we can enhance the trajectories by translating them in 3D space to help the robot adapt to different locations of the conveyor and the bin. Similarly, Enayati et al. [130] formally define the environment symmetry and investigate its utility to replicate demonstrations in robot manipulations tasks.

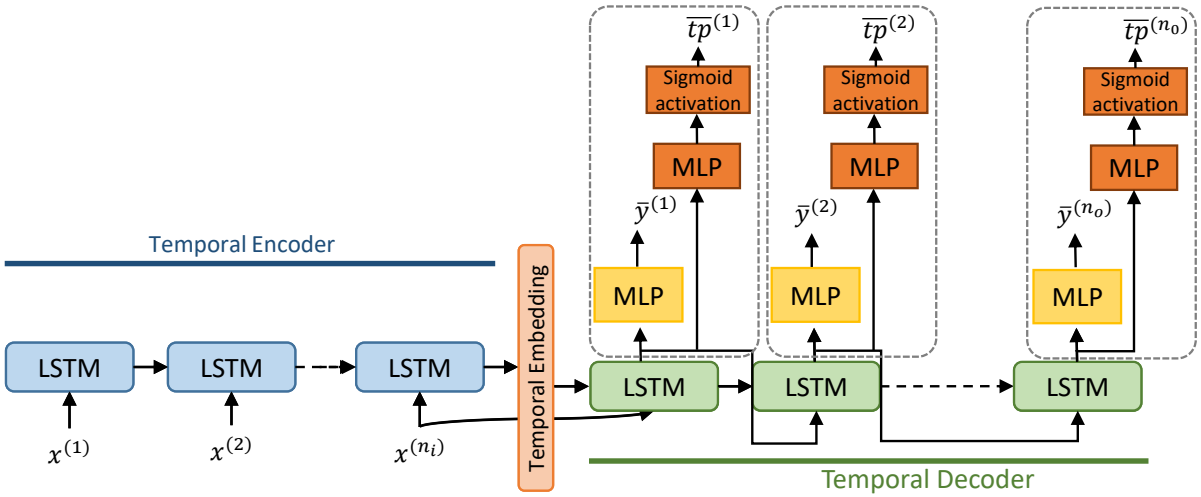


Figure 9.2 – Suggestion of a two-head representation model that also predicts the task progression tp.

9.2.2 Imitation Policy Learning

Our solution is a two-step process where the first phase trains a representation model of demonstrations and the second trains the imitation policy using RL and the imitation reward function. The time required to solve the imitation learning problem is primarily dominated by the time taken for the RL training phase. This is because the RL training process requires large number of iterations to optimise the imitation policy and the curriculum learning induced by the adaptive imitation reward function takes time to reach the right level of difficulty that leads the imitator to converge to the expert's behaviour. The representation learning phase, on the other hand, involves one-time supervised learning training and therefore takes much less time compared to the RL phase.

The findings of the last chapter indicate that tasks can be solved efficiently and quickly when the goal state is recognisable and a supplementary reward is given to the imitator upon reaching it. Nonetheless, requiring a recognisable goal state requires expert knowledge of the task, which contradicts our framework's objective of learning tasks solely from demonstrations, without any explicit knowledge of the task. In [131], a goal proximity function is trained on the demonstrations dataset to estimate the temporal distance to the goal and used for learning the imitation policy. Similarly, we suggest augmenting our representation model, as depicted in Fig 9.2, with an additional head that predicts the task progression tp, a value between 0 and 1 that can be computed as follows:

$$tp_t = \left(1 - \frac{T-t}{T}\right) \tag{9.2}$$

Where t and T are the current and terminal time-steps respectively. The training can be done by optimising the weighted sum of the losses $\mathcal{L}_1$ and $\mathcal{L}_2$ defined in Eq.9.3, as $\mathcal{L} = \alpha_1\mathcal{L}_1 + \alpha_2\mathcal{L}_2$, where α_1 and α_2 are the weighting coefficients.

$$\mathcal{L}_1 = \sqrt{\frac{1}{N}\sum_{1}^{N}(y - \overline{y})} \ , \qquad \mathcal{L}_2 = \sqrt{\frac{1}{N}\sum_{1}^{N}(tp - \overline{tp})} \tag{9.3}$$

After training the representation model, the imitation reward function outlined in Chapter 7, Eq.7.12, can be augmented by introducing an extra reward that reflects the imitator's progress in accomplishing the task. This reward increases as the imitator gets closer to the goal state, providing further guidance for solving the task at hand.

In our experiments the robot was operated by the trained policy through commanding the end-effector of the robot in the 3D space, which involves specifying the next desired position of the end-effector and using the inverse dynamics of the robot to apply movements in the joint space. This choice was favoured over commanding the joints of the robot to reduce the learning complexity. Commanding the end-effector is simpler to learn, as the agent only needs to learn how to move the end-effector to a specific position. This reduction in learning complexity comes with the cost of a reduction in the flexibility of the robot movements. This poses a problem, for instance, in the case of cluttered environments where the robot should learn how to avoid obstacles. Commanding the movements of multiple joints on the other hand allows the robot to utilise its multi-joint structure to perform complex manoeuvres. For example, consider a robot arm that needs to pick up objects in an environment with obstacles. Commanding the end-effector to move directly towards the target may not be feasible due to obstacles blocking the way. However, by commanding the joints, the robot can adjust its posture to avoid the obstacles. It should be noted, however, that commanding the joint space of the robot requires learning how to coordinate the movements of several joints to perform the task, which increases the learning complexity and the cost of training.

9.2.3 Learning from visual demonstrations

Our experiments were limited to demonstrations generated by an RL-expert model, with the spatial positions of objects in simulation readily available. The ultimate goal

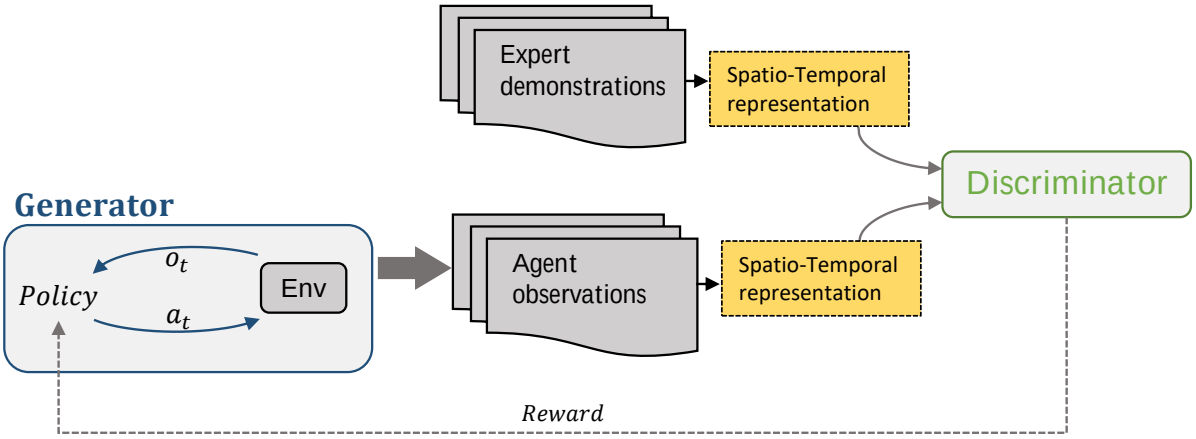


Figure 9.3 – Integrating the representation model introduced in Chapter 6 into GAIL algorithm. Passing demonstrations through the trained representation model, allows to reason about the spatio-temporal representation of demonstrations instead of the raw data.

of the research project, of which this thesis forms a part, is to enable robots to learn behavioural skills from observing human visual demonstrations while navigating their environment using a camera sensor. To achieve this, a pre-processing module of the camera feed is required, that can be plugged to the left end of our representation pipeline. This module will be responsible for identifying objects in the scene and determining their characteristics, such as spatial position and color. In the case of human demonstrations, the human hand also should be detected and tracked as in [132]. Once these characteristics have been extracted, they can be integrated into nodes and edges of the scene graph. Then, the representation model and the imitation policy can be trained as outlined in Chapter 6 and Chapter 7 without requiring any special changes to the backbone structure of the framework.

Nevertheless, the detection of objects and estimating their position in space using computer vision is susceptible to noise induced by the camera sensor and the environmental factors such as lighting conditions, occlusions and object interactions. To account for the inaccuracies induced by such noise, it might be necessary to incorporate methods for increasing the robustness of the representation model to noise. One such method is to augment the original demonstrations dataset by adding artificial variations to the original trajectories to make it more representative of real-world scenarios.

9.2.4 Integrating the Representation Model with GAIL

The proposed representation model can also be integrated with GAIL [64] for learning from visual demonstrations. GAIL's capability to learn from high-dimensional inputs sensory data (e.g. RGB data) is hindered by the the discriminator's tendency to differentiate between the imitator and expert's observations using task-irrelevant features, as demonstrated by Zolna et al. in [133]. This means that the discriminator may rely solely on the differences in appearances between the expert and the imitator's observations (e.g. human hand vs robot end-effector, background differences, etc.), without considering the underlying dynamics of the demonstrated task. One way to overcome this is by passing the expert demonstrations through our representation model as depicted in Fig 9.3, which encodes them into a lower-dimensional space capturing the relevant information for the task. The imitator's observations are also encoded using the same representation model, and the discriminator is trained to distinguish between the expert and imitator's representations.

9.3 Conclusion and Perspective

Our contribution in this thesis is mainly on introducing and implementing a novel framework that is modular and generic, with the primary purpose to enable robotic learning from state-only demonstrations. Its application to various tasks demonstrated promising potential. Our framework's flexibility allows for customisation to different settings, making it a versatile solution. The findings encourage further research to improve performance and expand the framework's capabilities to handle visual inputs and visual demonstrations. To this end, feasible and practical recommendations are provided in Section 9.2. We believe that our contribution represents a step forward in the field of robotic learning and imitation from observation, potentially opening up new opportunities for future developments.

The ultimate objective is to enable robots to learn from human videos, which has the potential to improve human-robot interaction and facilitate the acquisition of new skills by robots in a more natural and intuitive way. The alternative approach to the modular design that is adopted in our work, is to use a single model that takes raw input data, such as a frame or sequence of frames, and directly produces the desired actions without any intermediate steps. The model learns to extract features and patterns from the input data and chooses the action to take in a single step. Such a model is trainable end-to-end

and requires less engineering effort to implement but will require a considerable amount of data to train and it is hard to debug if it malfunctions. The modular design on the other hand breaks the overall problem to sub-problems and solves each one separately. In the case of imitation from human videos, the imitation problem can be divided into scene analysis using computer vision, learning a spatio-temporal representation of human behaviour, and learning the control policy. Utilising this approach can provide greater flexibility and ease of debugging compared to end-to-end approaches. Additionally, it enables learning from a smaller number of demonstrations and allows for the reuse of the preprocessing module across various tasks.

Our research highlights the potential of Machine Learning in equipping robots with the ability to learn and execute tasks solely based on demonstrations, without the need for prior expert knowledge. Unlike traditional programming approaches that require robotic experts to anticipate every possible scenario and program the robot accordingly, which can be time-consuming and error-prone, ML enables robots to learn from their experiences and adapt to new situations. This makes it possible for robots to acquire behavioural skills that are difficult or impossible to program explicitly. The progress in robot learning can lead to more efficient and cost-effective robotic systems. The incorporation of data collected from both observing human experts and robots' interactions with the environment will allow robots to learn and operate more efficiently, leading to increased versatility and usefulness across various applications, decreased manual intervention, and improved productivity. This will help to make robotics more accessible to a wider range of businesses, enabling them to benefit from the increased efficiency and flexibility that robots can provide.

Appendix A

Representation Learning Ablation Study

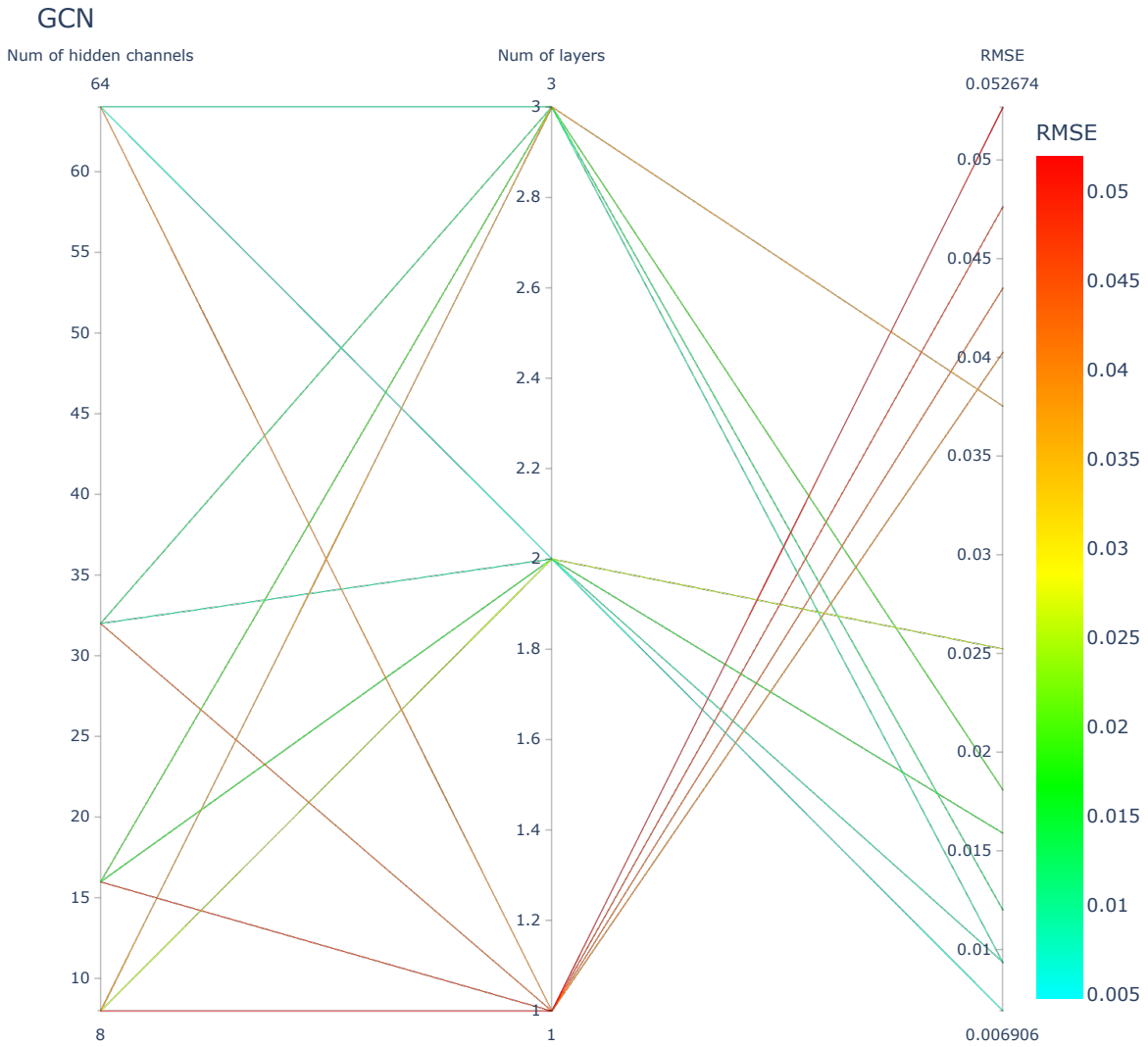


Figure A.1 – Enlarged depiction of the plot shown on the left-hand side in Figure 6.10-a.

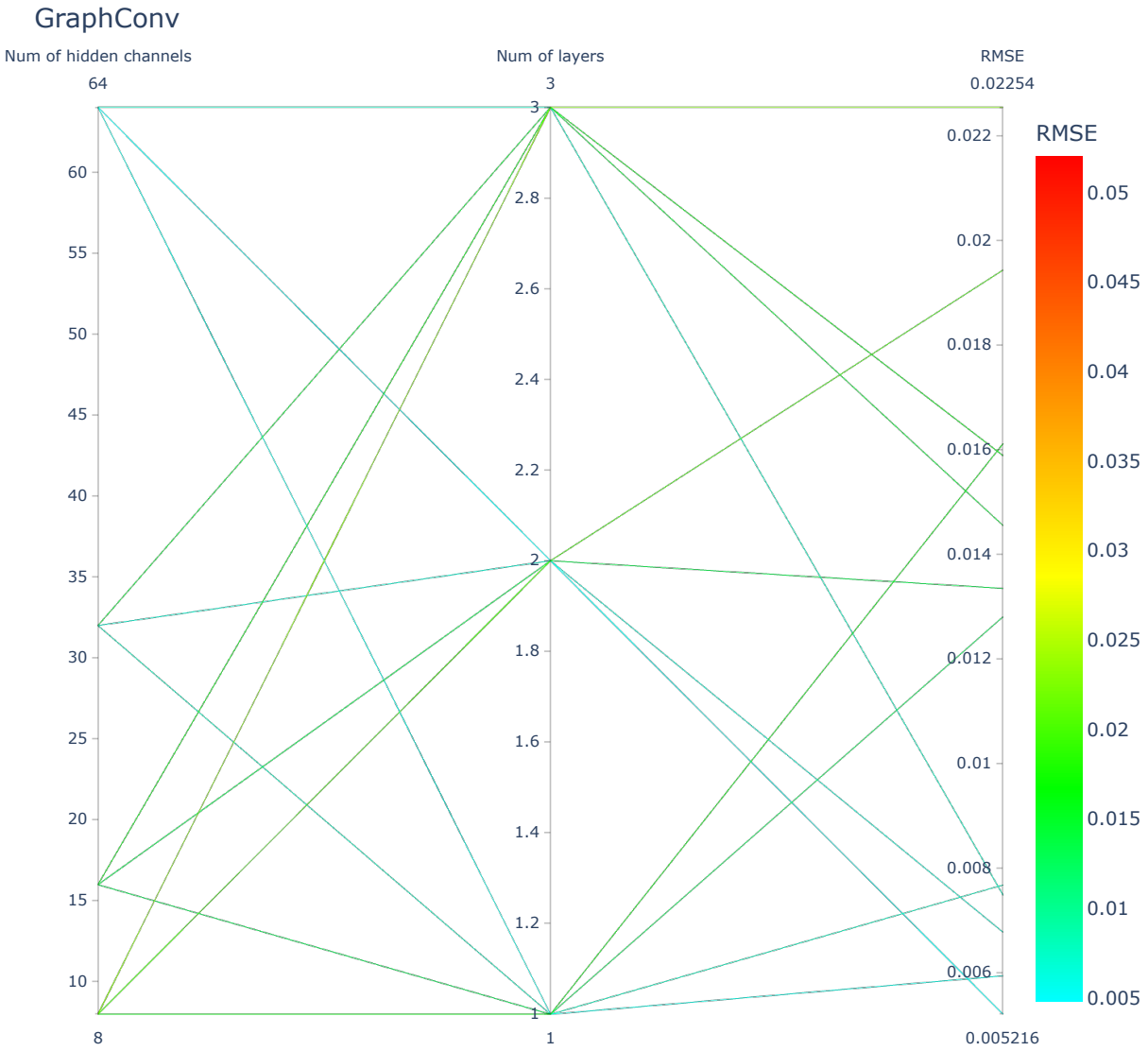


Figure A.2 – Enlarged depiction of the plot shown in the middle of Figure 6.10-a.

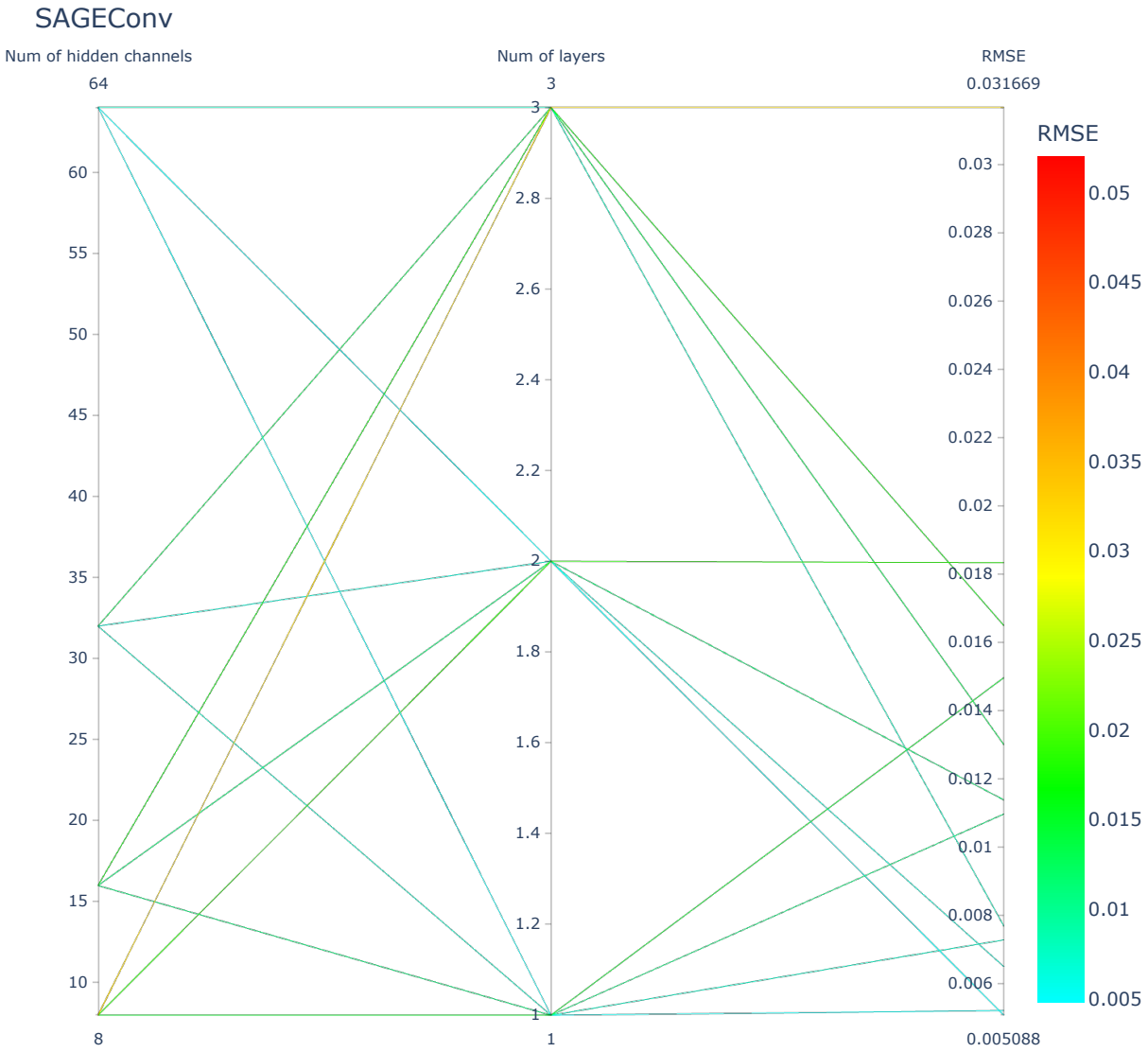


Figure A.3 – Enlarged depiction of the plot shown on the right-hand side in Figure 6.10-a.

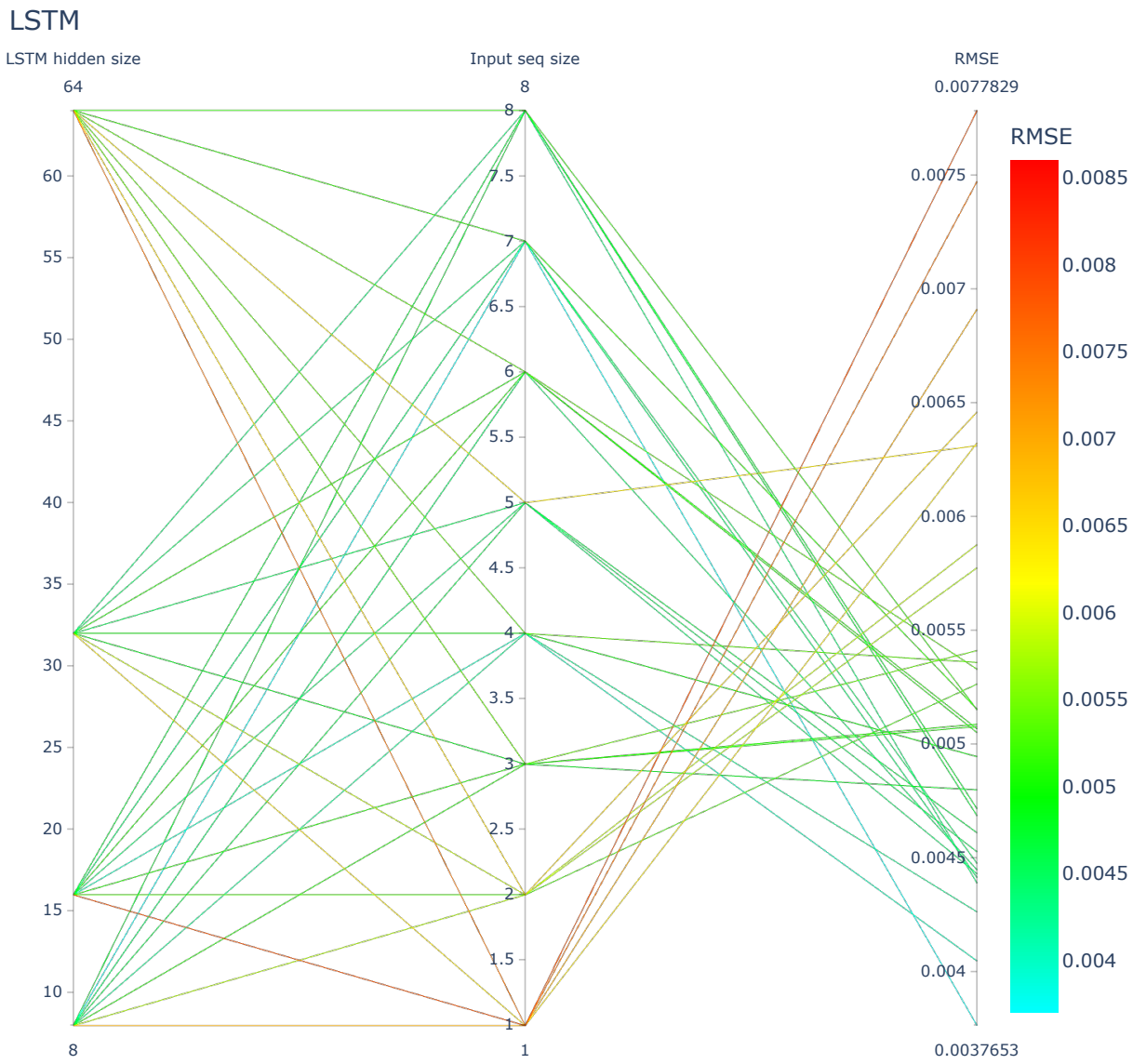


Figure A.4 – Enlarged depiction of the plot shown on the left-hand side in Figure 6.10-b.

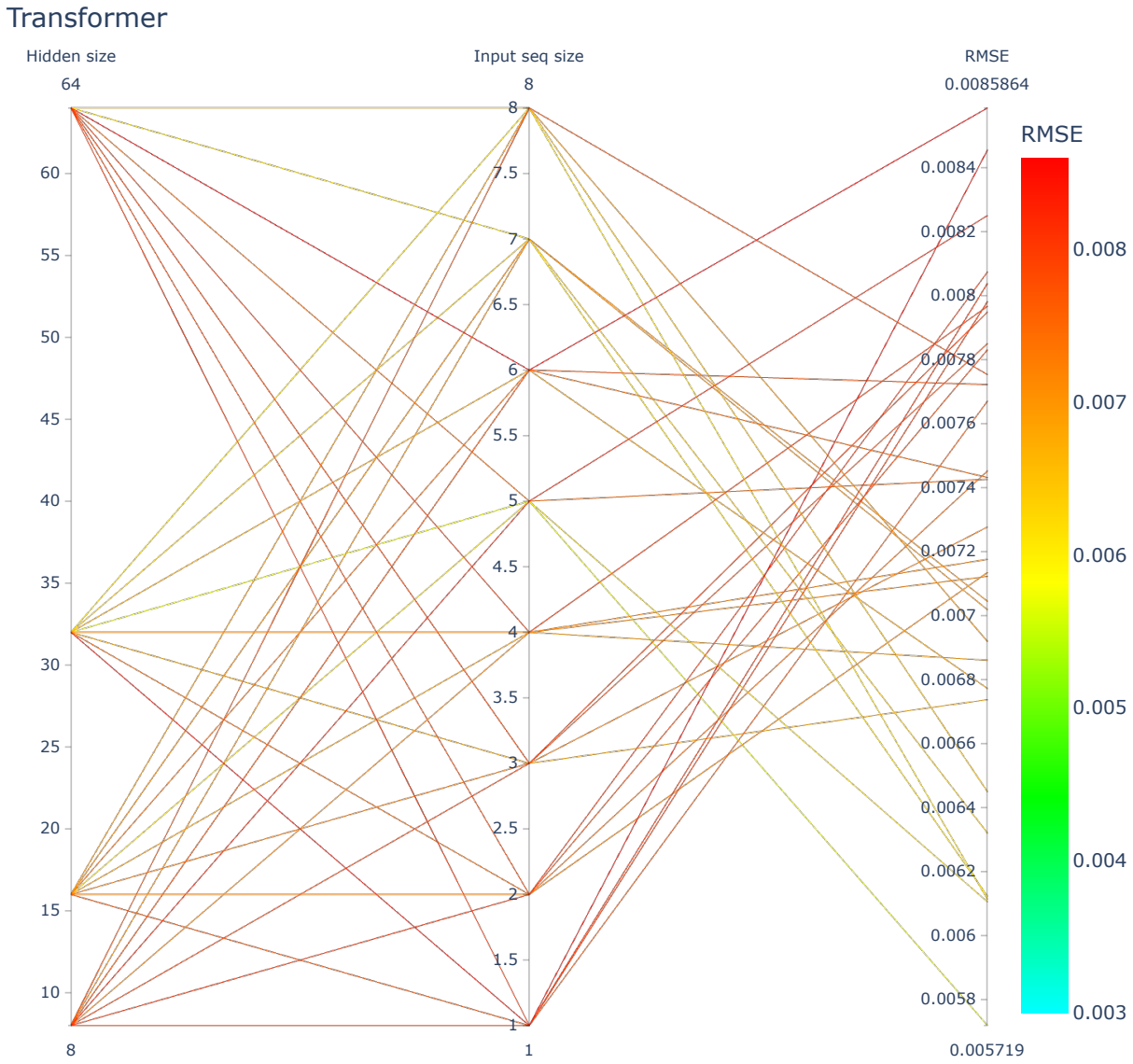


Figure A.5 – Enlarged depiction of the plot shown on the right-hand side shown in Figure 6.10-b.

Appendix B

Adaptive Imitation Reward Function

The following is an extension to the experiments about adaptive reward function presented in Chapter 7, Section 7.5.

Algo 5 details two other slightly different versions V2.0 and V2.1 of the adaptive reward function V2 presented in Section 7.5.2:

— V2.0: updates ε if the total number of non-zero rewards of a single episode is greater or equal to $ratio \times length(Episode)$.

— V2.1: updates ε based on the average of non-zero rewards over a queue of $n = 20$ recent episodes, and each time ε is updated, only the oldest item in the queue is removed.

Algo 6 details another variant of the adaptive reward function V3, introduced in Section 7.5.2, where only the oldest timestep in the last hw_{size} is removed when ε is updated. The results are shown in

The results are shown in Fig B.1, Fig B.2 and Fig. B.3 for reward functions V2.0, V2.1, and V3.1 respectively.

Algorithm 5: Adaptive Imitation Reward Function V2.0 & V2.1

Input: ε_{min}, ε_{max}, ε_{step}, $ratio$
$\varepsilon \leftarrow \varepsilon_{max}$;
while *Training is not done* **do**
  while *Episode is not done* **do**
    $a \leftarrow \pi(s_t)$;
    $(s_{t+1}, r_t, done) \leftarrow$ *excute the action* a;
    (r_t `is calculated according to Eq 7.12 using the current` ε)
  end
  begin Update ε **for Reward 2.0**
    if *Number of non-zero rewards in Episode* $\geq ratio \times length(Episode)$ **then**
      $\varepsilon \leftarrow \varepsilon - \varepsilon_{step}$
    end
  end
  begin Update ε **for Reward 2.1**
    Stack episodes in a queue Q of size $n = 20$;
    if *Q is full* ***and*** *number of non-zero rewards in Q* $\geq ratio \times length(Episodes\ in\ Q)$ **then**
      $\varepsilon \leftarrow \varepsilon - \varepsilon_{step}$;
      Remove only the oldest episode in Q ;
    end
  end
end

Algorithm 6: Adaptive Imitation Reward Function V3.1

Input: ε_{max}, $ratio$, hw_{size}
$\varepsilon \leftarrow \varepsilon_{max}$;
while *Training is not done* **do**
  while *Episode is not done* **do**
    $a \leftarrow \pi(s_t)$;
    $(s_{t+1}, r_t, done) \leftarrow$ *excute the action* a;
    (r_t `is calculated according to Eq 7.12 using the current` ε)
    begin Update ε **for Reward 3.1**
      Stack $d = \|s_t - s'_t\|_2$ in a queue Q of size $N = hw_{size}$;
      if *Q is full* ***and*** $size(\{d,\ d \in Q\ and\ d \leq \varepsilon\}) \geq ratio \times hw_{size}$ **then**
        $\varepsilon \leftarrow median(\{d, d \in Q\ and\ d \geq \varepsilon\})$;
        Remove the oldest time-step in Q ;
      end
    end
  end
end

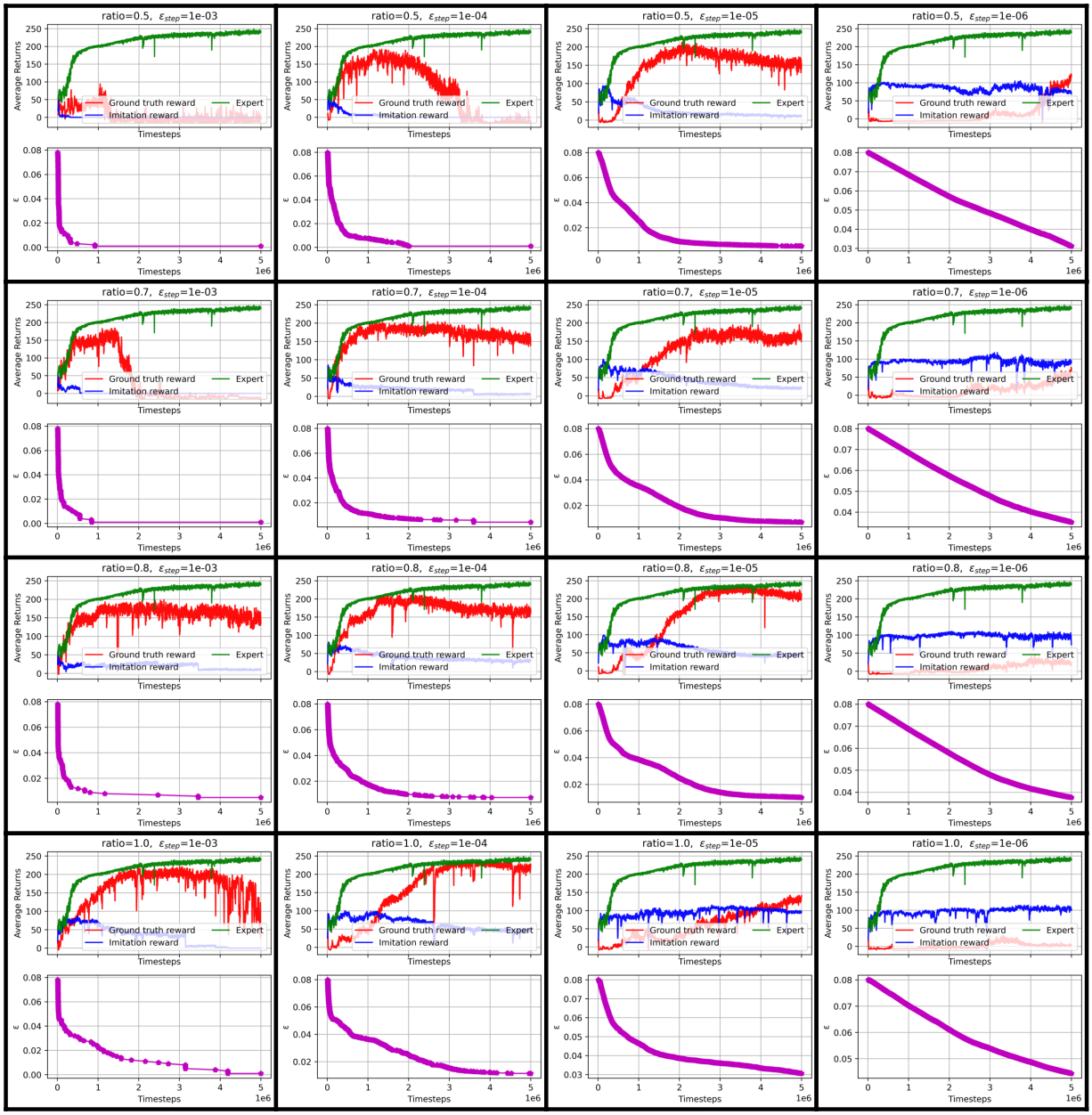


Figure B.1 – Results for Reward "V2.0". Each cell in the grid depicts the training performance and the evolution of ε for different values of ε_{step} and *ratio*. $\varepsilon_{step} \in \{1e-3, 1e-4, 1e-5, 1e-6\}$ and $ratio \in \{0.5, 0.7, 0.8, 1\}$.

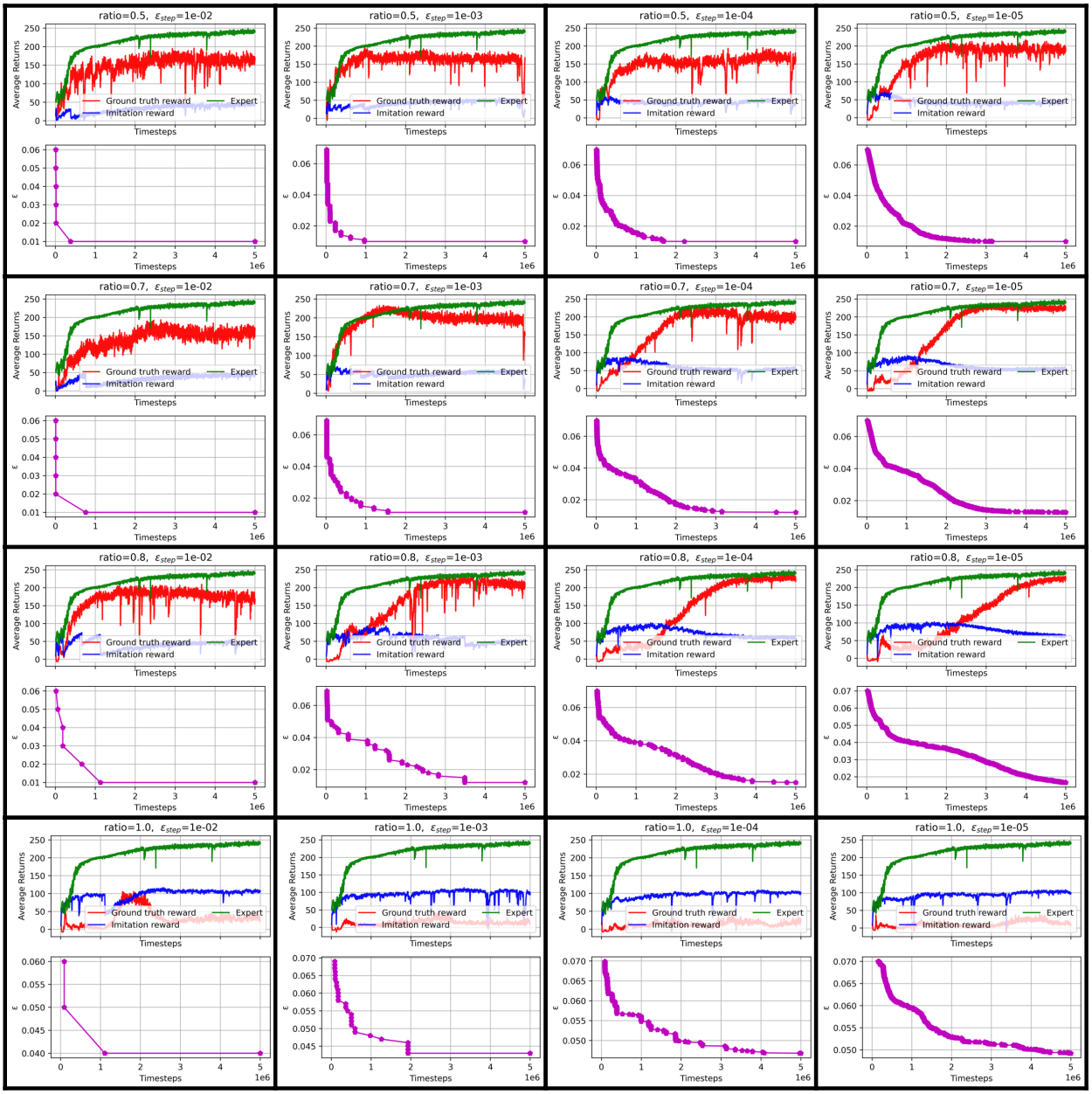


Figure B.2 – Results for Reward "V2.1". Each cell in the grid depicts the training performance and the evolution of ε for different values of ε_{step} and *ratio*. $\varepsilon_{step} \in \{1e-2, 1e-3, 1e-4, 1e-5\}$ and $ratio \in \{0.5, 0.7, 0.8, 1\}$.

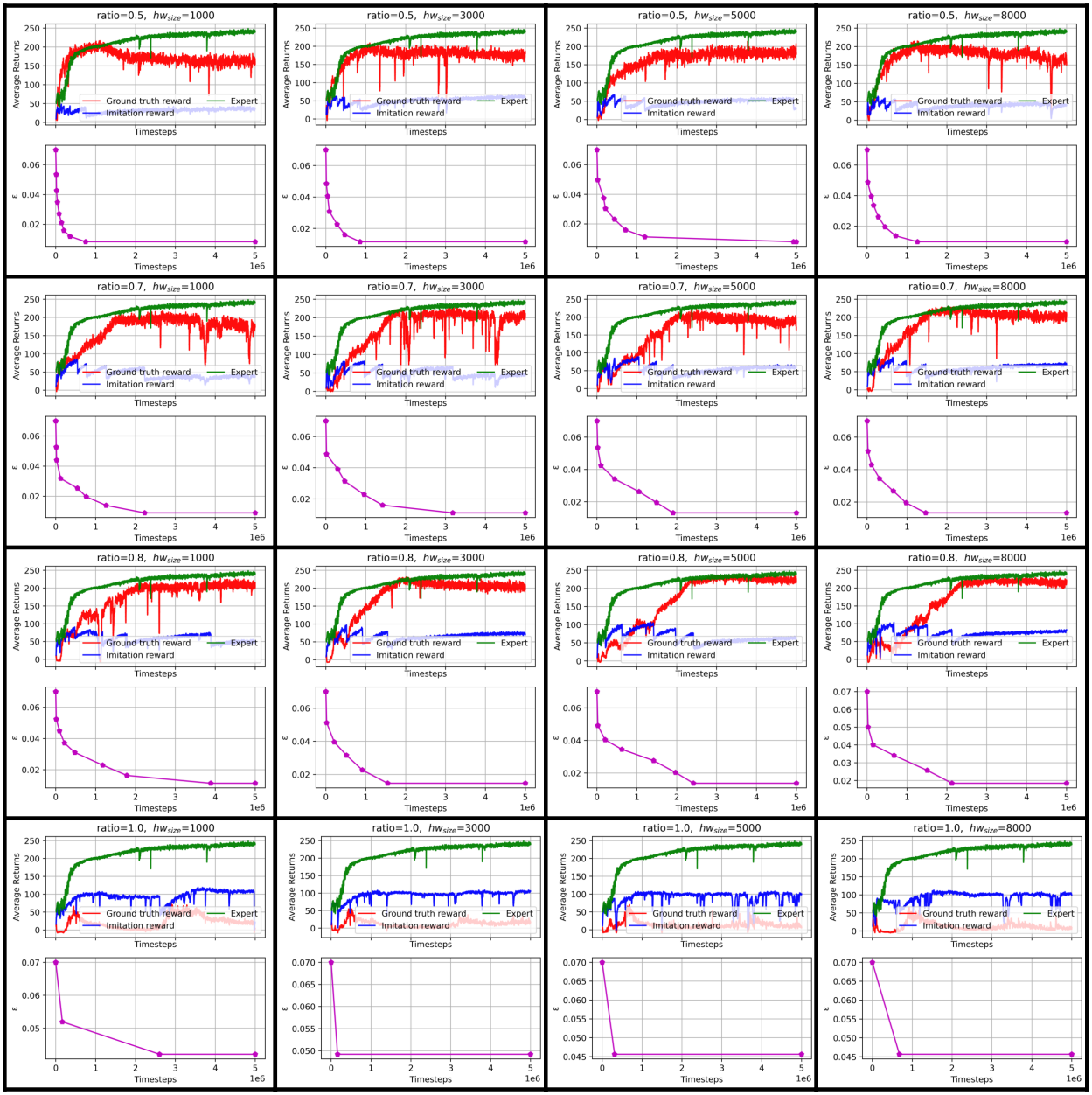


Figure B.3 – Results for Reward "V3.1". Each cell in the grid depicts the training performance and the evolution of ε for different values of hw_{size} and $ratio$. $\varepsilon_{step} \in \{1e3, 3e3, 5e3, 8e3\}$ and $ratio \in \{0.5, 0.7, 0.8, 1\}$.

Appendix C

Meta-World Benchmark

C.1 Meta-world Benchmark Tasks

Task label	Description	N. objs
assembly-v2	Pick up a nut and place it onto a peg	2
basketball-v2	Dunk the basketball into the basket	2
bin-picking-v2	Grasp the puck from one bin and place it into another bin	2
box-close-v2	Grasp the cover and close the box with it	2
button-press-topdown-v2 *	Press a button from the top	1
button-press-topdown-wall-v2 *	Bypass a wall and press a button from the top	1
button-press-v2 *	Press a button	1
button-press-wall-v2 *	Bypass a wall and press a button	2
coffee-button-v2 *	Push a button on the coffee machine	1
coffee-pull-v2	Pull a mug from a coffee machine	2
coffee-push-v2	Push a mug under a coffee machine	2
dial-turn-v2 *	Rotate a dial 180 degrees	1
disassemble-v2	pick a nut out of a peg	2
door-close-v2 *	Close a door with a revolving joint	1
door-lock-v2 *	Lock the door by rotating the lock clockwise	1
door-open-v2 *	Open a door with revolving joint	1
door-unlock-v2 *	Unlock the door by rotating the lock counter-clockwise	1
drawer-close-v2 *	Push and close a drawer	1
drawer-open-v2 *	Open a drawer	1
faucet-close-v2 *	Rotate the faucet clockwise	1
faucet-open-v2 *	Rotate the faucet counter-clockwise	1
hammer-v2 *	Hammer a screw on the wall	2
hand-insert-v2	Insert an object the into a hole	1
handle-press-side-v2 *	Press a handle down sideways	1
handle-press-v2 *	Press a handle down	1
handle-pull-side-v2 *	Pull a handle up sideways	1
handle-pull-v2 *	Pull a handle up	1
lever-pull-v2 *	Pull a lever down 90 degrees	1
peg-insert-side-v2 *	Insert a peg sideways	2
peg-unplug-side-v2	Unplug a peg sideways	2
pick-out-of-hole-v2 *	Pick up a puck from a hole	1
pick-place-v2	Pick and place a puck to a goal	1
pick-place-wall-v2 *	Pick a puck, bypass a wall and place the puck	2
plate-slide-back-side-v2 *	Get a plate from the cabinet sideways	2
plate-slide-back-v2 *	Get a plate from the cabinet	2
plate-slide-side-v2 *	Slide a plate into a cabinet sideways	2
plate-slide-v2 *	Slide a plate into a cabinet	2
push-back-v2 *	Pull a puck to a goal	1
push-v2	Push the puck to a goal	1
push-wall-v2 *	Bypass a wall and push a puck to a goal	2
reach-v2	Reach a goal position	0
reach-wall-v2	Bypass a wall and reach a goal	1
shelf-place-v2	Pick and place a puck onto a shelf	2
soccer-v2	Kick a soccer into the goal	2
stick-pull-v2	Grasp a stick and pull a box with the stick.	2
stick-push-v2 *	Grasp a stick and push a box using the stick	2
sweep-into-v2	Sweep a puck into a hole	1
sweep-v2 *	Sweep a puck off the table	1
window-close-v2 *	Push and close a window	1
window-open-v2 *	Push and open a window	1

Table C.1 – The list of the Meta-World benchmark tasks. The tasks marked with an asterisk are those used in the generalisation experiments of our imitation learning approach in Chapter 8.

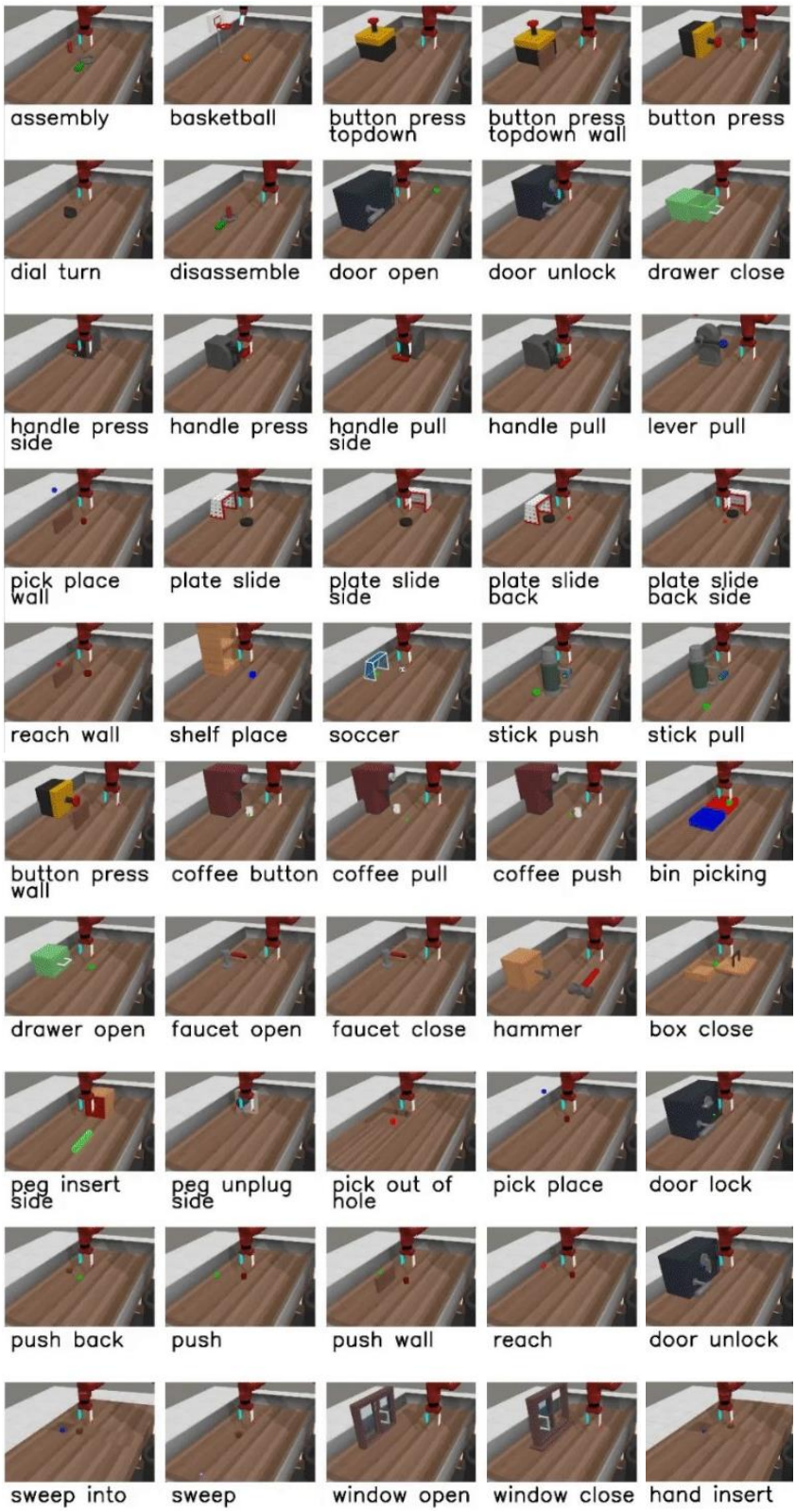


Figure C.1 – Depiction of the Meta-World benchmark environments.

C.2 Expert-RL training

Description	**Detail**
Environment Hyperparameters	
Episode maximum length	450
seed	17
Algorithm Hyperparameters	
Learning rate	3e-4
Buffer size	1e6
Batch size	256
Discount factor	0.99
Soft update coefficient	0.005
Gradient steps per rollout	1
Neural Network Hyperparameters	
Hidden layers	(128,128)
Activation function	ReLu

Table C.2 – The Expert-RL training hyperparameters using SAC method.

C.2.1 Using only the current observation as input

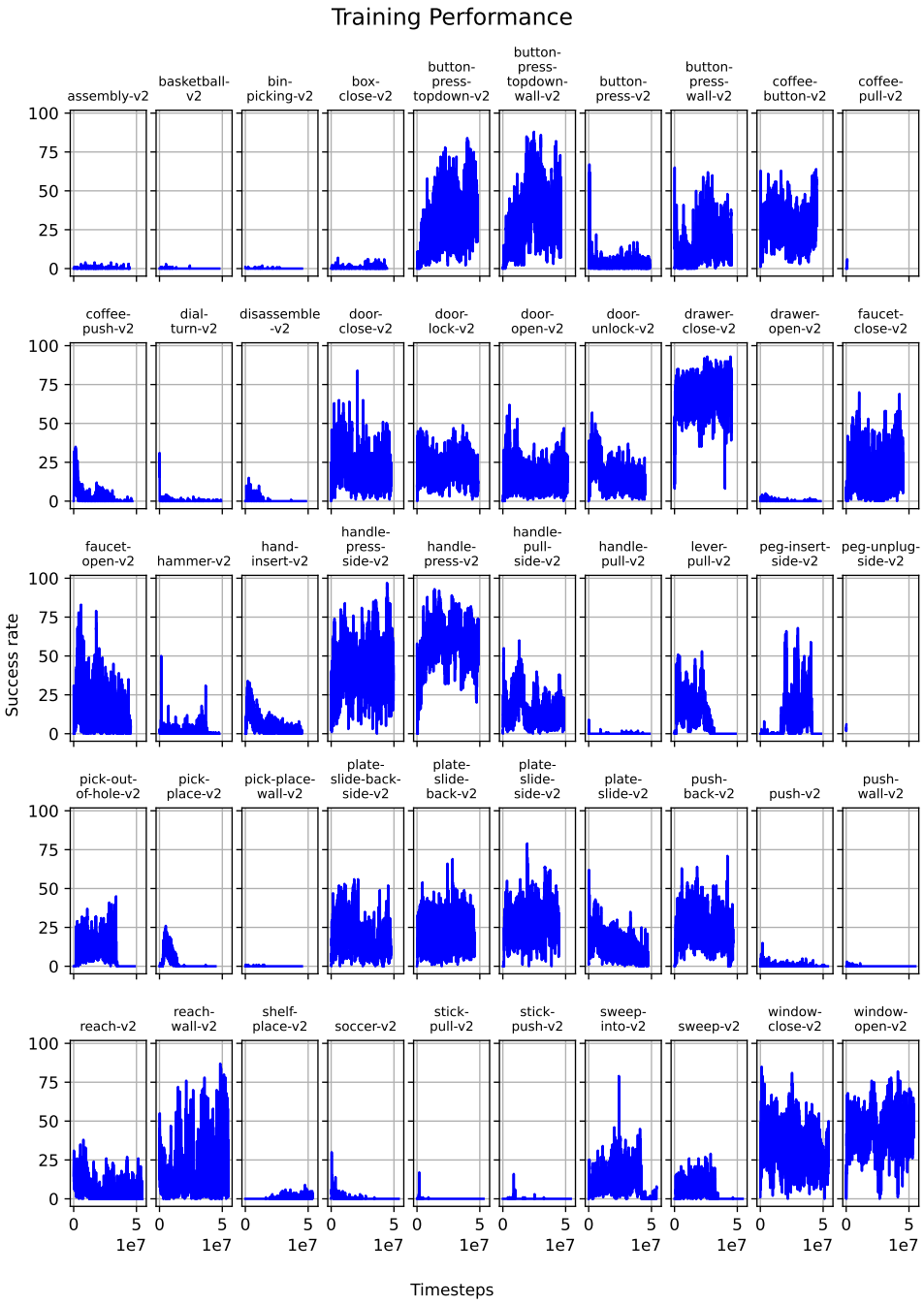


Figure C.2 – Training performance of RL-Expert models trained on the ground truth reward functions of each task in the Meta-World benchmark. The input of the policy is limited to the current state of objects at each time-step.

C.2.2 Using the previous and current observation as input

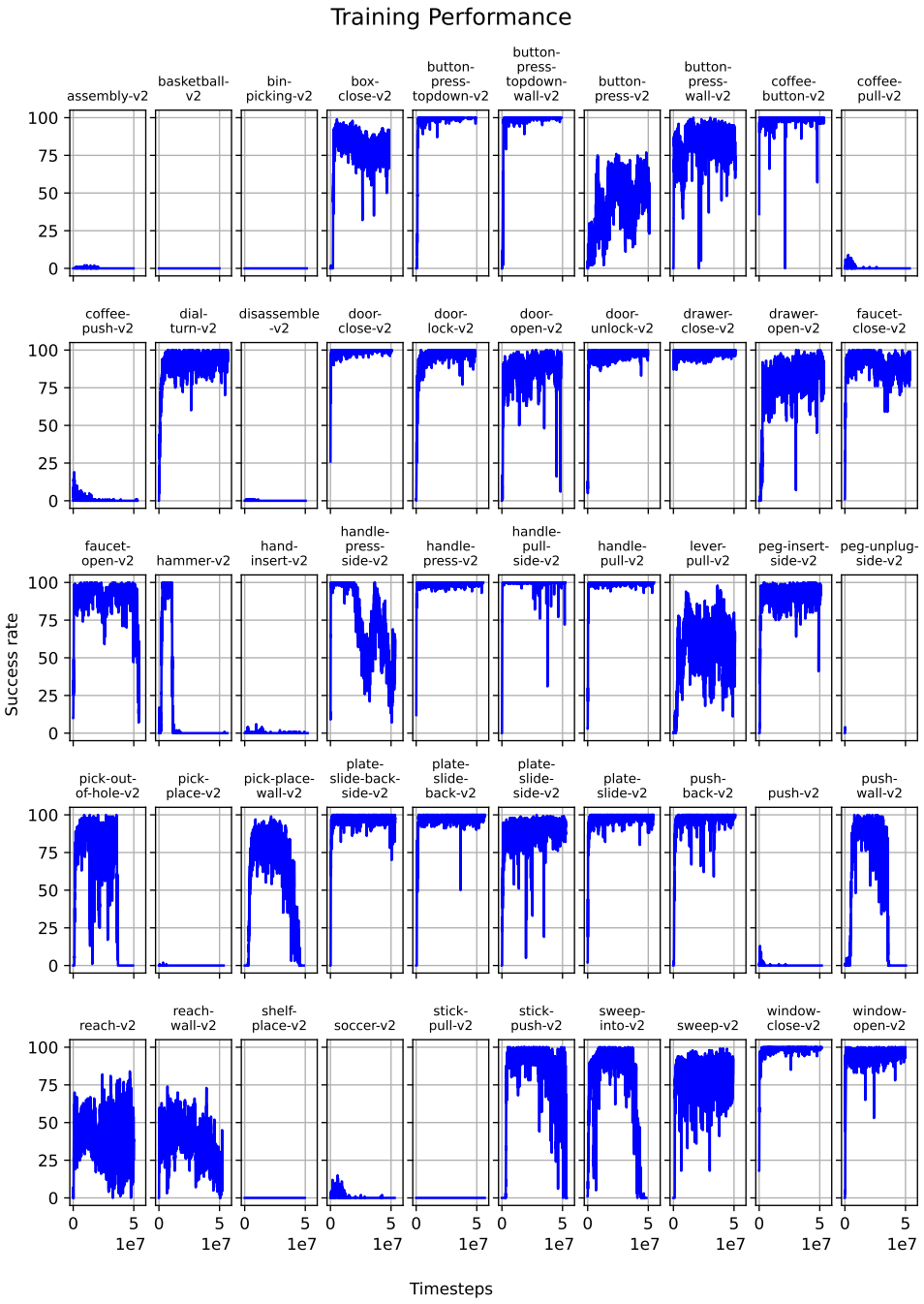


Figure C.3 – Training performance of RL-Expert models trained on the ground truth reward functions of each task in the Meta-World benchmark. The input of the policy consists of the current and previous state of objects at each time-step.

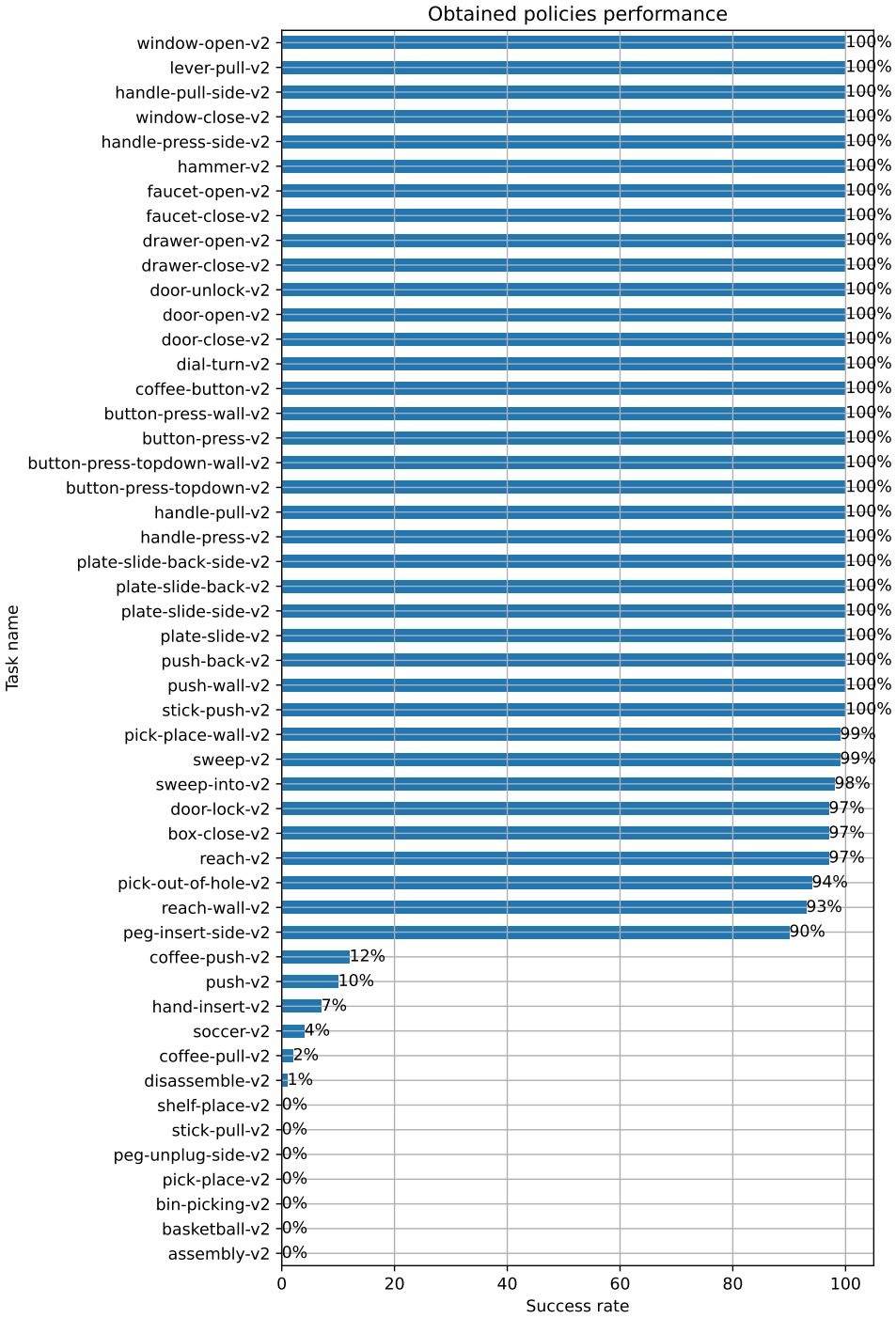


Figure C.4 – Success rate of trained RL-Expert models.

C.3 Demonstrations

	Number of time-steps in demonstrations			
env_name	mean	std	min	max
button-press-topdown-v2	58.6	4.7	51	68
button-press-topdown-wall-v2	58.2	5.1	50	70
button-press-v2	39.8	1.9	37	43
button-press-wall-v2	38.2	1.8	35	42
coffee-button-v2	24.8	3.2	19	30
dial-turn-v2	29.7	1.6	27	35
door-close-v2	51.9	3.5	46	59
door-lock-v2	27.2	5.7	18	44
door-open-v2	72.2	4.1	65	81
door-unlock-v2	34.8	15.9	26	120
drawer-close-v2	16.5	0.5	15	18
drawer-open-v2	42.8	1.0	42	46
faucet-close-v2	46.3	2.5	42	53
faucet-open-v2	45.7	2.2	42	51
hammer-v2	50.2	6.4	39	75
handle-press-side-v2	25.0	24.4	16	299
handle-press-v2	19.3	3.9	12	30
handle-pull-side-v2	28.1	0.8	27	30
handle-pull-v2	38.0	5.7	35	87
lever-pull-v2	48.0	3.7	41	55
peg-insert-side-v2	52.3	3.6	47	64
pick-out-of-hole-v2	72.1	23.9	59	325
pick-place-wall-v2	56.4	7.5	46	98
plate-slide-back-side-v2	37.4	6.1	26	48
plate-slide-back-v2	33.5	0.7	32	34
plate-slide-side-v2	41.3	4.4	37	86
plate-slide-v2	48.4	2.8	43	55
push-back-v2	55.3	3.8	50	67
push-wall-v2	59.9	13.7	48	162
stick-push-v2	55.6	2.2	51	61
sweep-v2	74.6	6.0	66	93
window-close-v2	50.0	4.1	43	62
window-open-v2	47.3	7.0	34	62

Table C.3 – The statistics of the demonstrations generated by the RL-Expert models.

C.4 Representation Learning Experiments

	Input sequence size = 3			Input sequence size =7		
env_name	RMSE	MAE	Max_error	RMSE	MAE	Max_error
button-press-topdown-v2	4.9	2.8	73.9	3.6	2.3	27.1
button-press-topdown-wall-v2	4.2	2.9	32.7	3.1	2.3	18.5
button-press-v2	1.4	1.0	8.0	1.1	0.8	8.5
button-press-wall-v2	1.2	0.8	8.0	1.0	0.7	5.8
coffee-button-v2	1.4	1.0	9.1	1.3	0.8	9.0
dial-turn-v2	2.2	1.4	16.8	2.2	1.3	19.5
door-close-v2	1.3	0.9	9.2	1.1	0.8	7.8
door-lock-v2	1.9	1.4	12.2	1.7	1.1	11.9
door-open-v2	2.3	1.5	21.2	2.2	1.3	32.8
door-unlock-v2	5.9	2.8	53.5	6.5	2.8	61.3
drawer-close-v2	1.0	0.7	5.1	1.2	0.9	8.9
drawer-open-v2	1.1	0.7	8.4	1.1	0.7	11.3
faucet-close-v2	2.2	1.4	14.3	2.2	1.5	12.8
faucet-open-v2	1.8	1.2	12.0	1.6	1.1	9.1
hammer-v2	2.6	1.5	17.9	2.3	1.4	18.9
handle-press-side-v2	4.2	2.2	49.4	4.1	2.3	45.1
handle-press-v2	3.0	2.1	16.1	2.9	2.0	14.0
handle-pull-side-v2	1.5	1.0	13.7	1.3	0.8	9.6
handle-pull-v2	1.6	1.1	13.5	1.5	1.1	11.3
lever-pull-v2	1.6	1.1	12.2	1.5	1.0	9.2
peg-insert-side-v2	3.2	2.1	22.3	3.2	2.2	18.3
pick-out-of-hole-v2	3.5	2.1	47.9	3.0	1.9	32.3
pick-place-wall-v2	4.4	2.7	38.0	3.3	2.0	38.8
plate-slide-back-side-v2	0.7	0.5	4.4	0.7	0.5	3.3
plate-slide-back-v2	0.8	0.6	4.6	0.6	0.4	2.5
plate-slide-side-v2	1.4	1.0	11.2	1.3	0.9	7.7
plate-slide-v2	1.3	0.9	7.6	1.2	0.8	5.7
push-back-v2	1.9	1.3	13.2	1.6	1.1	13.3
push-wall-v2	2.2	1.5	15.4	1.9	1.2	14.9
stick-push-v2	1.6	1.0	11.0	1.5	0.9	10.3
sweep-v2	2.0	1.4	12.2	1.4	0.9	7.4
window-close-v2	1.1	0.7	11.4	1.1	0.7	11.9
window-open-v2	2.4	1.6	12.4	2.3	1.5	15.2

Table C.4 – The representation learning results. Values are given in millimetres and can be also interpreted as a percentage of the maximal opening of the gripper, which is equivalent to $100mm$.

C.5 Imitation Learning Experiments

Description	Detail
Algorithm Hyperparameters	
Learning rate	3e-4
Buffer size	1e6
Batch size	256
Discount factor	0.99
Soft update coefficient	0.005
Gradient steps per rollout	1
Neural Network Hyperparameters	
Hidden layers	(256,256)
Activation function	ReLu

Table C.5 – The RL model training hyperparameters.

BIBLIOGRAPHY

[1] Richard S. Sutton and Andrew G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed., MIT Press, 2018.

[2] Abhijit Gosavi, « Reinforcement Learning: A Tutorial Survey and Recent Advances », *in*: *INFORMS Journal on Computing* 21 (2009), pp. 178–192.

[3] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller, « Playing Atari with Deep Reinforcement Learning », *in*: *arXiv:1312.5602 [cs]* (2013).

[4] Xiang Li, Weiwei Shang, and Shuang Cong, « Model-Based Reinforcement Learning For Robot Control », *in*: *International Conference on Advanced Robotics and Mechatronics (ICARM)*, IEEE, 2020, pp. 300–305.

[5] Pieter Abbeel, Morgan Quigley, and Andrew Y. Ng, « Using inaccurate models in reinforcement learning », *in*: *Proceedings of the 23rd international conference on Machine learning - ICML '06*, ACM Press, 2006, pp. 1–8.

[6] Anusha Nagabandi, Gregory Kahn, S. Fearing Fearing, and Sergey Levine, « Neural Network Dynamics for Model-Based Deep Reinforcement Learning with Model-Free Fine-Tuning », *in*: *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 7559–7566.

[7] Christopher G. Atkeson and Juan Carlos Santamaria, « A comparison of direct and model-based reinforcement learning », *in*: *Proceedings of International Conference on Robotics and Automation*, vol. 4, 1997, 3557–3564 vol.4.

[8] Xiaogang Ruan, Peng Li, Xiaoqing Zhu, Hejie Yu, and Naigong Yu, « End-to-End Autonomous Exploration with Deep Reinforcement Learning and Intrinsic Motivation », *in*: *Computational Intelligence and Neuroscience* 2021 (2021), pp. 1–15.

[9] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra, « Continuous control with deep reinforcement learning », *in*: *arXiv:1509.02971 [cs, stat]* (2019).

[10] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine, « Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor », *in*: *arXiv:1801.01290 [cs, stat]* (2018).

[11] Marc Peter Deisenroth, « A Survey on Policy Search for Robotics », *in*: *Foundations and Trends in Robotics* 2 (2011), pp. 1–142.

[12] Junzi Zhang, Jongho Kim, Brendan O'Donoghue, and Stephen Boyd, « Sample Efficient Reinforcement Learning with REINFORCE », *in*: *Proceedings of the AAAI Conference on Artificial Intelligence* 35 (2021), pp. 10887–10895.

[13] Nicolas Heess, Gregory Wayne, David Silver, Timothy Lillicrap, Tom Erez, and Yuval Tassa, « Learning Continuous Control Policies by Stochastic Value Gradients », *in*: *Advances in Neural Information Processing Systems*, vol. 28, Curran Associates, Inc., 2015.

[14] Oriol Vinyals, Igor Babuschkin, Wojciech M. Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H. Choi, Richard Powell, Timo Ewalds, Petko Georgiev, Junhyuk Oh, Dan Horgan, Manuel Kroiss, Ivo Danihelka, Aja Huang, Laurent Sifre, Trevor Cai, John P. Agapiou, Max Jaderberg, Alexander S. Vezhnevets, Rémi Leblond, Tobias Pohlen, Valentin Dalibard, David Budden, Yury Sulsky, James Molloy, Tom L. Paine, Caglar Gulcehre, Ziyu Wang, Tobias Pfaff, Yuhuai Wu, Roman Ring, Dani Yogatama, Dario Wünsch, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy Lillicrap, Koray Kavukcuoglu, Demis Hassabis, Chris Apps, and David Silver, « Grandmaster level in StarCraft II using multi-agent reinforcement learning », *in*: *Nature* 575 (2019), pp. 350–354.

[15] Shai Shalev-Shwartz, Shaked Shammah, and Amnon Shashua, « Safe, Multi-Agent, Reinforcement Learning for Autonomous Driving », *in*: *arXiv:1610.03295 [cs, stat]* (2016).

[16] Jens Kober, J. Andrew Bagnell, and Jan Peters, « Reinforcement learning in robotics: A survey », *in*: *The International Journal of Robotics Research* 32 (2013), pp. 1238–1274.

[17] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov, « Proximal Policy Optimization Algorithms », *in*: *arXiv:1707.06347 [cs]* (2017).

[18] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz, « Trust Region Policy Optimization », *in*: *Proceedings of the 32nd International Conference on Machine Learning*, PMLR, 2015, pp. 1889–1897.

[19] Matthias Plappert, Rein Houthooft, Prafulla Dhariwal, Szymon Sidor, Richard Y. Chen, Xi Chen, Tamim Asfour, Pieter Abbeel, and Marcin Andrychowicz, « Parameter Space Noise for Exploration », *in*: *arXiv:1706.01905 [cs, stat]* (2017).

[20] Stephen Dankwa and Wenfeng Zheng, « Twin-Delayed DDPG: A Deep Reinforcement Learning Technique to Model a Continuous Movement of an Intelligent Robot Agent », *in*: *Proceedings of the 3rd International Conference on Vision, Image and Signal Processing*, ACM, 2019, pp. 1–5.

[21] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine, « Soft Actor-Critic Algorithms and Applications », *in*: *arXiv:1812.05905 [cs, stat]* (2019).

[22] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel, « End-to-End Training of Deep Visuomotor Policies », *in*: *Journal of Machine Learning Research* 17 (2016), pp. 1–40.

[23] Chen Chen, Hsieh-Yu Li, Xuewen Zhang, Xiang Liu, and U-Xuan Tan, « Towards Robotic Picking of Targets with Background Distractors using Deep Reinforcement Learning », *in*: *2019 WRC Symposium on Advanced Robotics and Automation (WRC SARA)*, 2019, pp. 166–171.

[24] Yuchen Xiao, Sammie Katt, Andreas ten Pas, Shengjian Chen, and Christopher Amato, « Online Planning for Target Object Search in Clutter under Partial Observability », *in*: *2019 International Conference on Robotics and Automation (ICRA)*, 2019, pp. 8241–8247.

[25] Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, and Sergey Levine, « QT-Opt: Scalable Deep Reinforcement Learning for Vision-Based Robotic Manipulation », *in*: *arXiv:1806.10293 [cs, stat]* (2018).

[26] Shixiang Gu, Ethan Holly, Timothy Lillicrap, and Sergey Levine, « Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates »,

in: *International Conference on Robotics and Automation (ICRA)*, IEEE, 2017, pp. 3389–3396.

[27] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu, « Asynchronous Methods for Deep Reinforcement Learning », *in*: *Proceedings of The 33rd International Conference on Machine Learning*, PMLR, 2016, pp. 1928–1937.

[28] Mustafa Khaled, Beril Sirmaçek, Stefano Stramigioli, and Mannes Poel, « Towards Continuous Ccontrol For Mobile Robot Navigation: A Reinforcement Learning and Slam based Approach », *in*: *ISPRS Workshop Indoor 3D 2019*, 2019.

[29] Lei Tai, Giuseppe Paolo, and Ming Liu, « Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation », *in*: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017, pp. 31–36.

[30] Xuyi Qiu, Kaifang Wan, and Fusong Li, « Autonomous Robot Navigation in Dynamic Environment Using Deep Reinforcement Learning », *in*: *2019 IEEE 2nd International Conference on Automation, Electronics and Electrical Engineering (AUTEEE)*, 2019, pp. 338–342.

[31] Yuke Zhu, Roozbeh Mottaghi, Eric Kolve, Joseph J. Lim, Abhinav Gupta, Li Fei-Fei, and Ali Farhadi, « Target-driven visual navigation in indoor scenes using deep reinforcement learning », *in*: *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 3357–3364.

[32] Ahmad El Sallab, Mohammed Abdou, Etienne Perot, and Senthil Yogamani, « Deep Reinforcement Learning framework for Autonomous Driving », *in*: *Electronic Imaging* 29 (2017), pp. 70–76.

[33] Xinlei Pan, Yurong You, Ziyan Wang, and Cewu Lu, « Virtual to Real Reinforcement Learning for Autonomous Driving », *in*: *arXiv:1704.03952 [cs]* (2017).

[34] Patrick Wenzel, Torsten Schon, Laura Leal-Taixe, and Daniel Cremers, « Vision-Based Mobile Robotics Obstacle Avoidance With Deep Reinforcement Learning », *in*: *International Conference on Robotics and Automation (ICRA)*, IEEE, 2021, pp. 14360–14366.

[35] Guoguang Du, Kai Wang, Shiguo Lian, and Kaiyong Zhao, « Vision-based robotic grasping from object localization, object pose estimation to grasp estimation for parallel grippers: a review », *in*: *Artificial Intelligence Review* 54 (2021), pp. 1677–1734.

[36] Heli T. Hytti, « Characterization of digital image noise properties based on RAW data », *in*: *Image Quality and System Performance III*, vol. 6059, SPIE, 2006, pp. 86–97.

[37] Cecilia Aguerrebere, Julie Delon, Yann Gousseau, and Pablo Musé, *Study of the digital camera acquisition process and statistical modeling of the sensor raw data*, 2013, URL: `https://hal.archives-ouvertes.fr/hal-00733538`.

[38] Jorge Igual, « Photographic Noise Performance Measures Based on RAW Files Analysis of Consumer Cameras », *in*: *Electronics* 8 (2019), p. 1284.

[39] Taha Hasan Masood Siddique and Muhammad Usman, « 3D Object Localization Using 2D Estimates for Computer Vision Applications », *in*: *2021 Mohammad Ali Jinnah University International Conference on Computing (MAJICC)* (2021).

[40] Wenshuai Zhao, Jorge Peña Queralta, and Tomi Westerlund, « Sim-to-Real Transfer in Deep Reinforcement Learning for Robotics: a Survey », *in*: *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, 2020, pp. 737–744.

[41] Joshua 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*: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2017).

[42] Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel, « Sim-to-Real Transfer of Robotic Control with Dynamics Randomization », *in*: *2018 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2018, pp. 3803–3810.

[43] Yevgen Chebotar, Ankur Handa, Viktor Makoviychuk, Miles Macklin, Jan Issac, Nathan Ratliff, and Dieter Fox, « Closing the Sim-to-Real Loop: Adapting Simulation Randomization with Real World Experience », *in*: *2019 International Conference on Robotics and Automation (ICRA)*, 2019, pp. 8973–8979.

[44] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann, « Stable-Baselines3: Reliable Reinforcement Learning Implementations », *in*: *Journal of Machine Learning Research* 22 (2021), pp. 1–8.

[45] *Advanced Realtime Tracking GmbH & Co. KG*, URL: `https://ar-tracking.com/`.

[46] *Stereolabs*, URL: `https://www.stereolabs.com/`.

[47] Yassine EL Manyari, Laurent Dollé, and Patrick Le Callet, « Noisy Localization of Objects in Reinforcement Learning Framework: An Experimental Case Study on a Pushing Task », *in*: *2022 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, 2022, pp. 79–84.

[48] Alexander Pan, Kush Bhatia, and Jacob Steinhardt, « The Effects of Reward Misspecification: Mapping and Mitigating Misaligned Models », *in*: *arXiv:2201.03544 [cs, stat]* (2022).

[49] YuXuan Liu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine, « Imitation from Observation: Learning to Imitate Behaviors from Raw Video via Context Translation », *in*: *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 1118–1125.

[50] Zuyuan Zhu and Huosheng Hu, « Robot Learning from Demonstration in Robotic Assembly: A Survey », *in*: *Robotics* 7 (2018), p. 17.

[51] Yusuf Aytar, Tobias Pfaff, David Budden, Thomas Paine, Ziyu Wang, and Nando de Freitas, « Playing hard exploration games by watching YouTube », *in*: *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[52] Raunak P. Bhattacharyya, Derek J. Phillips, Changliu Liu, Jayesh K. Gupta, Katherine Driggs-Campbell, and Mykel J. Kochenderfer, « Simulating Emergent Properties of Human Driving Behavior Using Multi-Agent Reward Augmented Imitation Learning », *in*: *arXiv:1903.05766 [cs]* (2019).

[53] Ahmed Hussein, Mohamed Medhat Gaber, Eyad Elyan, and Chrisina Jayne, « Imitation Learning: A Survey of Learning Methods », *in*: *ACM Computing Surveys* 50 (2017), pp. 1–35.

[54] Boyuan Zheng, Sunny Verma, Jianlong Zhou, Ivor Tsang, and Fang Chen, « Imitation Learning: Progress, Taxonomies and Opportunities », *in*: *arXiv:2106.12177 [cs]* (2021).

[55] Takayuki Osa, Joni Pajarinen, Gerhard Neumann, J. Andrew Bagnell, Pieter Abbeel, and Jan Peters, « An Algorithmic Perspective on Imitation Learning », *in*: *Foundations and Trends in Robotics* 7 (2018), pp. 1–179.

[56] Stephane Ross, Geoffrey Gordon, and Drew Bagnell, « A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning », *in*: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, JMLR Workshop and Conference Proceedings, 2011, pp. 627–635.

[57] Stephane Ross, Narek Melik-Barkhudarov, Kumar Shaurya Shankar, Andreas Wendel, Debadeepta Dey, J. Andrew Bagnell, and Martial Hebert, « Learning monocular reactive UAV control in cluttered natural environments », *in*: *2013 IEEE International Conference on Robotics and Automation* (2013), pp. 1765–1772.

[58] Saurabh Arora and Prashant Doshi, « A Survey of Inverse Reinforcement Learning: Challenges, Methods and Progress », *in*: *arXiv:1806.06877 [cs, stat]* (2020).

[59] Dizan Vasquez, Billy Okal, and Kai O. Arras, « Inverse Reinforcement Learning algorithms and features for robot navigation in crowds: An experimental comparison », *in*: *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2014, pp. 1341–1346.

[60] Pieter Abbeel and Andrew Y. Ng, « Apprenticeship learning via inverse reinforcement learning », *in*: *Proceedings of the twenty-first international conference on Machine learning*, 2004, p. 1.

[61] Nathan D. Ratliff, J. Andrew Bagnell, and Martin A. Zinkevich, « Maximum margin planning », *in*: *Proceedings of the 23rd international conference on Machine learning*, 2006, pp. 729–736.

[62] Navid Aghasadeghi and Timothy Bretl, « Maximum entropy inverse reinforcement learning in continuous state spaces with path integrals », *in*: *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2011, pp. 1561–1566.

[63] Brian D. Ziebart, J. Andrew Bagnell, and Anind K. Dey, *Modeling Interaction via the Principle of Maximum Causal Entropy*, 2010.

[64] Jonathan Ho and Stefano Ermon, « Generative Adversarial Imitation Learning », *in*: *Advances in Neural Information Processing Systems*, vol. 29, 2016.

[65] Aravind Rajeswaran, Vikash Kumar, Abhishek Gupta, Giulia Vezzani, John Schulman, Emanuel Todorov, and Sergey Levine, « Learning Complex Dexterous Manipulation with Deep Reinforcement Learning and Demonstrations », *in*: *arXiv:1709.10087 [cs]* (2017).

[66] Ikechukwu Uchendu, Ted Xiao, Yao Lu, Banghua Zhu, Mengyuan Yan, Joséphine Simon, Matthew Bennice, Chuyuan Fu, Cong Ma, Jiantao Jiao, Sergey Levine, and Karol Hausman, « Jump-Start Reinforcement Learning », *in*: *arXiv:2204.02372 [cs]*, 2022.

[67] Matej Vecerík, Todd Hester, Jonathan Scholz, Fumin Wang, Olivier Pietquin, Bilal Piot, Nicolas Manfred Otto Heess, Thomas Rothörl, Thomas Lampe, and Martin A. Riedmiller, « Leveraging Demonstrations for Deep Reinforcement Learning on Robotics Problems with Sparse Rewards », *in*: *arXiv:1707.08817 [cs]* (2017).

[68] Todd Hester, Matej Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Dan Horgan, John Quan, Andrew Sendonaris, Ian Osband, Gabriel Dulac-Arnold, John Agapiou, Joel Leibo, and Audrunas Gruslys, « Deep Q-learning From Demonstrations », *in*: *Proceedings of the AAAI Conference on Artificial Intelligence* 32 (2018).

[69] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver, « Prioritized Experience Replay », *in*: *arXiv:1511.05952 [cs]* (2016).

[70] Faraz Torabi, Garrett Warnell, and Peter Stone, « Behavioral Cloning from Observation », *in*: *arXiv:1805.01954 [cs]* (2018).

[71] Ashvin Nair, Dian Chen, Pulkit Agrawal, Phillip Isola, Pieter Abbeel, Jitendra Malik, and Sergey Levine, « Combining self-supervised learning and imitation for vision-based rope manipulation », *in*: *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 2146–2153.

[72] Ashley Edwards, Himanshu Sahni, Yannick Schroecker, and Charles Isbell, « Imitating Latent Policies from Observation », *in*: *Proceedings of the 36th International Conference on Machine Learning*, PMLR, 2019, pp. 1755–1763.

[73] Alan Wu, A. J. Piergiovanni, and Michael S. Ryoo, « Model-based Behavioral Cloning with Future Image Similarity Learning », *in*: *Proceedings of the Conference on Robot Learning*, PMLR, 2020, pp. 1062–1077.

[74] Faraz Torabi, Garrett Warnell, and Peter Stone, « Generative Adversarial Imitation from Observation », *in*: *arXiv:1807.06158 [cs, stat]* (2019).

[75] Pierre Sermanet, Corey Lynch, Yevgen Chebotar, Jasmine Hsu, Eric Jang, Stefan Schaal, Sergey Levine, and Google Brain, « Time-Contrastive Networks: Self-Supervised Learning from Video », *in*: *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 1134–1141.

[76] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio, « Generative adversarial networks », *in*: *Communications of the ACM* 63 (2020), pp. 139–144.

[77] Zhengwei Wang, Qi She, and Tomás E. Ward, « Generative Adversarial Networks in Computer Vision: A Survey and Taxonomy », *in*: *ACM Computing Surveys* 54 (2021), 37:1–37:38.

[78] Gustavo H. de Rosa and João P. Papa, « A survey on text generation using generative adversarial networks », *in*: *Pattern Recognition* 119 (2021), p. 108098.

[79] Yukiya Hono, Kei Hashimoto, Keiichiro Oura, Yoshihiko Nankaku, and Keiichi Tokuda, « Singing Voice Synthesis Based on Generative Adversarial Networks », *in*: *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2019, pp. 6955–6959.

[80] Steven Liu, Tongzhou Wang, David Bau, Jun-Yan Zhu, and Antonio Torralba, « Diverse Image Generation via Self-Conditioned GANs », *in*: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, 2020, pp. 14274–14283.

[81] Lionel Blondé and Alexandros Kalousis, « Sample-Efficient Imitation Learning via Generative Adversarial Nets », *in*: *Proceedings of the Twenty-Second International Conference on Artificial Intelligence and Statistics*, PMLR, 2019, pp. 3138–3148.

[82] Karol Hausman, Yevgen Chebotar, Stefan Schaal, Gaurav Sukhatme, and Joseph J Lim, « Multi-Modal Imitation Learning from Unstructured Demonstrations using Generative Adversarial Nets », *in*: *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[83] Peter Henderson, Wei-Di Chang, Pierre-Luc Bacon, David Meger, Joelle Pineau, and Doina Precup, « OptionGAN: Learning Joint Reward-Policy Options Using Generative Adversarial Inverse Reinforcement Learning », *in*: *Proceedings of the AAAI Conference on Artificial Intelligence* 32 (2018).

[84] Wanxiang Li, Chu-Hsuan Hsueh, and Kokolo Ikeda, « Imitating Agents in A Complex Environment by Generative Adversarial Imitation Learning », *in*: *2020 IEEE Conference on Games (CoG)*, 2020, pp. 702–705.

[85] Lei Tai, Jingwei Zhang, Ming Liu, and Wolfram Burgard, « Socially Compliant Navigation Through Raw Depth Inputs with Generative Adversarial Imitation Learning », *in*: *2018 IEEE International Conference on Robotics and Automation (ICRA)* (2018).

[86] Ziyu Wang, Josh S. Merel, Scott E. Reed, Nando de Freitas, Gregory Wayne, and Nicolas Heess, « Robust Imitation of Diverse Behaviors », *in*: *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[87] Ilya Kostrikov, Kumar Krishna Agrawal, Debidatta Dwibedi, Sergey Levine, and Jonathan Tompson, « Discriminator-Actor-Critic: Addressing Sample Inefficiency and Reward Bias in Adversarial Imitation Learning », *in*: *arXiv:1809.02925 [cs, stat]*, 2018.

[88] Scott Fujimoto, Herke Hoof, and David Meger, « Addressing Function Approximation Error in Actor-Critic Methods », *in*: *International Conference on Machine Learning*, PLMR, 2018, pp. 1587–1596.

[89] Rohit Jena, Siddharth Agrawal, and Katia Sycara, « Addressing reward bias in Adversarial Imitation Learning with neutral reward functions », *in*: *arXiv:2009.09467 [cs, stat]* (2020).

[90] Adam Gleave, Mohammad Taufeeque, Juan Rocamonde, Erik Jenner, Steven H. Wang, Sam Toyer, Maximilian Ernestus, Nora Belrose, Scott Emmons, and Stuart Russell, « imitation: Clean Imitation Learning Implementations », *in*: *arXiv:2211.11972 [cs]*, 2022.

[91] Jun Jin, Laura Petrich, Masood Dehghan, and Martin Jagersand, « A Geometric Perspective on Visual Imitation Learning », *in*: *arXiv:2003.02768 [cs]* (2020).

[92] Deepak Pathak, Parsa Mahmoudieh, Guanghao Luo, Pulkit Agrawal, Dian Chen, Fred Shentu, Evan Shelhamer, Jitendra Malik, Alexei A. Efros, and Trevor Darrell, « Zero-Shot Visual Imitation », *in*: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, IEEE, 2018, pp. 2131–21313.

[93] Peter W. Battaglia, Jessica B. Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, Caglar Gulcehre, Francis Song, Andrew Ballard, Justin Gilmer, George Dahl, Ashish Vaswani, Kelsey Allen, Charles Nash, Victoria Langston, Chris Dyer, Nicolas Heess, Daan Wierstra, Pushmeet Kohli, Matt Botvinick, Oriol Vinyals, Yujia Li, and Razvan Pascanu, « Relational inductive biases, deep learning, and graph networks », *in*: *arXiv:1806.01261 [cs, stat]* (2018).

[94] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu, « A Comprehensive Survey on Graph Neural Networks », *in*: *IEEE Transactions on Neural Networks and Learning Systems* 32 (2021), pp. 4–24.

[95] Yixin Lin, Austin S. Wang, and Akshara Rai, « Efficient and Interpretable Robot Manipulation with Graph Neural Networks », *in*: *arXiv:2102.13177 [cs]* (2021).

[96] Tom Silver, Rohan Chitnis, Aidan Curtis, Joshua B. Tenenbaum, Tomás Lozano-Pérez, and Leslie Pack Kaelbling, « Planning with Learned Object Importance in Large Problem Instances using Graph Neural Networks », *in*: *Proceedings of the AAAI Conference on Artificial Intelligence* 35 (2021), pp. 11962–11971.

[97] Arbaaz Khan, Alejandro Ribeiro, Vijay Kumar, and Anthony G. Francis, « Graph Neural Networks for Motion Planning », *in*: *arXiv:2006.06248 [cs]*, 2020.

[98] Christian R. G. Dreher, Mirko Wächter, and Tamim Asfour, « Learning Object-Action Relations from Bimanual Human Demonstration Using Graph Networks », *in*: *IEEE Robotics and Automation Letters* 5 (2020), pp. 187–194.

[99] Tae Soo Kim and Austin Reiter, « Interpretable 3D Human Action Analysis with Temporal Convolutional Networks », *in*: *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2017, pp. 1623–1631.

[100] Jianfeng Xu, Kazuyuki Tasaka, and Hiromasa Yanagihara, « Beyond Two-stream: Skeleton-based Three-stream Networks for Action Recognition in Videos », *in*: *2018 24th International Conference on Pattern Recognition (ICPR)*, 2018, pp. 1567–1573.

[101] Pengfei Zhang, Cuiling Lan, Junliang Xing, Wenjun Zeng, Jianru Xue, and Nanning Zheng, « View Adaptive Neural Networks for High Performance Skeleton-Based Human Action Recognition », *in*: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 41 (2019), pp. 1963–1978.

[102] Chiara Plizzari, Marco Cannici, and Matteo Matteucci, « Skeleton-based Action Recognition via Spatial and Temporal Transformer Networks », *in*: *Computer Vision and Image Understanding* 208-209 (2021), p. 103219.

[103] Miao Feng and Jean Meunier, « Skeleton Graph-Neural-Network-Based Human Action Recognition: A Survey », *in*: *Sensors* 22 (2022), p. 2091.

[104] Will Hamilton, Zhitao Ying, and Jure Leskovec, « Inductive Representation Learning on Large Graphs », *in*: *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[105] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio, « Graph Attention Networks », *in*: *arXiv:1710.10903 [cs, stat]* (2018).

[106] Ilya Sutskever, Oriol Vinyals, and Quoc V Le, « Sequence to Sequence Learning with Neural Networks », *in*: *Advances in Neural Information Processing Systems*, vol. 27, 2014.

[107] Ramesh Nallapati, Bowen Zhou, Cicero dos Santos, Caglar Guulcehre, and Bing Xiang, « Abstractive Text Summarization using Sequence-to-sequence RNNs and Beyond », *in*: *Proceedings of the 20th SIGNLL Conference on Computational Natural Language Learning*, 2016, pp. 280–290.

[108] Kulothunkan Palasundram, Nurfadhlina Mohd Sharef, Nurul Amelina Nasharuddin, Khairul Azhar Kasmiran, and Azreen Azman, « Sequence to Sequence Model Performance for Education Chatbot », *in*: *International Journal of Emerging Technologies in Learning (iJET)* 14 (2019), pp. 56–68.

[109] Hana Yousuf, Michael Lahzi, Said A. Salloum, and Khaled Shaalan, « A systematic review on sequence-to-sequence learning with neural network and its models », *in*: *International Journal of Electrical and Computer Engineering (IJECE)* 11 (2021), pp. 2315–2326.

[110] Yuhuang Hu, Adrian Huber, Jithendar Anumula, and Shih-Chii Liu, « Overcoming the vanishing gradient problem in plain recurrent networks », *in*: *arXiv:1801.06105 [cs]* (2019).

[111] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin, « Attention is All you Need », *in*: *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[112] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer, « BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension », *in*: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 2020, pp. 7871–7880.

[113] Alexis Conneau and Guillaume Lample, « Cross-lingual Language Model Pretraining », *in*: *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[114] Wei-Cheng Chang, Hsiang-Fu Yu, Kai Zhong, Yiming Yang, and Inderjit S. Dhillon, « Taming Pretrained Transformers for Extreme Multi-label Text Classification », *in*: *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020, pp. 3163–3171.

[115] Salman Khan, Muzammal Naseer, Munawar Hayat, Syed Waqas Zamir, Fahad Shahbaz Khan, and Mubarak Shah, « Transformers in Vision: A Survey », *in*: *ACM Computing Surveys* 54 (2022), pp. 1–41.

[116] Biponjot Kaur and Sarbjeet Singh, « Object Detection using Deep Learning: A Review », *in*: *Proceedings of the International Conference on Data Science, Machine Learning and Artificial Intelligence*, 2022, pp. 328–334.

[117] Thomas N. Kipf and Max Welling, « Semi-Supervised Classification with Graph Convolutional Networks », *in*: *International Conference on Learning Representations*, 2023.

[118] Christopher Morris, Martin Ritzert, Matthias Fey, William L. Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe, « Weisfeiler and Leman Go Neural: Higher-Order Graph Neural Networks », *in*: *Proceedings of the AAAI Conference on Artificial Intelligence* 33 (2019), pp. 4602–4609.

[119] Matthias Fey and Jan Eric Lenssen, « Fast Graph Representation Learning with PyTorch Geometric », *in*: *arXiv:1903.02428 [cs, stat]* (2019).

[120] Sami Abu-El-Haija, Amol Kapoor, Bryan Perozzi, and Joonseok Lee, « N-GCN: Multi-scale Graph Convolution for Semi-supervised Node Classification », *in*: *Proceedings of The 35th Uncertainty in Artificial Intelligence Conference*, PMLR, 2020, pp. 841–851.

[121] Matthew Wilson and Tucker Hermans, « Learning to Manipulate Object Collections Using Grounded State Representations », *in*: *Proceedings of the Conference on Robot Learning*, PMLR, 2020, pp. 490–502.

[122] Rohit Jena, Changliu Liu, and Katia Sycara, « Augmenting GAIL with BC for sample efficient imitation learning », *in*: *Proceedings of the 2020 Conference on Robot Learning*, PMLR, 2021, pp. 80–90.

[123] Raghav Nagpal, Achyuthan Unni Krishnan, and Hanshen Yu, « Reward Engineering for Object Pick and Place Training », *in*: *arXiv:2001.03792 [cs]* (2020).

[124] Chao Yang, Xiaojian Ma, Wenbing Huang, Fuchun Sun, Huaping Liu, Junzhou Huang, and Chuang Gan, « Imitation Learning from Observations by Minimizing Inverse Dynamics Disagreement », *in*: *Advances in Neural Information Processing Systems*, vol. 32, Curran Associates, Inc., 2019.

[125] Sanmit Narvekar, Bei Peng, Matteo Leonetti, Jivko Sinapov, Matthew E. Taylor, and Peter Stone, « Curriculum learning for reinforcement learning domains: a framework and survey », *in*: *The Journal of Machine Learning Research* 21 (2022), pp. 7382–7431.

[126] Yassine El Manyari, Patrick Le Callet, and Laurent Dollé, « Imitation from Observation using RL and Graph-based Representation of Demonstrations », *in*: *2022 21st IEEE International Conference on Machine Learning and Applications (ICMLA)*, 2022, pp. 1258–1265.

[127] Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Avnish Narayan, Hayden Shively, Adithya Bellathur, Karol Hausman, Chelsea Finn, and Sergey Levine, « Meta-World: A Benchmark and Evaluation for Multi-Task and Meta Reinforcement Learning », *in*: *arXiv:1910.10897 [cs, stat]* (2021).

[128] Emanuel Todorov, Tom Erez, and Yuval Tassa, « MuJoCo: A physics engine for model-based control », *in*: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012, pp. 5026–5033.

[129] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba, « OpenAI Gym », *in*: *arXiv:1606.01540 [cs]* (2016).

[130] Amir M. Soufi Enayati, Zengjie Zhang, Kashish Gupta, and Homayoun Najjaran, « Exploiting Symmetry and Heuristic Demonstrations in Off-policy Reinforcement Learning for Robotic Manipulation », *in*: *arXiv:2304.06055 [cs]* (2023).

[131] Youngwoon Lee, Andrew Szot, Shao-Hua Sun, and Joseph J Lim, « Generalizable Imitation Learning from Observation via Inferring Goal Proximity », *in*: *Advances in Neural Information Processing Systems*, vol. 34, 2021, pp. 16118–16130.

[132] Yana Hasson, Gul Varol, Dimitrios Tzionas, Igor Kalevatykh, Michael J. Black, Ivan Laptev, and Cordelia Schmid, « Learning Joint Reconstruction of Hands and Manipulated Objects », *in*: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 11807–11816.

[133] Konrad Zolna, Scott Reed, Alexander Novikov, Sergio Gómez Colmenarejo, David Budden, Serkan Cabi, Misha Denil, Nando de Freitas, and Ziyu Wang, « Task-Relevant Adversarial Imitation Learning », *in*: *Proceedings of the 2020 Conference on Robot Learning*, PMLR, 2021, pp. 247–263.

COLLEGE ED MATHEMATIQUES
DOCTORAL STIC
PAYS DE LA LOIRE

Nantes Université

Titre : Une approche modulaire pour l'apprentissage par imitation générique à l'aide d'une représentation spatio-temporelle des démonstrations basée sur les graphes : Application à l'apprentissage robotique

Mot clés : Apprentissage par Imitation, Imitation par Observation, Apprentissage par Renforcement, Réseaux Neuronaux Graphiques, Modélisation Séquentielle

Résumé : L'apprentissage par renforcement et l'apprentissage par imitation permettent aux robots d'apprendre à effectuer des tâches de manière autonome, sans avoir besoin d'instructions explicites. Cette thèse examine les deux méthodes et les intègre dans un cadre modulaire et générique pour résoudre le problème d'apprentissage par imitation à partir d'observations. L'approche est mise en œuvre en deux étapes, en commençant par apprendre un modèle de représentation qui capture les caractéristiques spatiales et temporelles des démonstrations observées, suivi de l'application d'un algorithme RL prêt à l'emploi avec une fonction de récompense générique pour apprendre la politique d'imitation. Les résultats expérimentaux indiquent que la méthode proposée surpasse les méthodes de pointe et présente des capacités de généralisation prometteuses pour une gamme de tâches de manipulation, dépassant les méthodes génératives dans la plupart des cas.

Title: A Modular Framework for Generic Imitation Learning using Graph-based Spatio-Temporal Representation of Demonstrations: Application to Robotic Learning

Keywords: Imitation Learning, Imitation from Observation, Reinforcement Learning, Graph Neural Networks, Sequential modelling

Abstract: Reinforcement Learning and Imitation Learning allow robots to learn how to perform tasks independently, without the need for explicit instructions. This thesis examines both methods and integrates them into a modular and generic framework for solving the imitation learning from observation problem. The approach is implemented in two stages, beginning with learning a representation model that captures the spatial and temporal features of observed demonstrations, followed by applying an off-the-shelf RL algorithm with a task-agnostic reward function to learn the imitation policy. Experimental results indicate that the proposed method outperforms state-of-the-art methods and exhibits promising generalisation capabilities across a range of manipulation tasks, surpassing generative methods in most instances.