Abstract

Robot technology is one of the pillars of modern society. Advances in information, electronic, and mechanical fields enable us to build and program machines to perform tasks in very different contexts, such as industry, surgery, and space missions.
Specifically, in manufacturing, robots are mainly used to perform repetitive and unhealthy works like assembly, welding and material handling, thanks to their mechanical robustness and ability to repeatedly perform the same movements with high accuracy and precision.

While in the early day, robot systems were constrained in isolated and known environments. Over the past few decades, robots have been asked to solve tasks in dynamic and unknown/partially-known environments, where they must **coexist** and **cooperate** with humans, while solving different **dynamic** tasks [1] (e.g. pick a requested object, whose position is not known a priori).

In this scenario, the desired characteristics of such robotic systems are: **(a) Adaptability to new conditions**, i.e., the system must be able to easily adapt to dynamic changes in system and environmental conditions, performing *"intelligent" behaviors* to handle these new scenarios and solve the desired task; **(b) Adaptability to new tasks**, i.e., the system must be able to easily adapt to both new variations of a known task and completely new tasks by exploiting experience to infer actions and solve them;

These requirements, can be challenging to achieve with traditional robot programming techniques based on hand-written policies and control methods. These conventional techniques often require a meticulous analysis of process dynamics, the construction of an analytical model, and the derivation of a control law that meets specific design criteria. This design process is tedious and time-consuming, particularly when high-level perception systems (e.g., cameras, microphones, motion sensors) are used to infer the state of the environment (such as the unknown position of a desired object relative to the end-effector) and the intentions of the human operator.

In contrast, significant advancements have been made by leveraging *learning techniques*, where the control policy is learned from data. This data can be generated either by **agent experience** [2] or by **expert demonstration** [3]. In the case of agent experience, there is a trial-and-error procedure where the control policy generates actions executed by an agent, which interacts with the environment. The parameters are then tuned according to the effectiveness of the actions, based on their impact on the environment relative to the task to be solved. In the case of expert demonstration, the control policy parameters are directly tuned using a dataset containing examples of task execution. The goal is to replicate the tasks observed in the dataset.

Specifically this thesis is framed in the context of *Learning from Demonstration* (LfD), a learning approach based on expert demonstration. According to the requirements of adaptability the thesis focus on a specific aspect of LfD named *Multi-Task LfD*. In this case, the control policy is not trained to execute a single task (e.g., picking an object) with the goal of generalizing across different objects and initial conditions [4, 5]. Instead, it is trained to handle various variations of a specific task (e.g., picking an object from different possible locations) [6] or even entirely different tasks (e.g., a single control policy that solves both picking and placing tasks as well as assembly tasks) [7, 8]. The goal is to generalize not only with respect to the objects

being manipulated and the initial conditions but also with respect to the tasks themselves. This means that by leveraging the knowledge-sharing hypothesis, we can achieve a system capable of solving new variations.

In this scenario, the learning procedure is much more challenging because we need to include and define the **conditioning signal** (i.e., the signal that informs the policy about the task to execute, the object to manipulate, and the target placing location). Additionally, the environment can contain **multiple distractor objects** (e.g., objects that can potentially be manipulated but are not of interest for a given task variation).

Regarding the conditioning signal, there are at least two intuitive approaches. The first is through a natural language description of the task to be executed [9, 10, 7], and the second is through a video demonstration [6, 8]. In the former, the task is described using phrases that specify the task details, such as "Pick the red box and place it into the first bin". Given in input the phrase, the system must be able to infer the intent of the task (i.e., the pick-place operation) and the object of interest (e.g., red-box for picking and first bin for placing), and correlate this information with the environment and robot state to effectively control the robot. In the latter, another agent (either a robot or a human operator) performs the task in a different environment configuration, records this execution, and provides the video as input to the control policy. The control policy must then infer the intent from the video (i.e., the task to be performed, the object to be manipulated, and the final state) and control the robot to complete the task according to the agent's state, the environment's state, and the commanded task. Inspired by how humans can learn to replicate tasks by simply observing their execution, the main goal of this thesis is to develop a system capable of replicating tasks shown in a video demonstration. This involves addressing challenges related to extracting task-relevant information from the video, such as identifying the

manipulated object and its final position.

Regarding the issue of distractor objects, these are generally items that are not considered in manipulation operations, which simplifies the problem significantly. However, in the context proposed in this thesis, the problem is further complicated by the fact that the semantic meaning of an object of interest or a distractor is defined at run-time by the command itself. This means that if the initial configuration consists of four objects (e.g., four boxes of different colors), a specific object may or may not be of interest based on the command given to the robot.

The main contribution of this thesis is addressing the issue of distractor objects. Specifically, it was observed that a significant problem with the methods proposed in the literature is that while the learned control policy can generate valid trajectories, allowing the robot to reach, pick, and place objects, it often manipulates the wrong object. To address this problem, two main considerations have been made:

(1) The architectures proposed in current literature predominantly consist of end-to-end architectures, which translate high-level inputs such as images into corresponding actions. Consequently, the model must acquire an implicit representation capable of encoding both the task objective and the current state of the environment, including the location of the target object; **(2)** The learning procedure optimizes a metric that is not directly linked to task success but rather aims to replicate actions similar to those of the expert, on average.

These two aspects can lead to a control policy that is not able to effectively guide the robot toward the target object. Indeed, it was observed that one of the critical points during trajectory execution is the first steps. Small errors in these initial steps can result in reaching and consequently picking the wrong object.

Based on these considerations, this thesis evaluates the possibility of developing a system that explicitly reasons about the objects of interest (e.g., target object and placing location). The control module is

then directly informed with low-level information, such as the position of the target object.

To perform this explicit reasoning, a *Conditioned Object Detector* (COD) has been developed. This module, given the video demonstration and the current agent observation as input, predicts the class-agnostic bounding box related to the target object and the final placing location. This low-level positional information is then provided to the control module, which predicts the actions to perform.

The learning procedure is then divided into two steps. The first step involves training the *Conditioned Object Detector* (COD) module, which focuses on explicitly solving cognitive tasks, such as detecting regions of interest represented by the object to be manipulated and its final location. The second step involves training the *Object Conditioned Control Policy* (OCCP), which focuses on solving the control problem using low-level positional information that can be easily mapped into the corresponding actions.

The final system has been tested in both **multi-variation single-task** scenarios and **multi-variation multi-task** scenarios. Specifically, the system was evaluated on four different tasks: Pick-Place, Nut-Assembly, Stack-Block, and Button-Press. Each task had different variations based on the manipulated object and the final state. While the tasks share common properties, they also have specific characteristics. For example, the Nut-Assembly task involves contact-rich, precise manipulation, whereas Pick-Place can be solved in a much rougher manner.

Overall, the proposed methods demonstrated very promising behaviors and a general improvement over baseline methods that do not include object-related reasoning. This shows that solving manipulation tasks with an object-oriented approach can be an effective paradigm for LfD problems. Additionally, this approach provides interpretable information to the end user, as the predicted bounding boxes can be interpreted as the locations where the robot will move.

In conclusion, this thesis addresses the problem of leveraging object priors in the context of Multi-Task Imitation Learning. The proposed methods have been tested in both simulated and real-world scenarios, demonstrating their effectiveness.

Contents

ToDo.

- ToDo -

Chapter 1

Introduction

1.1 Application context

1.2 Motivation and thesis overview

1.3 State of the art

The chapter reviews the state of the art (SoTA) on the LfD problem. Specifically, Section 1.3.1 will focus on the formalization of the LfD problem. Then, Section 1.3.3 will present various approaches and methodologies for solving the LfD problem. This will include a detailed taxonomy of the different approaches, highlighting their pros and cons, and concluding with considerations relevant to the goals of this thesis.

1.3.1 Problem formulation

In this section, the problem of Learning from Demonstration, also known as Imitation Learning (IL), will be formalized.

The core idea behind IL is to program a robot by allowing a human expert to demonstrate how to solve a specific task. This approach avoids methods that demand intricate, handcrafted rules dictating the actions of machines and the dynamics of their operating environments, which require significant time and coding expertise. To achieve this goal, a way to transfer knowledge from the expert to the robot is necessary. One possibility is to leverage *data-driven* methods that can extrapolate control rules from demonstrated trajectories [3].

In this context, we need to define two general actors: the **expert**, who demonstrates the desired behaviors, and the **learner**, whose goal is to learn and replicate the behaviors demonstrated by the expert [3, 11]. Both the expert behaviors and the learner behaviors are described in terms of **policy**, generally denoted as π^E and π^L respectively.

Since IL relies on expert demonstrations, these are collected in a dataset $\mathcal{D}^E = \left\{ \left(\boldsymbol{\tau}_i^E, c_i \right) \right\}_{i=1}^N$, where:

- $\boldsymbol{\tau}_i^E$ is the i^{th} demonstrated trajectory, generated by the expert policy $\pi^E \sim \boldsymbol{\tau}_i^E$. It can be described as:
  - A *state-action sequence*, i.e., $\boldsymbol{\tau}_i = [s_0, a_0, \ldots, s_T, a_T]$, when the ground truth action performed by the expert is available.
  - A *state-only sequence*, i.e., $\boldsymbol{\tau}_i = [s_0, \ldots, s_T]$, when the ground truth action is not available.
- c_i is the *context-vector*, containing task-related information such as the initial state of the system s_0, the position of the target object, or a representation of the task to be executed (e.g., a natural language description of the task or video demonstrations).

The state s_t can be defined in various ways. RGB images have been used in different works, either from a third-person point of view [12], representing the robot and its workspace, or from a first-person point of view with the camera mounted on the robot [13] or the gripper [10].

Additionally, 3D point-clouds generated by RGB-D images can also be used [14]. This high-level state representation can be enriched with proprioceptive signals such as joint positions, velocities, and torques [4].

Regarding the policy, the predicted action, $\hat{a}_t$, can be generated into two distinct ways. In one case, it is described as a *deterministic function*, meaning that it directly determines the action as follows: $\hat{a}_t = \pi^L(s_t, c_i)$[13]. In the other case, it takes on a probabilistic definition, where it represents a probability distribution from which to sample the desired action: $\hat{a}_t \sim \pi^L(s_t, c_i)$ [8].

According to the definitions given in [3, 15], the learner policy π^L can be defined with respect to different abstraction levels:

- *Symbolic Characterization*, the policy maps states, and context to a sequence of options, i.e., $\pi : s_t, c \rightarrow [o_1, \ldots, o_T]$, where each option is a sequence of actions. With this representation, complex tasks can be decomposed into a sequence of simple movements. However, it is hard to achieve an accurate task segmentation and motion ordering;

- *Trajectory Characterization*, the policy maps context to trajectory, i.e., $\pi : c \rightarrow \boldsymbol{\tau}$. Because it allows the initial state to be mapped to a complete sequence of actions, this representation can be used to obtain the options in the Symbolic Representation. However, they need as many dynamic features as possible, that can be difficult to obtain;

- *State-Action Characterization*, the policy maps states(-context) to actions, i.e., $\pi : s_t, c \rightarrow a_t$. This representation makes it possible to map the current state directly to the corresponding action. However, it is easy for errors to accumulate in long-term processes. The action a_t can also be defined in various ways, depending on the type of control implemented by

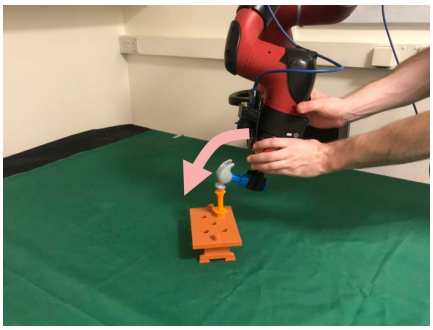
the method. Some methods operate in the joint-space, predicting motor control signals such as positions, velocities, and torques [4]. Other methods work in the operational space, where actions are assigned either an absolute pose with respect to a world frame $\mathcal{W}$, i.e., $a_i = \left[p_x^{\mathcal{W}}, p_y^{\mathcal{W}}, p_z^{\mathcal{W}}, r_x^{\mathcal{W}}, r_y^{\mathcal{W}}, r_z^{\mathcal{W}}\right]$ [8], or a displacement relative to the current gripper position, i.e., $a_i = \left[\Delta p_x^{\mathcal{W}}, \Delta p_y^{\mathcal{W}}, \Delta p_z^{\mathcal{W}}, \Delta r_x^{\mathcal{W}}, \Delta r_y^{\mathcal{W}}, \Delta r_z^{\mathcal{W}}\right]$ [13].

The expert and the learner, through their policies π^E and π^L, act on an environment modeled as a *Markov Decision Process* (MDP) [16]. An MDP is defined as a tuple (S, A, R, T, γ), where:

- $S \subseteq \mathbb{R}^n$ is the set of states (e.g., joint positions and/or images).
- $A \subseteq \mathbb{R}^n$ is the set of actions (e.g., desired end-effector pose, desired joint torques).
- $R(s, a, s')$ is the *reward function*, which expresses the immediate reward for executing action a in state s and transitioning to state s'.
- $T(s'|s, a)$ is the *transition function*, which defines the probability of reaching state s' after executing action a in state s. This distribution, which describes the *system dynamics*, can be given a priori or learned (Model-Based methods), or it may not be considered at all (Model-Free methods).
- $\gamma \in [0, 1]$ is the discount factor, expressing the agent's preference for immediate rewards over future rewards.

With respect to the given MDP definition, the reward function plays different roles depending on the approach used:

- In *Behavioral Cloning* (BC) methods, the reward function is not explicitly used. Instead, a surrogate loss function is employed.



(a) Example of kinesthetic teaching [17]

(b) Example of teleoperation [4]

Figure 1.1: Examples of direct demonstration

- In *Inverse Reinforcement Learning* (IRL), the reward function is learned, under the assumption that the expert acts (near-)optimally with respect to some unknown reward function.
- In *Generative Adversarial Imitation Learning* (GAIL) and *Learning from Observations* (LfO), the role of the reward function varies based on the specific method, as will be explained in Section TODO.

1.3.2 Source of demonstration

As stated in Section 1.3.1, IL methods rely on a dataset $\mathcal{D}^E$ of expert demonstrations. In this section, we will review the different ways to obtain these demonstrations, following the taxonomy proposed in [15].

Direct Demonstration

In the case of *Direct Demonstration*, the expert trajectory is a state-

action sequence, where the action is obtained directly from the robot. Specifically, the robot can be guided in task execution through *kinesthetic teaching* [18, 17], *teleoperation* [4, 19, 13, 7, 20, 21], or a *(hand-)written policy* [22, 6, 8, 23].

In kinesthetic teaching (Figure 1.1a), the human operator contacts and guides the robot, recording parameters such as the gripper pose, joint positions, and velocities. This has been one of the first approaches for the LfD problem [24, 25] because there is no need to consider differences in kinematics between human and robot. As a result, the data has less noise, and there is no need for expensive external tools for teleoperation. However, the robot must be passively controllable and require direct contact, which introduces safety problems and can be unintuitive for robots with multiple degrees of freedom.

In teleoperation (Figure 1.1b), the human operator remotely guides the robot with a joystick, control panel, or wearable device. These tools allow for higher safety since there is no direct contact between the robot and the human expert. Teleoperation systems have been used in various works. For example, the authors in [19, 26] proposed a teleoperation framework named Roboturk, which enables the collection of large-scale demonstration datasets [26, 5] for both simulated and real-world robots using a mobile phone as the controller. The authors in [4, 13, 7] used virtual reality controllers, allowing the human operator to intuitively move in the 3D environment, mapping the pose of the controllers to the corresponding gripper pose. This technology is of interest because it provides a safe and intuitive way to teleoperate a robot. However, the main drawback is the lack of haptic feedback, which can be mitigated by using haptic interfaces such as [27, 28].

Demonstrations collected with hand-written policies rely on the fact that the expert has access to ground truth information, such as the position of the object of interest. This assumption can be valid when a simulated environment is used to train, test, and validate the proposed methods, as seen in [6, 8, 23]. In these cases, the authors

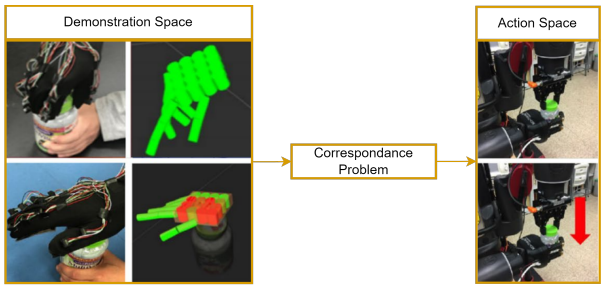
used a well-known simulation environment in the robotic learning community named Robosuite [29]. Here, demonstrations were collected to train methods using hand-written policies that have access to ground truth positional information about the object of interest, solving tasks such as Pick-Place, Nut-Assembly, Stack-Block, and more. Simulation environments facilitate the evaluation of the proposed methods and ensure reproducibility and consistent testing. The idea of using hand-engineered policies is not limited to simulation environments. Indeed, the authors in [22] used automatic grasping primitives combined with diagonal Gaussian distributions to collect demonstrations on a real-world robot. This approach aims to collect as many trajectories as possible with minimal human effort.

Indirect Demonstration

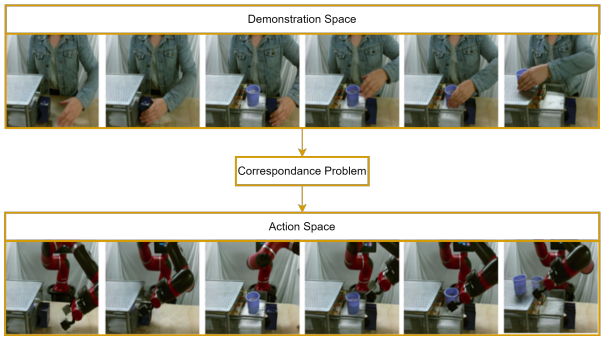
As discussed previously, in direct demonstration, an expert controls the learner agent and records the actions performed. The concept behind Indirect Demonstration (Figure 1.2) is to collect demonstrations that are completely disconnected from the target robotic platform. In the most promising scenario, a human demonstrator performs the desired tasks and records their operations. The learner, starting from this set of recordings, must be able to extrapolate the knowledge needed to replicate the observed tasks.

In this case, the expert demonstrations are state-only trajectories. Since the action space between the human demonstrator and the robot is different (consider just the different embodiment), it is not possible to directly use the human joint trajectories to minimize a supervised loss where the predicted value is related to the robot action space.

Initially, methods that follow this approach used wearable devices to capture human movement and record it [32, 30]. For example, in [32], the authors used a motion capture system to record the movement of a dancer and then transfer these trajectories to a humanoid robot.



(a) Example of indirect demonstration based on wearable device [30]



(b) Example of direct demonstration based on human video demonstration [31]

Figure 1.2: Examples of indirect demonstration

Similarly, in [30], the authors used a tactile glove [33] to record the movement performed by a human hand in the operation of opening bottles (Figure 1.2a).

In this line of research related to indirect demonstrations, there are also novel methods [31, 34, 35, 36, 37] that, inspired by the way humans learn by watching task execution, remove the assumption of having access to recorded human joint trajectories. In this case, the demonstrations are just videos of the human demonstrator (Figure

1.2b). Here, the system must infer from the video not only the intent of the task but also how this task can be solved and transform this information into its own action space.

Generally, methods based on wearable devices allow for very intuitive demonstrations, including critical information for manipulation tasks such as force and tactile information [30]. While methods based on just video demonstrations are very promising because they allow for the collection of demonstrations in the most intuitive and scalable way possible (potentially any video of a performed task can be used). However, both these approaches have to solve the *correspondence problem*, i.e., the system must be able to map motion captured in human space into the corresponding motion of the robot. In Section TODO, the different ways this problem has been solved in the context of visual demonstration will be explained in detail.

1.3.3 Learning from demonstration

This section is dedicated to presenting and analyzing various approaches for solving the LfD problem described in Section 1.3.1. Specifically, this section follows the taxonomy derived from studying different review papers [38, 39, 40, 15, 41, 11]. Figure 1.3 provides a graphical representation of the proposed taxonomy. The methods are first categorized based on the type of demonstration, either *State-Action* or *State-only*, followed by an overview of the different methodologies. The proposed taxonomy highlights the learning algorithm and main components for each methodology.

This chapter is divided into the following sections. Section 1.3.3.1 reviews methods for the fully-supervised learning methodology known as Behavioral Cloning.
Section 1.3.3.2 discusses methods under the umbrella of Inverse Reinforcement Learning, which solve the inverse optimization problem by first learning the reward function and then using it to guide policy

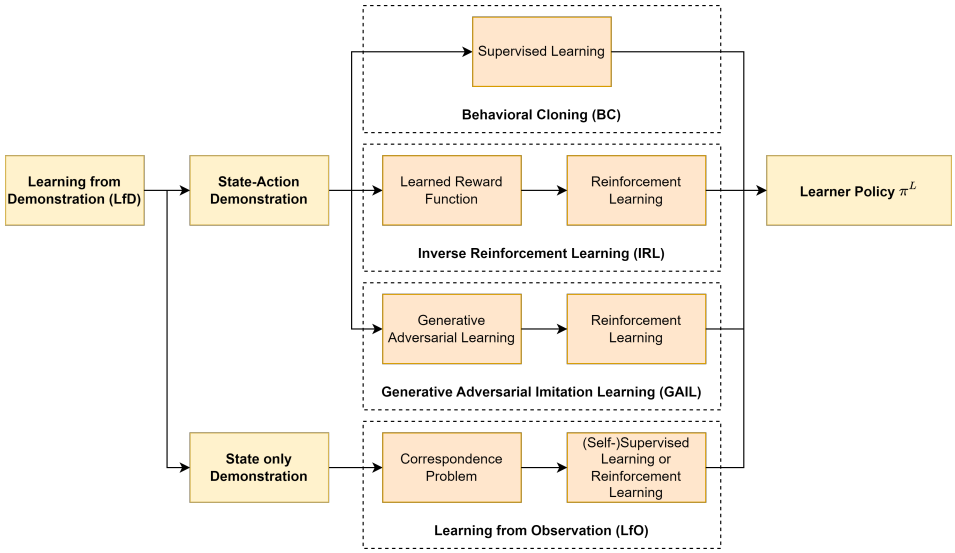


Figure 1.3: Taxonomy of LfD methods, divided based on type of demonstration and the learning algorithm used to learn the learner policy π^L

optimization following a reinforcement paradigm
Section 1.3.3.3 reviews methods leveraging the concept of *Generative Adversarial Learning* to optimize policy parameters and learn demonstrated behaviors, classified as Generative Adversarial Imitation Learning methods.
Finally, Section 1.3.3.4 covers the most recent methodology, Learning from Observation, characterized by optimizing the learner policy using state-only demonstrations.

1.3.3.1 Behavioral Cloning (BC)

Behavioral Cloning is one of the first approaches used to solve the LfD problem [42]. The high-level **supervised-learning** procedure

Algorithm 1 Abstract Algorithm for BC methods

Require: A set of expert demonstrations $\mathcal{D}^E$, a parameterized policy π_θ^L

Ensure: The optimal set of policy parameter θ^*

Optimize $\mathcal{L}$ w.r.t. policy parameter θ using $\mathcal{D}^E$

followed by BC methods is outlined in Algorithm 1. Generally, BC methods take as input the expert demonstration dataset $\mathcal{D}^E$ and a learner policy modeled as a parameterized function π_θ^L. The parameters θ can be either the weights of a neural network [42] or the parameters of a dynamic system [43]. As a supervised approach, the goal is to find the optimal parameters θ^* that can replicate the **ground-truth behaviors** contained in the dataset $\mathcal{D}^E$. This is achieved by solving an optimization problem, which can generally be described by Formula 1.1.

$$\theta^* = \underset{\theta}{argmin}\ \mathbb{E}_{(\boldsymbol{\tau},c)\sim\mathcal{D}^E}\ [\mathcal{L}((\boldsymbol{\tau},c),\ \pi_\theta^L)] \tag{1.1}$$

In the following section, different ways in which the general optimization problem described by Formula 1.1 is formulated and solved will be discussed.

Dynamical Movement Primitives

The *Dynamical-Movement Primitives* (DMPs) methods are among the first successful applications of the BC methodology to the LfD problem. Their success is attributed to their ease of implementation and efficiency in learning. DMPs do not require learning or estimating the system dynamics, nor do they require a reward function or interaction with the environment during the learning procedure, as they are supervised learning methods.

DMPs were first formalized in [43]. They derive the policy directly in the trajectory space, allowing for explicit modeling of constraints such as smooth convergence toward the goal state. The core idea behind DMPs, as proposed in [43, 44], is to model the trajectory as a **point-to-point attractor system**, described by the set of differential equations in Formula 1.2.

$$\begin{aligned} \tau\dot{y} &= \beta_s(\alpha_s(g-s)-y)+f(z) \\ \tau\dot{s} &= y \\ \tau\dot{z} &= -\alpha_z z,\ z(t)=z_0\ exp(-\tfrac{\alpha_z}{\tau}t) \end{aligned} \tag{1.2}$$

Here, β_s, α_s, and α_z are constants, s is the system state, z is the phase-variable function of time t, and f is the forcing term that describes the trajectory's non-linear behavior. Generally, f is a linear combination of basis functions $\psi_i(z)$ (e.g., Gaussian basis functions), such that $f(z(t)) = (g-s_0)\sum_{i=1}^{M}\psi_i(z(t))\omega_i z$. Essentially, a DMP describes a point-attractor system where the current system state s must converge to the goal state g, starting from s_0. In this context, the aim is to learn the set of weights $\{\omega_i, i=1,\ldots,M\}$, which can be obtained by solving a supervised learning problem with the loss function described in Formula 1.3.

$$\mathcal{L}_{DMP} = \sum_{t=0}^{T}(f_{target}(t)-f(z(t)))^2 \tag{1.3}$$

The function $f_{target}(t)$ is equal to $f_{target}(t) = \tau^2\ \ddot{s}_t^E - \beta_s(\alpha_s(g-s_t^E) - \tau\dot{s}_t^E)$ represents the evolution of the expert state s_t^E towards the goal state g, and represents the dynamic to mimic through the learned linear combination of basis function $f(z_t)$.

The initial DMPs formulation proposed in [43] has several issues that can be categorized as follows:

- **Handling stochasticity in demonstrations**: Different demonstrations can vary slightly due to differences in demonstrators, task completion methods, speeds, and paths. This variability creates a distribution in the demonstration space, requiring a method to manage it.

- **Defining different basis function**: In DMPs, the weights ω_i of the basis functions are learned. However, the force term can also be defined using other formalisms, such as Gaussian Mixture Models [45], Neural-Network Radial Basis Functions [46], or Gaussian Processes [47]. Therefore, the choice of how to model the force term is a hyper-parameter of the problem.

- **Managing arbitrary desired trajectories with intermediate via-points**: Once the behavior encoded in the demonstration is learned, generating novel trajectories that pass through new points (possibly defined by a human agent) is not possible. Therefore, a method to generalize to different waypoints is needed.

- **Handling high-dimensional inputs**: To use the DMPs algorithm, it is necessary to work in the robot space, recording joint and gripper trajectories through teleoperation or kinesthetic teaching. However, in complex scenarios involving interaction with objects that do not have fixed initial positions, it becomes essential to infer the initial object state from high-level inputs, such as images.

To address these drawbacks, several solutions have been proposed. Notably, the authors in [48] introduced the *Probabilistic Movement Primitives* (ProMPs) framework. This probabilistic framework offers an alternative movement primitive representation, capturing the variability across different demonstrations and degrees of freedom (DoFs)

through a covariance matrix. Specifically, the trajectory τ is modeled as a distribution: $\boldsymbol{\tau} = \prod_t \mathcal{N}(s(t)|\Psi(z(t))^T\omega, \Sigma_s)$, where Ψ is a time-dependent basis matrix.

Generally, modeling the problem in probabilistic terms has several advantages, particularly the ability to generalize to new goals by conditioning the learned distribution on a given novel goal state [49].

About the possibility to manage arbitrary desired trajectories, authors in [50] proposed a novel framework for learning movement primitives, named *Via-points Movement Primitives* (VMP). This is basically an extension of both DMPs (that can only adapt to new starts and goals, but cannot directly handle intermediate via-points) and ProMPs (that can adapt to via-points within the statistical distribution of the demonstrated trajectories). ToContinue

Despite all the successfully applications saw previsously, DMPs and all the variants have a very relevant limitation, that is related to the difficulty of handling high-dimensional input such as images. For this reason, the scientific community has focused the attention on methods that leverage deep architecture, that will be explained in detail in the following paragraphs.

Single-Task Imitation Learning

The *Single-Task Imitation Learning* refers to deep architecture designed to learn and replicate specific tasks from given demonstrations.

Interactive Imitation Learning

In *Interactive Imitation Learning* the learning process is augmented by interaction with a teacher, allowing for real-time feedback and adjustments to improve performance.

Multi-Task Imitation Learning

Multi-Task Imitation Learning enables the learning and execution of

multiple tasks from a set of demonstrations, highlighting the scalability and versatility of these methods.

Object-Oriented Imitation Learning
Object-Oriented Imitation Learning focuses on learning behaviors in relation to specific objects and their interactions, providing a more structured and contextual approach to imitation learning.

1.3.3.1.1 Discussion

1.3.3.2 Inverse Reinforcement Learning

The *Inverse Reinforcement Learning* (IRL) problem, also known as *Inverse Optimal Control*, is a LfD algorithm that addresses a significant challenge in Reinforcement Learning: designing an appropriate reward function for a given problem.

In many manipulation problems [51], using a binary and sparse reward function, where the reward is 1 if the task is completed correctly and 0 otherwise, can result in very long and inefficient training periods. For instance, in a reaching problem where the goal is to simply reach an object, the agent would receive a reward of 0 most of the time. More critically, actions that bring the agent closer to the object and actions that move it farther away would yield the same reward. In this simple example, a better reward function would measure the relative distance between the agent and the object. While this information can be easily obtained in simulations, it is often unavailable in real-world settings, forcing algorithm designers to make assumptions such as having prior knowledge about the object's location.

To address this issue, several research approaches have been proposed. One prominent approach is *Reward Shaping*. In this method, the goal is to make the reward function more informative by introducing intermediate rewards during the agent's experience. These

intermediate rewards are hand-engineered and require some degree of *domain knowledge*.

The second approach, which is the focus of this chapter, is Inverse Reinforcement Learning (IRL). In IRL, the main idea is to **learn** the reward function R which explains the expert behaviors contained in the dataset $\mathcal{D}^E$, and then use the learned R to train the agent using classic RL algorithms.

The IRL procedure was introduced in [52], and it is described by Algorithm 2. Essentially, the IRL algorithm is an iterative process where the parametrized reward function R_ω is first updated according to a given objective function $\mathcal{L}$. Subsequently, the updated reward function is used to update the learner's policy π_θ^L.

Algorithm 2 Classic feature matching IRL algorithm

Require: Dataset of expert trajectories $\mathcal{D}^E = \left\{\boldsymbol{\tau}_i^E\right\}_{i=1}^N$
Require: Reward function R_ω, policy π_θ^L
while policy improves **do**
Evaluate the state-action visitation frequency μ of the current policy π_θ^L
Evaluate the objective function $\mathcal{L}$, w.r.t. μ and the dataset $\mathcal{D}^E$
Update the reward-function parameters ω based on $\mathcal{L}$
Update the policy π_θ^L through an RL algorithm, using the updated R_ω
end while

Generally speaking, the IRL approach faces two main challenges:

- The learning process can be time-consuming and impractical for high-dimensional problems due to the double-nested optimization procedure;
- The IRL problem is *ill-posed* because multiple reward functions can produce the same set of actions.

Despite these practical and theoretical challenges, various significant scientific contributions have been made in this field.

To solve the problem of multiple solutions for the reward function, constraints have been added to the optimization problem. Specifically, based on the type of constraint, there are two approaches:

(a) The *Maximum Margin Prediction* (MMP) approach [53, 54].

(b) The *Maximum Entropy* (Max-Ent) approach [55, 56, 57].

The *MMP* methods assume that the demonstrated trajectories are **optimal** and operate in a **deterministic** setting. The aim is to find the cost function such that the reward of the demonstrated trajectories, $R(\boldsymbol{\tau}^E)$, is greater than the reward of all alternative trajectories, $R_\omega(\boldsymbol{\tau})$, by a certain margin m, solving the optimization problem formalized in Formula 1.4.

$$\max_{\omega,m} m \quad \text{s.t.} \quad R_\omega(\boldsymbol{\tau}^E) \geq \max(R_\omega(\boldsymbol{\tau})) + m \tag{1.4}$$

The main problem with this formulation is that it does not handle the case in which the expert behavior is sub-optimal, leading to an ambiguous notion of margin.

In the *Max-Ent* approach, the goal is to find the reward function parameters ψ that drive the policy to maximize the entropy, subject to feature expectation matching (Formula 1.5).

$$\max_{\psi} \mathcal{H}(\pi^{r_\psi}) \quad \text{s.t.} \quad \mathbb{E}_{\pi^{r_\psi}}[\mathbf{f}] = \mathbb{E}_{\pi^*}[\mathbf{f}] \tag{1.5}$$

The Max-Ent approach is the most popular in the IRL field since it removes the ambiguous aspects of the previous formulation. In the original work, the reward function was a linear combination of the features, i.e., $r_\psi = \psi^T \mathbf{f}$. However, this reward formulation is not suited for high-dimensional feature spaces, which may require the capability to model

non-linear reward structures. In [56], a deep neural network was used to model the reward function. In this work, the neural network maps the feature vector $\mathbf{f}$ to the reward value and is trained according to the Maximum-Entropy setting. Experimental results have shown that the ability to approximate highly non-linear reward functions is essential for successfully solving tasks in high-dimensional discrete state spaces. A generalization to continuous state spaces was proposed in [57].

In particular, [57] addressed the problem of learning a cost function in a high-dimensional continuous state space with **unknown dynamics**. Starting from the exponential trajectory distribution $p(\tau) = \frac{1}{Z}\exp(-c_\theta(\tau))$, the main difficulty is the estimation of the partition function Z, needed to compute the negative log-likelihood loss function (Formula 1.6).

$$\mathcal{L}_\theta = \frac{1}{N} \sum_{\tau_i^E \in D_{demo}} c_\theta(\tau_i^E) + \log(Z) \tag{1.6}$$

Since the dynamics are unknown, the idea was to estimate the partition function through the trajectories obtained from the current policy rollouts, with the hypothesis that, during the learning of the cost function, the current policy drives the distribution towards regions where samples are more useful.

Starting from these considerations, Algorithm 3 was proposed. The algorithm returns a *Linear-Quadratic Gaussian* [58] controller q_k, obtained by solving the problem in Formula 1.7.

$$\begin{aligned} q_k &= \arg\min_q \mathbb{E}[c_\theta(\tau)] - \mathcal{H}(\tau) \\ \text{s.t.} \quad & p(s_{t+1}|s_t, a_t) = \mathcal{N}(s_{t+1}; f_{x_t} x_t + f_{u_t} u_t, F_t) \end{aligned} \tag{1.7}$$

The cost function was parameterized according to a neural network, and experiments performed on real-robot manipulation tasks

Algorithm 3 Guided-Cost-Learning Algorithm [57]

Require: Initial controller $q_k(\tau)$
  for i = 1, . . . , N **do**
    Generate D_{traj} from $q_k(\tau)$
    $\mathcal{D}_{samp} \leftarrow \mathcal{D}_{samp} \bigcup \mathcal{D}_{traj}$
    Update the cost-function parameters, using $\mathcal{D}_{samp}$ and $\nabla_\theta \mathcal{L}(\theta)$
    Update the controller $q_{k+1}(\tau) \leftarrow q_k(\tau)$ according to [58] and using $\mathcal{D}_{traj}$
  end for

such as dish placement and pouring proved the effectiveness of the proposed method and the necessity of a non-linear representation of the cost function for complex problems, outperforming the classic MaxEnt method.

Recent methods have aimed to advance IRL towards more complex learning scenarios, such as learning a reward function from video demonstrations. In [59], the architecture shown in Figure 1.4 was proposed. The goal was to obtain the parameters Ψ of a cost function $C_\Psi(\hat{\boldsymbol{\tau}}, z_{goal})$, enabling the derivation of a sequence of actions by minimizing the cost function itself (Formula 1.8).

$$\mathbf{a}_{new} = \mathbf{a} - \eta \nabla_a C_\Psi(\hat{\boldsymbol{\tau}}, z_{goal}). \tag{1.8}$$

Different cost functions were proposed, all aiming to reduce the distance between the current predicted keypoints and the goal keypoint configuration. The trajectory $\hat{\boldsymbol{\tau}}$ is a sequence of predicted states $[\hat{s}_1, \ldots, \hat{s}_T]$, resulting from a learned dynamic model, $\hat{s}_{t+1} = f_{dyn}(\hat{s}_t, a_t)$.

Experimental results demonstrated that it is possible to learn a cost function from human/robot video demonstrations. However, the proposed setting was tested on a very simple reaching task, highlighting that much work remains to be done to establish the effectiveness or

ineffectiveness of IRL in complex real-robot manipulation tasks starting from video demonstrations.

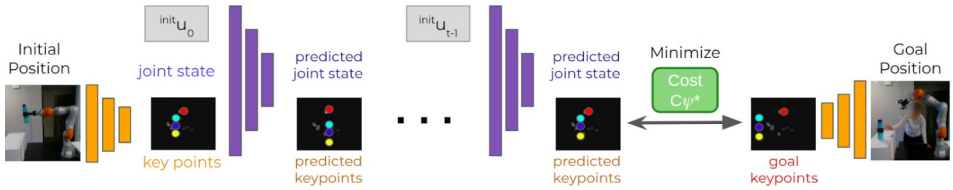


Figure 1.4: Architecture proposed in [59]

1.3.3.3 Generative Adversarial Imitation Learning

The *Generative Adversarial Imitation Learning* (GAIL) is a Learning from Demonstration (LfD) approach proposed by the authors of [60]. The rationale behind GAIL was to improve the Inverse Reinforcement Learning (IRL) setting, which is expensive to run due to the double-nested optimization procedure. To achieve this objective, the authors in [60] started from the Max-Ent formulation in Formula 1.9, and obtained a characterization of the learned policy. This characterization combines the learning of the reward function and the learning of the policy through a reinforcement learning algorithm. In Formula 1.9, $\psi(c)$ is a cost-regularizer, $\psi^*(c)$ is its conjugate, and ρ_π is the occupancy measure, i.e., the distribution of state-action pairs that the agent encounters when navigating the environment with policy π.

The next step was to choose an appropriate regularization function. In particular, by choosing the regularizer in Formula 1.11, the conjugate in Formula 1.12 can be obtained. This is the classic Adversarial Learning Loss, where the current policy π^L acts as the GAN generator, and D is the GAN discriminator, which must distinguish between state-action pairs generated either by the expert policy or by

the current policy.

$$IRL_{\psi}(\pi^E) = \underset{c \in \mathbb{R}^{S \times A}}{\arg\max} - \psi(c) + \left(\min_{\pi^L \in \Pi} - \mathcal{H}(\pi^L) + \mathbb{E}_{\pi^L}\left[c(s,a)\right] \right) - \mathbb{E}_{\pi^E}\left[c(s,a)\right] \tag{1.9}$$

$$RL \circ IRL_{\psi}(\pi^E) = \underset{\pi^L \in \Pi}{arg\ min} - \mathcal{H}(\pi^L) + \psi^*(\rho_{\pi^L} - \rho_{\pi^E}) \tag{1.10}$$

$$\begin{aligned} \psi_{GA}(c) &= \begin{cases} \mathbb{E}_{\pi^E}\left[g(c(s,a))\right] & \text{if } c < 0 \\ +\infty & \text{otherwise} \end{cases}, \\ g(x) &= \begin{cases} -x - \log(1 - e^x) & \text{if } c < 0 \\ +\infty & \text{otherwise} \end{cases} \end{aligned} \tag{1.11}$$

$$\psi^*_{GA}(\rho_{\pi^L} - \rho_{\pi^E}) = \underset{D \in (0,1)^{S \times A}}{max} \mathbb{E}_{\pi^L}\left[\log(D(s,a))\right] + \mathbb{E}_{\pi^E}\left[\log(1 - D(s,a))\right] \tag{1.12}$$

Based on these considerations, Algorithm 4 has been proposed. Specifically, the GAIL algorithm is an iterative procedure composed of two main steps. The first step involves updating the discriminator D, which must distinguish between trajectories produced by the learned policy τ_i^L and trajectories produced by the expert τ^E. The second step involves updating the learner policy π^L according to some reinforcement learning algorithm (e.g., TRPO was used in [60]). The policy is updated in such a way that the trajectories it generates become

Algorithm 4 Generative Adversarial Imitation Learning Algorithm

Require: Expert Trajectories $\tau^E \sim \pi^E$, initial policy π_θ^L, discriminator D_ω

for $i = 1, \ldots, N$ **do**

Sample trajectories, $\tau_i^L \sim \pi_\theta^L$

Update Discriminator, with the gradient

$$\hat{\mathbb{E}}_{\tau_i^L}\left[\nabla_\omega \log(D_\omega(s,a))\right] + \hat{\mathbb{E}}_{\tau^E}\left[\nabla_\omega \log(1 - D_\omega(s,a))\right]$$

Update Policy π_θ^L, with TRPO [61], using the cost-function $C(s,a) = \log(D_\omega(s,a))$, and the KL-constrained gradient step

$$\hat{\mathbb{E}}_{\tau_i}\left[\nabla_\theta \log \pi_\theta(a \mid s) Q(s,a)\right] - \lambda \nabla_\theta H\left(\pi_\theta\right)$$

$$Q(\bar{s},\bar{a}) = \hat{\mathbb{E}}_{\tau_i}\left[\log\left(D_w(s,a)\right) \mid s_0 = \bar{s}, a_0 = \bar{a}\right]$$

end for

indistinguishable from those of the expert for the discriminator, i.e., the learner produces state transitions that are similar to those of the expert.

In the seminal work [60], GAIL has proven to be more effective than classic IRL (Inverse Reinforcement Learning) algorithms [55, 62]. The authors evaluated the GAIL algorithm on nine classic simulated control tasks from the OpenAI Gym simulator [63]: Cartpole, Acrobot, MountainCar, HalfCheetah, Hopper, Walker, Ant, Humanoid, and Reacher.

The observation and control spaces for these tasks are detailed in Table 1.1, and the performance results are shown in Figure 1.5. From these results, it is evident that the GAIL algorithm overcomes classic IRL algorithms in terms of both pure reward and sample efficiency. Consequently, subsequent research has focused on improving the algorithm's efficiency in terms of environment interaction. This has been

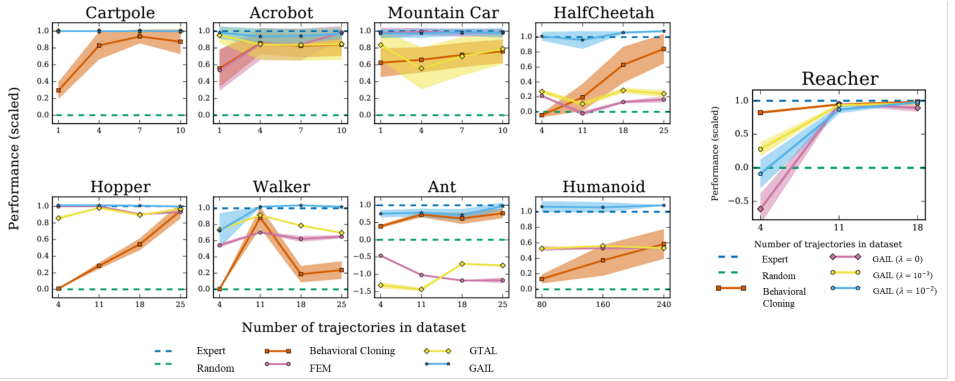


Figure 1.5: The performance comparison proposed in [60] is presented here. The y-axis shows the scaled reward, where the expert's reward is set to 1 and the random baseline is set to 0. The IRL baselines FEM and GTAL refer to the IRL algorithm described in [62], but with different cost functions.

achieved by replacing the model-free, on-policy TRPO algorithm with an off-policy RL algorithm, as seen in [64], or by modifying the reward function input to the RL algorithm [65, 66].

However, as noted in Table 1.1, the tested tasks are characterized by low-dimensional state spaces. More recent research [67, 68, 69, 70] has focused on testing the GAIL algorithms in high-dimensional state spaces, where the input is an image. Specifically, with respect to the Adversarial Imitation Learning setting, works of interest are [69, 70].

In [69], the authors focused on solving the **casual-confusion** problem. This problem occurs when the discriminator, during the learning process, focuses on task-irrelevant features between expert and policy generated transitions, for example a difference in the expert and agent embodiment like the gripper, this causes the rewards to become uninformative. To reduce the casual-confusion problem, in [69] two elements have been proposed:

Table 1.1: Observation and Action space for the tasks used in [60]

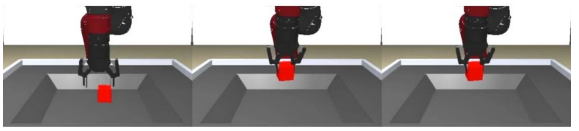
Task	**Observation space**	**Action space**
Cartpole	4 (continuous)	2 (discrete)
Acrobot	4 (continuous)	3 (discrete)
Mountain Car	2 (continuous)	3 (discrete)
Reacher	11 (continuous)	2 (continuous)
HalfCheetah	17 (continuous)	6 (continuous)
Hopper	11 (continuous)	3 (continuous)
Walker	17 (continuous)	6 (continuous)
Ant	111 (continuous)	8 (continuous)
Humanoid	376 (continuous)	17 (continuous)

1. A *regularization term*, with the aim to make the discriminator **unable** to distinguish between constraining sets I_E and I_A. These sets are composed of expert and agent observations, such that a sample can belong either to I_E or I_A, based on spurious features (e.g., a different gripper color);

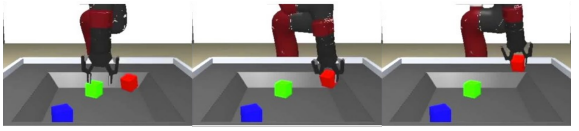
2. An *early-stopping policy* called Actor Early-Stopping (AES), that restarts the episode if the discriminator score at the current step exceeds the median score of the episode so far for $T_{patience}$ consecutive steps.

To prevent the discriminator from focusing on task-irrelevant features, the authors proposed a regularization term based on the constraining-set accuracy defined in Formula 1.13. The idea is that if the discriminator achieves an accuracy greater than $\frac{1}{2}$ on the constraining set, the maximized adversarial cost function should be inverted, as shown in Formula 1.14.

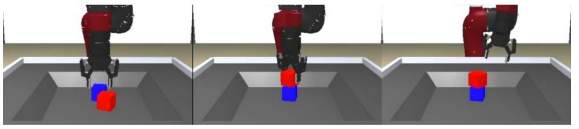
$$accuracy(I_E, I_A) = \frac{1}{2}\,\mathbb{E}_{s\in I_E}\left[\mathbf{1}_{D_\omega \geq \frac{1}{2}}\right] + \frac{1}{2}\,\mathbb{E}_{s\in I_A}\left[\mathbf{1}_{D_\omega < \frac{1}{2}}\right] \tag{1.13}$$



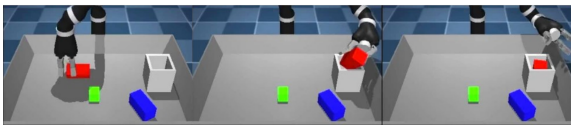
(a) Lifting task



(b) Lifting with distractors task



(c) Block Stacking task



(d) Block Insertion with distractors task

Figure 1.6: Tasks solved in [69]

$$\begin{aligned}\mathcal{L}_\psi\left(s_E, s_A, \hat{s}_E, \hat{s}_A\right) = G_\psi\left(s_E, s_A\right) \\ - \mathbf{1}_{\text{accuracy } (\hat{s}_E, \hat{s}_A) \geq \frac{1}{2}} G_\psi\left(\hat{s}_E, \hat{s}_A\right),\end{aligned}$$

$$\begin{aligned}\text{where} \quad G_\psi\left(s_E, s_A\right) = \sum_{i=1}^{N} \log D_\psi\left(s_E^{(i)}\right) \\ + \log\left[1 - D_\psi\left(s_A^{(i)}\right)\right]\end{aligned} \tag{1.14}$$

The proposed system was tested on 4 tasks (Figure 1.6), with the agent trained on each single task, according to the *Distributed Distri-*

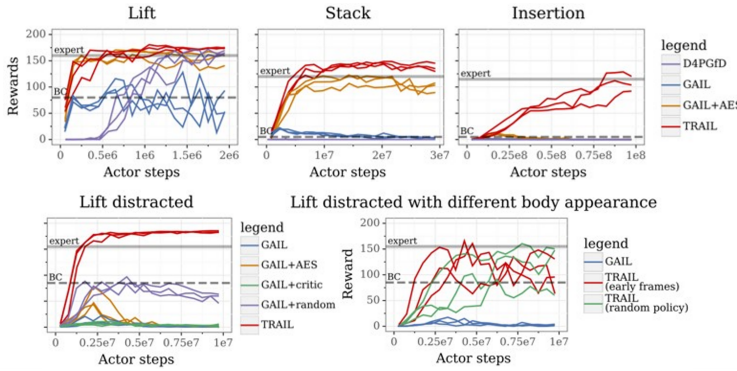


Figure 1.7: Experimental results on tasks without and with spurious features [69]

butional Deterministic Policy Gradients (D4PG) [71] RL algorithm, with reward-function $R(s_t) = -\log(1 - D_\omega(s_t))$. Experimental results have shown how the proposed system overcomes the GAIL [60] baseline, both in setting with spurious features and without spurious features (Figure 1.7).

The authors of [70] proposed a more data-efficient Adversarial Imitation Learning method. They leveraged a model-based approach within a high-dimensional state space. Instead of generating a dynamic model in the image space, i.e., training a generative model to produce the next image based on the current image and the performed action.Their method encodes observations defined in the image space into a corresponding latent space characterized by vectors of smaller dimensions. Then, they learn a dynamic model in that space, training a generative model to produce the next embedding based on the current encoded observation and the performed action. The proposed learning procedure is based on three main steps:

1. Learn the *Latent Dynamic Model*, $(\hat{\mathcal{U}}_\beta, \hat{\mathcal{T}}_\beta, q_{beta})$, by maximizing the Evidence Lower Bound (Formula 1.15), where $\hat{\mathcal{U}}_\beta$ is the

decoder, q_{beta} is the encoder, and $\hat{\mathcal{T}}_\beta$ is the transition model;

2. Train a *discriminator*, D_θ, by minimizing the Adversarial Loss function (Formula 1.16);

3. Train a *policy* π_θ^L, by maximizing the Value function (Formula 1.17).

$$\max_\beta \mathbb{E}_{q_\beta}\Bigg[\sum_t \log(\hat{\mathcal{U}}_\beta(s_t|z_t)) + \mathbb{D}_{KL}(q_\beta(z_t|s_t, z_{t-1}, a_{t-1})||\hat{\mathcal{T}}_\beta(z_t|z_{t-1}, a_{t-1}))\Bigg] \tag{1.15}$$

$$\min_\theta \mathbb{E}_{(z,a)\sim\rho^E(z,a)}\left[-\log(D_\theta(z,a))\right] + \mathbb{E}_{(z,a)\sim\rho^{\pi^L}_{\hat{\mathcal{T}}}}\left[-\log(1-D_\theta(z,a))\right] \tag{1.16}$$

$$\max_{\pi_\theta^L} V^K_{\theta,\beta}(z_t) = \max_{\pi_\theta^L} \mathbb{E}_{\pi_\theta^L,\hat{\mathcal{T}}_\beta}\Bigg[\sum_{\tau=t}^{t+K-1} \gamma^{\tau-t}\log(D_\theta(z_\tau^{\pi_\theta^L}, a_\tau^{\pi_\theta^L})) + \gamma^K V_\beta(z_{t+K}^{\pi_\theta^L})\Bigg] \tag{1.17}$$

With this learning setting the proposed system outperforms previous works such as [68, 64] both in terms of **data-efficiency** and **overall performance**, on a set of continous control tasks.

Generally speaking, the Generative Adversarial Imitation Learning has shown very promising performance in simulated control tasks and simulated robot manipulation tasks, even in complex high-dimensional state-space. However, it is not so clear, how these methods could perform in real-world robotic manipulation tasks, in terms of data-efficiency, generalization capability, and safety during real-world interactions.

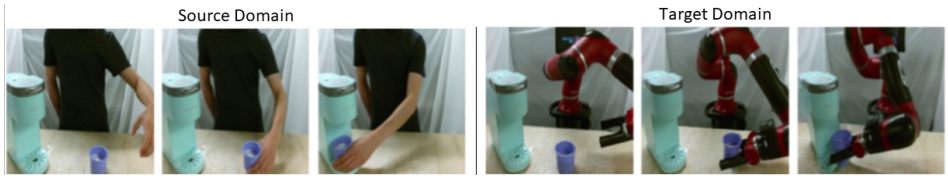


Figure 1.8: Representation of **embodiment mismatch problem**. (Left) The source domain represented by a video of human performing a task. (Right) The target domain, represented by the robot that executes the observed task

1.3.3.4 Learning from Observation

VEDERE CITAZIONI RECENTI MIMICPLAY,qian2024contrast

In all the previous sections, the methodologies assume access to the agent actions, working with state-action trajectories. In *Learning from Observation* (LfO), this assumption is relaxed, and methods for learning from **state-only** demonstrations are proposed. This approach has gained attention in recent years [34] because it theoretically allows a robotic system to be programmed as naturally as possible. Ideally, a robotic system should be able to reproduce a task by observing a human or another robot performing it, without access to the actions performed, unlike the methods described so far.

To address this problem, several questions need to be answered:

1. How can embodiment mismatches be resolved when the demonstrator has a different embodiment than the imitator?

2. How can the correspondence problem be handled when the demonstrator viewpoint differs from the imitator?

3. Once the perception subsystem issues are resolved, how is the policy π^L obtained?

The first question refers to the *correspondence problem* introduced in Section 1.3.2. This problem arises when the demonstrator embodiment differs from that of the learner, meaning that methods cannot directly use the recorded trajectories of the demonstrator.

One approach to solving this problem is to use methods that perform *image-to-image* translation. This involves using generative deep architectures to transform images of a subject in one domain (e.g., a human demonstrator) into images where the context remains the same, but the subject is different (e.g., the human demonstrator is replaced by the target robot) as depicted in Figure 1.8. This approach has been followed by authors in [31, 35, 72].

Specifically, the authors in [31, 35] used the Cycle-GAN architecture [73] to translate images from the source domain (human images) to the target domain (robot images) in an unsupervised manner. The work in [73] shifted the translation problem from a paired image setting, where each source domain image has a corresponding target domain image, to an unpaired image setting, where the source domain image does not have a corresponding target domain image.

To address this, Cycle-GAN introduces a novel learning procedure involving two translation models: $G : X \rightarrow Y$ and $F : Y \rightarrow X$. The first model maps inputs from the source domain to the target domain, while the second model maps inputs from the target domain back to the source domain. These two models are trained in an adversarial

setting by minimizing the loss function shown in Formula 1.18.

$$\begin{aligned}
\mathcal{L}(G, F, D_X, D_Y) &= \mathcal{L}_{GAN}(G, D_Y, X, Y) + \\
&\quad \mathcal{L}_{GAN}(F, D_X, Y, X) + \lambda \mathcal{L}_{cyc}(G, F) \\
\\
\mathcal{L}_{GAN}(Z, D_K, S, T) &= \mathbb{E}_{t \sim p_{data}(t)} \left[\log(D_K(t))\right] + \\
&\quad \mathbb{E}_{s \sim p_{data}(s)} \left[\log(1 - D_K(Z(s)))\right] \\
\\
\mathcal{L}_{cyc}(G, F) &= \mathbb{E}_{x \sim p_{data}(x)} \left[\|F(G(x)) - x\|_1\right] + \\
&\quad \mathbb{E}_{y \sim p_{data}(y)} \left[\|G(F(y)) - y\|_1\right]
\end{aligned} \tag{1.18}$$

Here, $\mathcal{L}_{GAN}$ is the adversarial loss component, where the discriminator D_K is trained to distinguish between real samples $t \in T$ and translated samples $Z(s)$. The generator Z is trained to generate samples that are as similar as possible to those in the target domain, starting from samples in the source domain. Meanwhile, $\mathcal{L}_{cyc}$ is a loss term that aims to maintain consistency between the generated samples and the ground truth one.

The application of this concept in the domain of interest, lead to a dataset for the source domain was composed of human demonstrations as well as a small amount of "random" data, in which the human moves around the scene but does not specifically attempt the task, while for the target domain, it consists of robot images executing randomly sampled actions in a few different settings.

The second question also addresses a variant of the correspondence problem, where, in addition to the embodiment mismatch, the problem of *different viewpoints* is also encountered (Figure 1.9a). This issue has been tackled in [74, 67].

In [74], a Convolutional Neural Network was trained using a *Triplet-Loss* [75]. The aim was to train a network to predict an embedding independent of the viewpoint, but containing only task-relevant fea-

tures. To achieve this, the network had to produce an embedding, $f(x)$, such that $||f(x_i^a) - f(x_i^p)||_2^2 + \alpha < ||f(x_i^a) - f(x_i^n)||_2^2$ for all $(f(x_i^a), f(x_i^p), f(x_i^n)) \in \mathcal{T}$, where $\mathcal{T}$ is the set of all possible triplets in the dataset. This implies that embeddings produced by samples from different viewpoints, but sharing the same time-step, (x_i^a, x_i^p), should be similar, while embeddings produced by samples from the same viewpoint, but at different time-steps, (x_i^a, x_i^n), should be different (Figure 1.9a).

In [67], a different approach was employed. Here, a *context translation problem* was addressed using an Encoder-Decoder architecture (Figure 1.9b). The proposed architecture was trained on pairs of demonstrations, $\mathcal{D}_i = [o_0^i, o_1^i, \ldots, o_T^i]$ and $\mathcal{D}_j = [o_0^j, o_1^j, \ldots, o_T^j]$, composed of visual observations. Samples in $\mathcal{D}_i$ come from the source context ω_i, while samples in $\mathcal{D}_j$ come from the target context ω_j. The model must output the observations in $\mathcal{D}_j$ conditioned on both $\mathcal{D}_i$ and the first observation o_0^j from the target domain.

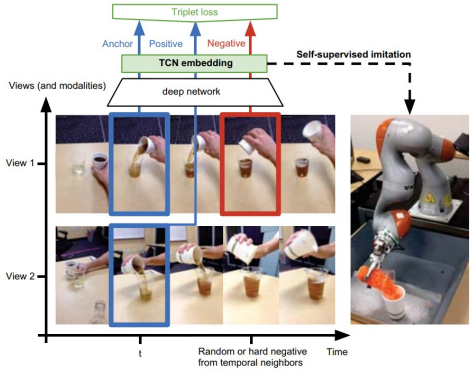
As will be explained next, the outputs of both the Time-Contrastive and the Context-Translation networks can be used to obtain an engineered reward function.

The third question concerns the method by which the final learned policy is derived. To present the different approaches, it is essential to distinguish between *Model-Free* and *Model-Based* methods (Figure 1.10).

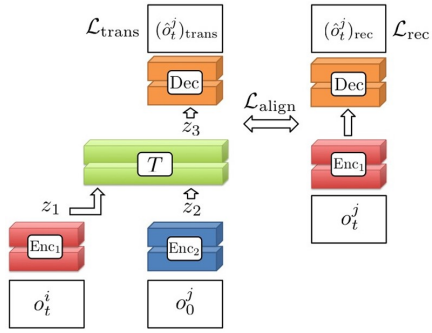
Model-Free

Model-Free methods are characterized by the fact that they do not leverage knowledge about the environment dynamics, which can either be given a priori or learned through data-driven approaches. A further classification must be done between *Reward Engineering* and *Adversarial Learning* approaches.

Reward Engineering methods are characterized by the use of a



(a) *Time-Contrastive network*, proposed in [74].



(b) *Context-Translation network*, proposed in [67]

Figure 1.9: Examples of how the mismatch between demonstrator viewpoint and learner viewpoint can be handled.

hand-designed reward function to train the policy according to the Reinforcement Learning paradigm [2]. Methods based on this approach include [67, 74, 35, 76].

In [67], the reward function is defined as in Formula 1.19.

$$R(o_t^l) = -\left\| Enc_1(o_t^l) - \frac{1}{n}\sum_{i=1}^{n} F(o_t^i, o_0^l) \right\|_2^2 - w_{rec}\left\| o_t^l - \frac{1}{n}\sum_{i=1}^{n} M(o_t^i, o_0^l) \right\|_2^2 \tag{1.19}$$

The first term is the classic Feature Tracking reward function, which aims to minimize the Euclidean Distance between the encoding of the current learner observation o_t^l and the encoding of the demonstration in the learner context. The second term penalizes the policy for experiencing observations that differ from the translated observation.

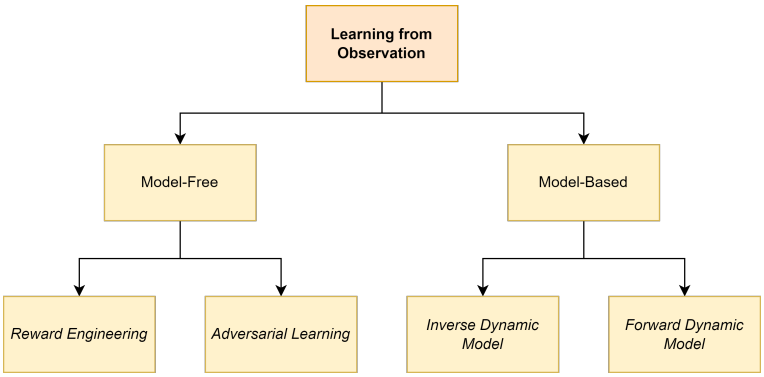


Figure 1.10: Learning from Observation taxonomy

In [74], the reward function is defined according to Formula 1.20.

$$R(\mathbf{v}_t, \mathbf{w}_t) = -\alpha \left\|\mathbf{w}_t - \mathbf{v}_t\right\|_2^2 - \beta\sqrt{\gamma + \left\|\mathbf{w}_t - \mathbf{v}_t\right\|_2^2} \tag{1.20}$$

Where $\mathbf{v}_t$ is the TCN embedding of the video demonstration at timestep t, and $\mathbf{w}_t$ is the TCN embedding produced by the robot observation (Figure 1.9a).

In [35], a **keypoint-representation** (Figure 1.11) is obtained for both the current robot observations z_t and each frame of the translated demonstration video $\{z_p^E\}_{p=1}^T$. The reward is then computed as in Formula 1.21.

$$\begin{aligned} R(z_t, z_{t+1}, z^E) = -\lambda_1 \min_p \left\|z_t - z_p^E\right\| - \\ \lambda_2 \min_p \left\|(z_{t+1} - z_t) - (z_{p+1}^E - z_p^E)\right\| \end{aligned} \tag{1.21}$$

The main idea is to generate actions that minimize the distance between the translated keypoints and the keypoints obtained from the current robot observation, thereby reproducing the demonstrated trajectory. The *min* operator is necessary because the robot and the

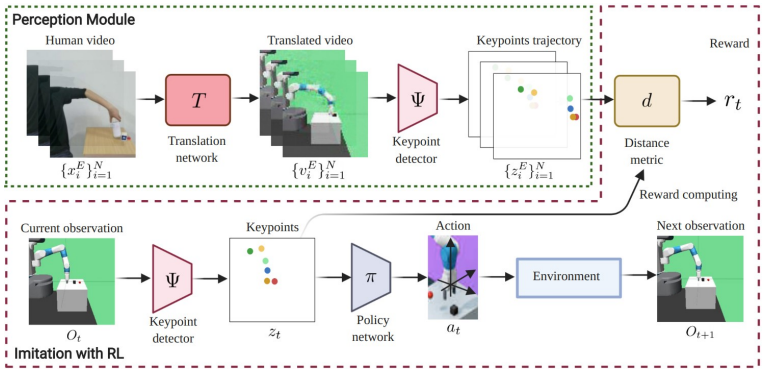


Figure 1.11: Architecture proposed by [35]

demonstration are not temporally aligned; there is no a priori knowledge about which demonstration frame corresponds to the current agent state.

In [76], the reward function was defined as $R(s_t) = -\frac{1}{k}\,||\phi(s_t) - g||_2^2$, where g is the goal embedding, defined as the mean embedding of the last frame of all the demonstration videos in the dataset, while $\phi(s_t)$ is the embedding of the current observation. Experimental results, on simulation data proved that the proposed method can be used to learn tasks from cross-embodiment demonstrations, outperforming baseline [74] in terms of both sample efficiency and performance.

Adversarial Learning methods rely on the Adversarial Learning paradigm and are closely related to the GAIL methods (Section 1.3.3.3). Unlike the methods discussed in the GAIL section, these methods do not assume access to the demonstrator's actions. Preliminary works in this area have been proposed in [77, 78].

The goal of authors in [77] was to demonstrate that the Adversarial Learning setting can be effectively used even without action information. To test this hypothesis, the authors conducted a series

Algorithm 5 GAIfO algorithm [78]

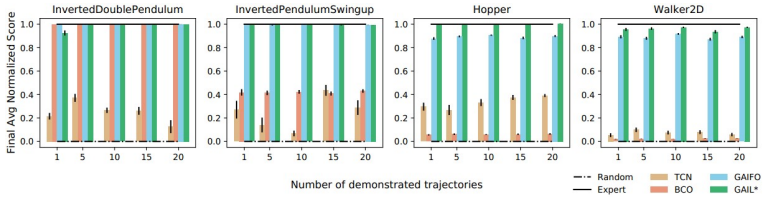
Require: Initial policy π_ϕ^L, Initial Discriminator D_θ
Require: State-only expert demonstration trajectories $\tau^E = \{(s, s')\}$

while Policy Improves **do**
  Execute π_ϕ^L and collect state transitions $\tau^L = \{(s, s')\}$
  Update D_θ, with $\mathcal{L}_{D_\theta} = -\ (\ \mathbb{E}_{\tau^L}[\log(D_\theta(s, s'))] + \mathbb{E}_{\tau^E}[\log(1 - D_\theta(s, s'))]\ )$
  Update π_ϕ^L, with reward $r_{\pi_\phi^L} = -\ (\ \mathbb{E}_{\tau^L}[\log(D_\theta(s, s'))]\ )$
end while

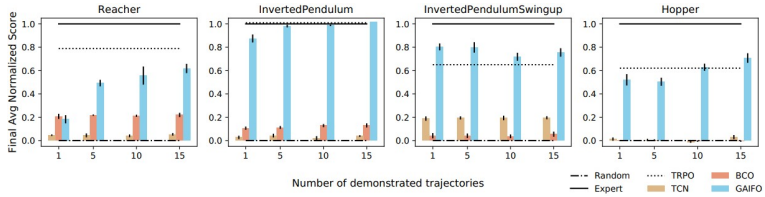
of experiments in simulation for a walking task, where the same RL policy was trained in two contexts: one where the Discriminator had access to the (state, action) pair, and another where the Discriminator had access to state-only demonstrations. The results showed no substantial difference between the two settings, supporting the hypothesis that the essential information for task learning is contained in the state.

The next significant work was proposed by the authors of [78], who formalized the *GAIfO* algorithm (Algorithm 5), an extension of GAIL [60] to state-only demonstrations. The proposed algorithm was used to train a network to solve tasks in a simulation environment [63], with both low-dimensional state representation and visual-state representation. Results regarding the number of demonstrated trajectories are reported in Figure 1.12.

As noted from the results, GAIfO outperforms previous observation-based methods [74, 79] in settings with a low number of expert trajectories. The main drawback of GAIfO is the **high number of environmental interactions** needed to learn a policy, as it uses the model-free TRPO [61] algorithm. This issue was addressed by DEALIO [80], which replaced the model-free algorithm with PILQR



(a) Experimental results in low-dimensional state space



(b) Experimental results in high-dimensional state space

Figure 1.12: Experimental results reported in [78].

[81], a model-based RL algorithm discussed next.

Model-Based

Model-Based methods leverage knowledge about the environment dynamics, which is learned through data-driven approaches. These methods can be further classified into *Inverse Dynamic Model* and *Forward Dynamic Model* approaches.

The *Inverse Dynamic Model* approach, given a transition (s_t, s_{t+1}), obtains a function M that maps state transitions to actions, i.e., $a_t = M(s_t, s_{t+1})$. In contrast, the *Forward Dynamic Model* approach, given a state-action pair (s_t, a_t), aims to learn a function F that generates the next state s_{t+1}, i.e., $s_{t+1} = F(s_t, a_t)$.

Inverse Dynamic Model methods include [82, 79, 83, 84].

In [82], the goal was to develop a system capable of tying a knot in a rope. A self-supervised learning approach was used to train a Convolutional Neural Network. Given a pair of images (I_t, I_{t+1}) representing

two successive rope states, the network was able to determine the action required to transition from state I_t to state I_{t+1}. The network was trained using a dataset of 30K tuples (I_t, a_t, I_{t+1}) collected via an exploratory policy.

Authors in [79] proposed a general approach, depicted in Figure 1.13, composed of two main parts: the learned Inverse Dynamic Model, M_θ, and the learned policy π_ϕ. The learning procedure is iterative. The model M_θ^i is updated by maximizing the probability $p_\theta(a_t|s_t, s_{t+1})$, using tuples (s_t, a_t, s_{t+1}) collected by the current policy. Once the dynamic model is updated, it infers the action $\tilde{a}_t$ given the demonstrations. The policy, having access to both state and action information, is then trained using classic Behavioral Cloning (BC) by optimizing the policy parameters through maximum-likelihood estimation $\phi^* = \underset{\phi}{argmax} \prod_{i=0}^{N} \pi_\phi^L(\tilde{a}_i|s_i)$.

In [83], a similar approach to [79] was used, but the agent's policy was trained using a combination of Behavioral Cloning and the Advantage Actor Critic (A2C) objective function [85] (Formula 1.22).

$$\begin{aligned} \mathcal{L}_\theta^{hyb} = \mathbb{E}_{s,a}\left[A(s)\log(a|s;\theta) + \alpha\ \mathcal{H}(\pi^L(.|s))\right] + \\ \mathbb{E}_{(\hat{s}_t,\hat{s}_{t+1})\sim D}\left[\log(\pi^L(M(\hat{s}_t,\hat{s}_{t+1})|\theta))\right]. \end{aligned} \tag{1.22}$$

The main drawback is the assumption of access to the reward function, which can limit its applicability in real-robot manipulation tasks.

In [84], the work in [86] was extended to state-only demonstrations, and the *State-Only Imitation Learning* (SOIL) algorithm was proposed. This method addresses complex dexterous manipulation tasks, such as object reallocation, tool use, in-hand manipulation, and door opening, using a simulated humanoid hand. A neural network representing the Inverse Dynamic Model was trained by minimizing the L2-loss, given the action performed during the policy rollout. The policy was then updated according to the *Demo Augmented Policy*

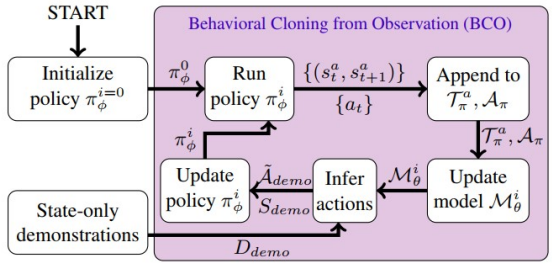


Figure 1.13: Representation of the learning procedure proposed by [79]

Gradient (DAPG) method [86], adapted for state-only demonstrations, which follows the gradient updates described by Formula 1.23.

$$g_{SOIL} = g + \lambda_0 \, \lambda_1^k \sum_{(s_t, \tilde{a}_t) \in D'} \nabla_\theta \log \pi_\theta^L(\tilde{a}_t, s_t), \tag{1.23}$$

where g is the Natural Policy Gradient term. The idea is to leverage the demonstrations at the beginning of the training and then exploit the RL algorithm to improve the behavior. Experiments performed in simulation showed that, compared to pure RL, the proposed method converges faster and produces more human-like behaviors.

Forward Dynamic Model methods include [31, 80].

In [31], once the human video demonstration was translated into the corresponding robot video, the policy was learned using the model-based RL algorithm SOLARIS [87]. This algorithm optimizes a controller using the Linear-Quadratic Regulator (LQR) procedure. The policy optimization occurs in a low-dimensional, highly regularized *latent space*, generated using Variational Inference [88]. Starting from a sequence of observations and actions, a Global Dynamic Model over the latent trajectory is obtained. Then, given the Latent Dynamic Model, a Linear-Gaussian Controller is derived using LQR-FLM [58]. Real-world robotic experiments demonstrated that with just **2 hours**

of robot interaction, it was possible to outperform previous works such as [74, 79] and classic BC algorithms in tasks like "coffee making" (Figure 1.8) and cup-retrieving, where the robot retrieves a cup from a closed drawer.

In [80], the sample inefficiency problem of GAIfO [78] was addressed. The approach combined the adversarial learning setting with state-only demonstrations, which had shown promising results (Figure 1.12), with a more data-efficient RL algorithm like PILQR [81]. PILQR's core is the LQR optimization procedure. Generally, it returns a *linear-gaussian controller* (Formula 1.24) that optimizes a *quadratic-cost function* (Formula 1.25) under the assumption of *linear-gaussian dynamics* (Formula 1.26).

$$\pi(a_t|s_t) = \mathcal{N}(K_t s_t + k_t, S_t) \tag{1.24}$$

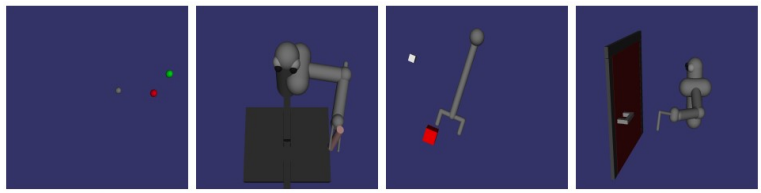
$$c(s_t, a_t) = \begin{bmatrix} s_t \\ a_t \end{bmatrix}^T C_t \begin{bmatrix} s_t \\ a_t \end{bmatrix} + \begin{bmatrix} s_t \\ a_t \end{bmatrix}^T c_t + cc_t \tag{1.25}$$

$$s_{t+1} \sim P(s_{t+1}|s_t, a_t) = \mathcal{N}(F_t \begin{bmatrix} s_t \\ a_t \end{bmatrix} + f_t, \Sigma_t) \tag{1.26}$$

To use this framework, the linear-gaussian dynamic model was fitted using the current policy rollouts. Then, to obtain a quadratic cost function as needed by LQR, the dynamic model was used to express the modified discriminator output (Formula 1.27) as a function of the pair (s_t, a_t).

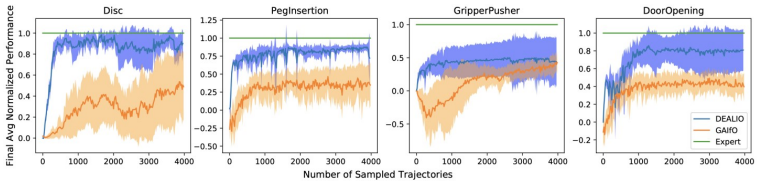
$$D_\theta(s_t, s_{t+1}) = \frac{1}{2} \begin{bmatrix} s_t \\ s_{t+1} \end{bmatrix}^T C^{ss}(s_t, s_{t+1}) \begin{bmatrix} s_t \\ s_{t+1} \end{bmatrix} + \begin{bmatrix} s_t \\ s_{t+1} \end{bmatrix}^T c^{ss}(s_t, s_{t+1}) \tag{1.27}$$

Experiments performed in simulation with low-dimensional state spaces showed promising results (Figure 1.14b) in terms of sample



(a) Disc (b) PegInsertion (c) GripperPusher (d) DoorOpening

(a) Control Tasks solved in [80]



(b) Performance of DEALIO [80] compared against GAIfO [78], with respect to the number of trajectories sampled during the learning process.

Figure 1.14: DEAILO: (1.14a) Control Tasks, (1.14b) Performance Level

efficiency compared to the GAIfO baseline. However, improvements are needed to: **(1)** Reduce variance to make the learning process more reliable, **(2)** Increase overall performance, **(3)** Adapt the algorithm for real-world robot manipulation tasks.

Generally, LfO methods have demonstrated interesting features, such as generating a policy from state-based information alone. This supports the hypothesis that the primary source of information for task learning is the sequence of state transitions. Extrapolating the valuable information to perform actions that induce the desired behavior may not be trivial, especially if the state space is represented by images of a human operator. This complexity leads to the design of architectures composed of different stages, which not only increase

the complexity of the system itself but also the amount and diversity of data required for their training. Furthermore, many methods of interest have been tested in simulated or relatively simple scenarios, still leaving open whether these methods can be used in real-world complex robotic manipulation tasks.

1.3.4 Graph Neural Network for planning systems

Chapter 2

Conditioned object detector

In this chapter the COD is going to be described. Specifically, Section 2.1 will outline the detection problem being addressed. Section 2.2 will detail the proposed architecture designed to solve the described problem. Section 2.3 will discuss the experimental setup and present the results obtained from testing the proposed architecture.

2.1 Problem formulation

2.2 Architecture

2.3 Experiments

In this section, the performed experiments are going to be described. Specifically, in Section 2.3.1 the dataset used for training procedure will be described. Section 2.3.2 will report the obtained results.

2.3.1 Dataset

2.3.2 Results

This section presents the obtained results, divided into two main blocks. The first block (Section 2.3.2.1) discusses the results of the method trained to detect only the target object. The second block (Section 2.3.2.1) covers the results of the method trained to detect both the target object and the final (placing) location. For each method, results are reported for two different scenarios: first, where the method is trained in a single-task multi-variation scenario; and second, where the method is trained in a multi-task multi-variation scenario.

2.3.2.1 Target object detector

Single-task multi-variation scenario

Multi-task multi-variation scenario

2.3.2.2 Target object and final position detector

Single-task multi-variation scenario

Multi-task multi-variation scenario

Chapter 3

Object conditioned control policy

In this chapter the OCCP is going to be described. Specifically, Section 3.1 will outline the problem being addressed. Section 3.2 will detail the proposed architecture designed to solve the described problem. Section 3.3 will discuss the experimental setup and present the results obtained from testing the proposed architecture.

3.1 Problem formulation

3.2 Architecture

This section describes the entire policy architecture, specifically focusing on how the COD (Chapter 2) is integrated. There are two control architectures, differing in how the final control module predicts actions. Section 3.2.1 details the architecture that uses a single control module to predict actions for both the reaching and placing phases. In contrast, Section 3.2.2 describes the architecture that divides the con-

trol module into two distinct parts: one for computing actions during the reaching phase, and another for the placing phase.

3.2.1 Single control module

3.2.2 Double control modules

3.3 Experimental results

In this section, the performed experiments are going to be described. Specifically, in Section 3.3.1 the dataset used for training procedure will be described. Section 3.3.2 will report the obtained results.

3.3.1 Dataset

3.3.2 Results

This section presents the obtained results, divided into two main blocks. The first block (Section 3.3.2.1) discusses the results of the method described in Section3.2.1. The second block (Section 3.3.2.2) covers the results of the method described in 3.2.2. For each method, results are reported for two different scenarios: first, where the method is trained in a single-task multi-variation scenario; and second, where the method is trained in a multi-task multi-variation scenario.

3.3.2.1 Single control module

Single-task multi-variation scenario

Multi-task multi-variation scenario

3.3.2.2 Double control modules

Single-task multi-variation scenario

Multi-task multi-variation scenario

Chapter 4

Graph Neural Network for heuristic estimation

Chapter 5

Real world application

This chapter details the validation of the proposed methods in a real world scenario. Specifically, Section 5.1 describes the experimental setup. Section 5.2 discusses the dataset used to train the system. Finally, Section 5.3 presents the results obtained.

5.1 Experimental Setting

5.2 Dataset

5.3 Results

Chapter 6

Conclusions

Bibliography

[1] S. Bini, G. Percannella, A. Saggese, and M. Vento, "A multi-task network for speaker and command recognition in industrial environments," *Pattern Recognition Letters*, vol. 176, pp. 62–68, 2023.

[2] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[3] T. Osa, J. Pajarinen, G. Neumann, J. A. Bagnell, P. Abbeel, J. Peters *et al.*, "An algorithmic perspective on imitation learning," *Foundations and Trends® in Robotics*, vol. 7, no. 1-2, pp. 1–179, 2018.

[4] T. Zhang, Z. McCarthy, O. Jow, D. Lee, X. Chen, K. Goldberg, and P. Abbeel, "Deep imitation learning for complex manipulation tasks from virtual reality teleoperation," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 5628–5635.

[5] A. Mandlekar, D. Xu, J. Wong, S. Nasiriany, C. Wang, R. Kulkarni, L. Fei-Fei, S. Savarese, Y. Zhu, and R. Martín-Martín, "What matters in learning from offline human demonstrations for robot manipulation," in *Conference on Robot Learning*. PMLR, 2022, pp. 1678–1690.

[6] S. Dasari and A. Gupta, "Transformers for one-shot visual imitation," in *Conference on Robot Learning*. PMLR, 2021, pp. 2071–2084.

[7] A. Brohan, N. Brown, J. Carbajal, Y. Chebotar, J. Dabis, C. Finn, K. Gopalakrishnan, K. Hausman, A. Herzog, J. Hsu, J. Ibarz, B. Ichter, A. Irpan, T. Jackson, S. Jesmonth, N. J. Joshi, R. Julian, D. Kalashnikov, Y. Kuang, I. Leal, K. Lee, S. Levine, Y. Lu, U. Malla, D. Manjunath, I. Mordatch, O. Nachum, C. Parada, J. Peralta, E. Perez, K. Pertsch, J. Quiambao, K. Rao, M. S. Ryoo, G. Salazar, P. R. Sanketi, K. Sayed, J. Singh, S. Sontakke, A. Stone, C. Tan, H. T. Tran, V. Vanhoucke, S. Vega, Q. Vuong, F. Xia, T. Xiao, P. Xu, S. Xu, T. Yu, and B. Zitkovich, "RT-1: robotics transformer for real-world control at scale," in *Robotics: Science and Systems XIX, Daegu, Republic of Korea, July 10-14, 2023*, K. E. Bekris, K. Hauser, S. L. Herbert, and J. Yu, Eds., 2023. [Online]. Available: https://doi.org/10.15607/RSS.2023.XIX.025

[8] Z. Mandi, F. Liu, K. Lee, and P. Abbeel, "Towards more generalizable one-shot visual imitation learning," in *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, 2022, pp. 2434–2444.

[9] S. Stepputtis, J. Campbell, M. Phielipp, S. Lee, C. Baral, and H. Ben Amor, "Language-conditioned imitation learning for robot manipulation tasks," *Advances in Neural Information Processing Systems*, vol. 33, pp. 13 139–13 150, 2020.

[10] O. Mees, L. Hermann, E. Rosete-Beas, and W. Burgard, "Calvin: A benchmark for language-conditioned policy learning for long-horizon robot manipulation tasks," *IEEE Robotics and Automation Letters (RA-L)*, vol. 7, no. 3, pp. 7327–7334, 2022.

[11] M. Zare, P. M. Kebria, A. Khosravi, and S. Nahavandi, "A survey of imitation learning: Algorithms, recent developments, and challenges," *IEEE Transactions on Cybernetics*, 2024.

[12] S. James, M. Bloesch, and A. J. Davison, "Task-embedded control networks for few-shot imitation learning," in *2nd Annual Conference on Robot Learning, CoRL 2018, Zürich, Switzerland, 29-31*

October 2018, Proceedings, ser. Proceedings of Machine Learning Research, vol. 87. PMLR, 2018, pp. 783–795. [Online]. Available: http://proceedings.mlr.press/v87/james18a.html

[13] E. Jang, A. Irpan, M. Khansari, D. Kappler, F. Ebert, C. Lynch, S. Levine, and C. Finn, "Bc-z: Zero-shot task generalization with robotic imitation learning," in *Conference on Robot Learning*. PMLR, 2022, pp. 991–1002.

[14] M. Shridhar, L. Manuelli, and D. Fox, "Perceiver-actor: A multi-task transformer for robotic manipulation," in *Conference on Robot Learning*. PMLR, 2023, pp. 785–799.

[15] B. Fang, S. Jia, D. Guo, M. Xu, S. Wen, and F. Sun, "Survey of imitation learning for robotic manipulation," *International Journal of Intelligent Robotics and Applications*, vol. 3, no. 4, pp. 362–369, 2019.

[16] O. Kroemer, S. Niekum, and G. Konidaris, "A review of robot learning for manipulation: Challenges, representations, and algorithms," *The Journal of Machine Learning Research*, vol. 22, no. 1, pp. 1395–1476, 2021.

[17] E. Johns, "Coarse-to-fine imitation learning: Robot manipulation from a single demonstration," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 4613–4619.

[18] R. Caccavale, M. Saveriano, A. Finzi, and D. Lee, "Kinesthetic teaching and attentional supervision of structured tasks in human–robot interaction," *Autonomous Robots*, vol. 43, no. 6, pp. 1291–1307, 2019.

[19] A. Mandlekar, Y. Zhu, A. Garg, J. Booher, M. Spero, A. Tung, J. Gao, J. Emmons, A. Gupta, E. Orbay *et al.*, "Roboturk: A crowdsourcing platform for robotic skill learning through imitation," in *Conference on Robot Learning*. PMLR, 2018, pp. 879–893.

[20] F. Ebert, Y. Yang, K. Schmeckpeper, B. Bucher, G. Georgakis, K. Daniilidis, C. Finn, and S. Levine, "Bridge Data: Boosting Generalization of Robotic Skills with Cross-Domain Datasets," in *Proceedings of Robotics: Science and Systems*, New York City, NY, USA, June 2022.

[21] A. Mandlekar, S. Nasiriany, B. Wen, I. Akinola, Y. Narang, L. Fan, Y. Zhu, and D. Fox, "Mimicgen: A data generation system for scalable robot learning using human demonstrations," in *Conference on Robot Learning*. PMLR, 2023, pp. 1820–1864.

[22] S. Dasari, F. Ebert, S. Tian, S. Nair, B. Bucher, K. Schmeckpeper, S. Singh, S. Levine, and C. Finn, "Robonet: Large-scale multi-robot learning," in *Conference on Robot Learning*. PMLR, 2020, pp. 885–897.

[23] M. Chang and S. Gupta, "One-shot visual imitation via attributed waypoints and demonstration augmentation," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 5055–5062.

[24] D. Lee and C. Ott, "Incremental kinesthetic teaching of motion primitives using the motion refinement tube," *Autonomous Robots*, vol. 31, pp. 115–131, 2011.

[25] M. Saveriano, S.-i. An, and D. Lee, "Incremental kinesthetic teaching of end-effector and null-space motion primitives," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015, pp. 3570–3575.

[26] A. Mandlekar, J. Booher, M. Spero, A. Tung, A. Gupta, Y. Zhu, A. Garg, S. Savarese, and L. Fei-Fei, "Scaling robot supervision to hundreds of hours with roboturk: Robotic manipulation dataset through human reasoning and dexterity," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2019, pp. 1048–1055.

[27] C. Systems, "Cyberforce." [Online]. Available: http://www.cyberglovesystems.com/cyberforce

[28] D. Systems, "Touch." [Online]. Available: https://it.3dsystems.com/haptics-devices/touch

[29] Y. Zhu, J. Wong, A. Mandlekar, R. Martín-Martín, A. Joshi, S. Nasiriany, and Y. Zhu, "robosuite: A modular simulation framework and benchmark for robot learning," *arXiv preprint arXiv:2009.12293*, 2020.

[30] H. Liu, C. Zhang, Y. Zhu, C. Jiang, and S.-C. Zhu, "Mirroring without overimitation: Learning functionally equivalent manipulation actions," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, pp. 8025–8033, Jul. 2019.

[31] L. Smith, N. Dhawan, M. Zhang, P. Abbeel, and S. Levine, "AVID: Learning Multi-Stage Tasks via Pixel-Level Translation of Human Videos," in *Proceedings of Robotics: Science and Systems*, Corvalis, Oregon, USA, July 2020.

[32] S. Nakaoka, A. Nakazawa, F. Kanehiro, K. Kaneko, M. Morisawa, H. Hirukawa, and K. Ikeuchi, "Learning from observation paradigm: Leg task models for enabling a biped humanoid robot to imitate human dances," *The International Journal of Robotics Research*, vol. 26, no. 8, pp. 829–844, 2007.

[33] H. Liu, X. Xie, M. Millar, M. Edmonds, F. Gao, Y. Zhu, V. J. Santos, B. Rothrock, and S.-C. Zhu, "A glove-based system for studying hand-object manipulation via joint pose and force sensing," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 6617–6624.

[34] F. Torabi, G. Warnell, and P. Stone, "Recent advances in imitation learning from observation," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence,*

IJCAI-19. International Joint Conferences on Artificial Intelligence Organization, 7 2019, pp. 6325–6331. [Online]. Available: https://doi.org/10.24963/ijcai.2019/882

[35] H. Xiong, Q. Li, Y.-C. Chen, H. Bharadhwaj, S. Sinha, and A. Garg, "Learning by watching: Physical imitation of manipulation skills from human videos," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2021, pp. 7827–7834.

[36] C. Wang, L. Fan, J. Sun, R. Zhang, L. Fei-Fei, D. Xu, Y. Zhu, and A. Anandkumar, "Mimicplay: Long-horizon imitation learning by watching human play," in *Conference on Robot Learning*. PMLR, 2023, pp. 201–221.

[37] Z. Qian, M. You, H. Zhou, X. Xu, H. Fu, J. Xue, and B. He, *IEEE Transactions on Automation Science and Engineering*, 2024.

[38] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of artificial intelligence research*, vol. 4, pp. 237–285, 1996.

[39] B. D. Argall, S. Chernova, M. Veloso, and B. Browning, "A survey of robot learning from demonstration," *Robotics and autonomous systems*, vol. 57, no. 5, pp. 469–483, 2009.

[40] A. Hussein, M. M. Gaber, E. Elyan, and C. Jayne, "Imitation learning: A survey of learning methods," *ACM Computing Surveys (CSUR)*, vol. 50, no. 2, pp. 1–35, 2017.

[41] B. Zheng, S. Verma, J. Zhou, I. Tsang, and F. Chen, "Imitation learning: Progress, taxonomies and opportunities," *arXiv preprint arXiv:2106.12177*, 2021.

[42] D. A. Pomerleau, "Alvinn: An autonomous land vehicle in a neural network," *Advances in neural information processing systems*, vol. 1, 1988.

[43] A. Ijspeert, J. Nakanishi, and S. Schaal, "Learning attractor landscapes for learning motor primitives," *Advances in neural information processing systems*, vol. 15, 2002.

[44] A. J. Ijspeert, J. Nakanishi, H. Hoffmann, P. Pastor, and S. Schaal, "Dynamical movement primitives: learning attractor models for motor behaviors," *Neural computation*, vol. 25, no. 2, pp. 328–373, 2013.

[45] W. Si, N. Wang, and C. Yang, "Composite dynamic movement primitives based on neural networks for human–robot skill transfer," *Neural Computing and Applications*, vol. 35, no. 32, pp. 23 283–23 293, 2023.

[46] J. Li, J. Wang, S. Wang, and C. Yang, "Human–robot skill transmission for mobile robot via learning by demonstration," *Neural Computing and Applications*, vol. 35, no. 32, pp. 23 441–23 451, 2023.

[47] Y. Fanger, J. Umlauft, and S. Hirche, "Gaussian processes for dynamic movement primitives with application in knowledge-based cooperation," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2016, pp. 3913–3919.

[48] A. Paraschos, C. Daniel, J. R. Peters, and G. Neumann, "Probabilistic movement primitives," *Advances in neural information processing systems*, vol. 26, 2013.

[49] M. Saveriano, F. J. Abu-Dakka, A. Kramberger, and L. Peternel, "Dynamic movement primitives in robotics: A tutorial survey," *The International Journal of Robotics Research*, vol. 42, no. 13, pp. 1133–1184, 2023.

[50] Y. Zhou, J. Gao, and T. Asfour, "Learning via-point movement primitives with inter-and extrapolation capabilities," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2019, pp. 4301–4308.

[51] D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke *et al.*, "Scalable deep reinforcement learning for vision-based robotic manipulation," in *Conference on robot learning*. PMLR, 2018, pp. 651–673.

[52] P. Abbeel and A. Y. Ng, "Apprenticeship learning via inverse reinforcement learning," in *Proceedings of the twenty-first international conference on Machine learning*, 2004, p. 1.

[53] N. D. Ratliff, J. A. Bagnell, and M. A. Zinkevich, "Maximum margin planning," in *Proceedings of the 23rd international conference on Machine learning*, 2006, pp. 729–736.

[54] N. D. Ratliff, D. Silver, and J. A. Bagnell, "Learning to search: Functional gradient techniques for imitation learning," *Autonomous Robots*, vol. 27, no. 1, pp. 25–53, 2009.

[55] B. D. Ziebart, A. L. Maas, J. A. Bagnell, A. K. Dey *et al.*, "Maximum entropy inverse reinforcement learning." in *Aaai*, vol. 8. Chicago, IL, USA, 2008, pp. 1433–1438.

[56] M. Wulfmeier, P. Ondruska, and I. Posner, "Maximum entropy deep inverse reinforcement learning," *arXiv preprint arXiv:1507.04888*, 2015.

[57] C. Finn, S. Levine, and P. Abbeel, "Guided cost learning: Deep inverse optimal control via policy optimization," in *International conference on machine learning*. PMLR, 2016, pp. 49–58.

[58] S. Levine and P. Abbeel, "Learning neural network policies with guided policy search under unknown dynamics," *Advances in neural information processing systems*, vol. 27, 2014.

[59] N. Das, S. Bechtle, T. Davchev, D. Jayaraman, A. Rai, and F. Meier, "Model-based inverse reinforcement learning from visual demonstrations," in *Conference on Robot Learning*. PMLR, 2021, pp. 1930–1942.

[60] J. Ho and S. Ermon, "Generative adversarial imitation learning," *Advances in neural information processing systems*, vol. 29, 2016.

[61] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International conference on machine learning*. PMLR, 2015, pp. 1889–1897.

[62] J. Ho, J. Gupta, and S. Ermon, "Model-free imitation learning with policy optimization," in *International conference on machine learning*. PMLR, 2016, pp. 2760–2769.

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

[64] I. Kostrikov, K. K. Agrawal, D. Dwibedi, S. Levine, and J. Tompson, "Discriminator-actor-critic: Addressing sample inefficiency and reward bias in adversarial imitation learning," in *International Conference on Learning Representations*, 2018.

[65] J. Fu, K. Luo, and S. Levine, "Learning robust rewards with adverserial inverse reinforcement learning," in *International Conference on Learning Representations*, 2018.

[66] S. K. S. Ghasemipour, R. Zemel, and S. Gu, "A divergence minimization perspective on imitation learning methods," in *Conference on Robot Learning*. PMLR, 2020, pp. 1259–1277.

[67] Y. Liu, A. Gupta, P. Abbeel, and S. Levine, "Imitation from observation: Learning to imitate behaviors from raw video via context translation," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 1118–1125.

[68] S. Reddy, A. D. Dragan, and S. Levine, "Sqil: Imitation learning via reinforcement learning with sparse rewards," in *International Conference on Learning Representations*, 2019.

[69] K. Zolna, S. Reed, A. Novikov, S. G. Colmenarejo, D. Budden, S. Cabi, M. Denil, N. de Freitas, and Z. Wang, "Task-relevant adversarial imitation learning," in *Conference on Robot Learning*. PMLR, 2021, pp. 247–263.

[70] R. Rafailov, T. Yu, A. Rajeswaran, and C. Finn, "Visual adversarial imitation learning using variational models," *Advances in Neural Information Processing Systems*, vol. 34, pp. 3016–3028, 2021.

[71] G. Barth-Maron, M. W. Hoffman, D. Budden, W. Dabney, D. Horgan, D. Tb, A. Muldal, N. Heess, and T. Lillicrap, "Distributed distributional deterministic policy gradients," *arXiv preprint arXiv:1804.08617*, 2018.

[72] J. Li, T. Lu, X. Cao, Y. Cai, and S. Wang, "Meta-imitation learning by watching video demonstrations," in *International Conference on Learning Representations*, 2021.

[73] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2223–2232.

[74] P. Sermanet, C. Lynch, Y. Chebotar, J. Hsu, E. Jang, S. Schaal, S. Levine, and G. Brain, "Time-contrastive networks: Self-supervised learning from video," in *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2018, pp. 1134–1141.

[75] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet: A unified embedding for face recognition and clustering," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 815–823.

[76] K. Zakka, A. Zeng, P. Florence, J. Tompson, J. Bohg, and D. Dwibedi, "Xirl: Cross-embodiment inverse reinforcement learning," in *Conference on Robot Learning*. PMLR, 2022, pp. 537–546.

[77] J. Merel, Y. Tassa, D. TB, S. Srinivasan, J. Lemmon, Z. Wang, G. Wayne, and N. Heess, "Learning human behaviors from motion capture by adversarial imitation," *arXiv preprint arXiv:1707.02201*, 2017.

[78] F. Torabi, G. Warnell, and P. Stone, "Generative adversarial imitation from observation," *arXiv preprint arXiv:1807.06158*, 2018.

[79] ——, "Behavioral cloning from observation," in *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, 2018, pp. 4950–4957.

[80] ——, "Dealio: Data-efficient adversarial learning for imitation from observation," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2021, pp. 2391–2397.

[81] Y. Chebotar, K. Hausman, M. Zhang, G. Sukhatme, S. Schaal, and S. Levine, "Combining model-based and model-free updates for trajectory-centric reinforcement learning," in *International conference on machine learning*. PMLR, 2017, pp. 703–711.

[82] A. Nair, D. Chen, P. Agrawal, P. Isola, P. Abbeel, J. Malik, and S. Levine, "Combining self-supervised learning and imitation for vision-based rope manipulation," in *2017 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2017, pp. 2146–2153.

[83] X. Guo, S. Chang, M. Yu, G. Tesauro, and M. Campbell, "Hybrid reinforcement learning with expert state sequences," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 3739–3746.

[84] I. Radosavovic, X. Wang, L. Pinto, and J. Malik, "State-only imitation learning for dexterous manipulation," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2021, pp. 7865–7871.

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

[86] A. Rajeswaran, V. Kumar, A. Gupta, G. Vezzani, J. Schulman, E. Todorov, and S. Levine, "Learning complex dexterous manipulation with deep reinforcement learning and demonstrations," in *Proceedings of Robotics: Science and Systems*, Pittsburgh, Pennsylvania, June 2018.

[87] M. Zhang, S. Vikram, L. Smith, P. Abbeel, M. Johnson, and S. Levine, "Solar: Deep structured representations for model-based reinforcement learning," in *International Conference on Machine Learning*. PMLR, 2019, pp. 7444–7453.

[88] D. P. Kingma and M. Welling, "Auto-Encoding Variational Bayes," in *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014.

List of Figures

List of Tables