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 inferred from data. This data can be generated either by **agent experience** [2] or by **expert demonstrations** [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 demonstrations, the control policy parameters are directly tuned using a dataset containing examples of task execution. Here, the goal is to replicate the tasks observed in the dataset.

Given this background, the thesis is framed in the context of *Learning from Demonstration* (LfD), a learning approach based on expert demonstrations. 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 it is possible to achieve a system capable of solving new variations by leveraging the knowledge-sharing hypothesis.

In this scenario, the learning procedure is much more challenging because there is the 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, they are typically defined as items present in the scene but never involved in manipulation operations. Modern deep architectures can handle this scenario effectively, as they can easily learn to ignore these objects since they do not participate in any manipulation tasks. However, in the context proposed in this thesis, the problem is

further complicated by the fact that the semantic meaning of an object (i.e., target or 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 primary contribution of this thesis is tackling the challenge of distractor objects. A key issue identified in existing literature is **target misidentification**, where the learned control policy generates valid trajectories, enabling the robot to reach, pick, and place objects, but frequently manipulates the wrong object.

To solve this problem, two main considerations were made:

(1) Architectures proposed in the current literature are predominantly **end-to-end**, translating high-dimensional inputs, such as images, into corresponding low-dimensional actions. As a result, the model must learn an implicit representation that encodes both the task objective and the current state of the environment, including the location of the target object.

(2) The learning procedure optimizes an **action-centric metric**, meaning that it is not directly linked to task success but instead focuses on mimicking the expert's actions on average. This action-focused optimization can lead to poor encoding of critical information, such as object positions.

These two factors can result in a control policy that fails to effectively guide the robot toward the target object. In particular, it was observed that the early stages of trajectory execution are critical. Even small errors during these initial steps can cause the robot to reach and ultimately pick the wrong object.

Based on these considerations, this thesis explores the development of a **modular** architecture instead of an end-to-end approach. This architecture features modules specifically designed for reasoning about the objects of interest (e.g., target

object and placement location). The outputs of these reasoning modules are then integrated to simplify the learning problem for the Control Module. This module is now informed by low-dimensional information, such as the position of the target object, which may be more effectively utilized during the learning process, especially in light of the action-centric cloning loss.

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 category-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 extensively tested in simulation environments, then it was also validated on a real-world robotic platform.

Regarding the simulated environment, the system was evaluated on both **multi-variation single-task** scenarios and **multi-variation multi-tasks** scenarios, considering four simulated 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, the proposed method was tested also in a real-world environment, where the complexity of the problem is heightened by the presence of less and noisy data collected through teleoperation. Even under these challenging conditions, the proposed method demonstrated its effectiveness in addressing both the cognitive and control problems. This confirms that the object prior can be successfully applied in real-world scenarios, enabling the development of a reliable system despite limited and noisy trajectory data.

Contents

Will robots inherit the earth? Yes, but they will be our children.

- Marvin Lee Minsky -

Chapter 1

Introduction

1.1 Motivation and thesis overview

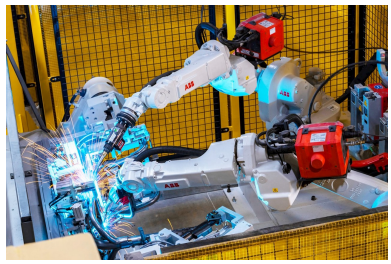
This section presents the primary motivation for this thesis and provides an overview of its structure.

A key objective in robotics is to develop autonomous robots that can perform a wide range of manipulation tasks, such as pick-and-place and assembly operations, in response to specific commands. These tasks often involve varying degrees of freedom, like object categories, for a specific task there can different objects of interest, and positions, the initial state of the environment (how and where the object are arranged) is not fixed.

In traditional industrial settings, manually coded control rules are effective for managing situations where robots follow fixed and repetitive paths. This is possible because the categories and positions of objects are known in advance, and the robot workspace remains constant over time (e.g., the workcell shown in Figure 1.1).

However, the modern landscape of social robotics and contemporary industrial applications requires a higher level of **adaptability** and **flexibility**. Robots are now expected to operate in environments shared with human operators, receiving commands and interacting with them in a collaborative or cooperative manner. For example, a robot may be tasked with picking up a tool and delivering it to an operator. This requires the robot to not only recognize different object categories and

estimate their positions but also to correlate the outcomes of environmental analysis with the commands received, adapting its actions accordingly to exhibit "intelligent" behaviors.



(a) Robots involved in arc welding operation

(b) Robot involved in loading operation

Figure 1.1: Industrial Robots: example of applications

To address this problem, the scientific community has focused on evaluating the use of *data-driven* approaches. These methods include Imitation Learning algorithms, which leverage data from examples of desired behaviors, often referred to as demonstrations, to enable a robot to replicate the demonstrated tasks.

Multi-Task Imitation Learning methods, as highlighted in [11, 6, 8, 7], are particularly promising due to their ability to meet the requirements of both high adaptability and flexibility. These methods use a **multi-task dataset**, which contains demonstrations for n different tasks (e.g., pick-and-place, nut-assembly, etc.), to fit a *single control function* denoted as $\pi_\theta^L(a_t|s_t, c_{m_i})$. This function maps the current observed state s_t and the command c_{m_i} into the corresponding action a_t to be executed by a physical robot. The command c_{m_i} refers to the m^{th} variation of the i^{th} task (e.g., the pick-and-place task can have different variations based on the target object and the target placing spot).

While these methods have shown significant potential, challenges remain in both single-task and multi-task scenarios. As reported in [11, 12], existing systems struggle with identifying critical task points, such as determining when to close the gripper near an object or when to open it near the placement area, rather than understanding the broader intent of the task.

Moreover, performance drops occur when scenes involve distractor objects, i.e., objects that do not contribute to task execution. This underscores the need for enhanced capabilities to correctly correlate commands with the results of scene analysis to identify target objects.

In the context presented so far, this thesis addresses the problem of *Visual-Conditioned Multi-Task Imitation Learning*. The goal is to develop a system that can advance the realization of a *versatile collaborative robot* suitable for industrial applications. This robot should be capable of executing tasks directed by a human operator and acquiring new skills based on a limited number of demonstrations, building upon its existing knowledge. To illustrate potential applications, consider a collaborative workspace where a human operator could instruct the robot to provide a tool or engage in assembly tasks. Specifically, the focus is on exploring methods that rely on *visual inputs*, where the current state s_t is an image depicting the robot's workspace, and the command c_{m_i} is given in terms of video demonstrations of an agent (e.g., robot or human) executing the desired task.

This thesis addresses the problem of Visual-Conditioned Multi-Task Imitation Learning from a different perspective. Current state-of-the-art methods typically address this challenge using end-to-end architectures trained with an action-centric behavioral cloning loss. In these methods, the system receives high-level inputs (such as images of a task demonstration and the agent current observation) and generates corresponding low-level control actions.

However, in scenarios involving multiple tasks or variations of tasks, manipulation contains both control and cognitive problems. Specifically, two main challenges must be addressed:

1. **Command Analysis and Understanding**: The first challenge is analyzing the given command. From a high-level task command, the system needs to understand the **intent** (e.g., picking and placing versus assembling), identify the relevant objects (e.g., selecting the blue box instead of the red one), and recognize the required skill (e.g., reaching, following, or picking).

2. **Action Generation**: The second challenge is correlating features from the observed state with the information derived from the command. The goal is to create an intermediate representation that integrates relevant information from both inputs (e.g., focusing on the portion of the image containing the target object). This representation should highlight key details necessary for inferring the correct action based on both the current state and the command.

Addressing these challenges with end-to-end architectures is particularly difficult, especially in scenes with multiple similar objects that could either be distractors or the actual object of interest depending on the task. Indeed, as shown by preliminary results and evaluation of state-of-the-art methods, there is a recurring issue of **target object misidentification**. While these methods can produce control policies that result in smooth and reasonable behaviors for high-level tasks like pick-and-place or nut-assembly, they often manipulate the wrong object, failing to complete the task correctly.

To address this issue, this thesis proposes an approach that explicitly tackles the two challenges described before, introducing modular architectures, rather than end-to-end systems. These modular architectures are composed of modules that are specifically designed to handle the tasks of command analysis and understanding, as well as action generation.

Specifically, this approach tries to leverage the well-known capabilities of deep architectures in solving problems like object detection to introduce **object priors**. These object priors can provide crucial information to the control system about the regions of interest in the workspace where objects of interest are located, thereby improving the **robustness** of the system against distractor objects as well as the **interpretability** of the system, since the cognitive module can return information about the region of the image where the robot is intended to move.

In conclusion, this thesis is organized as follows. Section 1.2 presents an overview of the classic methods used to address the Learning from Demonstration problem. Chapter 2 introduces

the Conditioned Object Detection module, designed to solve the cognitive aspect of the decision-making problem. Chapter 3 presents the Object Conditioned Control Policy, aimed at solving the control aspect of the decision-making problem. Chapter 4 details the validation of these methods in a real-world scenario.

Chapter **??** presents an initial exploration of how learning algorithms could enhance the capabilities of the proposed system, specifically by analyzing the performance of systems based on Graph Neural Networks for heuristic estimation, in order to understand whether these methods can be integrated into the proposed architecture to generalize to multi-step tasks.

Finally, Section 6 provides a summary of the results and discusses potential future research directions.

1.2 Learning from Demonstrations

The chapter provides a comprehensive review of LfD problem, offering an overview of the problem itself and the progression of methodologies developed to address it over time. Specifically, Section 1.2.1 will focus on the formal definition of the LfD problem. Following that, Section 1.2.3 will discuss various approaches and methodologies used to solve the problem. This section will include a detailed taxonomy of these approaches, highlighting their respective advantages and disadvantages, and concluding with considerations that are relevant to the objectives of this thesis.

1.2.1 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 that define the actions to perform based on the knowledge of the dynamic of the environment, 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 infer control rules from demonstrated trajectories [3].

In this context, two general actors must be defined: the **expert**, who demonstrates the desired behaviors, and the **learner**, whose goal is to learn and replicate the behaviors demonstrated by the expert [3, 13]. 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 $\boldsymbol{\tau}_i^E \sim \pi^E$. 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 [14], representing the robot and its workspace, or from a first-person point of view with the camera mounted on the robot [11] or on the gripper [10]. Additionally, 3D point-clouds generated by RGB-D images can also be used [15]. 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: $\hat{a}_t = \pi^L(s_t, c_i)$[11]. 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, 16], 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., $a_i = \left[\Delta p_x^{\mathcal{W}}, \Delta p_y^{\mathcal{W}}, \Delta p_z^{\mathcal{W}}, \Delta r_x^{\mathcal{W}}, \Delta r_y^{\mathcal{W}}, \Delta r_z^{\mathcal{W}}\right]$ [11].

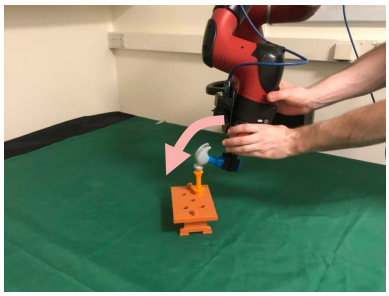
The expert and the learner, through their policies π^E and π^L, act on an environment modeled as a *Markov Decision Pro-*

cess (MDP) [17]. 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.
- 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 1.2.3.3 and Section 1.2.3.4.



(a) Example of kinesthetic teaching [18]

(b) Example of teleoperation [4]

Figure 1.2: Examples of direct demonstration

1.2.2 Source of demonstration

As stated in Section 1.2.1, IL methods rely on a dataset $\mathcal{D}^E$ of expert demonstrations. In this section, the different way this dataset can be collected are going to be discussed [16].

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* [19, 18], *teleoperation* [4, 20, 11, 7, 21, 22], or a *(hand-)written policy* [23, 6, 8, 24].

In kinesthetic teaching (Figure 1.2a), 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 [25, 26] 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.

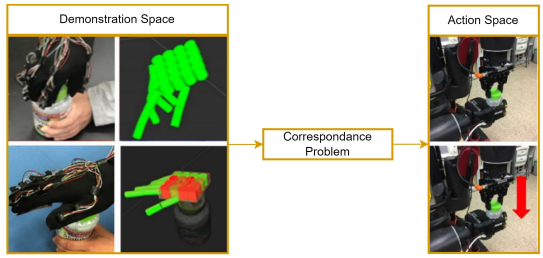
In teleoperation (Figure 1.2b), the human operator remotely guides the robot with a joystick, control panel, or wearable device. These tools allow for higher safety since there is no direct contact between the robot and the human expert. Teleoperation systems have been used in various works. For example,

the authors in [20, 27] proposed a teleoperation framework named Roboturk, which enables the collection of large-scale demonstration datasets [27, 5] for both simulated and real-world robots using a mobile phone as the controller. The authors in [4, 11, 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 [28, 29].

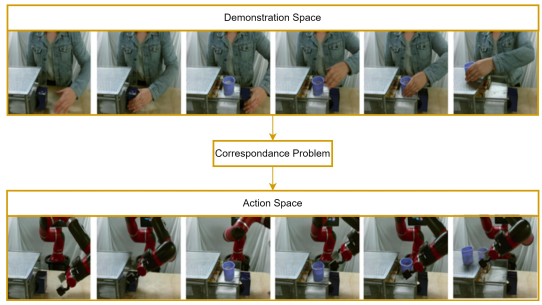
Demonstrations collected with hand-written policies rely on the fact that the expert has access to ground truth information, such as the position of the object of interest. This assumption can be valid when a simulated environment is used to train, test, and validate the proposed methods, as seen in [6, 8, 24]. In these cases, the authors used a well-known simulation environment in the robotic learning community named Robosuite [30]. 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 [23] 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 1.3) 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 record-



(a) Example of indirect demonstration based on wearable device [31]



(b) Example of direct demonstration based on human video demonstration [32]

Figure 1.3: Examples of indirect demonstration

ings, 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 [33, 31]. For example, in [33], the authors used a motion capture system to record the movement of a dancer and then transfer these trajectories to a humanoid robot. Similarly, in [31], the authors used a tactile glove [34] to record the movement performed by a human hand in the operation of opening bottles (Figure 1.3a).

In this line of research related to indirect demonstrations, there are also novel methods [32, 35, 36, 37, 38] 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 1.3b). 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 [31]. 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 1.2.3.4, the different ways this problem has been solved in the context of visual demonstration will be explained in detail.

1.2.3 Methodologies

This section is dedicated to presenting and analyzing various approaches for solving the LfD problem described in Section 1.2.1. Specifically, the proposed taxonomy is derived from studying different review papers [39, 40, 41, 16, 42, 13]. Figure 1.4 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 1.2.3.1 reviews methods for the fully-supervised learning methodology known as Behavioral Cloning.
Section 1.2.3.2 discusses methods under the umbrella of Inverse Reinforcement Learning, which solve the inverse optimization problem by first learning the reward function and then us-

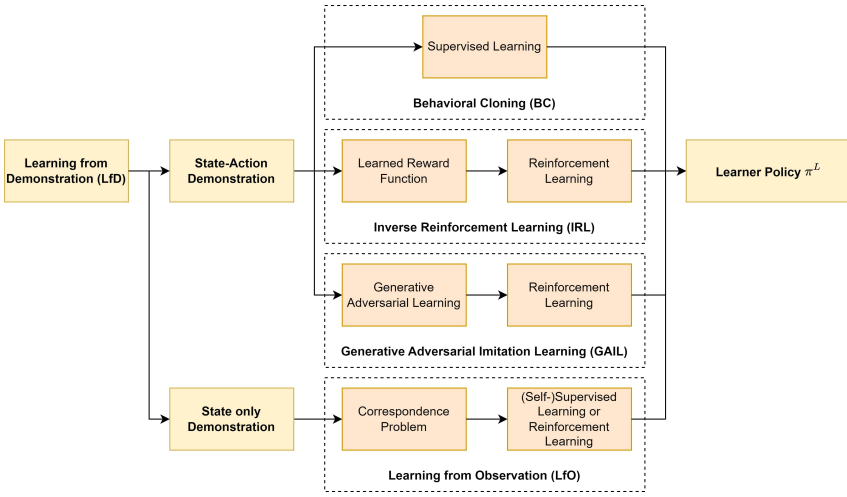


Figure 1.4: Taxonomy of LfD methods, divided based on type of demonstration and the learning algorithm used to learn the learner policy π^L

ing it to guide policy optimization following a reinforcement paradigm.
Section 1.2.3.3 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 1.2.3.4 covers the most recent methodology, Learning from Observation, characterized by optimizing the learner policy using state-only demonstrations. Each paragraph will present the research in the same way, i.e., the proposed works will be presented in a temporal order, highlighting the evolution of techniques and approaches over time. It is important to note that with respect to the problem managed in this thesis, the most relevant and most related literature is the one preseneted in Section 1.2.3.1. However, for sake of clarity and completeness, all the other approaches will be described with their pros/cons, with some considerations with respect to the proposal of this thesis.

1.2.3.1 Behavioral Cloning

Behavioral Cloning is one of the first approaches used to

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 $\mathcal{D}^E$

solve the LfD problem [43]. The high-level **supervised-learning** procedure 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 [43] or the parameters of a dynamic system [44]. 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 [44]. 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 [44, 45], 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 [44] 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 [46], Neural-Network Radial

Basis Functions [47], or Gaussian Processes [48]. Therefore, the choice of how to model the force term is a hyperparameter 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 [49] 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 [50].

To manage arbitrary desired trajectories, the authors in [51] proposed a novel framework for learning movement primitives, named *Via-points Movement Primitives* (VMP). VMP extends both Dynamic Movement Primitives (DMPs), which can only adapt to new start and goal positions but cannot directly handle intermediate via-points, and Probabilistic Movement Primitives (ProMPs), which can adapt to via-points within the statistical distribution of the demonstrated trajectories.

To achieve generalization across trajectories, the authors of VMP modeled the trajectory as a combination of two terms: the *elementary trajectory* h and a *shape modulation* f, expressed as $y(x) = h(x) + f(x)$ (Figure 1.5). The elementary trajectory serves as the foundational path that directly connects the start and goal of the demonstrated motion, and it can follow the formulation of a linear trajectory with constant velocity (Formula 1.4) or a minimum jerk trajectory (Formula 1.5). Specifically, the elementary trajectory can directly connect with linear segments two points that can be either the start and goal point, or the start and a via-point as well as the via-point and the goal point.

$$\begin{aligned} y(x) &= (y_0 - g)\, x + g + f(x) \\ h(x) &= (y_0 - g)x + g \end{aligned} \tag{1.4}$$

$$\begin{aligned} y(x) &= \textstyle\sum_{k=0}^{5} a_k x^k + f(x) \\ h(x_0) = y_0 - f_0,\ \dot{h}(x_0) &= \dot{y}_0 - \dot{f}_0,\ \ddot{h}(x_0) = \ddot{y}_0 - \ddot{f}_0 \\ h(x_1) = y_1 - f_1,\ \dot{h}(x_1) &= \dot{y}_1 - \dot{f}_1,\ \ddot{h}(x_1) = \ddot{y}_1 - \ddot{f}_1 \end{aligned} \tag{1.5}$$

In contrast, the shape modulation $f(x)$ is a linear model $f(x) = \psi(x)^T w + \epsilon_f$, where $\psi(x)$ is a Radial Basis Function and w is the set of learnable weights. This modulation term allows for adjustments to the trajectory to accommodate specific via-points.

In this framework, the learning procedure consists of two main components:

1. **Learning the shape modulation**, which involves learning the prior probability distribution of the parameter vectors w, characterized by the mean μ_w and variance Σ_w.

2. **Modifying the elementary trajectory**, which entails determining whether the requested via-point can be accommodated by adjusting the shape modulation or by introducing a new linear segment into the elementary trajectory.

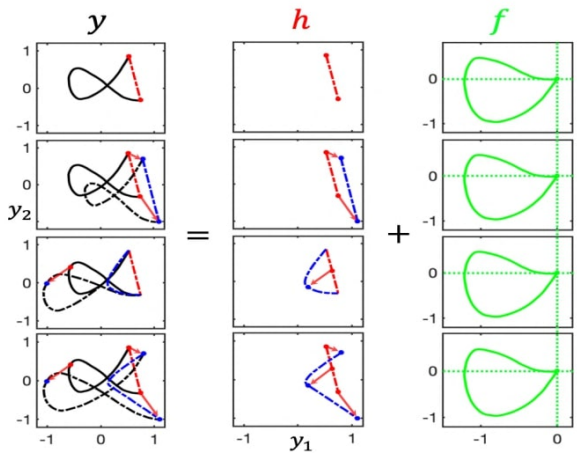


Figure 1.5: Graphical representation of the idea behind VMP [51]. The final trajectory y is represented as the sum of two components: the elementary trajectory h, and the shape modulation f. The elementary trajectory can directly connect two points (e.g., start and goal points) with a linear segment.

To achieve this, VMP calculates the conditional probability of the shape modulation parameters given the desired via-point. If this probability exceeds a certain threshold η, the shape modulation is adjusted to ensure the trajectory passes through the via-point. However, if the probability is below η, indicating that the desired via-point is unlikely based on the learned prior probability of w, VMP will modify the elementary trajectory instead.

Overall, the VMP framework enhances the adaptability of movement primitives by enabling the robot to learn and adjust its trajectories based on a limited number of demonstrations and specified via-points, making it a practical solution for various robotic applications.

In addition to these methodological advancements, several works have leveraged DMPs to solve specific robot manipulation problems [52, 19, 53]. In all of these cited works, DMPs have been employed to address the problem of task decomposition, specifically the identification of different skills (e.g., pick, pour, place, reach, etc.) involved in a task.

For instance, in the preliminary work [52], the authors used

a set of predefined motion primitives modeled as DMPs. The objective was to recognize these primitives within the demonstrated trajectory using an Expectation-Maximization algorithm. Similarly, in [19], DMPs were utilized to learn and reproduce motion primitives from demonstrations during the kinesthetic teaching of structured tasks. In this work, action segmentation was performed based on either object proximity or explicit human commands. When the robot manager identifies a new action, the corresponding DMP is learned, and the sequence of tasks is organized into a hierarchical structure.

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

This paragraph will review the research conducted in the context of Single-Task Imitation Learning. Specifically, within the scope of robotic manipulation problems, the term "Single-Task" indicates that the learned policy π^L can perform only the specific task it has been trained on. For example, if the task involves a pick-and-place operation with a fixed place position, the model cannot handle variations in the place location. Additionally, the focus will be primarily on methods that use high-dimentional state representations, such as images, processed by deep architectures to solve the problem.

In this scenario, the scientific literature extends far back in time. One of the seminal works in this field was proposed in 1988 by Pomerleau, who introduced *ALVINN* [43]. ALVINN is an autonomous vehicle driving system based on a Neural Network that predicts the steering angle from a synthetic camera image input. The network was trained on pairs of (image, steering angle), with the training procedure framed as a supervised classification problem. This was achieved by discretizing the steering angle into 45 units. Pomerleau's work immediately highlighted the issue of **compounding error**, which

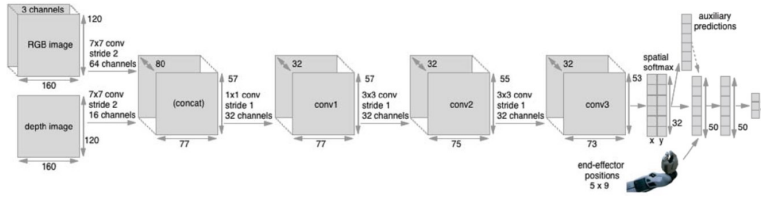


Figure 1.6: Architecture proposed in [4]

arises from the **covariate shift phenomenon**. This issue occurs because an action a_t influences the subsequent state s_{t+1}, which becomes the next sample, thereby violating the i.i.d. assumption of Supervised Learning. This results in a test-data distribution that may differ from the training one. This phenomenon has significant consequences on the expected performance of the system and is addressed by methods discussed in the paragraph on *Interactive Imitation Learning*.

Despite the covariate-shift problem, [4] showed that very interesting performance can be obtained in the context of Robot Manipulation, by means of Behavioral Cloning and high quality demonstrations given by teleportation system. In this work, a Convolutional Neural Network was trained to predict the desired linear-velocity, angular-velocity of the end-effector, and the binary gripper state (open/close), given in input the current RGB-D observation of the scene, and the position of three points of the end-effector, during the last 5 time-steps (Figure 1.6). The system was tested on 10 tasks, and the performance are reported in Table 1.1. The proposed system achieved a high success rate while evaluating all the tasks. The tests were carried out from different initial conditions but still quite similar to those present in the training set (e.g., the initial object positions have been uniformly distributed within the training regime, with random local variations around these positions). The analysis of failure cases showed that the leading cause of errors was the inability to detect critical points in the task execution, such as closing/opening the gripper to pick/place the object or detect the position of the object of interest in order to avoid collision with it.

Generally speaking, when working with these types of systems, there are various aspects and design choices to consider.

Table 1.1: Statistics of Training set, and Test Success rate [4]

Task	**Reaching**	**Grasping**	**Pushing**	**Plane**	**Cube**	**Nail**	**Grasp-and-Place**	**Grasp-Drop-Push**	**Grasp-Place-x2**	**Cloth**
#demo	200	180	175	319	206	215	109	100	60	100
demo duration (min)	13.7	11.1	16.9	25.0	12.7	13.6	12.3	14.5	11.6	10.1
Test success rate (%)	91.6	97.2	98.9	87.5	85.7	87.5	96.0	83.3	80.0	97.4

These include the selection of the architecture (e.g., temporal-dependent or independent), the type of demonstration (human-generated or machine-generated), the quantity of demonstrations, and so on. The authors in [5] identify a set of challenges in *Learning from Offline Human Demonstrations* and propose an extensive study, offering valuable insights for future work.

Specifically, the authors tested three Imitation Learning algorithms and two Offline Reinforcement Learning algorithms in both simulated and real-world manipulation tasks (Figure 1.7). The study primarily focused on analyzing the following aspects:

1. *Source of Demonstrations*: Comparing system performance with demonstrations generated by hand-written policies (MG), proficient human demonstrators (PH), and multiple human demonstrators with varying levels of expertise in teleoperation (MH).

2. *Observation Space*: Comparing the performance of methods based on the type of input, whether low-dimensional (gripper pose, gripper fingers position, object position) or image-based (visual observation, gripper pose, gripper fingers position).

Specifically, by extrapolating the most important results from this study (Table 1.2), it can be observed that the most promising architecture is the recurrent BC-RNN, particularly for datasets composed of multi-human demonstrations (MH). This architecture outperforms even state-of-the-art offline reinforcement learning algorithms, which tend to struggle with datasets containing trajectories of varying quality.

In conclusion, for real-world tasks, the BC-RNN architecture achieved a success rate of **96.7**% on the lift task, **73.3**%

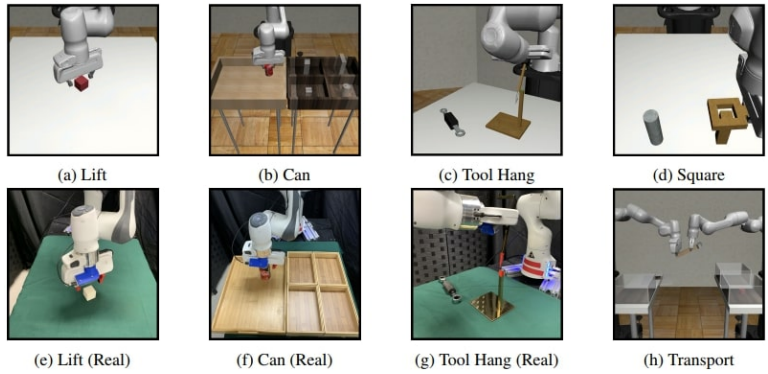


Figure 1.7: The set of tasks presented in the benchmark [5].

on the can task, and **3.3**% on the tool-hang task. These results are noteworthy, as they essentially demonstrate the feasibility of training a proficient system from offline human demonstrations. However, several aspects must be considered, such as the simplicity of the test scenarios, which include only one object without any distractors, and the potential benefit of incorporating contact information for contact-rich manipulation tasks like the tool-hang.

After this work, further research in the field of Single Task Imitation Learning, focused on exploring novel architecture [54] and learning paradigm [55, 56, 57].

Regarding the learning paradigm, the contribution presented in [55] is particularly noteworthy. The authors introduced R3M (Reusable Representations for Robotic Manipulation), exploring the potential of leveraging pre-trained visual backbones in the context of robotic manipulation. In traditional computer vision tasks, such as object detection, it is standard practice to use backbones pre-trained on large, general datasets and fine-tune them for specific problems, significantly reducing training time. However, this approach is not as widely adopted in robotic manipulation, mainly due to the vastly different evaluation scenarios, including variations in tasks, robot embodiments, and environments. This raises the question of whether the fine-tuning approach is also applicable to robotic manipulation problems.

To address this question, the authors in [55] started with

Table 1.2: Results are presented on tasks performed in a low-dimensional observation space for simulated environments. PH refers to *Proficient Human*, which represents trajectories collected by a single expert human demonstrator with extensive experience in teleoperating the robot. MH refers to *Multi Human*, which represents trajectories collected by multiple human operators with varying levels of expertise in teleoperation.

Dataset	BC	BC-RNN	BCQ	CQL
Lift (PH)	$\mathbf{100.0 \pm 0.0}$	$\mathbf{100.0 \pm 0.0}$	$\mathbf{98.0 \pm 1.6}$	52.0 ± 13.0
Can (PH)	$\mathbf{97.3 \pm 1.9}$	$\mathbf{98.0 \pm 0.9}$	86.7 ± 2.5	0.7 ± 0.9
Square (PH)	62.0 ± 4.9	$\mathbf{82.0 \pm 0.0}$	41.3 ± 4.1	-
Transport (PH)	55.3 ± 6.2	$\mathbf{72.0 \pm 4.3}$	0.7 ± 0.9	-
Tool Hang (PH)	20.0 ± 5.9	$\mathbf{67.3 \pm 4.1}$	3.3 ± 0.9	-
Lift (MH)	$\mathbf{100.0 \pm 0.0}$	$\mathbf{100.0 \pm 0.0}$	93.3 ± 0.9	11.3 ± 9.3
Can (MH)	85.3 ± 0.9	$\mathbf{96.0 \pm 1.6}$	77.3 ± 6.8	0.0 ± 0.0
Square (MH)	46.0 ± 1.6	$\mathbf{76.7 \pm 3.4}$	17.3 ± 7.5	-
Transport (MH)	18.7 ± 2.5	$\mathbf{42.0 \pm 1.6}$	0.0 ± 0.0	-

the Ego4D dataset [58], which contains over 3,500 hours of video footage of people engaging in a wide range of tasks, from cooking to socializing to assembling objects, across more than 70 locations worldwide. They trained a ResNet50 [59] to generate a robust representation that could be leveraged in robotic manipulation tasks. The authors proposed a self-supervised learning procedure designed to capture the following aspects:

- *Temporal Dynamics*: The learned representation should account for features related to physical interactions. To achieve this, the authors introduced a **Time Contrastive Loss**, implemented as an InfoNCE loss, which creates similar embeddings for time-adjacent frames while keeping frames that are far apart in time or from different videos distinct.

- *Semantic Meaning*: The learned representation should encode information related to the task itself. To capture this, the authors introduced a language prediction module. Given the representation for the first frames of a task k (ϕ_0^k), the representation for the i^{th} frame of the same or a different task (ϕ_i^j), and a language description l referring to task k, the system must output a score indi-

cating whether the transition from ϕ_0^k to ϕ_i^j corresponds to the description l. In this way, the system is trained to encode a representation that contains semantic features related to the task itself, enabling it to determine whether a given representation corresponds to a specific task.

In testing, the authors demonstrated that using the pre-trained R3M representation improves the overall success rate while requiring fewer demonstrations. On the Meta-World benchmark, with just 5 demonstrations, R3M achieved a success rate of nearly 60%, compared to 30% for a system trained from scratch.

Authors in [56], introduced the paradigm of "Diffusion Learning" in the context of policy learning for robotic manipulation. Generally speaking, the concept behind the Diffusion learning paradigm is to model the output of a network as a *denoising process*. This means that,starting from x^k sampled from Gaussian noise, the denoising process performs K iterations of denoising to produce a series of intermediate samples with decreasing levels of noise, $x_k, x_{k-1}, \ldots, x_0$, until a desired noise-free output x_0 is formed. This process follows the equation in Formula 1.6, where ϵ_θ is the noise prediction network whose parameters are trained during learning.

$$x^{k-1} = \alpha \left(x^k - \gamma \epsilon_\theta(x^k, k) + \mathcal{N}(0, \sigma^2 I) \right) \tag{1.6}$$

The authors adapted this concept to the context of robot manipulation, beginning with the observation that the denoising process can be applied to an action a_t^k. However, a challenge arises from the need for the model to generate an action conditioned on the current observation. This implies that the model must learn the conditional probability distribution $p(a_t \mid o_t)$ rather than the joint data distribution $p(a_t, o_t)$. To address this issue, the authors proposed the architecture illustrated in Figure 1.8.

Specifically, in both architectures, the model takes the latest T_o steps of observation data O_t (observation horizon) as input and predicts T_p steps of actions (prediction horizon), of which T_a steps of actions are executed on the robot without re-planning.

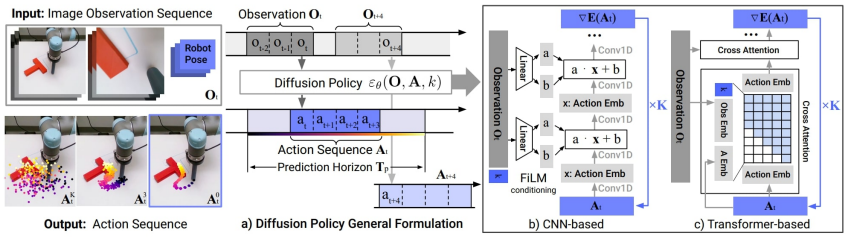


Figure 1.8: Architecture presented in [56]. (a) General formulation, at time step t, the policy inputs the latest T_o steps of observation data O_t and outputs T_a steps of actions A_t. (b) CNN-based Diffusion Policy, the observation feature O_t is conditioned using FiLM [60]. Starting with A_t^K from Gaussian noise, the noise-prediction network ϵ_θ iteratively subtracts noise to obtain the denoised action sequence A_t^0. (c) Transformer-based Diffusion Policy, the observation embedding O_t is fed into a multi-head cross-attention layer within each decoder block, with causal attention applied to constrain each action embedding to attend only to itself and prior actions.

After training the system by minimizing the Mean Squared Error Loss, interesting results were obtained during testing. Specifically, the tasks depicted in Figure 1.7 showed near-perfect results across all tasks (Lift, Can, Square, Transport, and Tool Hang), demonstrating the diffusion model's ability to better handle intrinsically noisy demonstrations from the MH set. Additionally, the Transformer-based architecture achieved an average success rate of 100% on the two most complex tasks in the benchmark, Transport and Tool Hang.

Another important work related to Single-Task Imitation Learning is presented in [57]. The authors address the issue of compounding errors, but instead of using the Interactive Learning Paradigm (discussed in the following paragraph), they propose a novel approach. They observed that a trajectory can be decomposed into a limited number of key states or waypoints. By linearly interpolating between these waypoints, it is possible to reconstruct a demonstrated trajectory with a certain degree of accuracy, thereby mitigating the effect of compounding errors. This is because the network predicts far fewer future states compared to methods that predict every step of the

trajectory. Building on this insight, the authors introduced the "Automatic Waypoint Extraction" (AWE) system. This system, which acts as a pre-processing tool, takes as input a trajectory τ and decomposes it into a certain number of waypoints. From these waypoints, an approximate trajectory $\hat{\tau}$ can be interpolated, with a controlled error margin. Specifically, to decompose the original trajectory τ authors formalized the optimization problem in Formula 1.7, where they basically wants to find the minimum number of waypoints W such that a given reconstruction loss $\mathcal{L}$ is lower than a certain margin η.

$$\min_{W} |W| \; s.t. \mathcal{L}((f(W), \tau)) \leq \eta \tag{1.7}$$

They solved this problem by implementing a Dynamic Programming-based algorithm, which identifies the shortest subsequence such that the reconstruction loss is less than η, while ensuring that the points in the subsequence are restricted to the original trajectory. The authors applied this preprocessing tool in conjunction with state-of-the-art architectures, such as the Diffusion Policy [56], to the same tasks presented in [5]. Notably, they observed that, compared to the near-optimal results of [56], the Diffusion Policy combined with the AWE system achieves good performance with significantly less data. Specifically, in the Square task, the system achieved an average success rate of 91.7% with only 100 demonstrations, compared to 82.0% with the Diffusion Policy alone.

In conclusion, there is significant research interest in the field of Imitation Learning for robotic manipulation tasks. However, the primary limitation of these Single-Task methods is that they fall short of the ideal concept of a **general-purpose robot** capable of solving any prompted task, which is the focus of this thesis. To address this limitation, **Multi-Task Imitation Learning** systems have been proposed. Indeed, as stated in Section 1.1, one goal is to have an adaptable and general system able to perform multiple-prompted task. These methods will be discussed in details in the Section 3.1 of the core Chapter 3. Despite this, the discussion of these methods is crucial, as the approaches presented later in this thesis build upon the foundational concepts introduced here.

Interactive Imitation Learning

The Interactive Imitation Learning approach encompasses all methods specifically designed to address or mitigate the compounding-error phenomenon, which was first described in [43].

As previously mentioned, this problem arises from the fact that, while Behavioral Cloning (BC) primarily follows a supervised learning procedure, it does not satisfy the i.i.d. assumption. This is because the agent's actions influence subsequent observations, creating a dependency between samples in the training set. As a result, when the agent interacts with the environment, even small errors can lead to new observations that fall outside the training distribution, potentially leading to an unrecoverable situation that the robot cannot resolve autonomously.

The significance of this problem was first formalized in [61]. The authors observed that if a system makes an error with probability ϵ in a task with a time horizon of T, then, due to the compounding of errors, a supervised learner incurs a quadratic total cost of $O(\epsilon\, T^2)$, instead of the expected linear cost of $O(\epsilon\, T)$. The quadratic term arises because, at any given time step t, the agent's state is influenced by errors made in the previous $t-1$ steps. This cumulative effect breaks the independence assumption typically held in the i.i.d. setting.

To attenuate this problem, interactive supervised learning algorithms have been proposed, such as the well-known *DAgger* [62]. Algorithm 2 describes the DAgger procedure. It is an aggregation strategy, based on the idea to train the policy π^L under the state-distribution induced by the policy itself, but with the correct action performed by the expert. The main problem with DAgger is that it requires the expert to interact with the system during the training, introducing both **safety** and **data-efficiency** problems, especially when the system does not provide the human expert with sufficient control authority during the sampling process [63].

Human-Guided DAgger (HG-DAgger) [64] is an enhancement of the traditional DAgger strategy, where a human expert oversees the rollout of the current policy. If the agent moves into an unsafe region of the state space, the expert steps in to guide the system back to safety. specifically, HG-DAgger

Algorithm 2 DAgger Algorithm [62]

Require: Initial Dataset $\mathcal{D} \leftarrow \emptyset$, Initial policy π_1^L
Ensure: The best policy π_i^L
  for $i = 1, \dots N$ **do**
    Sample $T - step$ trajectories using π_i^L
    Let $\mathcal{D}_i = (s_t, \pi^E(s_t))$, state s_t visited by policy π_i^L, and actions given by the expert
    Aggregate Dataset, $\mathcal{D} \leftarrow \mathcal{D} \bigcup \mathcal{D}_i$
    Train policy π_i^L on $\mathcal{D}$
    Let $\pi_{i+1}^L = \beta_i \pi^E + (1 - \beta_i)\pi_i^L$
  end for

was proposed in the context of autonomous vehicle driving, however, in [11], it was shown that HG-DAgger is particularly effective in robotic manipulation tasks. The study found that, given the same total number of episodes, a policy trained exclusively on expert demonstrations has a significantly lower success rate than one trained on a dataset that includes both expert demonstrations and expert corrections.

In the context of Interactive Learning for Robot Manipulation, other works of interest include [65, 66].

In [66], a human expert provides both **corrective** and **evaluative** feedback. The former consists in the human that takes control of the robot to adjust the trajectory, the latter consists in a scalar weight q, set to 1 if the trajectory is satisfactory, 0 if the trajectory is not satisfactory, α if the trajectory is adjusted by the expert, where α is the ratio between non-corrected and corrected samples. Then a Neural Network was trained by minimizing a weighted version of the maximum-likelihood, $\mathcal{L}(a_t, s_t) = -q \ log(\pi_\theta^L(a_t|s_t))$. Real-world experiments show that with a training time of **41 minutes**, including environmental reset, it was possible to have an agent capable of performing tasks such as picking up a cube or pulling a plug.

1.2.3.2 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 [67], 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 [68], and it is described by Algorithm 3. 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.

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.

Algorithm 3 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

- 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 [69, 70].

(b) The *Maximum Entropy* (Max-Ent) approach [71, 72, 73].

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 1.8.

$$\max_{\omega,m} m \quad \text{s.t.} \quad R_\omega(\boldsymbol{\tau}^E) \geq \max(R_\omega(\boldsymbol{\tau})) + m \tag{1.8}$$

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 1.9).

$$\max_{\psi} \mathcal{H}(\pi^{r_\psi}) \quad \text{s.t.} \quad \mathbb{E}_{\pi^{r_\psi}}[\mathbf{f}] = \mathbb{E}_{\pi^*}[\mathbf{f}] \tag{1.9}$$

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 [72], 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 [73].

In particular, [73] 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 1.10).

$$\mathcal{L}_\theta = \frac{1}{N} \sum_{\tau_i^E \in D_{demo}} c_\theta(\tau_i^E) + \log(Z) \tag{1.10}$$

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 4 was proposed. The algorithm returns a *Linear-Quadratic Gaussian* [74] controller q_k, obtained by solving the problem in Formula

Algorithm 4 Guided-Cost-Learning Algorithm [73]

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 [74] and using $\mathcal{D}_{traj}$
  end for

1.11.

$$\begin{aligned} q_k &= \arg\min_q \mathbb{E}[c_\theta(\tau)] - \mathcal{H}(\tau) \\ \text{s.t.} \quad p(s_{t+1}|s_t, a_t) &= \mathcal{N}(s_{t+1}; f_{x_t} x_t + f_{u_t} u_t, F_t) \end{aligned} \tag{1.11}$$

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

Recent methods have aimed to advance IRL towards more complex learning scenarios, such as learning a reward function from video demonstrations. In [75], the architecture shown in Figure 1.9 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 1.12).

$$\mathbf{a}_{new} = \mathbf{a} - \eta \nabla_a C_\Psi(\hat{\boldsymbol{\tau}}, z_{goal}). \tag{1.12}$$

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.

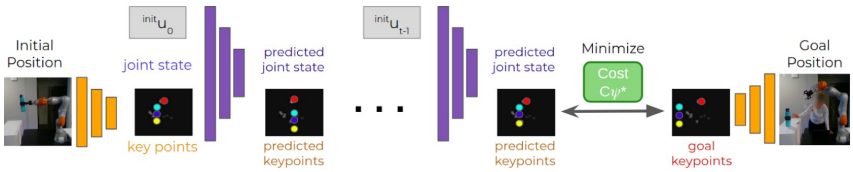


Figure 1.9: Architecture proposed in [75]

1.2.3.3 Generative Adversarial Imitation Learning

The *Generative Adversarial Imitation Learning* (GAIL) is a Learning from Demonstration (LfD) approach proposed by the authors of [76]. 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 [76] started from the Max-Ent formulation in Formula 1.13, 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 1.13, $\psi(c)$ 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 1.15, the conjugate in Formula 1.16 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.

$$IRL_{\psi}(\pi^E) = \underset{c \in \mathbb{R}^{S \times A}}{\arg\max} - \psi(c) + \left(\min_{\pi^L \in \Pi} - \mathcal{H}(\pi^L) + \mathbb{E}_{\pi^L}\left[c(s,a)\right] \right) - \mathbb{E}_{\pi^E}\left[c(s,a)\right] \tag{1.13}$$

$$RL \circ IRL_{\psi}(\pi^E) = \underset{\pi^L \in \Pi}{arg\ min} - \mathcal{H}(\pi^L) + \psi^*(\rho_{\pi^L} - \rho_{\pi^E}) \tag{1.14}$$

$$\begin{aligned} \psi_{GA}(c) &= \begin{cases} \mathbb{E}_{\pi^E}\left[g(c(s,a))\right] & \text{if } c < 0 \\ +\infty & \text{otherwise} \end{cases}, \\ g(x) &= \begin{cases} -x - \log(1 - e^x) & \text{if } c < 0 \\ +\infty & \text{otherwise} \end{cases} \end{aligned} \tag{1.15}$$

$$\psi^*_{GA}(\rho_{\pi^L} - \rho_{\pi^E}) = \underset{D \in (0,1)^{S \times A}}{max} \mathbb{E}_{\pi^L}\left[\log(D(s,a))\right] + \mathbb{E}_{\pi^E}\left[\log(1 - D(s,a))\right] \tag{1.16}$$

Based on these considerations, Algorithm 5 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 [76]). The policy is updated in such a way that the trajectories it generates become 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 [76], GAIL has proven to be more effective than classic IRL (Inverse Reinforcement Learning) algorithms [71, 78]. The authors evaluated the GAIL algorithm on nine classic simulated control tasks from the OpenAI Gym

Algorithm 5 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 [77], 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

simulator [79]: Cartpole, Acrobot, MountainCar, HalfCheetah, Hopper, Walker, Ant, Humanoid, and Reacher.

The observation and control spaces for these tasks are detailed in Table 1.3, and the performance results are shown in Figure 1.10. 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 achieved by replacing the model-free, on-policy TRPO algorithm with an off-policy RL algorithm, as seen in [80], or by modifying the reward function input to the RL algorithm [81, 82].

However, as noted in Table 1.3, the tested tasks are characterized by low-dimensional state spaces. More recent research [83, 84, 85, 86] 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 [85, 86].

In [85], the authors focused on solving the **casual-confusion** problem. This problem occurs when the discriminator, during the learning process, focuses on task-irrelevant features be-

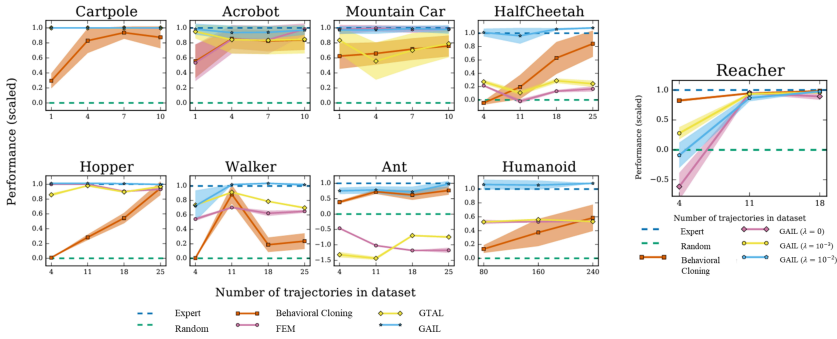


Figure 1.10: The performance comparison proposed in [76] is presented here. The y-axis shows the scaled reward, where the expert's reward is set to 1 and the random baseline is set to 0. The IRL baselines FEM and GTAL refer to the IRL algorithm described in [78], but with different cost functions.

Table 1.3: Observation and Action space for the tasks used in [76]

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)

tween 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 [85] two elements have been proposed:

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 1.17. 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 1.18.

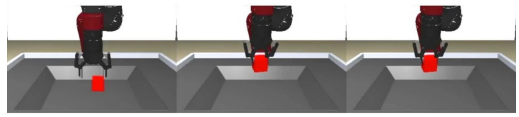
$$accuracy(I_E, I_A) = \frac{1}{2}\,\mathbb{E}_{s\in I_E}\left[\mathbf{1}_{D_\omega \geq \frac{1}{2}}\right] + \frac{1}{2}\,\mathbb{E}_{s\in I_A}\left[\mathbf{1}_{D_\omega < \frac{1}{2}}\right] \quad (1.17)$$

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

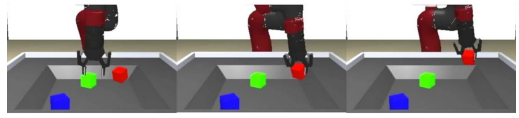
$$\begin{aligned} \text{where } \; G_\psi\left(s_E, s_A\right) &= \sum_{i=1}^{N} \log D_\psi\left(s_E^{(i)}\right) \\ &\quad + \log\left[1 - D_\psi\left(s_A^{(i)}\right)\right] \end{aligned} \quad (1.18)$$

The proposed system was tested on 4 tasks (Figure 1.11), with the agent trained on each single task, according to the *Distributed Distributional Deterministic Policy Gradients* (D4PG) [87] 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 [76] baseline, both in setting with spurious features and without spurious features (Figure 1.12).

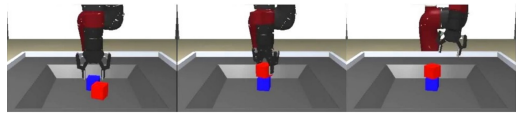
The authors of [86] proposed a more data-efficient Adversarial Imitation Learning method. They leveraged a model-based approach within a high-dimensional state space. Instead of generating a dynamic model in the image space, i.e., training a generative model to produce the next image based on the current 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.



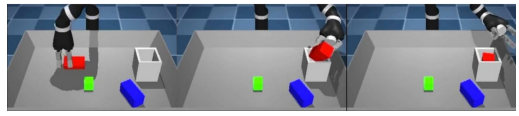
(a) Lifting task



(b) Lifting with distractors task



(c) Block Stacking task



(d) Block Insertion with distractors task

Figure 1.11: Tasks solved in [85]

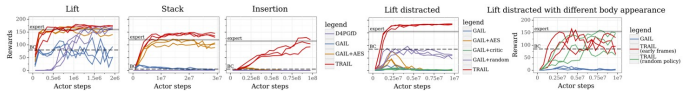


Figure 1.12: Experimental results on tasks without and with spurious features [85]

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 1.19), 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 1.20);

3. Train a *policy* π_θ^L, by maximizing the Value function (Formula 1.21).

$$\max_{\beta} \mathbb{E}_{q_\beta} \Bigg[\sum_t \log(\hat{\mathcal{U}}_\beta(s_t|z_t)) + \mathbb{D}_{KL}(q_\beta(z_t|s_t, z_{t-1}, a_{t-1})||\hat{\mathcal{T}}_\beta(z_t|z_{t-1}, a_{t-1})) \Bigg] \tag{1.19}$$

$$\min_{\theta} \mathbb{E}_{(z,a)\sim\rho^E(z,a)} [-\log(D_\theta(z,a))] + \mathbb{E}_{(z,a)\sim\rho^{\pi^L}_{\hat{\mathcal{T}}}} [-\log(1 - D_\theta(z,a))] \tag{1.20}$$

$$\max_{\pi^L_\theta} V^K_{\theta,\beta}(z_t) = \max_{\pi^L_\theta} \mathbb{E}_{\pi^L_\theta, \hat{\mathcal{T}}_\beta} \Bigg[\sum_{\tau=t}^{t+K-1} \gamma^{\tau-t} \log(D_\theta(z_\tau^{\pi^L_\theta}, a_\tau^{\pi^L_\theta})) + \gamma^K V_\beta(z_{t+K}^{\pi^L_\theta}) \Bigg] \tag{1.21}$$

With this learning setting the proposed system outperforms previous works such as [84, 80] 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.

1.2.3.4 Learning from Observation

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 [35] 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:

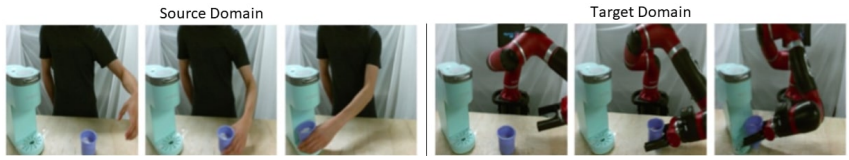


Figure 1.13: Representation of **embodiment mismatch problem**. (Left) The source domain represented by a video of human performing a task. (Right) The target domain, represented by the robot that executes the observed task

1. 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 1.2.2. 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 1.13. This approach has been followed by authors in [32, 36, 88].

Specifically, the authors in [32, 36] used the Cycle-GAN architecture [89] to translate images from the source domain (human images) to the target domain (robot images) in an unsupervised manner. The work in [89] 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.

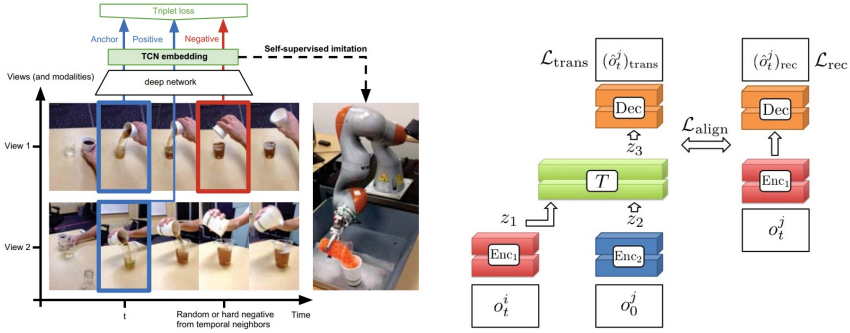
To address this, Cycle-GAN introduces a novel learning procedure involving two translation models: $G : X \to Y$ and $F : Y \to X$. The first model maps inputs from the source domain to the target domain, while the second model maps inputs from the target domain back to the source domain. These two models are trained in an adversarial setting by minimizing the loss function shown in Formula 1.22.

$$\begin{aligned}
\mathcal{L}(G, F, D_X, D_Y) &= \mathcal{L}_{GAN}(G, D_Y, X, Y) + \\
&\quad \mathcal{L}_{GAN}(F, D_X, Y, X) + \lambda \mathcal{L}_{cyc}(G, F) \\
\\
\mathcal{L}_{GAN}(Z, D_K, S, T) &= \mathbb{E}_{t \sim p_{data}(t)} \left[\log(D_K(t))\right] + \\
&\quad \mathbb{E}_{s \sim p_{data}(s)} \left[\log(1 - D_K(Z(s)))\right] \\
\\
\mathcal{L}_{cyc}(G, F) &= \mathbb{E}_{x \sim p_{data}(x)} \left[\|F(G(x)) - x\|_1\right] + \\
&\quad \mathbb{E}_{y \sim p_{data}(y)} \left[\|G(F(y)) - y\|_1\right]
\end{aligned} \tag{1.22}$$

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 1.14a). This issue has been tackled in [90, 83].



(a) *Time-Contrastive network*, proposed in [90].

(b) *Context-Translation network*, proposed in [83]

Figure 1.14: Examples of how the mismatch between demonstrator viewpoint and learner viewpoint can be handled.

In [90], a Convolutional Neural Network was trained using a *Triplet-Loss* [91]. The aim was to train a network to predict an embedding independent of the viewpoint, but containing only task-relevant features. 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 1.14a).

In [83], a different approach was employed. Here, a *context translation problem* was addressed using an Encoder-Decoder architecture (Figure 1.14b). 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.

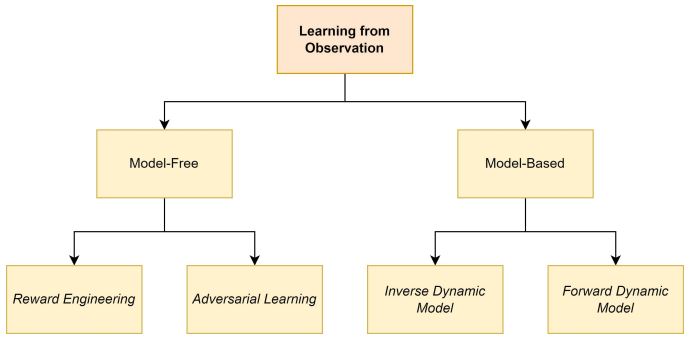


Figure 1.15: Learning from Observation taxonomy

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 1.15).

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 hand-designed reward function to train the policy according to the Reinforcement Learning paradigm [2]. Methods based on this approach include [83, 90, 36, 92].

In [83], the reward function is defined as in Formula 1.23.

$$R(o_t^l) = -\left\| Enc_1(o_t^l) - \frac{1}{n}\sum_{i=1}^{n} F(o_t^i, o_0^l) \right\|_2^2 - w_{rec}\left\| o_t^l - \frac{1}{n}\sum_{i=1}^{n} M(o_t^i, o_0^l) \right\|_2^2 \tag{1.23}$$

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

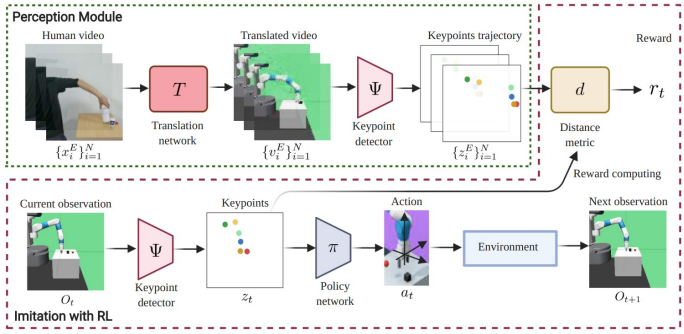


Figure 1.16: Architecture proposed by [36]

penalizes the policy for experiencing observations that differ from the translated observation.

In [90], the reward function is defined according to Formula 1.24.

$$R(\mathbf{v}_t, \mathbf{w}_t) = -\alpha \left\|\mathbf{w}_t - \mathbf{v}_t\right\|_2^2 - \beta\sqrt{\gamma + \left\|\mathbf{w}_t - \mathbf{v}_t\right\|_2^2} \tag{1.24}$$

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 1.14a).

In [36], a **keypoint-representation** (Figure 1.16) 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 1.25.

$$\begin{aligned} R(z_t, z_{t+1}, z^E) = -\lambda_1 \min_p \left\|z_t - z_p^E\right\| - \\ \lambda_2 \min_p \left\|(z_{t+1} - z_t) - (z_{p+1}^E - z_p^E)\right\| \end{aligned} \tag{1.25}$$

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

In [92], the reward function was defined as $R(s_t) = -\frac{1}{k}\left|\left|\phi(s_t) - g\right|\right|_2^2$, where g is the goal embedding, defined as the mean embedding of the last frame of all the demonstration videos in

Algorithm 6 GAIfO algorithm [94]

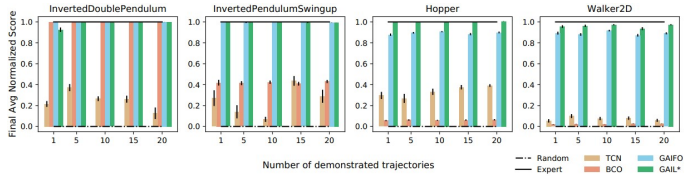
Require: Initial policy π_ϕ^L, Initial Discriminator D_θ
Require: State-only expert demonstration trajectories $\tau^E = \{(s, s')\}$
  while Policy Improves **do**
    Execute π_ϕ^L and collect state transitions $\tau^L = \{(s, s')\}$
    Update D_θ, with $\mathcal{L}_{D_\theta} = -\ (\ \mathbb{E}_{\tau^L}[\log(D_\theta(s,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

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 [90] in terms of both sample efficiency and performance.

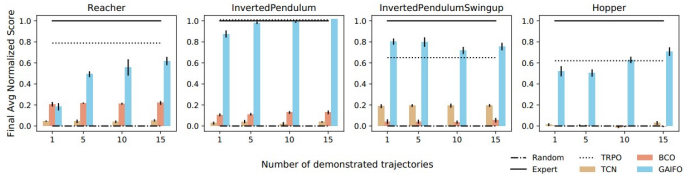
Adversarial Learning methods rely on the Adversarial Learning paradigm and are closely related to the GAIL methods (Section 1.2.3.3). Unlike the methods discussed in the GAIL section, these methods do not assume access to the demonstrator's actions. Preliminary works in this area have been proposed in [93, 94].

The goal of authors in [93] 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 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 [94], who formalized the *GAIfO* algorithm (Algorithm 6), an extension of GAIL [76] to state-only demonstrations. The proposed algorithm was used to train a network to solve tasks in a simulation environment [79], with both low-dimensional state



(a) Experimental results in low-dimensional state space



(b) Experimental results in high-dimensional state space

Figure 1.17: Experimental results reported in [94].

representation and visual-state representation. Results regarding the number of demonstrated trajectories are reported in Figure 1.17.

As noted from the results, GAIfO outperforms previous observation-based methods [90, 95] 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 [77] algorithm. This issue was addressed by DEALIO [96], which replaced the model-free algorithm with PILQR [97], 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 [98, 95, 99, 100].

In [98], 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 [95] proposed a general approach, depicted in Figure 1.18, 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 [99], a similar approach to [95] was used, but the agent's policy was trained using a combination of Behavioral Cloning and the Advantage Actor Critic (A2C) objective function [101] (Formula 1.26).

$$\begin{aligned}\mathcal{L}_\theta^{hyb} = \mathbb{E}_{s,a}\left[A(s)\log(a|s;\theta) + \alpha\ \mathcal{H}(\pi^L(.|s))\right] + \\ \mathbb{E}_{(\hat{s}_t,\hat{s}_{t+1})\sim D}\left[\log(\pi^L(M(\hat{s}_t,\hat{s}_{t+1})|\theta))\right].\end{aligned} \tag{1.26}$$

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

In [100], the work in [102] 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

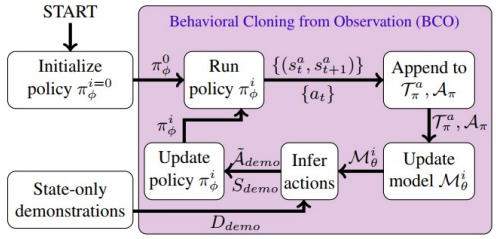


Figure 1.18: Representation of the learning procedure proposed by [95]

the action performed during the policy rollout. The policy was then updated according to the *Demo Augmented Policy Gradient* (DAPG) method [102], adapted for state-only demonstrations, which follows the gradient updates described by Formula 1.27.

$$g_{SOIL} = g + \lambda_0 \, \lambda_1^k \sum_{(s_t, \tilde{a}_t) \in D'} \nabla_\theta \log \pi_\theta^L(\tilde{a}_t, s_t), \tag{1.27}$$

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 [32, 96].

In [32], once the human video demonstration was translated into the corresponding robot video, the policy was learned using the model-based RL algorithm SOLARIS [103]. 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 [104]. 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 [74]. Real-world robotic experiments demonstrated that with just **2 hours** of robot interaction, it was possible to outperform previous works such as [90, 95] and classic BC algorithms in tasks

like "coffee making" (Figure 1.13) and cup-retrieving, where the robot retrieves a cup from a closed drawer.

In [96], the sample inefficiency problem of GAIfO [94] was addressed. The approach combined the adversarial learning setting with state-only demonstrations, which had shown promising results (Figure 1.17), with a more data-efficient RL algorithm like PILQR [97]. PILQR's core is the LQR optimization procedure. Generally, it returns a *linear-gaussian controller* (Formula 1.28) that optimizes a *quadratic-cost function* (Formula 1.29) under the assumption of *linear-gaussian dynamics* (Formula 1.30).

$$\pi(a_t|s_t) = \mathcal{N}(K_t s_t + k_t, S_t) \tag{1.28}$$

$$c(s_t, a_t) = \begin{bmatrix} s_t \\ a_t \end{bmatrix}^T C_t \begin{bmatrix} s_t \\ a_t \end{bmatrix} + \begin{bmatrix} s_t \\ a_t \end{bmatrix}^T c_t + cc_t \tag{1.29}$$

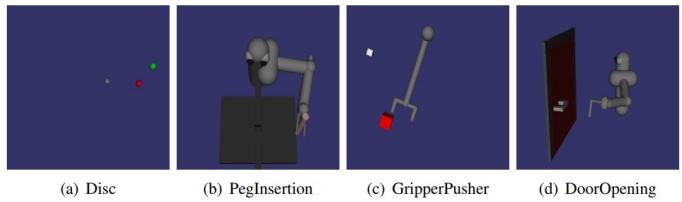
$$s_{t+1} \sim P(s_{t+1}|s_t, a_t) = \mathcal{N}(F_t \begin{bmatrix} s_t \\ a_t \end{bmatrix} + f_t, \Sigma_t) \tag{1.30}$$

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 1.31) as a function of the pair (s_t, a_t).

$$D_\theta(s_t, s_{t+1}) = \frac{1}{2} \begin{bmatrix} s_t \\ s_{t+1} \end{bmatrix}^T C^{ss}(s_t, s_{t+1}) \begin{bmatrix} s_t \\ s_{t+1} \end{bmatrix} + \begin{bmatrix} s_t \\ s_{t+1} \end{bmatrix}^T c^{ss}(s_t, s_{t+1}) \tag{1.31}$$

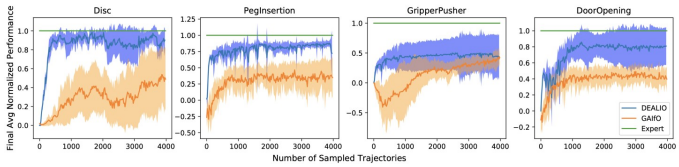
Experiments performed in simulation with low-dimensional state spaces showed promising results (Figure 1.19b) in terms of sample efficiency compared to the GAIfO baseline. However, improvements are needed to: **(1)** Reduce variance to make the learning process more reliable, **(2)** Increase overall 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



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

(a) Control Tasks solved in [96]



(b) Performance of DEALIO [96] compared against GAIfO [94], with respect to the number of trajectories sampled during the learning process.

Figure 1.19: DEAILO: (1.19a) Control Tasks, (1.19b) Performance Level

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.

1.2.4 Target object misidentification problem

Section 1.2.3 provides a detailed overview of classic methodologies for solving the LfD problem. However, all the discussed methods are evaluated in single-task scenarios, meaning that the learned policies can only replicate the specific task they are trained on, with limited generalization to new configurations.

In collaborative environments where a robot may need to perform various tasks as requested by a human operator, these systems may fail due to their inability to generalize across different tasks.

To address this limitation, the Multi-Task Imitation Learning problem has been introduced in literature. As mentioned in Section 1.1, the goal of these methods is to train a single conditioned control policy that, given the current state of the robot and the task to be performed, can generate the correct action. These methods will be described in detail Chapter 3, where both the conditioning modalities language-based and video-based will be reviewed.

However, this thesis specifically focuses on video-conditioned methods. Experimental results in Section 3.4 demonstrate that state-of-the-art methods in Video-Conditioned Multi-Task Imitation Learning, such as TOSIL [6] and MOSAIC [8], suffer from the problem of **target misidentification** in scenarios with multiple similar objects, where the semantic role of the objects (e.g., distractor or object of interest) changes based on the prompted task. This means that, while these methods can generate reasonable trajectories and complete tasks, they often manipulate the wrong object. Specifically, it was observed that the backbone failed to produce an embedding that encodes the position of the target object at the beginning of the task. In fact, by performing ground-truth actions for just two steps at the very start, the model was able to complete the task correctly. This highlights a lack in learning features related to the cognitive aspect of the task, specifically, the position of the object to be manipulated.

To overcome this issue, and building upon the observations made in Section 1.1 regarding the cognitive and control tasks involved in multi-task manipulation, this thesis proposes a modular approach. Rather than using an end-to-end architecture that maps high-dimensional inputs directly to the action space, the problem is divided into two sub-problems: the cognitive and control problems, each addressed by specialized modules. The cognitive module, described in Chapter 2, introduces object priors related to the position of the object to be manipulated. Specifically, a Conditioned Object Detection problem

is formulated and solved using a Conditioned Convolutional Neural Network, which predicts a category-agnostic bounding boxes for the objects of interest for the specific prompted variation.

The proposed COD module is then integrated into a state-of-the-art framework to validate the hypothesis that adding inferred object priors can resolve the target misidentification issue. This leads to a novel modular architecture, detailed in Chapter 3, capable of performing both *multi-variation* and *multi-task* manipulation robustly. Additionally, it generates human-readable information, enhancing the interpretability of the learned policies by providing insights into the robot's intended movements.

Chapter 2

Proposed conditioned object detector

As discussed in Section 1.1, this thesis proposes a modular approach to solve the Visual-Conditioned Multi-Task Imitation Learning Problem. This chapter focuses on the module responsible for addressing an instance of the "Cognitive Task" relevant to the context under consideration. Specifically, the approach aims to leverage an **object-prior** to enable the system to ignore distractor objects in the scene and focus on the target region of interest. To achieve this goal, a preliminary step is to develop a system capable of computing this object-prior. Therefore, this chapter will present and discuss the design of the Conditioned Object Detector (COD), which is specifically designed to solve a **novel variation** of the well-known Object Detection problem. Unlike traditional object detection tasks, where a deep architecture produces bounding boxes for all objects in the scene, this variation requires generating bounding boxes only for the regions of interest based on the demonstrated task.

Section 2.1 will review relevant methods related to this task. Section 2.2 outlines the detection problem being addressed. Section 2.3 describes the proposed architecture for solving this problem. Finally, Section 2.4 discusses the experimental setup and presents the results obtained from testing the proposed architecture.

2.1 Related Works

This section will review the preliminary research conducted in the context of Object-Oriented Imitation Learning (Section 2.1.1) to understand how previous studies have leveraged object priors to solve the Learning from Demonstration (LfD) problem. Additionally, it will examine research in the context of Visual-Question Answering (Section 2.1.2) to highlight the specific challenges addressed by the COD module, while also presenting the methods that have primarily inspired the proposed solution.

2.1.1 Object-Oriented Imitation Learning

All the methods discussed in Section 1.2.3 share a common characteristic: they are end-to-end systems that take high-level inputs, such as images, and directly generate the corresponding actions as output. While this approach can be sufficient in scenarios where the scene is simple, meaning there are no distracting objects, or if there are distracting objects, they can be easily identified by the system because they are consistently not involved in the manipulation, this end-to-end approach may struggle in more complex environments. Specifically, it can encounter difficulties when the robot workspace contains objects that are similar to each other, especially if these objects are involved in manipulation for some task variations.

Based on this consideration, this paragraph will describe methods that leverage object priors. This means that the control policy is informed not only by the embedding of the agent scene, which is obtained from a deep architecture, but also by object-level information, such as the bounding boxes of objects in the scene, obtained from an object detector.

The concept of leveraging object priors has been explored in both earlier works [105, 106] and more recent approaches [107, 108, 109, 110].

One of the preliminary works on leveraging object priors was presented by the authors of [105]. This work primarily focuses on the challenge of generalization in Learning from Demonstration (LfD) systems that use an end-to-end approach.

In such systems, a task-specific model is trained to predict actions based on raw visual observations. The authors found that while it is possible to achieve good *instance-level generalization*, meaning the model can solve tasks with varying initial configurations using a limited number of samples, achieving *category-level generalization* is more challenging. Category-level generalization refers to the model ability to solve tasks involving different objects. To achieve this, the dataset must include a large number of trajectories involving a wide variety of objects. For instance, if the task is to pour the contents of a bottle into a cup, the dataset should contain trajectories with different types of cups and bottles. However, constructing such a dataset is time-consuming and costly. Moreover, the potential of well-known large datasets in classical computer vision tasks, such as object detection, is not fully utilized.

To address this issue, the authors proposed a paradigm shift by introducing a robotic vision framework that operates on sets of objects rather than raw pixel data. This framework leverages prior datasets to learn a generic object concept model, thereby enhancing category-level generalization without requiring an extensive and diverse dataset. The framework is illustrated in Figure 2.1 and is composed of several stages:

- **Meta-Attention**, which is basically a Region Proposal Network (RPN) [111], trained on the well known MSCOCO [112] dataset. The RPN generates objects proposals, i.e., region of the image that possibly contain an object.

- **Task-Specific Attention**, which aims to learn what are the object of interest with respect to the task in hand. This module is parametreized as a vector w such that the attention paid to o^i is proportional to $e^{w^T f(o^i)}$.

- **Soft Attention**, this module gives a probabilistic meaning to the atttention map obtained from the Task-Specific Attention. Specifically, a Boltzmann distribution is used to map the attention weights to a probability for each object proposal, i.e., $p\left(o^i \mid w_j\right) = \frac{e^{w_j^\top \frac{f\left(o^i\right)}{\|f(o^i)\|_2}}}{\sum_{i=0}^{N} e^{w_j^\top \frac{f(o^i)}{\|f(o^i)\|_2}}}$.

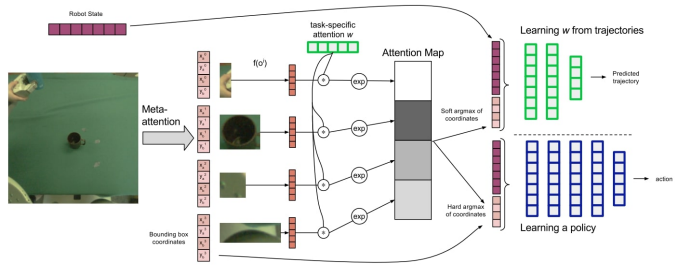


Figure 2.1: Robotic vision framework proposed in [105]. The framework is divided into different stages: **Meta-Attention**: Generates object proposals from an input image, trained on an object detection dataset, and shared across tasks; **Task-Specific Attention**: Focuses on relevant objects for a task using the meta-attention's semantic features; **Soft Attention**: Distributes attention as probabilities over object proposals using a Boltzmann distribution; **Movement Prediction Network**: Combines attended object information with the robot's state to predict the next action.

- **Movement Prediction Network**, this module predicts the next robot action, given the attended object information from the soft attention, and the robot state represented by the joint and end-effector state.

This preliminary work focused mainly on two tasks: • *Pouring Task*: The robot is required to pour contents from a bottle into a mug. The challenge is to locate the mug from an image without being explicitly provided its location, especially when different mugs are used during training and testing. • *Sweeping Task*: The robot must sweep an object (e.g., a plastic orange) into a dustpan, with both objects starting in different positions. This task requires the robot to adapt its approach based on the relative positions of the objects. During testing, the authors focused on *Category Generalization* and the ability to *Ignore Distractor Objects*. For the former, the system was trained with only one type of mug and evaluated with other mugs (Figure 2.2). The results showed that the system successfully poured the contents into the correct mug, thanks to the learned task-specific attention weight that highlighted the mug

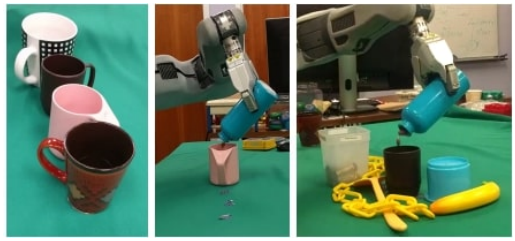


Figure 2.2: Pouring task setting proposed in [105]. (Left) Mugs used for evaluation. Note that only the brown mug was seen during training. Center: Successful pouring into the pink mug. (Right) Pouring into the brown mug in a cluttered environment that was not seen during training.

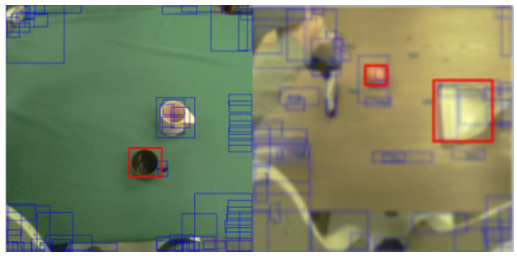


Figure 2.3: The region proposals (meta-attention) are drawn in blue and the task-specific attended regions are drawn in red. For the Pouring task with distractor mug (pink) and target mug (brown), the attention locks on to the brown mug as its position defines the trajectory. For the sweeping task, two attention vectors are used, one attends to the orange and one attends to the dustpan.

features. For the latter, the authors designed a test where two mugs were present in the scene (Figure 2.3). Since the model did not receive any conditioning signal indicating which mug to use, the authors fine-tuned the attention weight on trajectories where only the brown mug was used, demonstrating that this mechanism could focus on more specific features, such as the mug color.

In summary, this preliminary work demonstrated that leveraging object priors can facilitate category-level generalization by utilizing large, well-known datasets for the object-detection problem. However, the experimental setup was relatively sim-

ple, even in scenarios with distractor objects. The proposed system could handle distractor objects only after specific fine-tuning and was not able to dynamically discriminate between objects of interest and distractors based on task variations.

A recent work that follows a similar approach is proposed in [108]. In this work, the authors introduced VIOLA (Visuomotor Imitation via Object-centric Learning) (Figure 2.4), an architecture inspired by the ideas presented in [105]. VIOLA uses an RPN and a ResNet18 [59] to generate object proposals and produce a spatial feature map, respectively. It then constructs a *per-step feature* vector, composed of three key elements: a **global context feature** that encodes the current task stage, an **eye-in-hand visual feature** to mitigate occlusion, and a **proprioceptive feature** that captures the robot state. These per-step features are concatenated to form a **history of observations**, which is designed to capture temporal dependencies and dynamic changes in object states. This tensor is then fed into a Transformer [113], which leverages its intrinsic attention mechanism to automatically focus on the object of interest.

This method was first evaluated in a simulation environment across three tasks, as depicted in Figure 2.5. Various testing scenarios were considered, including different object placements, the introduction of distractor objects, and changes in camera position. Generally speaking, VIOLA outperformed all baselines across these testing conditions, further demonstrating the utility of object priors in enhancing the robustness of such methods. However, similar to [105], the testing scenarios were relatively simple, with clear distinctions between distractors and objects of interest. The distractors were objects never seen during the demonstration and were not involved in manipulation, making them relatively easy for the model to discriminate.

In the works discussed so far, the approach has primarily focused on leveraging object priors to directly predict the actions that the robot must perform. However, a different approach was proposed in [107], where the authors introduced an alternative interpretation of object-centric concept. Instead of focusing on the robot perspective, they shifted the emphasis

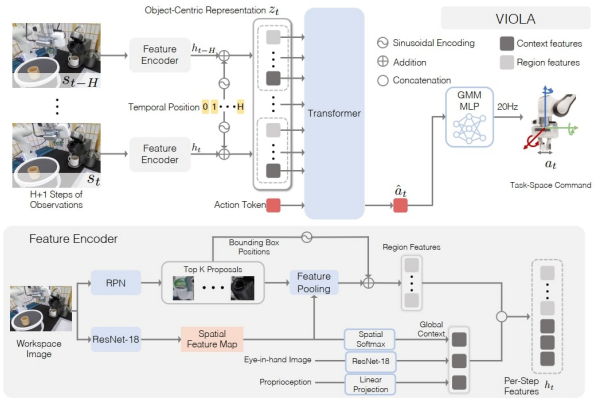


Figure 2.4: The VIOLA architecture proposed in [108]. (Top) The overall control architecture is based on a Transformer module that processes a stack of *per-step features* h_t, obtained from the Feature Encoder, to generate a final action embedding, which is then input into a GMM policy. (Bottom) The Feature Encoder builds both local and global features. Local features correspond to regions of interest extracted by the RPN. Global features are obtained by processing the workspace image, the image from the camera on the gripper, and proprioceptive information.

to the object perspective, proposing PLATO (Predicting Latent Affordances Through Object-Centric Play). PLATO is a learning framework that learns a **latent affordance space**, which describes how an object can be used (e.g., a block being grasped, a door knob being turned, or a drawer being opened).

The authors argue that learning these affordances (i.e., what happens to the object) rather than plans (i.e., what happens to the robot) from play leads to a simpler and more robust task representation that can operate over varying time horizons. This, in turn, results in more effective policies at test time. This paradigm shift allows the policy to reason about the environment more effectively: given access to an affordance (e.g., the door knob being turned) and a goal (e.g., an opened door), the policy can work backwards to infer the behavior needed to exploit that affordance (e.g., reaching the knob and rotating the gripper to turn it).

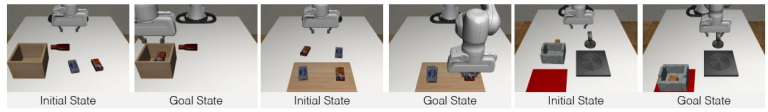


Figure 2.5: Simulation tasks on which the VIOLA [108] method was tested. (Left) Sorting task. (Center) Stacking task. (Right) BUDS-Kitchen task

To reach this objective authors started from the observation that a single-object manipulation is composed of the following three phases:

1. **Pre-interaction**, when the robot prepares to interact with an object (e.g., reaching for a block).

2. **Interaction**, when the robot and the object engage in joint actions (e.g., pushing or pulling the block).

3. **Post-interaction**, when the robot separates from the object, and the object may come to rest (e.g., the block stops moving).

Given these three phases, the algorithm learns a **latent affordance distribution**. Specifically, the architecture comprises three learnable modules: E, E', and π. E models the posterior distribution, mapping the full interaction trajectory τ^i to the corresponding latent affordance distribution, from which the affordance embedding z is sampled. E' is the prior module used during rollout. It takes the current state and the goal state as input and generates the affordance embedding z'. This module is trained to match the posterior distribution modeled by E. π represents the current policy, which generates the action a^i given the current state s^i, the desired goal o_g, and the latent embedding z.

These three modules are trained end-to-end by minimizing the loss function in Formula 2.1, which includes three terms. The first two terms correspond to the policy π, ensuring it matches the ground-truth trajectories in the interaction and pre-interaction phases. The last term is the KL-divergence, used to train the posterior and prior modules E and E'.

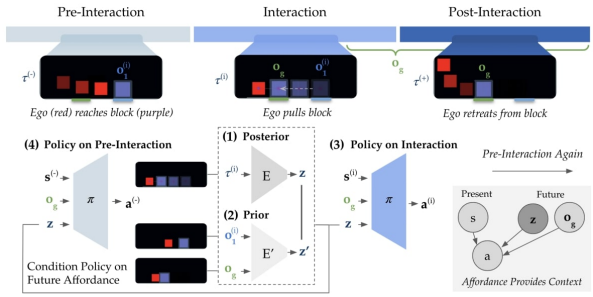


Figure 2.6: PLATO architecture proposed in [107]. The architecture is composed of different stages. (1) The posterior encoder E encodes the interaction sequence $\tau^{(i)}$ into the affordance z. (2) The prior encoder E' encodes the object initial state $o_1^{(i)}$ and goal state o_g to predict z, with o_g sampled after the interaction. (3) The policy is trained to output actions during the interaction period conditioned on the affordance. Simultaneously, (4) it is trained to output actions during the pre-interaction period conditioned on the "future" affordance.

$$\begin{aligned}\mathcal{L}_{PLATO} = &- \log\left(\pi\left(a_{1:H}^{(i)} \mid s_{1:H}^{(i)}, o_g, z\right)\right) - \\ &\alpha \log\left(\pi\left(a_{1:H}^{(-)} \mid s_{1:H}^{(-)}, o_g, z\right)\right) + \\ &\beta\, \mathrm{KL}\left(p(z) \| p\left(z'\right)\right)\end{aligned} \tag{2.1}$$

Finally, this method was tested in a variety of scenarios, including both single-object and multiple-object manipulation with different manipulation primitives (Figure 2.7). However, in the multi-object scenarios, the system was only tested on single-object manipulation primitives.

This work is particularly noteworthy as it demonstrates that a policy can be learned by solving an inverse problem, starting from object affordances and deriving the corresponding robot trajectories. It also shows that the policy can be conditioned based on the desired goal state. However, certain aspects were not addressed in this work, such as the potential presence of distractor objects and tasks requiring the manipulation of multiple objects. Additionally, the affordances were learned in the object space (i.e., with known object poses)

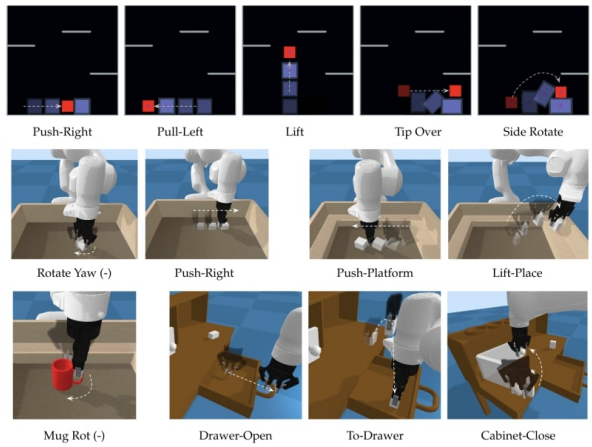
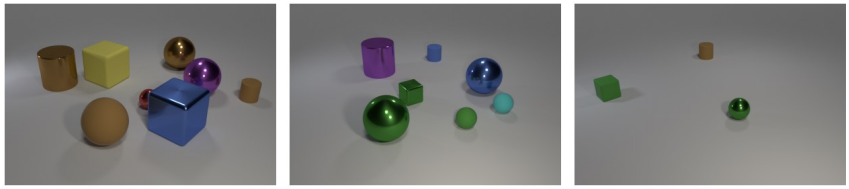


Figure 2.7: Testing scenarios and primitives proposed in [107]. (Top) **Block2D** Environment primitive examples. (Center) **Block3D** and **Block3DPlatform** primitive examples. (Bottom) The left image shows an example primitive in **Mug3D-Platforms**. The right three images show sample tasks from **Playroom3D**.

rather than in the high-level image space.

The works discussed in this paragraph highlight the significant research efforts aimed at modeling the manipulation problem from an object-centric perspective. These efforts either focus on the affordances of the object (i.e., the possible movements the object allows) or introduce object priors defined by regions of interest that may contain the object to be manipulated, with results demonstrated in both simulated and real-world environments. However, the methods presented so far primarily address single-task scenarios, where distractor objects can be easily identified as they remain constant across demonstrations. In contrast, this thesis proposes a solution for a more challenging scenario in which the robot operates in a multi-variation environment. This environment includes multiple similar objects, which may serve as either targets or distractors depending on the specific task variation.



Q: How many small spheres are the same color as the big rubber cube?
A: 0
Q-type: count
Size: 9

Q: Is the tiny ball made of the same material as the large purple ball?
A: yes
Q-type: equal_material
Size: 12

Q: How many small green things are behind the green rubber sphere in front of the blue thing that is in front of the large purple metal cylinder?
A: 1
Q-type: count
Size: 18

Q: The cylinder that is the same size as the blue metallic sphere is what color?
A: purple
Q-type: query_color
Size: 9

Q: What is the color of the matte thing that is left of the thing behind the tiny green thing behind the tiny shiny sphere?
A: green
Q-type: query_color
Size: 15

Q: There is a thing in front of the tiny block; is its color the same as the matte object in front of the tiny brown rubber thing?
A: yes
Q-type: equal_color
Size: 17

Figure 2.8: Example of Vision-Question Answering problem get from the CLEVR [114] dataset. It is possible to observe how for a given image multiple different questions can be done. As well as, questions covers different reasoning skills such as attribute identification, counting, comparison, multiple attention, and logical operations

2.1.2 Visual-Question Answering

The *Visual-Question Answering* (VQA) problem refers to the ability of a system of "reasoning" over an image, based on a given query expressed in terms of text. This means that, given in input an arbitrary image and an arbitraty textual prompt the system must be able to generate an answer to the given prompt (Figure 2.8).

The key aspect related to this problem is related to the fact that for a given image different questions can be made, this means that there is not an a priori mapping between the input image and the corresponding answer, but the system must be able to dynamically adapt its **attention** on specific part of the image based on the given query, meaning that the system must be able to correlate in an effective way the encoding of the image and the encoding of the prompt, in order to create a **conditioned** embedding that can be effectively used to generate a valid answer.

Based on these preliminary considerations, various solutions have been proposed using both Convolutional Neural Network architectures [60] and novel Transformer architectures [115, 116, 117].

A significant breakthrough in this field was introduced in [60], where the FiLM (Feature-wise Linear Modulation) conditioning layer was proposed. The core idea behind FiLM is to adaptively modify the output of a neural network by applying an affine transformation to its intermediate features based on some input. Specifically, the FiLM layer learns two functions, f and h, which produce the coefficients $\gamma_{i,c}$ and $\beta_{i,c}$, respectively, based on an input x_i, as defined in Equation 2.2.

$$\gamma_{i,c} = f_c(x_i) \quad \beta_{i,c} = h_c(x_i) \tag{2.2}$$

These coefficients are then used to modulate the c^{th} activation map $\mathbf{F}_{i,c}$ through a *feature-wise affine transformation*, as described in Equation 2.3.

$$FiLM(\mathbf{F}_c|\gamma_c, \beta_c) = \gamma_c \mathbf{F}_c + \beta_c \tag{2.3}$$

An example of integrating the FiLM layer into a deep architecture is shown in Figure 2.9, illustrating the implementation proposed in [60]. In this configuration, the prompt is encoded using a Gated Recurrent Unit (GRU), and the resulting embedding is fed into a Multi-Layer Perceptron (MLP), which outputs the coefficients $\gamma_{i,c}$ and $\beta_{i,c}$. These coefficients are subsequently used to modulate the activations of N Residual Blocks (ResBlocks).

The FiLM layer has also been effectively applied in the context of Language-Conditioned Policy Learning (Section 3.1.1) [11, 7]. Starting from a textual prompt describing a desired task, the FiLM layer modulates the activation maps of the convolutional blocks in the network that encodes the robot's observations. This allows the network to focus on specific parts of the image relevant to the task (Figure 2.10).

Since the introduction and expansion of Transformer models [113], there has been a major shift towards solving the Visual Question Answering (VQA) problem by utilizing these architectures. Transformers excel at capturing intra-modal relationships through dependency modeling and promoting alignment and interaction between multiple modalities. This has led to the development of Vision-Language Transformers capable of directly processing multi-modal inputs, such as images

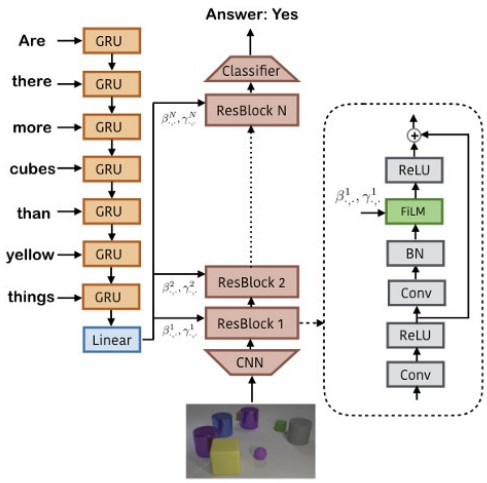


Figure 2.9: Proposed integration of the FiLM conditioning layer. Here, the Linear Modulation is applied to modify the activation maps of the ResNet blocks, while a linear layer generates the modulation coefficients β and γ based on the embedding derived from the textual prompt. This enables the model to conditionally adjust the activation maps according to the context provided by the input query

and text. An example is the LLaVA model proposed in [117], which represents the current state-of-the-art in VQA. However, despite its capability to handle complex queries, LLaVA relies on a Large Language Model (LLM) with a minimum size of 7 billion parameters, requiring a high-performance server. This makes real-time deployment in real-world systems, such as robotic control, challenging or even impractical.

2.2 Problem formulation

As outlined in previous chapters, the primary objective of this thesis is to validate the possibility of using object priors to enhance the system ability to ignore distractor objects during task execution, while simultaneously providing human-readable information about the robot behavior.

In Section 2.1.1, various methods that utilize object pri-

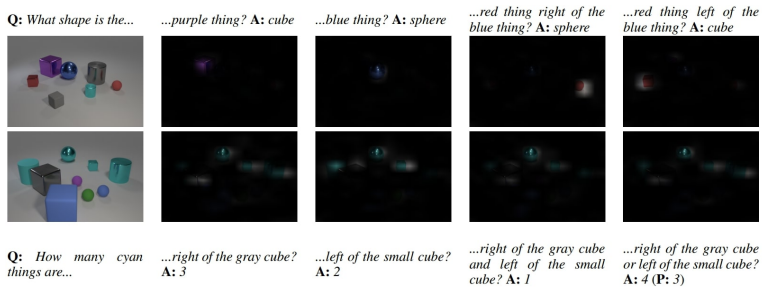


Figure 2.10: Visualizations of the distribution of locations used by the model for its globally max-pooled features, from which its final MLP makes predictions. FiLM correctly localizes the object referenced by the answer (top) or all objects referenced by the question (bottom). However, the localization is less accurate when the model provides an incorrect answer (rightmost bottom).

ors in the context of Imitation Learning have been discussed. However, all the methods described in that section rely on pre-trained object detectors to identify regions of interest, i.e., areas that potentially contain objects, regardless of the object class (e.g., box, nut, bin, peg, etc.) or its semantic state (i.e., whether the object is relevant to the task or merely a distractor). Furthermore, none of these methods are designed for multi-variation scenarios, where the semantic state of an object (whether it is relevant or a distractor) is dynamically determined by the requirements of the specific task.

In the context of interest, the object detection problem can be formulated as follows: given a pair consisting of an agent observation and a command, (o_t^a, c_{m_i}), a parameterized function f_θ, must generate a set of bounding boxes (bb), $\{bb_t^j | j \in C\}$ that identifies **specific** regions of interest, i.e., $\{bb_t^j | j \in C\} = f_\theta(o_t^a, c_{m_i})$. Here, C represents a set of classes assigned to the bounding boxes, which differs from the traditional object detection classes. Instead of predicting the object category (e.g., box, nut, bin, etc.), the focus is on predicting the **semantic attribute** associated with each object. In the most general case, these attributes are:

- *Target*, it indicates that the bounding box refers to the

object to be manipulated in the current task variation c_{m_i}.

- *No-Target*, it refers to any other object in the scene that is not being manipulated.

- *Target-Place*, it represents the final region of interest for the manipulated object, which varies depending on the task. For example, in a pick-and-place task, it refers to the bin where the object is placed; in a nut-assembly task, it refers to the peg where the object is inserted; and in a button-press task, it refers to the region where the button has been pressed.

- *No-Target-Place*, it refers to any other region that is not relevant to the current task variation, such as other bins in a pick-and-place task or other pegs in a nut-assembly task.

This example highlights how the semantic attribute assigned to an object changes depending on the requested task c_m. In the first case, the green box and the first bin are marked as targets (green boxes), while other objects are labeled as distractors (red boxes). In the second case, even though the configuration remains the same, the roles of the objects changes. This is because, in the second task, the demonstrator manipulates the red box and places it into a different location.

Since the goal is to explicitly manage the changing semantic meaning of objects, traditional object detectors, which predict bounding boxes and object categories for all objects in the scene, are not suitable. Instead, a novel, specialized architecture is required to implement the function f_θ. The details of this implementation are provided in Section 2.3.

2.3 Proposed Architecture

This section describes the implementation of the COD module, which implements the parameterized function f_θ.
As described in the previous paragraph this function must generate a set of object category agnostic bounding-boxes $\left\{bb_t^j | j \in C\right\}$,

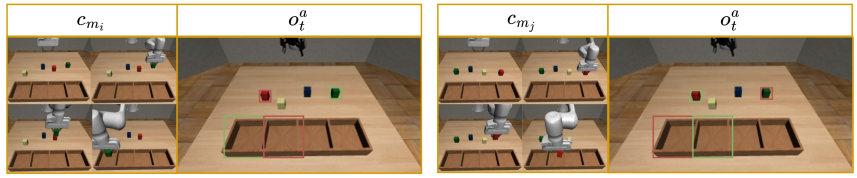


Figure 2.11: Example of bounding-boxes assignmennt, where green boxes refer to target object and placing location while red boxed refers to no-target and no-target-place. (Left) The demonstrator manipulates the green box, placing it into the first bin. (Reight) The demonstrator manipulates the red box, placing it into the second bin. Note, how for a given agent environment state, the semantic attribute between objects changes.

where $bb_t^j \in \mathcal{R}^4$ is a vector of 4 elements containing the coordinates of the bounding-box defined as the upper-left (ul) and bottom-right (br) corners $\left[x^{ul}, y^{ul}, x^{br}, y^{br}\right]$. To each predicted bounding-box is assigned a class that refers to the semantic attribute associated with the object for a given command. The bounding-boxes are generated by taking in input:

- *Agent Observation*, $o_t^a \in \mathcal{R}^{H \times W \times 3}$, which is a RGB image which represents both the state of the agent as well as the state of the environment.

- *Demonstrator Command*, $c_{m_i} \in \mathcal{R}^{T \times H \times W \times 3}$, which is a sequence of T frames sampled from a video of a demonstrator performing the requested task.

In the field of Computer Vision, Object Detection is a well established and extensively researched problem, with modern techniques achieving high performance in both detection and classification tasks. However, the methods used in previous work [110, 108] are not applicable here. The objective in this case is not to detect all objects in the scene, but rather to identify "target locations", which can vary depending on the specific task at hand (Figure 2.11). Additionally, the method must be **category-agnostic**, meaning it does not need to answer the question, "What category does the object belong to?" Instead, the primary goal is to determine, "Where is the target

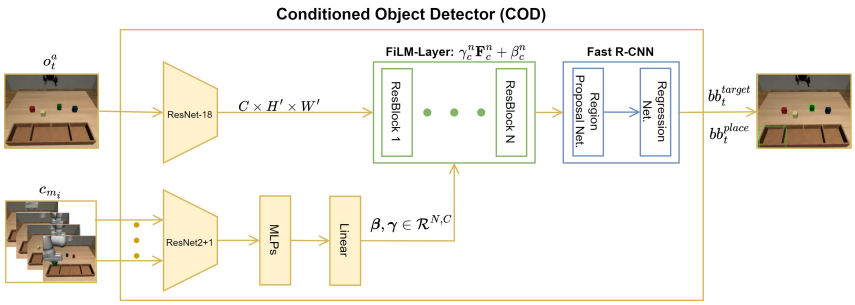


Figure 2.12: The proposed *Conditioned Object Detector* architecture takes as input the pair (o_t^a, c_{m_i}), where o_t^a represents the agent observation and c_{m_i} represents the demonstration frames. The agent observation is encoded using ResNet-18, while the demonstration frames are encoded through ResNet2+1. The FiLM conditioning layer is employed to inject information from the command c_{m_i} into the feature maps extracted from the observation. Finally, Fast R-CNN generates the bounding boxes based on the conditioned input.

object located?" This allows the approach to be easily adapted to scenarios involving multiple object categories.

To solve the given problem, inspiration was drawn from another well-known task in computer vision, namely, "Visual Question Answering" [60]. In this task, the system generates an answer to a given query (e.g., a textual question) based on an input image. What distinguishes this approach is its ability to **selectively focus on specific segments of the image**, guided by the content of the question, as illustrated in Figure 2.10.

The proposed approach is rooted in the concept that, similar to the architecture presented in [60], which directs attention to specific regions of an image in response to input queries, our model aims to achieve a similar capability. Specifically, the model is designed to focus on certain regions of the image based on the task command c_{m_i}. The proposed architecture is shown in Figure 2.12.

The architecture consists of the following modules:

- *ResNet-18* [59], used as a feature extractor for the agent observation o_t^a, represented as an RGB image.

- *ResNet2+1* [118], employed as a feature extractor for the command description c_{m_i}. This description consists of 4 frames sampled from the demonstration video. The architecture is designed to extract both spatial and temporal features, which are then flattened and passed through a Multilayer Perceptron (MLP) network.

- The FiLM-Layer, as introduced in [60], is responsible for integrating the features from both the agent observation and the command description. It modulates the c^{th} activation through a feature-wise affine transformation, expressed as in Formula 2.3. Here, γ_c and β_c are parameters generated by the linear module based on the input.

- *Fast R-CNN* [111], a two-stage anchor-based object detector, is used to predict the bounding box position based on the feature maps generated by the preceding module.

The entire architecture is trained using the classic object-detection loss function $\mathcal{L} = w_1\mathcal{L}_{reg} + w_2\mathcal{L}_{cls} + w_3\mathcal{L}_{class}$, where:

- $\mathcal{L}_{reg}$ is the *L1-loss* function that compares the predicted bounding box offset with the ground truth offset.

- $\mathcal{L}_{cls}$ is a *binary cross-entropy loss* that allows the model to learn the difference between foreground and background bounding boxes.

- $\mathcal{L}_{class}$ is a cross-entropy loss designed for object classification.

Once the architecture has been fully described, we can introduce the experiments conducted to evaluate the proposed model. A detailed explanation of these experiments, along with the results, will be provided in Section 2.4.

2.4 Experiments

In this section, the performed experiments are going to be described. Specifically, in Section 2.4.1 the dataset used for training procedure will be described. Section 2.4.2 will report the obtained results.

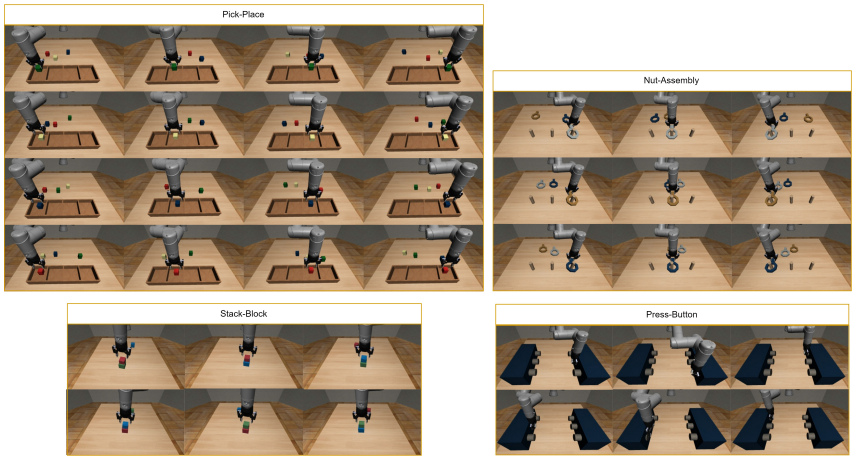


Figure 2.13: Examples of tasks and variations taken in consideration for the methods validation. The images report the final system state. For the pick-place, nut-assembly and stack-block tasks the variation is defined with respect to the target object and the placing location, while for the press-button the variation is defined according to the button to press.

2.4.1 Dataset

To validate the proposed architecture, a simulated dataset was generated, focusing on four primary tasks: *Pick-Place*, *Nut-Assembly*, *Stack-Block*, and *Press-Button*. Each task consists of multiple variations. Specifically, the Pick-Place task has 16 variations, Nut-Assembly has 9 variations, Stack-Block has 6 variations, and Press-Button has 6 variations. Each task and its variations are graphically described in Figure 2.13.

For each task, 100 trajectories were collected for both the agent (UR5e robot) and the demonstrator (Franka-Emika Panda robot) using hand-written policies that take as input ground-truth information about the object position. Across these trajectories, the position of the objects of interest varies. The workspace for each task is divided as follows:

- **Pick-Place**: The bins are fixed in position, while the boxes can change according to the following rule. The workspace is divided into 4 slots, parrallel to the bins, each with a width of 6 cm and a height of 15 cm. A slot

is randomly selected, and the box is randomly spawned within the chosen slot, ensuring that each slot is filled with only one object.

- **Nut-Assembly**: The pegs are fixed in position, while the nuts vary in placement. A spawn region of 75×10 cm is defined, and the nuts are randomly spawned in this region, ensuring they do not collide with one another.

- **Stack-Block**: The placing box is spawned in a region of 16×5 cm, while the target box is spawned in a region of 24×5 cm, ensuring no collisions between the boxes.

- **Press-Button**: The two sets of buttons are placed within a region of 4×4 cm.

The bounding boxes are generated automatically using a procedure that assumes prior knowledge of the object's pose and dimensions, as well as the camera's pose and intrinsic parameters. With this information, a 3D bounding box can be constructed around the object in continuous world space. The corners of this bounding box are then projected into the 2D discrete camera space via a sequence of transformations, as described in Equation 2.4.

$$\begin{aligned} \tilde{p}_{object}^{camera} &= T_{world}^{camera} \times \tilde{p}_{object}^{world} \\ px_x &= \frac{p_x}{p_z} \cdot f + \frac{w}{2} \\ px_y &= \frac{p_y}{p_z} \cdot f + \frac{h}{2} \end{aligned} \tag{2.4}$$

In this equation, $T_{world}^{camera} \in \mathcal{R}^{4\times4}$ represents the homogeneous transformation matrix that defines the camera orientation and position relative to the world frame. The term $\tilde{p}_{target}^{src} = [p_x, p_y, p_z, 1]$ denotes the homogeneous position vector of the *target* frame relative to a given *src* frame. For instance, $\tilde{p}_{object}^{world}$ describes the position of the object relative to the world frame.

The parameters w and h correspond to the image resolution width and height, respectively. The camera's focal length f is given by $f = \frac{0.5 \times h}{\tan(fov_y)}$, where fov_y is the vertical field of view.

Finally, px_x and px_y represent the pixel coordinates obtained in the x and y axes, respectively.

2.4.2 Results

This section presents the obtained results, divided into two main blocks. The first block (Section 2.4.2.1) discusses the results of the method trained to detect only the target object. The second block (Section 2.4.2.1) 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.

For all the test configurations described below, the testing procedure was consistent. Specifically, for each task and its variations, 10 rollouts were performed. During each rollout, the robot was controlled by a hand-written policy, and the predicted bounding box was compared with the ground truth, obtained using the same procedure as described earlier.

The metrics used to evaluate the methods were *Precision@0.5* (Equation 2.5) and *Recall@0.5* (Equation 2.6).

$$Pre = \frac{TP}{TP + FP} \tag{2.5}$$

$$Rec = \frac{TP}{TP + FN} \tag{2.6}$$

A *True Positive* (TP) sample is defined as a predicted bounding box for the target class with an Intersection over Union (IoU) greater than or equal to 0.5 when compared with the ground-truth bounding box. A *False Positive* (FP) sample is defined as a predicted bounding box for the target class with an IoU less than 0.5. A *False Negative* (FN) sample is defined as a target object for which no bounding box was predicted.

The metrics were computed considering only the target bounding boxes, i.e., the bounding boxes predicted for the target object or the target location. Among all the candidates generated by the network, only the one with the highest predicted class score was considered.

Table 2.1: Results of the CTOD module obtained in the single-task multi-variation scenario. Performances are reported in terms of *Precision* (Prec), *Recall* (Rec) with an IoU threshold of 0.5 and the Average IoU (IoU_{avg})

Task	Precision@0.5	Recall@0.5	$\mathbf{IoU_{avg}}$
Pick-Place	0.770	1.00	0.628
Nut-Assembly	0.985	1.00	0.789
Stack-Block	0.995	1.00	0.848
Press-Button	0.997	1.00	0.899

2.4.2.1 Target object detector

This first set of validation tests is related to the training of the COD module in a setting where it must predict the location of the target-object. This means that, with respect to the semantic attributes defined in Section 2.2, the set is restricted to just two classes "target" and "no-target". In this case, the method will be named *Conditioned Target Object Detector* (CTOD).

Single-task multi-variation scenario
In the single-task multi-variation scenario, a specific model was trained for each task described in Figure 2.13. This approach allows for evaluating the model ability to handle tasks of increasing complexity. Initially, the model is evaluated in a simpler scenario, where it predicts the target position of a specific category of objects (e.g., only boxes in pick-and-place, or only nuts in nut-assembly), thereby limiting variability in object categories.

Table 2.1 presents the overall performance of the CTOD module in terms of Precision and Recall, providing a comprehensive evaluation of the system ability to accurately identify target objects (Precision) and consistently detect them across multiple rollouts (Recall). The results demonstrate that the module successfully identifies target objects with precise bounding boxes and maintains a high level of consistency across rollouts, achieving excellent precision and recall metrics.

In particular, the Recall remains consistently at **1.00**, indicating the absence of false negatives, meaning the system

Table 2.2: Results of the CTOD module obtained in the multi-task multi-variation scenario. Performances are reported in terms of *Precision* (Prec), *Recall* (Rec) with an IoU threshold of 0.5 and the Average IoU (IoU_{avg})

Task	**Precision@0.5**	**Recall@0.5**	**IoU$_{avg}$**
Pick-Place	0.652	1.00	0.563
Nut-Assembly	0.948	1.00	0.726
Stack-Block	0.894	1.00	0.708
Press-Button	0.977	1.00	0.825

always generates a bounding box classified as the "target" object. Precision values are also high, exceeding **0.90** for three tasks: Nut-Assembly, Stack-Block, and Press-Button. However, a lower Precision is observed for the Pick-Place task. This is due to the task longer motion duration, both in terms of time and space, which introduces varying scales of the target object throughout the motion, increasing the complexity of predicting accurate bounding boxes. Specifically, out of the 2612 false positives generated for the Pick-Place task, **2125** were produced after the picking phase, while only **487** occurred during the reaching phase.

Generally, the obtained performance demonstrate that the module not only generates bounding boxes with the correct "target" classification but also produces highly accurate bounding boxes that closely match the ground truth. Figure 2.14 shows examples of predictions for all tasks during the execution of a rollout. As observed, during the approaching phase, crucial for avoiding target misidentification, the bounding box remains accurately positioned around the target object. However, during the robot motion, the bounding box slightly shifts, occasionally resulting in false positives.

Multi-task multi-variation scenario

In the multi-task multi-variation scenario, a single model was trained to perform all four tasks. This setup enables a more rigorous validation of the COD module in a complex environment, where objects of different shapes and sizes are involved

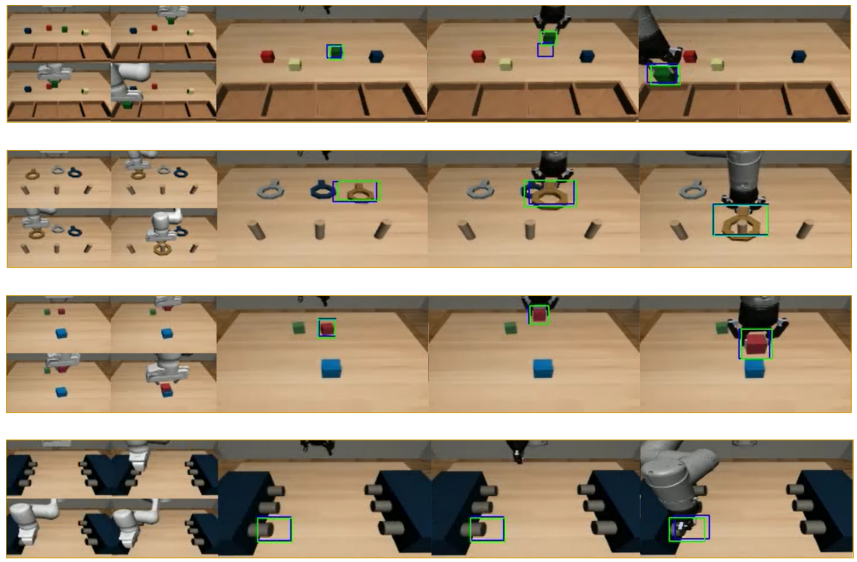


Figure 2.14: Example of predictions generated by the CTOD module in the single-task scenario. The blue-box is the predicted one, while the green is the ground truth.

across various tasks. Table 2.2 presents the overall performance of the COD module in terms of Precision and Recall. In this scenario, the system consistently generates bounding boxes for the "target" class (achieving a Recall of 1.00 for all tasks) with an average overlap greater than 0.5. A similar trend as in the single-task scenario is observed, with a lower Precision for the Pick-Place task due to the same reasons previously explained.

Furthermore, compared to the single-task setting, a slight reduction in Average IoU is observed, which can be attributed to the increased complexity of the task. In this multi-task scenario, a single Fast R-CNN must predict bounding boxes across a diverse set of manipulation tasks, where objects may be similar but appear at different scales in the input images. This variation adds further complexity to the prediction process.

2.4.2.2 Target object and final position detector

This set of validation tests focuses on training the COD module in the complete scenario described in Section 2.2. In this setting, the COD module is trained to predict bounding boxes

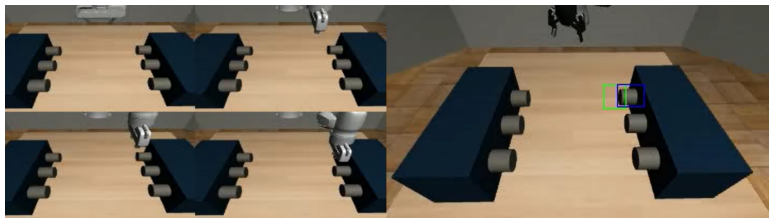


Figure 2.15: Example of Target Bounding Box and Final Placing Position Definition for the Press-Button Task. In this task, the target bounding box is represented by the green bounding box, which encloses the button that needs to be pressed. The blue bounding box represents the final placing position, indicating the location where the robot's end-effector should be positioned to successfully press the button.

Table 2.3: Results of the COD module obtained in the single-task multi-variation scenario. Performances are reported in terms of *Precision* (Prec), *Recall* (Rec) with an IoU threshold of 0.5 and the Average IoU (IoU_{avg}) for both the bounding-box of the "target" and the "target-place" classes.

Task	**Precision@0.5**	**Recall@0.5**	**IoU$_{avg}$** (target)	**IoU$_{avg}$** (target-place)
Pick-Place	0.667	1.00	0.374	0.826
Nut-Assembly	0.958	1.00	0.705	0.909
Stack-Block	0.979	1.00	0.787	0.827
Press-Button	0.967	1.00	0.799	0.683

for both the target object and the final location of interest. Specifically, for the three tasks involving the "pick-and-place" primitive (i.e., Pick-Place, Nut-Assembly, and Stack-Block), the final location corresponds to the target bin, peg, and block to be stacked, respectively. In contrast, for the Press-Button task, the target location is defined as the bounding box of the initial button, translated to the final position of the pressed button (Figure 2.15).

Single-task multi-variation scenario

Table 2.3 summarizes the performance of the COD module in the single-task, multi-variation evaluation setting. It can be observed that, even in this scenario, the COD module con-

Table 2.4: Results of the COD module obtained in the multi-task multi-variation scenario. Performances are reported in terms of *Precision* (Prec), *Recall* (Rec) with an IoU threshold of 0.5 and the Average IoU (IoU_{avg}) for both the bounding-box of the "target" and the "target-place" classes.

Task	**Precision@0.5**	**Recall@0.5**	**IoU$_{avg}$** (target)	**IoU$_{avg}$** (target-place)
Pick-Place	0.735	1.00	0.457	0.825
Nut-Assembly	0.951	1.00	0.685	0.898
Stack-Block	0.867	1.00	0.628	0.799
Press-Button	0.925	1.00	0.734	0.687

sistently identifies both the target object and the target placement location. Specifically, the average Intersection over Union (IoU_{avg}) is generally higher for the target-placement class. This is because, for the Pick-Place, Nut-Assembly, and Stack-Block tasks, the placement location is fixed, making the detection problem easier for the conditioning module to address.

Regarding the "target" class, the IoU_{avg} is lower compared to Table 2.1, with a significant drop observed in the Pick-Place task. However, by examining the distribution of false positives, it can be noted that, out of **7565** false-positive cases, **2001** occurred during the reaching phase, while the remaining **5564** occurred during the placing phase.

It is worth noting that the module can still identify the region of the target object. In fact, by lowering the IoU threshold to 0.1, the number of false positives during the reaching phase drops to 155, indicating that the system is able to detect the region of interest, albeit with lower precision. Nevertheless, for the task at hand, having consistently precise bounding boxes is not critical, as the control module is designed to be robust against minor errors or imprecisions in bounding box predictions.

Multi-task multi-variation scenario

The final evaluation setting essentially confirms the trends observed in the previous one. Table 2.4 summarizes the results obtained from the COD module trained in the multi-task scenario. In this case, a slight drop in Precision is again observed,

which corresponds to a decrease in the average Intersection over Union (IoU_{avg}). This decline can be attributed to the same reasons discussed in the previous multi-task setting, with the added complexity that the module must also predict the bounding box for the target placement location.

In conclusion, this paragraph has introduced the COD module, a Conditioned-Convolutional Neural Network designed to address a novel object-detection problem. The module predicts category-agnostic bounding boxes for the two key regions in a manipulation task: the region where the object to be manipulated is located and the final region where the task is completed (e.g., placing the box, assembling the nut, stacking the block, or pressing the button). In a highly generalized multi-variation scenario, where the object position is not predefined and the semantic attribute assigned to the object is dynamically specified by a command, which is given through a video demonstration of another agent performing the desired task.

The results demonstrate that the proposed module effectively handles both single-task and multi-task scenarios, generating bounding boxes that are correctly classified and accurately identify the regions of interest.

In Chapter 3, it will be shown how the positional information from this command can be effectively utilized to inform the control module about the regions of interest, addressing the problem of target misidentification.

2.5 Conclusion

In conclusion, this chapter proposed and detailed the Conditioned Object Detector (COD), marking the first step in developing a modular architecture for Visual-Conditioned Multi-Task Imitation Learning. Specifically, this module addresses a key aspect of Cognitive Task design: identifying the region of interest in an agent's scene, based on a demonstrated task. The goal is to identify the relevant target object and final placement location in scenarios where multiple objects and placing locations are possible. The specific semantic attributes of these objects (i.e., target or distractor) are determined at runtime

through task demonstration.

To tackle this challenge, a conditioned convolutional architecture was introduced and tested in both single-task multi-variation and multi-task multi-variation scenarios.

The results demonstrated that the proposed COD architecture effectively identified objects of interest, consistently achieving a Recall@0.5 score of **1.00** across all tasks and testing scenarios. This means that for every frame, a bounding box corresponding to the target class was always present.

Regarding Precision@0.5, there was variability across different tasks and scenarios, ranging from a minimum of **0.652** for the Pick-Place task in the multi-task multi-variation scenario to a maximum of **0.997** for the Press-Button task in the single-task multi-variation scenario. Despite this variation, the system was able to consistently identify the target object, with an average IoU always greater than 0.3. While this might appear low, further analysis of the generated bounding boxes revealed that the system successfully identified the target object during the initial stages of the rollout, particularly in the reaching phase, when conditioning on the target object is critical. However, a notable number of false positives occurred during the manipulation and placing phases, where the conditioning on the target object weakens, suggesting that the trained policy must be robust to errors in bounding box generation during these phases.

Chapter 3

Proposed object conditioned control policy

In this chapter, the main contribution of this thesis will be presented: the *Object Conditioned Control Policy* (OCCP). As outlined in Section 1.1, the central challenge addressed in this thesis is *target-object misidentification* within Visual-Conditioned Multi-Task Imitation Learning systems. To tackle this issue, a modular control architecture is proposed, based on the hypothesis that separating the cognitive task from the control task into distinct modules can improve the overall robustness and interpretability of these systems.

Specifically, Section 3.1 will provide an extensive and detailed review of prior research in both Multi-Task Imitation Learning and Object-Oriented Imitation Learning, which are the two fields most closely related to the proposed approach. Section 3.2 will outline the problem being addressed, while Section 3.3 will describe the proposed architecture designed to solve this problem. Finally, Section 3.4 will discuss the experimental setup and present the results obtained from testing the proposed architecture.

3.1 Related Works

This Section will review and discuss the prior research performed to address the proble of Multi-Task Imitation Learning (Section 3.1.1) and Object-Oriented Imitation Learning (Sec-

tion 2.1.1), that are the two topics monstly related to the thesis proposal.

3.1.1 Multi-Task Imitation Learning

The methods discussed in the Section 1.2.3 describe architectures and approaches specifically designed to solve a single task, with limited generalization to the object category (e.g., picking blocks rather than balls) and initial state (e.g., the object starting position). For instance, a system trained for pick-and-place operations cannot be repurposed for tasks like assembly operations. The methods described here address these limitations, solving the problem of *Multi-Task Imitation Learning* (MTIL).

Before starting to present and describe the different methods and approaches, it is necessary to describe the problem. Starting from a reformulation of the dataset used to train the system and the learned policy. Indeed, in Section 1.2.1, the expert dataset $\mathcal{D}^E$ has been introduced. Based on the problem to solve this dataset can composed in different way. Specifically, in the context of MTIL, the dataset $\mathcal{D}^E$ can be seen as a composition of n datasets, $\mathcal{D}^E = \{\mathcal{D}_1, \ldots, \mathcal{D}_n\}$, where $\mathcal{D}_i = \left\{(\tau^j_{m_i},\ c_{m_i}), j = 1, \ldots, N,\ m_i \in \mathcal{M}_i\right\}$ is the *single-task dataset*, composed of:

- N expert demonstration for each m^{th} variation of the i^{th} task, where M_i is the number of variation for the i^{th} task.

- Agent trajectories $\tau^j_{m_i} = (s_0, a_0, s_1, a_1, \ldots, a_{T-1}, s_T)$. where s_t is the state at time t and a_t is the corresponding action (Section 1.2.1).

- Task-conditioning signal c_{m_i} for the m^{th} variation of i^{th} task, which describes the desired task in terms of video demonstrations [14, 119, 6, 8], natural language description [9, 11, 10, 120, 121, 7, 15] or multi-modal prompt, that exploits both visual information (e.g., an image of the target object) and text information that contains the information related to the action to be performed [110].

The goal of Multi-Task Imitation Learning is to learn a *conditioned policy* $\pi_\theta^L(a_t|s_t, c_{m_i})$, that is able to map the current state and command into the corresponding action. Depending on how the policy is defined, various loss functions come into play. In the case of deterministic policies, the learning process focuses on minimizing the Mean-Squared Error (refer to Formula 3.1). However, for probabilistic policies, the learning process centers around minimizing the Negative Log-likelihood (refer to Formula 3.2). This approach aims to enhance the probability of correctly executing the action.

$$\mathcal{L}(\tau_{m_i}^j, c_{m_i}, \pi_\theta^L) = \frac{1}{T}\sum_{t=0}^{T}(a_t - \pi_\theta^L(s_t, c_{m_i}))^2 \tag{3.1}$$

$$\mathcal{L}(\tau_{m_i}^j, c_{m_i}, \pi_\theta^L) = -\frac{1}{T}\sum_{t=0}^{T} log(\pi_\theta^L(a_t|s_t, c_{m_i})) \tag{3.2}$$

The following sections will describe the various approaches proposed to address the problem. Figure 3.1 illustrates the taxonomy used for the Multi-Task Imitation Learning methods. Specifically, the methods are categorized based on the type of generalization required by the algorithm (Few-Shot vs. Zero-Shot). For Few-Shot generalization, the Meta-Learning paradigm will be discussed, as it is most relevant to the problem at hand. For Zero-Shot generalization, the methods are further divided based on the type of conditioning signal used, whether it is provided through natural language descriptions or visual information.

Few-Shot MTIL refers to approaches designed to train a model on a variety of tasks so that it can effectively solve a new task using only a small number of samples and consequently requires only a few back-propagation steps [122]. In this context, one of the most significant learning paradigms, especially relevant to robotic manipulation, is *Meta-Learning*. The goal of Meta-Learning is to train a model that can "learn to learn," meaning it develops a set of general weights θ that, while not directly usable for solving any specific task within a distribution of tasks $\mathcal{T}$, can be quickly adapted through a few backpropagation steps to solve a given task within that

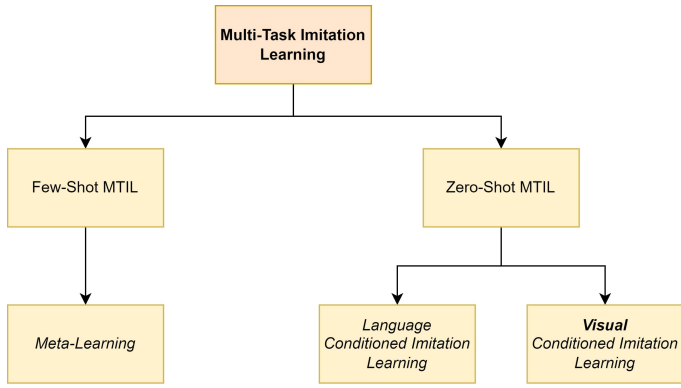


Figure 3.1: Multi-Task Imitation Learning Taxonomy

distribution, $\mathcal{T}_i \in \mathcal{T}$. One of the most popular Meta-Learning algorithms is the *Model-Agnostic Meta-Learning* (MAML) algorithm [122], described in Algorithm 7. The MAML algorithm follows an iterative learning procedure consisting of two steps:

- **Meta-Learning**: During this phase, task-specific weights θ_i are computed for each sampled task $\mathcal{T}_i$. Specifically, the *meta-parameters* θ are updated according to the gradient obtained from evaluating the loss function on the i^{th} task $\mathcal{T}_i$, where the function f is parameterized by the meta-parameters θ.

- **Meta-Adaptation**: In this phase, the meta-parameters are further refined. The loss function f, now parameterized by the task-specific parameters for the i^{th} task, is used to adjust the meta-parameters based on the gradients derived from the sum of the loss functions evaluated on the task-specific weights. This process provides feedback to the meta-parameters θ from each task, leading to a generalized point that can be easily adapted to new tasks (Figure 3.2).

The MAML algorithm is the base for different methods which apply Few-Shot Imitation Learning in the context of Behavioral Cloning [123, 12, 124].

In [123], MAML algorithm was used to prove the effectiveness of Meta-Learning in the context of real robot manipula-

Algorithm 7 Model-Agnostic Meta-Learning (MAML) [122]

Require: Distribution over tasks $p(\mathcal{T})$
  Randomly initialize θ
  while $i = 1, \dots N$ **do**
    Sample batch of tasks $\mathcal{T}_i \sim p(\mathcal{T})$
    for all $\mathcal{T}_i$ **do**
      Evaluate $\nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$ w.r.t. K examples
      Compute adapted parameters with gradient descent: $\theta_i' = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$
    end for
    Update $\theta \leftarrow \theta - \beta \nabla_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta_i'})$
  end while

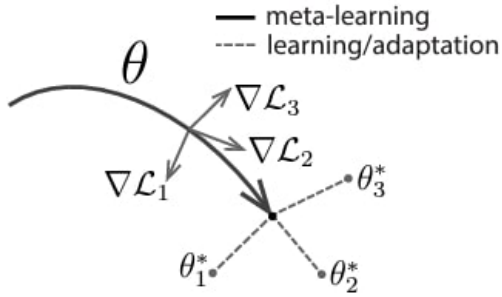


Figure 3.2: Diagram of MAML algorithm, which optimizes for a representation θ that can quickly adapt to new tasks

tion, with visual observations, as opposite to [125]. A Convolutional Neural Network was trained by following the Algorithm 7, using as loss-function the Mean Squared Error, computed between the predicted action and the ground truth one. For real-robot experiments a dataset of **1300** placing demonstrations (i.e., place an holded object in a target container), containing near to **100** different objects, was collected through teleportation. The trained system was tested by performing the adaptation step on one video demonstration, over 29 new objects, moreover, between the video demonstration and the actual execution, the objects configuration was changed. In this setting the system reached the **90**% of success rate, outperforming baseline methods based on LSTM [125], and contextual network (i.e., a Convolutional Neural Network that takes

in input the current observation and the image representing the target state).

In [12], the *Domain Adaptive Meta-Learning* algorithm (DAML) was proposed with the goal of learning to infer a policy from a single human demonstration. To achieve it, a two-step algorithm was proposed. In the first-step, called **Meta-Learning step**, given in input, for each task $\mathcal{T}$, a set of human demo $\mathcal{D}_{\mathcal{T}}^{h}$ and a set or robot demo $\mathcal{D}_{\mathcal{T}}^{r}$ (Figure 3.3), the *initial policy parameters* θ and the *adaptive loss* parameters ψ are learned, solving the problem in Formula 3.3.

$$\min_{\theta,\psi} \sum_{\mathcal{T}\sim p(\mathcal{T})} \sum_{\mathbf{d}^h \sim D_{\mathcal{T}}^{h}} \sum_{\mathbf{d}^r \sim D_{\mathcal{T}}^{r}} \mathcal{L}_{BC}(\theta - \alpha \nabla_{\theta} \mathcal{L}_{\psi}(\theta, \mathbf{d}^h), \mathbf{d}^r) \tag{3.3}$$

The outer loss is the classic supervised Behavioral Cloning loss, defined as $\mathcal{L}_{BC}(\phi, \mathbf{d^r}) = \sum_t \log(\pi_\phi(a_t \mid s_t, o_t))$. The inner loss, $\mathcal{L}_\psi$, is a learned **adaptive loss**. Specifically, $\mathcal{L}_\psi$ is used during Meta-Adaptation, where the policy parameters are updated by evaluating the gradients derived from $\mathcal{L}_\psi$. This process involves using a video of a human demonstrating a new task $\mathcal{T}$ as input, leading to the policy update defined by $\phi_{\mathcal{T}} = \theta - \alpha \nabla_\theta \mathcal{L}_\psi(\theta, \mathbf{d}^h)$. This adaptive loss is the key component proposed in DAML. To use it effectively, it is necessary to learn the parameters ψ, observing how there is no direct correspondence between the human video demonstration and the robot's ground truth actions. To address this challenge, the authors of DAML observed that while the policy learns to produce appropriate actions through the $\mathcal{L}_{BC}$ loss, the adaptive loss should instead adjust the perceptual aspect of the policy, focusing on human motion and the manipulated object. Based on this insight, the authors implemented the function $\mathcal{L}_\psi$ using a 1D Temporal Convolutional Network (Figure 3.4). The convolutional layers are applied to a stack of embeddings generated by the policy π across different frames of the video demonstrations. The parameters of this module are learned during the meta-training phase, following the weight update process described in Formula 3.4. The objective of $\mathcal{L}_\psi$ is to generate task-specific policy parameters $\phi_{\mathcal{T}}$ that guide the policy to produce effective actions.

Figure 3.3: Tasks performed in [12]. (Top row) Human demonstration, (Bottom row) robot demonstration. (Left) Placing task, (Middle) pushing task, (Right) pick-and-place task.

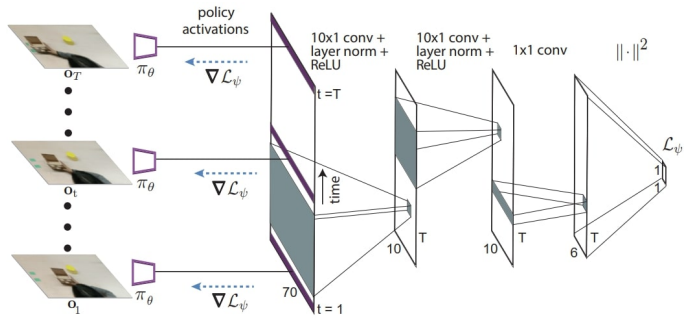


Figure 3.4: The Temporal Adaptation Loss architecture applies 1D temporal convolutional layers to the stacked embeddings generated by the policy π from the frames of the human video demonstration.

$$
\begin{aligned}
(\theta, \psi) &\leftarrow (\theta, \psi) - \beta \nabla_{\theta,\psi} \mathcal{L}_{\mathrm{BC}} \left(\phi_{\mathcal{T}}, \mathbf{d}^r\right) \\
\phi_{\mathcal{T}} &= \theta - \alpha \nabla_\theta \mathcal{L}_\psi \left(\theta, \mathbf{d}^h\right)
\end{aligned}
\tag{3.4}
$$

Experimental evaluation on tasks such as placing, pushing, and pick-and-place, has shown that:

- The system was able to generalize across both new objects and objects configuration starting from only a single human demonstration;
- A performance degradation was observed in large domain-shift experiments, such as novel backgrounds and different camera view-points.

Meta-Learning algorithms have demonstrated intriguing properties, notably their capacity for few-shot generalization to

novel objects and object configurations. However, it has been observed that during the adaptation step, these methods tend to lose their effectiveness in performing other tasks. This limitation has underscored the need for the development of Multi-Task Imitation Learning methods, which aim to address these shortcomings and enable more versatile task execution in complex scenarios. These kind of methods will be discussed in the following paragraphs.

Zero-Shot MTIL refers to approaches that aim to train a model capable of solving different tasks without any further adaptation or backpropagation steps. This approach addresses a key issue in Meta-Learning methods, which is the problem of forgetting how to solve previous tasks after adapting to a new one. The goal is to develop a single policy that can handle multiple tasks in a zero-shot manner.

In this context, a crucial design choice is how to convey the desired task to the policy. The literature identifies two main methods to address this problem:

1. *Language Conditioned*: These methods leverage natural language descriptions of tasks to inform the model about the task to be executed.

2. *Visual Conditioned*: These methods use visual information (e.g., goal-state images, video demonstrations) to provide the model with the task instructions.

Language Conditioned, as said a possible and intuitive way to inform the policy about which task to execute is through natural language description. Indeed, by looking to the phrase "Pick the blue cube and place it in the red bow", there are both information about the action to perform (pick and place) and which are the objects involved (blue cude and red bow). Consequently, through training a neural network with a diverse set of tasks, the system should exhibit the ability to generalize its understanding to both previously unseen objects within familiar tasks and entirely novel tasks composed of the fundamental actions practiced during training. This approach showcases the potential for **robust** and **adaptable human-robot interaction** in real-world scenarios.

Foundational research, such as that by [9] and [11], has sought to explore the previously mentioned hypothesis. Notably, [9] introduced an innovative architectural framework, depicted in Figure 3.5, marking the first instance of seamlessly integrating language, vision, and control tasks. This model is composed of two critical components: a *Semantic Model*, which generates a task embedding denoted as e by processing the initial scene image and the accompanying text command, and a *Control Model*, which generates the control signal using the current robot state r_t and the task embedding e.

A crucial aspect of such architectures is the management of the visual state, represented by the image I, and the command v, to create a meaningful embedding e that encapsulates both the current scene state and the desired command. Specifically, the image is first processed using a pre-trained object detection network (Faster R-CNN [111]) to identify salient objects in the robot's environment. The detected objects are represented by feature vectors, which include class and bounding box information. Concurrently, the language command is embedded into a suitable representation using a language embedding technique (e.g., GloVe [126]), with the command vector V encoded by a recurrent GRU unit. To associate the objects identified with the sentence embedding s, a likelihood value is computed for each object proposal $a_i = w_a^T f_a([\mathrm{f}_i, s])$, and a probability distribution is computed over the candidates $\mathbf{a} = \mathrm{softmax}([a_0, \ldots, a_c])$. Finally, the task embedding e is formed by a fully connected layer that takes as input the sentence embedding s and the weighted sum of object candidate features $e' = \sum_{i=0}^{c} f_i a_i$.

Training of this model was conducted on two fundamental tasks, namely "Picking" and "Pouring", within scenarios featuring multiple objects of the same category, which served as distractors (see Figure 3.6). The subsequent testing experiments demonstrated the system's capability to successfully complete the picking task 98 out of 100 times and the pouring task 85 out of 100 times within novel scenarios. These results serve as compelling evidence of the efficacy and potential of language-conditioned methodologies in the field.

Authors in [11] make a step towards a more general agent,

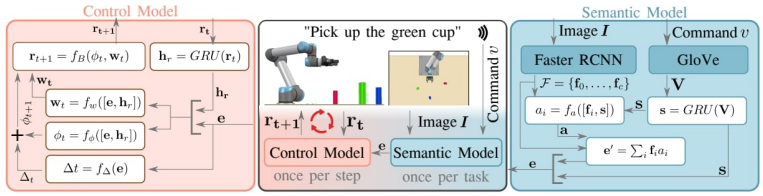


Figure 3.5: Architecture proposed in [9]. The *Semantic Model* takes in input the image I and the command v, generating a command conditioned embedding e. The *Control Module* receives in input the embedding e and the current robot state r_t and produces the next control signal.

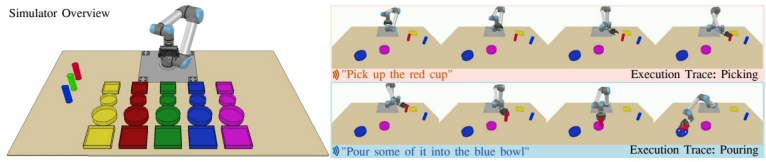


Figure 3.6: (Left) Set of object used in [9]. (Right) Sample of task execution (right)

by proposing a large-scale dataset containing **100** diverse manipulation tasks. The demonstrations were collected through both expert teleoperation and shared autonomy process (HG-DAgger [64]). The demonstrated tasks were related to pick-and-place, grasp, pick-and-drag, pick-and-wipe, and push skills. The dataset was used to train the network in Figure 3.7. As it can be noted the samples were composed by the current robot observation, and a conditioning represented by either a ***natural language description*** or a ***human video demonstration***. The idea was that training a conditioned policy over the current observation o_t, and a task representation c, , it would allow the policy to generalize over new tasks in a zero-shot manner (i.e., without any fine-tuning). Specifically, in contrast to the previous method, this approach does not rely on pre-trained object detectors to identify candidate regions. Instead, the Task Embedding is directly injected into the Feature Maps generated by the Convolutional Neural Network ResNet-18. This injection is carried out through the FiLM layer [60].

Experimental results shown that, over 28 held-out tasks,

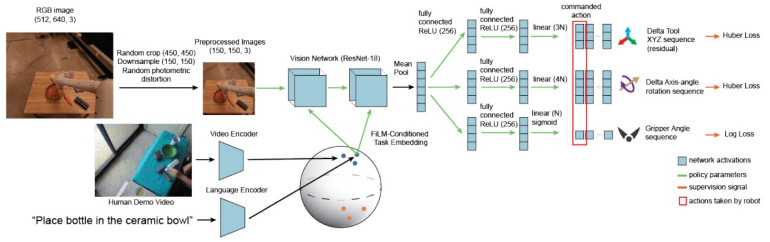


Figure 3.7: Architecture proposed in [11]. Here, the Task Embedding is injected directly in the Feature Maps generated by the ResNet-18.

containing both completely new objects, and known objects but in different tasks, an average success rate of **38%** was reached in the easiest setting, with only one distractor and with natural language instruction. The success rate dropped to **4%** in the hardest setting with 4 distractors and video conditioning.

Foundational research in the field of Language-Conditioned Multi-Task Imitation Learning has demonstrated promising results in zero-shot generalization. However, the robustness of their performance remains a challenge. Subsequent research, as highlighted in [7, 10, 121], has focused on enhancing performance.

In particular, the authors of [7] sought to investigate whether the transfer of knowledge from extensive, diverse, and task-agnostic datasets, which has enabled modern machine learning models to excel in zero or few-shot learning scenarios for new and specific tasks, is applicable within the realm of robotics. This inquiry arises due to the presence of high-capacity architectures capable of assimilating knowledge from such large datasets. To explore this prospect, the authors in [7] introduced a comprehensive dataset comprising over 130,000 demonstrations collected across more than 700 household tasks (Figure 3.8). Additionally, they proposed a Language-Conditioned Transformer-based architecture (Figure 3.9). Here the authors made relevant modification in the architecture, with respect to the previous BC-Z architecture [11].Indeed, they modified the Visual Module by using an EfficientNet [127] instead of the ResNet-18. The instruction was encoded using the Univer-

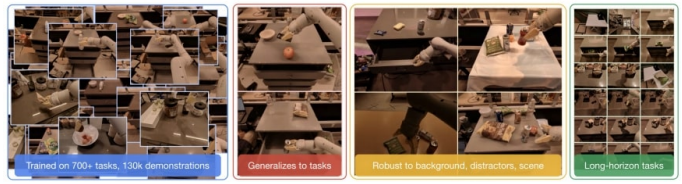


Figure 3.8: Examples of household scenarios in RT-1 large-scale dataset.

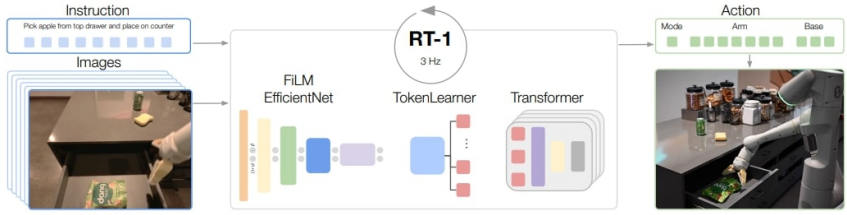


Figure 3.9: RT-1 Language-Conditioned Transformer based architecture proposed in [7].

sal Sentence Encoder [128], and the policy was implemented through a Transformer. Additionally, the authors addressed the real-time constraints of robot control. To accelerate inference time and achieve a frequency of 3 Hz, the authors employed a TokenLearner module [129], which utilizes an attention mechanism to select the most relevant tokens, thereby reducing the number of tokens that the underlying control module must process to infer the action.

It is worth noting the intriguing findings presented in Table 3.2. The model exhibits robustness in replicating tasks it

Table 3.1: Distribution of tasks in large-scale dataset proposed in [7]

Skill	Count	Description	Example Instruction
Pick Object	130	Lift the object off the surface	pick iced tea can
Move Object Near Object	337	Move the first object near the second	move pepsi can near rxbar blueberry
Place Object Upright	8	Place an elongated object upright	place water bottle upright
Knock Object Over	8	Knock an elongated object over	knock redbull can over
Open Drawer	3	Open any of the cabinet drawers	open the top drawer
Close Drawer	3	Close any of the cabinet drawers	close the middle drawer
Place Object into Receptacle	84	Place an object into a receptacle	place brown chip bag into white bowl
Pick Object from Receptacle and Place on the Counter	162	Pick an object up from a location and then place it on the counter	pick green jalapeno chip bag from paper bowl and place on counter
	9	Skills trained for realistic, long instructions	open the large glass jar of pistachios pull napkin out of dispenser grab scooper
Total	744		

Table 3.2: Results reported in [7] by training the same model RT-1 with different dataset size

				Generalization			
Models	% Tasks	% Data	Seen Tasks	All	Unseen Tasks	Distractors	Backgrounds
RT-1	100	**100**	97	73	76	**83**	59
RT-1	100	**51**	71	50	52	**39**	59
RT-1	100	**37**	55	46	57	**35**	47
RT-1	100	**22**	59	29	14	**31**	41

has encountered before, and it even performs reasonably well on tasks it hasn't seen previously. However, a notable decline in performance becomes apparent when the model is exposed to novel backgrounds and scenarios featuring distracting objects, especially when the available data for these situations is limited. This observed trend holds significance because, unlike domains such as Computer Vision and Natural Language Processing where gathering large-scale datasets is relatively straightforward, the collection of real-world robotic datasets is a laborious and time-consuming endeavor. Furthermore, these real-world robotic datasets often have limited applicability to other research due to **disparities in action space, robot morphology, and scene representation**, as pointed out by [7].Therefore, the aspiration is to develop a system capable of replicating tasks with minimal demonstrations specifically gathered from the particular robot in use.
Furthermore, the decline in success performance when faced with distractor objects emphasizes that addressing the policy-learning problem in an end-to-end manner, which involves mapping high-dimensional and high-level inputs like images and text directly to low-dimensional, low-level outputs such as the actions to be executed, may not be the most effective approach. This is because such models might lack the necessary perceptual components that enable them to initially recognize the target object within the scene, subsequently navigate towards it, and commence the manipulation process. This process aligns with how humans approach manipulation tasks [130].

As we have seen up to now, all the proposed methods were tested on different robotic platforms with different scenarios and environments. As in other Computer Vision problem, there are no well now benchmark that are used by the re-

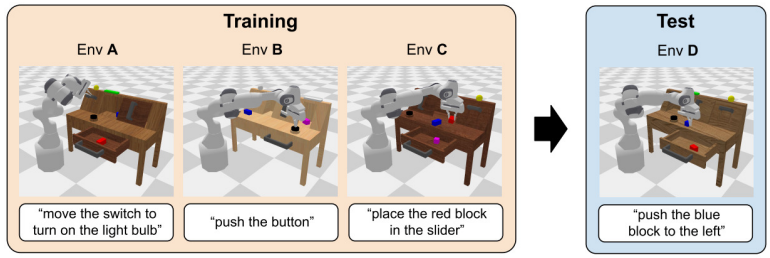


Figure 3.10: Environment proposed in CALVIN benchmark [**?**]. The environments have different textures and all static elements such as the sliding door, the drawer, the light button, and switch are positioned differently

searchers around the world. To solve this problem, authors in [10] proposed CALVIN (Composing Actions from Language and Vision), which is an open-source simulated benchmark designed for learning long-horizon language-conditioned tasks in robotic manipulation. Specifically, CALVIN propose a set of 34 manipulation tasks in 4 different environments (Figure 3.10).
This benchmark was used by [121, 131, 132]. Specifically, the work proposed in [132] is actually the best performing method on the CALVIN benchmark. The authors proposed an architecture that is able to handle goals described in terms of both natural language description and goal image, moreover they used a Diffusion Transformer Model as policy (Figure 3.11). To reach this goal, the authors had to solve first of all the problem related to how to force the Multimodal Transformer Encoder to generate the same behavior indipendently from the goal modality. To solve this problem authors of MDT proposed two auxiliary self-supervised loss functions:

- *Masked Generative Foresight* (MGF) is a reconstruction-loss function designed to ensure that the MDT generates embeddings capable of guiding the robot behavior consistently across different modalities. This means that the tokens generated by the MDT can be used to construct image patches representing feature states, whether the goal is specified in terms of an image or a language description. Specifically, given the latent embedding of the MDT encoder for state s_i and goal g, MGF trains a Vi-

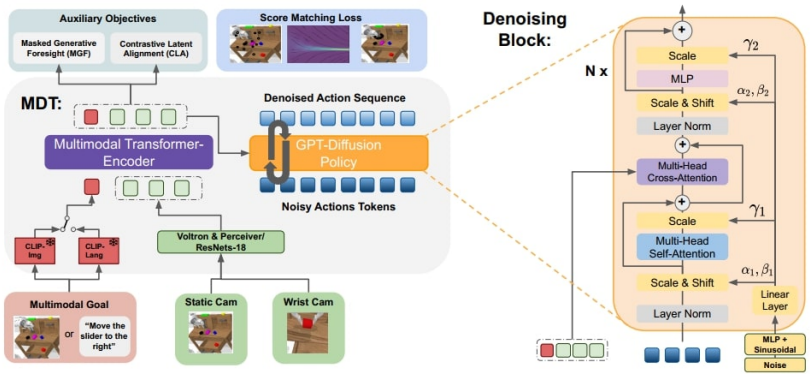


Figure 3.11: The Multimodal Diffusion Transformer (MDT) architecture proposed in [132]

sion Transformer (ViT) to reconstruct a sequence of 2D image patches $(u_1, \ldots, u_U) = \text{patch}(s_{i+v})$ corresponding to the future state s_{i+v}.

- *Contrastive Latent Alignment* (CLA) is a contrastive loss term designed to encourage the MDT to align the embeddings generated from a goal image with those generated from a language description. For each training sample (s_i, a_i) paired with a multimodal goal specification $G_{s_i,a_i} = \{o_i, l_i\}$, CLA reduces the image and language goals to vectors z_i^o and z_i^l, respectively. The CLA loss is then computed using the InfoNCE loss, based on the cosine similarity $C(z_i^o, z_i^l)$ between the image-goal conditioned state embedding z_i^o and the language-goal conditioned state embedding z_i^l.

As mentioned, this method was tested on the CALVIN benchmark, specifically in the $ABCD \rightarrow D$ test scenario, where the model was trained on the $ABCD$ environments and tested on environment D. The CALVIN benchmark comprises a set of 1000 rollouts, with each rollout consisting of a sequence of 5 commands. It is noteworthy that the MDT architecture successfully completed 80% of the rollouts up to the fifth command. In particular, the MGF loss was observed to have a significant impact on system performance, improving the success rate for the fifth command from 69.8% to 79.4%. This

demonstrates that making the embedding informative about the system's evolution can meaningfully guide the diffusion system, which predicts actions over a certain time horizon into the future.

In the context of Language Conditioned MTIL, other important works to cite include [133, 15]. Specifically, the authors in [133] introduced CLIP-Port, a two-stream architecture that explicitly models the two key tasks in language-conditioned imitation learning: reasoning about **what to do** and reasoning about **how to do it**. The former task, referred to as *semantic reasoning*, is derived from the text-based command and is handled using a pre-trained CLIP architecture [134]. The latter task, known as *spatial reasoning*, is managed by leveraging the Transporter architecture [135]. The overall architecture (Figure 3.12) is an Encoder-Decoder framework that ultimately outputs an **affordance map**, which identifies the locations for executing pick or place operations (Figure 3.13).

During testing, this method was not directly compared with other Language Conditioned MTIL approaches. However, the results obtained allow for several observations. Notably, for tabletop manipulation tasks, the ability to reason using both spatial and semantic features enables a high success rate (ranging from 80% to 90%) on tasks involving seen object attributes, such as color, even with a relatively low number of demonstrations (100). However, similar to the results reported in Table 3.2, CLIP-Port also struggles with entirely novel tasks, as evidenced by a significant drop in performance when dealing with unseen object attributes and a lower number of demonstrations.

In conclusion, as discussed in this paragraph, considerable research effort has been dedicated to addressing the problem of Language Conditioned MTIL, particularly in terms of architectural designs and learning strategies. However, it remains challenging to draw definitive conclusions from these various approaches, as they are generally not evaluated on a common benchmark. Despite this issue, some trends can be observed. There is clearly room for improvement in the generalization capabilities of these methods, particularly in handling novel scenarios and tasks. Additionally, data efficiency remains a

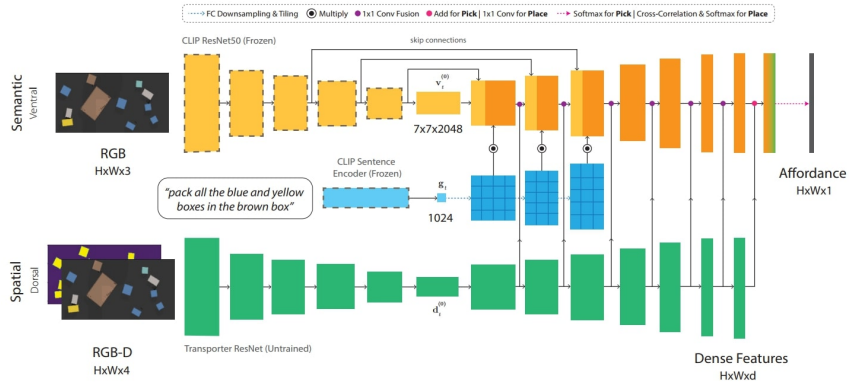


Figure 3.12: The Dual Stream architecture proposed in [133] consists of two parallel streams: a semantic stream and a spatial stream. The semantic stream utilizes a frozen CLIP ResNet50 to encode the RGB input, with the decoder layers conditioned by tiled language features from the CLIP sentence encoder. Meanwhile, the spatial stream encodes the RGB-D input, and its decoder layers are laterally fused with those of the semantic stream. The final output is a map of dense pixel-wise features, which is used to predict pick or place affordances.

concern, as there is a noticeable decline in performance as the number of expert trajectories decreases. Another challenge is the ability to manage cluttered scenes with **relevant distractors**, objects that may have been manipulated during training but must be ignored in the target task, which can lead to occlusion and/or confusion.

Visual Conditioned refers to methods that use visual information as a conditional signal. This information can be represented either as an image depicting the desired goal state or as a sequence of frames illustrating how the task should be performed. The idea behind this kind of methods is to build systems that are able to replicate tasks by observing the execution performed by other agents, like human can learn task by mimiking the execution of other humans.

Preliminary works, such as those by [14] and [119], introduced architectures conditioned on the first and last frames of task demonstrations. Specifically, in [14], the authors proposed TecNets (Task-Embedded Control Networks), an architecture

Figure 3.13: Example of affordance map. (Left) The top-view input image. (Center) The affordance map for the pick-operation, since the task is to grab cherries the map highlights the two pickable cherries. (Right) The affordance map for the place operation.

illustrated in Figure 3.14. TecNets consist of two main components:

- *Task-Embedding Network*: The purpose of this network is to create an embedding space where demonstrations of the same task are closely clustered, while embeddings of different tasks are kept as distant as possible. This network is trained using a contrastive loss function, defined in Formula 3.5, which ensures that the embedding of the k^{th} sample of the j^{th} task is similar to the "sentence" representing that task (i.e., the average of the embeddings of the j^{th} task) and maximizes the distance from the sentences of other tasks.

$$\mathcal{L}_{emb} = \sum_{\tau_k^j \in \mathcal{T}^j} \sum_{\mathcal{T}^i \notin \mathcal{T}^j} \max\left[0, \text{margin} - s_k^j \cdot s^j + s_k^j \cdot s^i\right] \tag{3.5}$$

- *Control Network*: This module implements the policy π, which takes as input the current observation o and the sentence s of the desired task. It then generates the corresponding action for the robot, conditioned on the task.

The results from this preliminary work demonstrated the feasibility of training a vision-conditioned system capable of solving tasks in a zero-shot manner. The system achieved a success rate of **86.31**% on a simulated 2D reaching task and

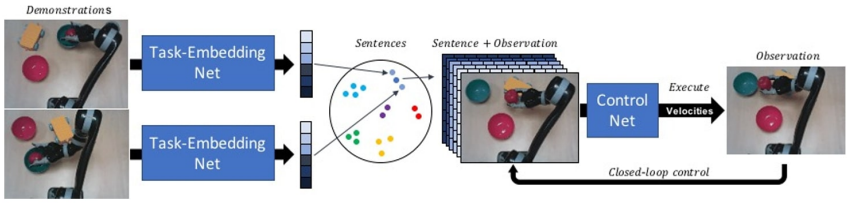


Figure 3.14: Architecture proposed in [14]. The *Task Embedding Net* generates the embedding representing the task to be performed given the goal image. The *Control Net* implements the policy, and takes in input the current observation and the tiled task embedding

77.25% on a simulated pushing task. However, it is important to note a significant limitation of these approaches: the conditioning signal often fails to adequately capture critical information about the optimal strategy for solving the task. This shortcoming arises because the conditioning signal does not fully encode the strategies needed for effective task execution.

The methodologies introduced in [6] and [8] aim to advance the concept of an ideal multi-task agent, capable of replicating new tasks based on a single demonstration, often performed by another agent (e.g., a robot or a human).

In [6], the authors focused on training a vision-based control architecture (depicted in Figure 3.15) that can replicate a task demonstrated through a set of frames sampled from a video of another agent performing the task. The architecture relies on two main components:

- *ResNet-18* [59], which serves as the backbone for extracting visual embeddings from the current agent state and the demonstration frames.

- *Transformer* [113], which uses the attention mechanism to combine embeddings from the demonstration frames with the current visual observation.

This architecture was trained using multiple loss terms, specifically:

- *Behavioral Cloning* term $\mathcal{L}_{BC}$ (Formula 3.6), a supervised loss function used to learn the action to perform given the agent's observation and the demonstrated task. Specifically, the loss function is a negative log-likelihood, computed over a mixture of learned logistic distributions, where the i^th component is parameterized according to the mean and variance (μ_i, σ_i) which are estimated through a MLPs.

$$\mathcal{L}_{BC} = -ln(\sum_{i=0}^{k} \alpha_i(\phi_t) P(a_t, \mu_i(\phi_t), \sigma_i(\phi_t))) \tag{3.6}$$

- *Inverse Model Regularizer* term $\mathcal{L}_{inv}$ (Formula 3.7), a regularization term where the output is the action a_t, given two consecutive representations $\tilde{\phi}_t$ and $\tilde{\phi}_{t+1}$, effectively solving the inverse control problem.

$$\mathcal{L}_{inv} = -ln(\sum_{i=0}^{k} \alpha_i(\tilde{\phi}_t, \tilde{\phi}_{t+1}) logistic(\mu_i(\tilde{\phi}_t, \tilde{\phi}_{t+1}), \sigma_i(\tilde{\phi}_t, \tilde{\phi}_{t+1}))) \tag{3.7}$$

- *Point Prediction Auxiliary* term $\mathcal{L}_{point}$, a simple regression loss that, given the current representation ϕ_t, predicts the position of the gripper in the next $t + H$ steps. This loss is used to improve the explainability of the actions performed by the robot.

The authors specifically trained and tested the architecture on the *pick-place* task, which consists of a total of 16 variations (Figure 3.16). In the proposed evaluation setting, the TOSIL architecture achieved a success rate of $\mathbf{88.8}\% \pm \mathbf{5.0}\%$, demonstrating the system's ability to accurately capture the requested variations represented in the demonstration frames, even when faced with differences in robot embodiment.

In [8], the authors presented MOSAIC (Multi-task OneShot imitation with self-AttentIon and Contrastive learning), which is an improvement of the work proposed in [6] from both the architectural point-of-view and from the evaluation setting. Indeed, from the architectural point of view, MOSAIC is designed to model a demonstration-conditioned policy, denoted

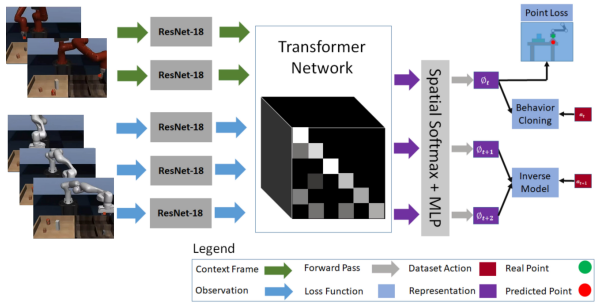


Figure 3.15: Transformer based architecture proposed in [6]. The Transformer network is used to create task-specific representation, given context and observation features computed with ResNet-18.

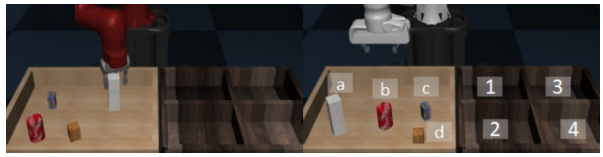


Figure 3.16: Validation setting proposed in [6]. The 16 tasks consist of taking an object (a-b) to a bin (1-4). On the left there is the demonstrator robot, while on the right there is the agent robot.

as $\pi_\theta^L(a_t|s_t, c)$, with c representing the current task demonstration, such as a video depicting a robot performing the task. Specifically, the architecture is an optimization of TOSIL since, where the encoder-decoder Transformer architecture has been substituted with an encoder-only architecture leveraging the self-attention mechanism to correlate the embeddings coming from the current agent observation and the one coming from the demonstration frames (Figure 3.17). This adjustment allows to have a lower number of parameters and consequently improve the inference time.

To handle this complex scenario involving both multiple variations and multiple tasks, the authors proposed training the system using a combination of loss functions. Specifically, in addition to the $\mathcal{L}_{BC}$ loss function defined in Formula 3.6 and the $\mathcal{L}_{inv}$ loss function defined in Formula 3.7, the authors

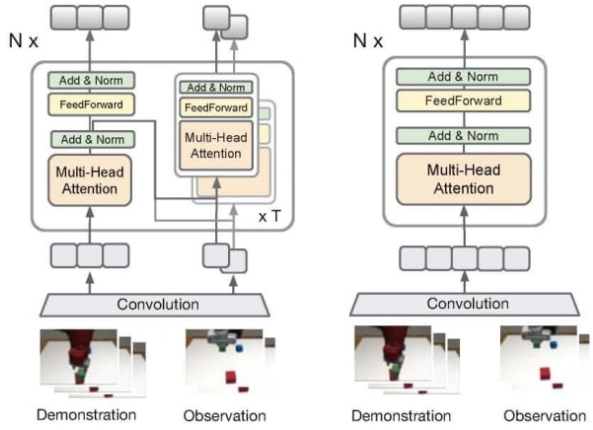


Figure 3.17: (Left) The TOSIL architecture, as proposed in [6]. (Right) The MOSAIC architecture, as introduced in [8]. In the MOSAIC architecture, the original encoder-decoder Transformer architecture has been replaced with an encoder-only architecture featuring self-attention modules.

introduced a contrastive loss term implemented through the InfoNCE objective (Formula 3.8). The goal of this loss is to enable the system to create similar representations for time-adjacent frames within a given task while maximizing the distance from representations corresponding to other tasks. In this context, q is the anchor embedding obtained from a randomly sampled frame in a given batch B, k_+ is the positive example, which is a nearby frame from a different view of the batch (i.e., the original batch with a different data augmentation applied), and k_i represents the negative samples, obtained from any other frame in the batch.

$$\mathcal{L}_{Rep} = \log\left(\frac{\exp(q^T W k_+)}{\exp(q^T W k_+) + \sum_{i=1}^{F-1}\exp(q^T W k_i)}\right) \tag{3.8}$$

To test the model, a dataset containing **seven distinct tasks** and **61 variations** (Figure 3.18) was generated by executing hand-written policies in a simulation environment. This dataset was employed to train and compare the proposed MOSAIC architecture against other one-shot (meta-)imitation learn-

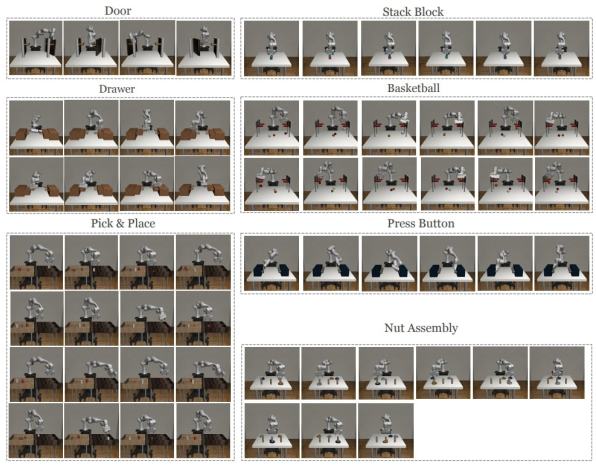


Figure 3.18: The evaluation tasks proposed in [8] consist of a total of 7 tasks with 61 semantic variations.

ing methods, such as [12] and [6]. The results, presented in Table 3.3, demonstrate that MOSAIC surpasses previous methods in the Single-Task Zero-Shot Imitation Learning setting, particularly when addressing individual tasks with multiple variations and testing on previously unseen scenarios (i.e., novel object configurations).

Furthermore, when evaluated in a Multi-Task setting, where the system is trained on all tasks and subsequently tested on each task separately, MOSAIC exhibits the capability to partially replicate the demonstrated tasks. It is important to note that transitioning from a single-task to a multi-task context introduces inherent challenges. Despite being familiar with the tasks used during training, the success rate tends to decrease for almost all tasks. This phenomenon underscores the necessity for training procedures and architectures capable of generating embeddings that accurately represent both the task itself and its various sub-tasks. Such capacity is essential for reusing these embeddings when executing new instances or entirely novel tasks.

In this context, further improvements were introduced by the authors of [24, 136]. Specifically, the authors in [24] analyze the shortcomings of existing methods like TOSIL and MOSAIC, which often struggle due to issues such as the DAgger

Table 3.3: Results obtained in single-task and multi-task one-shot imitation learning [8].

Task	Setup	DAML [12]	TOSIL [6]	MOSAIC [8]
Open door	single	23.3 ± 5.2	57.9 ± 7.1	**67.1 ± 5.5**
	multi	10.8 ± 5.4	49.2 ± 6.0	**68.3 ± 6.3**
Open drawer	single	15.4 ± 5.5	57.5 ± 3.9	**65.4 ± 3.4**
	multi	3.3 ± 1.4	53.3 ± 4.0	**55.8 ± 3.6**
Press button	single	62.8 ± 3.9	56.4 ± 2.4	**71.7 ± 3.9**
	multi	1.7 ± 0.7	63.3 ± 3.5	**69.4 ± 3.4**
Pick-and-Place	single	0.0 ± 0.0	74.4 ± 2.1	**88.5 ± 1.1**
	multi	0.0 ± 0.0	19.5 ± 0.4	**42.1 ± 2.3**
Stack block	single	10.0 ± 1.8	13.3 ± 2.6	**79.3 ± 1.8**
	multi	0.0 ± 0.0	34.4 ± 3.4	**70.6 ± 2.4**
Basketball	single	0.4 ± 0.3	12.5 ± 1.6	**67.5 ± 2.7**
	multi	0.0 ± 0.0	6.9 ± 1.3	**49.7 ± 2.2**
Nut assembly	single	2.2± 1.4	6.3 ± 1.9	**55.2 ± 2.8**
	multi	0.0 ± 0.0	6.3 ± 1.3	**30.7 ± 2.5**

problem (distribution shift from offline training), last-centimeter errors in fine motor control (e.g., collisions when the robot reaches the target object to pick), and misalignment with task contexts rather than the actual tasks (e.g., the system links the trajectories to the objects present in the scene rather than focusing on what is demonstrated by the expert).

To overcome these issues, the authors proposed a modular approach that separates task inference, i.e., understanding the intent of the demonstrations, from task execution, i.e., generating the actions to perform during the rollout. To achieve this, they introduced AWDA (Attributed Waypoints and Demonstration Augmentation), a modular framework for visual demonstration represented in Figure 3.19. AWDA is based on two main concepts:

- *Attributed Waypoints*, designed to mitigate task execution errors, primarily those related to the distributional shift problem. These are classic waypoints (i.e., sequences of 6D poses of the end-effector) augmented with attributes, such as whether there is an object in the gripper. The robot moves between these generated waypoints using hand-defined motion primitives, which helps eliminate

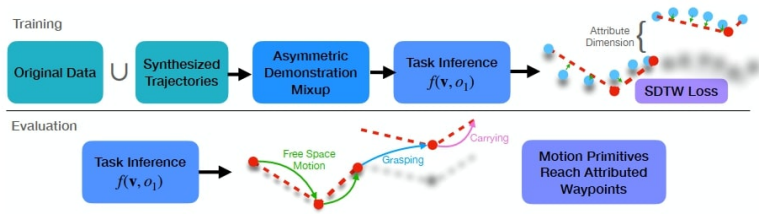


Figure 3.19: AWDA framework proposed in [24]. The task inference network $f(v, o)$ predicts a sequence of attributed waypoints (red dots) that are achieved by hand-defined motion primitives. The original data are augmented with free-space motion trajectories and asymmetric demonstration mixup in order to reduce the correlation between tasks and task content.

small errors during rollout execution that could otherwise cause the system to deviate from the correct trajectory.

- *Demonstration Augmentation*, introduced to decouple tasks from task contexts. Specifically, two types of augmentation are introduced: **Asymmetric Demonstration Mixup**, which generates novel samples by mixing samples from two distinct trajectories according to Formula 3.9, where v and $\tilde{v}$ are video demonstrations of two distinct tasks, and o and $\tilde{o}$ are agent observations in two distinct contexts. **Additional Demonstrations via Trajectory Synthesis** involves generating free-space motions for the robot in various contexts by sampling a small number of points (1 to 3) uniformly at random within the agent's workspace and moving the end effector sequentially through these points using an inverse kinematics solver. Training samples are created by pairing each trajectory with itself, requiring the model to focus on the motion of the arm and ignore background elements to make correct predictions.

$$v'_t = \alpha v_t + (1-\alpha)\tilde{v}_0 \quad o'_t = \alpha o_t + (1-\alpha)\tilde{o}_0 \tag{3.9}$$

Using this framework, the authors trained the same Transformer-

Table 3.4: Success rates achieved using the MOSAIC and AWDA methods. In this scenario, training is conducted on 5 out of the 6 tasks, with the remaining task reserved for testing.

Methods	Door	Drawer	Button	Blocks	Basketball	Nut Assembly
MOSAIC	**0.05**	**0.15**	**0.05**	0	0	0
AWDA	**0.10**	**0.29**	0.01	0	0	**0.02**

based architecture as in [6], achieving notable results. Specifically, on the pick-place task represented in Figure 3.16, AWDA reached a success rate of **100**% on two held-out variations, i.e., the system was trained on 14 variations and tested on the remaining 2. However, when tested on completely novel tasks, i.e., tasks never seen during training, the system struggled to succeed, as reported in Table 3.4.

In conclusion, significant research efforts have been dedicated to addressing the challenge of Visual Conditioned Multi-Task Imitation Learning (VC-MTIL). The aim is to develop systems capable of solving a given task in a zero-shot manner, starting from just a single video demonstration. Specifically, while these systems show promising results in solving new instances of seen tasks and unseen variations of known tasks, they struggle to handle a multi-task multi-variation setting, exhibiting a consistent drop in performance. Additionally, they face difficulties in solving entirely new tasks.

Furthermore, the work discussed in this section predominantly involves experiments conducted in simulation environments, leaving unanswered the question of whether these systems can be effectively applied in real-world settings, where demonstrations might also come in the form of human video demonstrations.

3.2 Problem formulation

As described in Section 3.1, this thesis primarily addresses the problem of Visual-Conditioned Multi-Task Imitation Learning. The goal is to train a single conditioned control function, $\pi_\theta(a_t|o_t^a, c_m)$, that can guide a robotic agent in solving both

variations of a given task and entirely different tasks. Where, the input consists of a command c_m, represented as a video demonstration of the requested task, along with the current observation of the agent o_t^a.

The approach proposed in this thesis is based on the observation that solving this problem involves two key tasks:

- *Command analysis*: This task involves solving a cognitive problem, where the system must interpret the high-level task command, understand the task intent, identify the relevant objects, and recognize the required actions.

- *Action generation*: This task involves solving a control problem, where the system must correlate the information from the command analysis with the agent environmental state to generate a valid action that moves the robot toward completing the requested task.

As demonstrated in the comprehensive review of related works (Section 3.1), the Visual-Conditioned MTIL problem is typically addressed using end-to-end architectures, which are trained with an action-centric behavioral cloning loss. While these systems are often able to control the robot and produce **reasonable trajectories** to complete tasks like pick-and-place, they may manipulate the wrong object, indicating a limitation in the cognitive ability to correctly identify the relevant object.

In this thesis, a modular approach is adopted. Specifically, Figure 3.20 illustrates the differences between a general end-to-end architecture and the proposed modular architecture. In the end-to-end architecture, the *Backbone Module*, which can be any deep learning architecture used in state-of-the-art methods, takes both the agent observation o_t^a and the command c_{m_i} as input. It generates an embedding z_t that must encapsulate all the necessary information for the *Control Module* to produce a valid action. This includes details derived from both the command, such as the position of the object of interest, the task being solved, its variations, as well as the state of the agent itself.

In contrast, the modular approach utilizes two backbone modules. The first, the *Command Analysis Backbone*, explic-

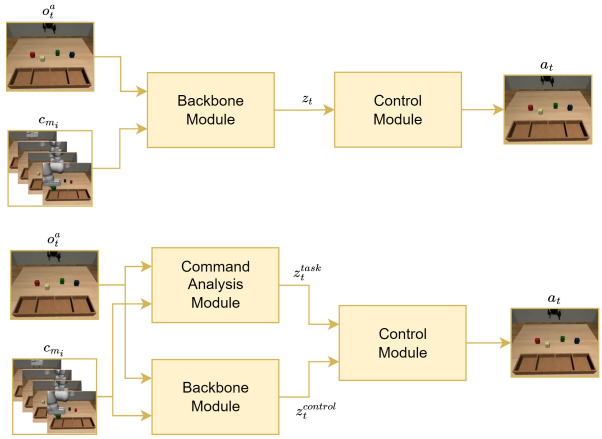


Figure 3.20: (Upper) General end-to-end architecture, where the *Backbone Module* takes as input both the agent observation and the command. It generates an embedding z_t that must contain information related to both the command and the control. (Bottom) In the modular architecture, there are two backbone modules: the *Command Analysis Module*, which generates the task-embedding z_t^{task}, and the *Backbone Module*, which is trained to generate only the control-embedding $z_t^{control}$.

itly solves the cognitive task, producing task-relevant information (z_t^{task}) such as the position of the object of interest. The second, the *Control Backbone Module*, generates a control embedding ($z_t^{control}$), which directly encodes information relevant to the action to be performed. In this modular approach, the *Control Module* takes both the task-relevant information (z_t^{task}) and the control embedding ($z_t^{control}$) as inputs.

The underlying assumption of this approach is that by separating the problem into two components, cognitive and control, and designing task-specific modules trained independently of each other, the system becomes more robust. For example, the Command Analysis Module is trained specifically for the cognitive task (e.g., the Conditioned Object Detection task described in Chapter 2), while the Control Backbone is trained for the control task. This separation allows the final Control Module to be informed by optimal embeddings generated by

modules trained on task-specific problems.

Section 2.2 provides an example of a cognitive problem that can be addressed by the Command Analysis Backbone. Here, the focus is on describing the problem solved in order to learn the final control policy π_θ. Specifically, the goal is to learn the parameters of the policy π_θ using a supervised-learning approach.

The first step is defining the dataset. As explained in Section 1.2.3.1, in multi-task, multi-variation scenario, there are n distinct tasks, denoted by $\mathcal{T} = \{T_1, T_2, \ldots, T_n\}$, where each task T_i is associated with a set of variations $\mathcal{M}_i$. For each task, a specific dataset $\mathcal{D}_i = ((c_m, \tau_m), m \in \mathcal{M}_i)$ is constructed, containing pairs of demonstrator videos c_m and corresponding target trajectories τ_m for each variation. The demonstrator video consists solely of visual observations, represented as $c_m = \left\{o_1^d, o_2^d, \ldots, o_{T'}^d\right\}$, while the target trajectory includes both observations and associated actions: $\tau_m = \{(o_1^a, a_1), \ldots, (o_T^a, a_T)\}$.

Building upon the complete dataset $\mathcal{D} = \{\mathcal{D}_1, \ldots, \mathcal{D}_n\}$, the optimal parameters θ^* are obtained by solving the minimization problem described in Formula 3.10, where the loss function $\mathcal{L}$ is minimized.

$$\theta^* = \arg\min_\theta \mathcal{L}(\pi_\theta, \mathcal{D}) \tag{3.10}$$

Section 3.3 will present the specific instance of the proposed architecture, along with the loss function and modules used. The results of the experiments will be discussed in Section 3.4.

3.3 Proposed Architecture

This section provides the effective implementation of the general modular architecture described in Section 2.2. Since the Command Analysis task being solved is Conditioned-Object Detection (Chapter 2), the focus here is on how the COD module is effectively integrated into the control framework.

The COD module is integrated into two different architectures, which vary in the number of control modules they use. Section 3.3.1 outlines an architecture that employs a single control module to predict actions for the entire trajectory. In

contrast, Section 3.3.2 describes an architecture that splits the control module into two distinct parts: one for computing actions during the reaching phase, and another for the final phase, where the specific primitive depends on the task.

3.3.1 Single control module

The architecture (Figure 3.21) composed of a single control module is essentially an instance of the general modular architecture depicted in Figure 3.20. Specifically, the Command Analysis Module is replaced by the CTOD module (Section 2.4.2.1), which takes as input the current agent observation o_t^a and the command c_m, producing a task embedding represented by the *target-object bounding box*, i.e., $z_t^{task} = bb_t^{target}$.

The Backbone Module, responsible for generating the control embedding $z_t^{control}$, is replaced by the same backbone used in MOSAIC. This backbone is a combination of a Convolutional Network, which encodes the agent observation o_t^a and the demonstration frames c_m, and a Self-Attention mechanism to create correlated agent and command embeddings (Section 1.2.3.1).

Finally, the Control Module follows the same implementation as in MOSAIC [8]. In this case, the actions generated by the control module are sampled from a multivariate logistic distribution (Equation 3.11), where the distribution parameters μ_i and σ_i are estimated by MLPs implementing the Control Module.

$$a_t \sim \sum_{i=1}^{m} \alpha(z_t)\, logistic(\mu_i(z_t), \sigma_i(z_t)) \tag{3.11}$$

Here, the embedding z_t is a concatenation of $z_t^{control}$, generated by the MOSAIC Backbone, and bb_t^{target}, produced by the CTOD Module.

3.3.2 Double control modules

Regarding the architecture composed of multiple control modules, the discussion must begin with the key observation that the tasks considered can be roughly divided into two phases.

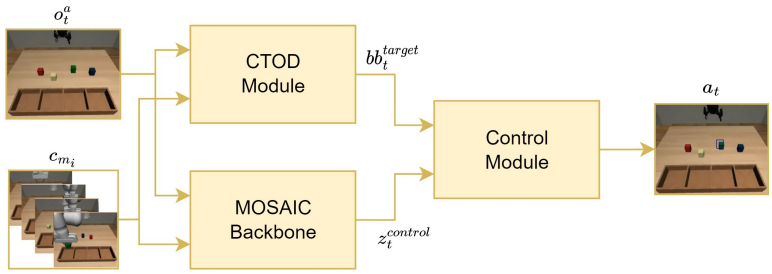


Figure 3.21: Proposed Single-Control Module Architecture. In contrast to the general architecture described in Figure 3.20, the Command Analysis module is replaced by the CTOD module, which generates the bounding box related to the target object. The chosen backbone is the MOSAIC architecture [8]. The control module is now informed by both low-level positional information (bb_t^{target}) and a control-oriented embedding ($z_t^{control}$), enabling it to make more informed decisions.

The first is generally a *reaching phase*, where the robot must reach a target location. The second phase varies based on the task: placing for Pick-Place, assembly for Nut-Assembly, stacking for Stack-Block, and pushing for Press-Button. Based on this, a dual control module architecture is proposed, with each control module trained to learn the primitive associated with each phase. The rationale behind this approach is that training modules specifically for simpler atomic primitives can result in a more robust and reliable control system.

Figure 3.22 illustrates the overall architecture. The main difference can be observed in the bounding boxes received by the control modules. Specifically, the MOSAIC Backbone generates the control embedding $z_t^{control}$, as before. However, the Command Analysis Module, now referred to as the COD Module, generates both the bounding box for the target object, bb_t^{target}, and the bounding box for the final placing position, bb_t^{place}.

The bb_t^{target} is provided as input to the *Reaching Control* module, while the bb_t^{place} is supplied to the *Placing Control* module.

Additionally, each control module operates based on an enabling signal s^{en}, which is set to 1 at the beginning of the

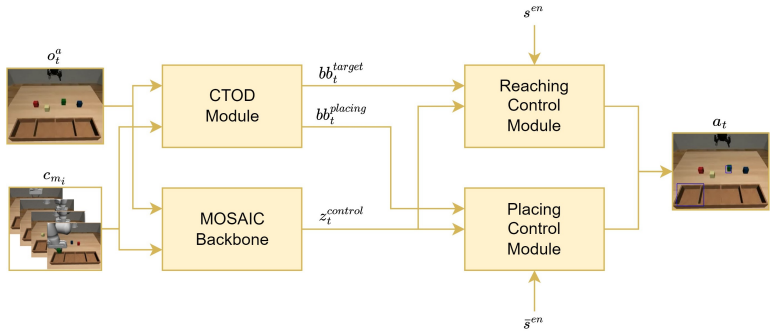


Figure 3.22: Proposed Double-Control Module Architecture. In this architecture, the Control Module is split into two distinct modules, each responsible for learning a specific primitive: the *reaching* primitive and the *placing* primitive. The first module takes as input the bounding box corresponding to the target object (bb_t^{target}), while the second module receives the bounding box related to the final placing location ($bb_t^{placing}$). This separation allows for specialized control during both the reaching and placing phases.

rollout and remains active until the Reaching Control module generates its first prediction for the closing command. After this point, s^{en} is set to 0, and control is transferred to the Placing Control module.

3.4 Experimental results

In this section, the performed experiments are going to be described. Specifically, in Section 3.4.1 the dataset used for training procedure will be described. Section 3.4.2 will report the obtained results.

3.4.1 Dataset

The dataset used for training the control architectures is described in this section. It consists of the same tasks outlined in Section 2.4.1, following the same collection procedure. For each task variation, 100 trajectories were collected for both the

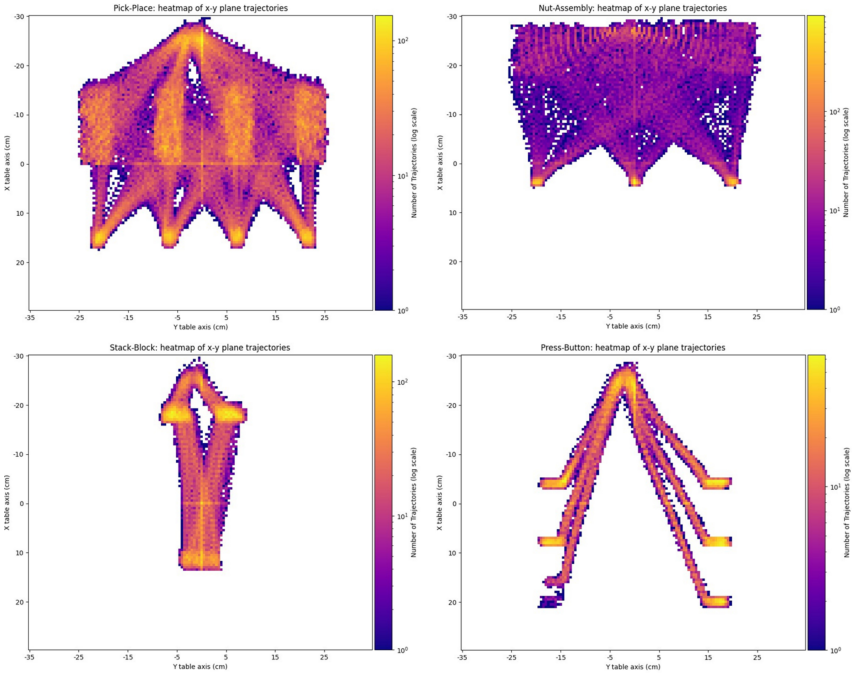


Figure 3.23: The distribution of trajectories for the different tasks along the x-y axis of the table workspace.

agent and the demonstrator. Each trajectory presents a novel object configuration, according to the rules defined in Section 2.4.1.

In this section, the dataset is analyzed in terms of the distribution of the action space, in order to highlight the challenges from a control perspective.

Figure 3.23 illustrates the distribution of the trajectories followed by the robot. Each plot represents a density map illustrating the frequency with which the robot's end-effector passed through specific points across all variations and trajectories. Notably, the use of a simulated environment enables complete coverage of the workspace. This particular aspect of the workspace coverage will be further discussed in Chapter 4 where the real-world system will be discussed.

Notably, each task exhibits a **multi-modal behavior**. This is evident across different variations, as the final placing position changes with each variation. Additionally, during the

reaching phase, the robot traverses almost all possible locations within the working area.

It is important to note that, for a given task, the trajectories exhibit significant variation in length. Specifically, for the Pick-Place task, the average trajectory length is 71.21 ± 7.69 frames. In the Nut-Assembly task, the average length is 62.24 ± 7.47 frames, while for the Stack-Block task, the average is 63.07 ± 1.51 frames. Finally, in the Press-Button task, the average trajectory length is 40.45 ± 8.17 frames.

This variability poses a challenge, as the likelihood of errors increases with task progression, and compounding errors become more pronounced over time. Additionally, this multi-modal behavior complicates the learning process. The dataset lacks any prior distribution (i.e., the robot does not consistently approach the target object from the same direction), making it challenging for the architecture to identify and leverage consistent patterns during training.

3.4.2 Results

This section presents the obtained results, divided into two main blocks. The first block (Section 3.4.2.1) discusses the results of the method described in Section3.3.1. The second block (Section 3.4.2.2) covers the results of the method described in 3.3.2. 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.

The tests are conducted following the procedure outlined in Section 2.4.2. For each variation, 10 independent runs are performed, each with a novel initial objects configuration. This approach assesses the system robustness with respect to different initial state configurations. Additionally, each set of 10 rollouts is repeated three times to evaluate the overall robustness and consistency of the model.

Various evaluation metrics are considered, either generally defined for each task or task-specific. The general metrics include:

- *Reaching*: The ratio of successful attempts where the

robot reaches the target object across all rollouts.

- *Picking*: The ratio of successful attempts where the robot picks the correct object across all rollouts.
- *Success*: The ratio of successful task completions across all rollouts.
- *Reaching Wrong*: The ratio of instances where the robot reaches an object other than the target across all rollouts.
- *Picking Wrong*: The ratio of instances where the robot picks the wrong object across all rollouts.
- *Success with Wrong Object*: The ratio of task completions where the robot manipulates the wrong object.

Task-specific metrics, particularly for tasks like Pick-Place and Nut-Assembly, include:

- *Place Wrong with Wrong Object*: The number of times the robot completes the task by placing the wrong object in the wrong position.
- *Place Wrong with Correct Object*: The number of times the robot completes the task by placing the correct object in the wrong position.

These metrics provide a comprehensive evaluation of the robot performance, capturing all major error cases and giving a full picture of its behavior.

3.4.2.1 Single control module

In this section the results obtained with the architecture composed of a Single Control Module will be described. As in Chapter 2, there are two testing scnearios, the first one trained with a single-task multi-variation scenario, while the second one with a multi-task multi-variation setting, with an increasing level of complexity.

Table 3.6: The single-task performance of the baseline methods, TOSIL [6] and MOSAIC [8], was evaluated. For each model, additional tests were conducted by generating the first 2 steps and 10 steps using the hand-written controller.

Task	Model	GT Action	Reaching [%]	Picking [%]	Success [%]	Reaching Wrong [%]	Picking Wrong [%]	Success with Wrong Object [%]
Pick-Place	MOSAIC	0	62.90±0.95	62.08±0.95	58.75±1.87	36.67±0.95	36.67±0.95	**37.71±0.72**
		2	89.17±1.30	88.12± 0.62	84.17± 1.57	11.25± 1.25	11.25± 1.25	11.46± 1.30
		10	99.79±0.36	98.54±0.72	**95.63±0.63**	1.25±0.63	1.25±0.63	0.41±0.36
	TOSIL	0	33.12±1.08	27.92± 0.72	26.87± 0.62	63.96± 2.36	63.75± 2.72	**66.04±1.57**
		2	69.17±1.30	60.83±2.52	59.38±2.17	29.17±0.95	28.96±1.30	30.83±2.20
		10	98.58±0.36	92.71± 1.44	**90.00±1.08**	2.70± 1.30	2.70± 1.30	1.25± 0.62
Nut-Assembly	MOSAIC	0	38.89±1.11	36.67±1.11	33.33±1.11	59.26±1.69	55.93±1.69	**51.48±2.31**
		2	85.19± 4.49	83.33± 5.09	78.89± 4.01	13.33± 4.84	11.85± 2.31	11.11± 2.22
		10	100.00±100.00	99.26±1.28	**90.74±2.79**	0.00±0.00	0.00±0.00	0.00±0.00
	TOSIL	0	36.30± 1.28	35.19± 2.31	31.11± 1.92	63.33± 1.11	62.59± 1.28	**56.30±3.57**
		2	83.33±2.22	82.96±2.31	77.78±2.22	16.67±2.22	16.67±2.22	14.44±1.11
		10	100.00± 0.00	99.26± 1.28	**88.89±2.22**	0.00± 0.00	0.00± 0.00	0.00± 0.00
Stack-Block	MOSAIC	0	60.56±0.96	60.56±0.96	53.33±1.66	39.44±0.96	39.44±0.96	**36.11±0.96**
		2	91.11± 0.96	90.56± 0.96	73.89± 4.19	7.70± 0.96	7.70± 0.96	7.20± 0.96
		10	100.00±0.00	99.44±0.96	**77.78±1.92**	0.00±0.00	0.00±0.00	0.00±0.00
	TOSIL	0	69.44± 0.96	60.00± 0.16	48.89± 2.54	42.78± 0.96	42.78± 0.96	**41.67±1.66**
		2	90.56±0.96	87.78±0.96	79.44±1.92	12.22±0.96	11.67±0.00	11.67±0.00
		10	100.00± 0.00	99.44± 0.96	**88.89±1.92**	1.66± 1.66	1.11± 0.96	0.00± 0.00
Press-Button	MOSAIC	0	100.00±0.00	-	**100.00±0.00**	0.00±0.00	-	0.00±0.00
		2	100.00± 0.00	-	100.00, 0.00	0.00± 0.00	-	0.00± 0.00
		10	100.00±0.00	-	100.00±0.00	0.00±0.00	-	0.00±0.00
	TOSIL	0	83.33± 1.66	-	**83.33±1.66**	17.78± 1.93	-	16.67± 1.67
		2	80.56±1.92	-	80.56±1.92	20.00±2.88	-	18.33±2.89
		10	92.22± 0.96	-	81.67± 3.33	9.44± 0.96	-	9.44± 0.96

Single-task multi-variation scenario

The discussion of the results begins with the single-task multi-variation scenario, focusing on the baseline methods TOSIL [6] and MOSAIC [8]. Specifically, Table 3.5 presents the performance of these baseline methods.

As observed, both baseline methods suffer from the issue of **target-object misidentification**. This is evident from the *Success with Wrong Object* column, where the success rate involving wrong objects is significant. Figure 3.24 provides an example of a pick-and-place rollout in which the task is technically completed, but the wrong object is manipulated.

To investigate the cause of these errors, test rollouts were performed using the first 2 and 10 actions generated by a hand-written policy, which has access to ground-state information. As noted, the success rate improves substantially by applying just 2 ground-truth actions. This supports the hypothesis outlined in Section 3.2, suggesting that the end-to-end architecture trained with an action-centric loss produces a suboptimal embedding z for the cognitive task. The embedding fails to suf-

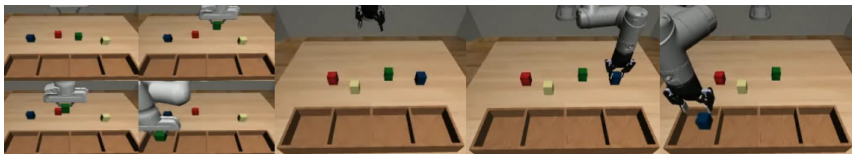


Figure 3.24: Example of task rollout with incorrect object manipulation. In this scenario, the robot successfully completes the task by placing an object in the first bin. However, instead of manipulating the correct object (the green box), the robot mistakenly picks up the blue box. This illustrates a situation where the robot executes the task's final action correctly but selects the wrong object during manipulation.

Table 3.7: The single-task performance of the proposed MOSAIC-CTOD module is compared to the MOSAIC and MOSAIC-GT-BB baselines. MOSAIC-GT-BB refers to the MOSAIC model, where the Control Module receives the ground-truth target bounding box as input.

Task	**Method**	**Reaching** [%]	**Picking** [%]	**Success** [%]	**Reaching Wrong** [%]	**Picking Wrong** [%]	**Success with Wrong Object** [%]
Pick-Place	MOSAIC	62.90±0.95	62.08±0.95	58.75±1.87	36.67±0.95	36.67±0.95	37.71±0.72
	MOSAIC-GT-BB	100.00± 0.00	97.71± 0.72	76.46± 3.20	0.00± 0.00	0.00± 0.00	0.00± 0.00
	MOSAIC-CTOD	98.12±0.62	91.88±4.88	**77.11±5.60**	1.04±0.36	1.04±0.36	1.04±0.36
Nut-Assembly	MOSAIC	38.89±1.11	36.67±1.11	33.33±1.11	59.26±1.69	55.93±1.69	51.48±2.31
	MOSAIC-GT-BB	100.00± 0.00	98.89± 1.11	**70.37±1.69**	0.00± 0.00	0.00± 0.00	0.00± 0.00
	MOSAIC-CTOD	98.89±1.11	97.41±2.31	64.07±0.64	0.00±0.00	0.00±0.00	0.00±0.00
Stack-Block	MOSAIC	60.56±0.96	60.56±0.96	53.33±1.66	39.44±0.96	39.44±0.96	36.11±0.96
	MOSAIC-GT-BB	100.00± 0.00	99.44± 0.96	90.00± 2.89	0.00± 0.00	0.00± 0.00	0.00± 0.00
	MOSAIC-CTOD	100.00±0.00	100.00±0.00	**91.67±2.88**	0.00±0.00	0.00±0.00	0.00±0.00
Press-Button	MOSAIC	100.00±0.00	-	**100.00±0.00**	0.00±0.00	-	0.00±0.00
	MOSAIC-GT-BB	92.22± 2.54	-	90.56± 1.92	3.88± 0.96	-	3.88± 0.96
	MOSAIC-CTOD	97.22±1.92	-	95.56±1.92	2.77±0.96	-	2.77±0.96

ficiently inform the control policy about the correct position of the target object.

Based on this consideration, the thesis proposal to inform the control module with both a low-level positional information (e.g., the bouding-box of the target object) and a control embedding generated by the MOSAIC-backbone has been tested. During this test, two models variations have been considered, the first named *MOSAIC-GT-BB* is basically the MOSAIC model, with the Control Module that receives in input the control-embedding $z^{control}$ and the ground-truth bouding-box. The second, named *MOSAIC-CTOD* is the architecture described in Section 3.3.1.

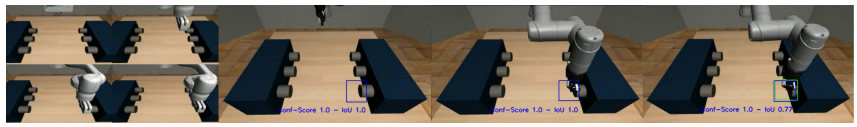


Figure 3.25: Example of unsuccessful Press-Button rollout. In this scenario, the robot successfully reaches the target button using the predicted bounding box (blue). However, due to instability in predictions during the pushing phase, the robot is unable to complete the pressing action, resulting in an unsuccessful task execution.

The results are summarized in Table 2.1. Several key observations can be made from these findings. For tasks involving multiple similar objects that change positions across different demonstrations (such as Pick-Place, Nut-Assembly, and Stack-Block), the use of positional information significantly enhances the system robustness. This enables the robot to consistently reach the target object and improves the overall success rate.

In contrast, for the Press-Button task, although the MOSAIC-CTOD method achieves a high success rate (95.56% on average), it does not overcome the baseline method, which consistently solves the task with a 100% success rate. This is because, once the button is reached, the positional information does not provide any additional guidance on how to complete the task. As a result, the robot tends to get stuck near the button, failing to execute the pushing action (Figure 3.25).

Furthermore, the inclusion of positional information introduces a novel type of error. Specifically, since the robot behavior is conditioned by the bounding box, any error in its prediction can cause the robot to move incorrectly (Figure 3.26). This error is significant, as in the Pick-Place task, the metric "Place Wrong with Correct Object" reaches 11.25%, and for the Nut-Assembly task, the same metric averages 22.22%.

To address this issue, the Double-Control Module architecture has been proposed (Section 3.3.2), with the corresponding results presented in Section 3.4.2.2.

Multi-task multi-variation scenario

As in the previous paragraph, an evaluation of the two base-

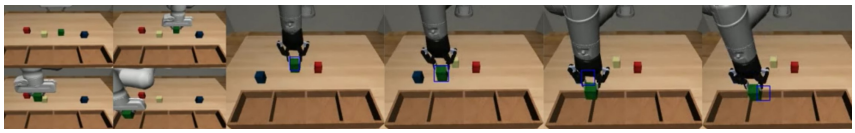


Figure 3.26: Example of unsuccessful Pick-Place rollout. In this case, the robot fails to complete the task due to errors in the bounding box predictions. These inaccuracies cause the robot to move in the wrong direction, leading to an unsuccessful execution of the task.

line methods was also conducted in the multi-task setting. The results are summarized in Table 3.8. The same general trends and behaviors observed in the single-task scenario are present here as well. Specifically, both baselines demonstrate the ability to produce reasonable trajectories that allow the robot to complete the task, though they manipulate the wrong object.

Furthermore, when comparing the results from Table 3.5 with those in Table 3.8, it is evident that the success rate decreases across all methods in the more complex multi-task setting. This highlights a potential issue with task balancing during the learning process.

After evaluating the baseline, the proposed MOSAIC-CTOD was tested, using the same variations as in the previous section. Table 3.9 summarizes the results obtained with the inclusion of positional information. Compared to the baseline, a general improvement is observed. However, caution is required when training the system in a multi-task setting. Specifically, when comparing the single-task performance (Table 3.7) to the multi-task performance, there is a noticeable drop in success rates for the same tasks. This decline is particularly evident in the system's ability to execute the "pick" primitive, especially when the nut object is involved. This observation underscores the importance of incorporating regularization techniques during multi-task learning to manage the varying complexities of different tasks.

Table 3.8: The multi-task performance of the baseline methods, TOSIL [6] and MOSAIC [8] was evaluated. For each model, additional tests were conducted by generating the first 2 steps and 10 steps using the hand-written controller.

Task	Model	GT Action	Reaching [%]	Picking [%]	Success [%]	Reaching Wrong [%]	Picking Wrong [%]	Success with Wrong Object [%]
Pick-Place	MOSAIC	0	25.83±1.30	23.96±0.72	22.71±0.72	65.63±2.87	63.96±1.90	**67.92±1.30**
		2	61.04±2.19	58.13±2.86	56.46±3.14	36.04±1.90	35.42±1.90	36.25±1.08
		10	97.77±0.36	95.00±0.62	**87.50±1.25**	2.50±0.62	2.29±0.95	2.29±0.72
	TOSIL	0	35.42±0.72	27.08±1.30	26.46±1.80	60.21±1.57	59.58±1.44	59.58±1.44
		2	62.57±0.64	48.19±1.26	48.06±1.26	31.88±3.13	31.81±3.13	37.08±2.82
		10	98.13± 0.63	86.18± 1.88	85.90± 2.19	1.88± 0.63	1.88± 0.63	1.88± 0.63
Nut-Assembly	MOSAIC	0	32.96±1.28	30.74±2.31	28.53±2.31	56.30±4.49	48.15±4.20	41.67±1.66
		2	76.67±1.11	73.33±1.92	65.56±4.00	19.63±0.64	17.41±1.28	17.04±1.69
		10	100.00±0.00	97.78±1.11	82.96±1.28	0.00±0.00	0.00±0.00	0.00±0.00
	TOSIL	0	27.78±2.94	24.44±2.94	22.59±3.57	70.37±2.57	67.78±2.22	**64.81±2.57**
		2	87.04±1.70	85.43±2.23	80.86±2.83	12.96±1.70	12.96±1.70	12.96±1.70
		10	100.00± 0.00	98.27± 0.57	**94.94± 0.57**	0.00± 0.00	0.00± 0.00	0.00± 0.00
Stack-Block	MOSAIC	0	57.78±1.92	57.78±1.92	55.56±2.54	42.22±1.92	42.22±1.93	41.11±1.67
		2	95.56±0.96	95.56±0.96	89.44±1.92	4.44±0.96	4.44±0.96	4.44±0.96
		10	100.00±0.00	100.00±0.00	**92.22±0.96**	0.00±0.00	0.00±0.00	0.00±0.00
	TOSIL	0	54.44±0.96	54.44±0.96	48.89±0.96	46.11±0.96	45.56±0.96	**45.56±0.96**
		2	87.59±0.85	87.59±0.85	74.81±1.70	12.41±0.85	12.41±0.85	12.41±0.85
		10	100.00± 0.00	100.00± 0.00	82.22± 4.19	0.00± 0.00	0.00± 0.00	0.00± 0.00
Press-Button	MOSAIC	0	78.33±1.66	-	77.22±2.54	22.78±2.54	-	22.78±2.54
		2	77.22±0.96	-	76.67±1.67	23.33±1.66	-	23.33±1.66
		10	82.78±2.54	-	**80.00±3.33**	20.00±3.33	-	20.00±3.33
	TOSIL	0	70.56±7.52	-	60.59±11.34	27.22±2.47	-	**27.22±2.47**
		2	74.44±2.55	-	60.37±4.24	25.56±2.55	-	25.56±2.55
		10	75.93± 2.51	-	60.00± 1.67	25.74± 0.85	-	25.74± 0.85

3.4.2.2 Double control modules

In this section the results obtained with the Double Control Module (Section 3.3.2) are going to be discussed.

Single-task multi-variation scenario

As discussed in Section 3.3.2, the introduction of the Double Control Module (DCM) was motivated by the need to reduce errors associated with incorrect object placement. These errors arise from inaccurate bounding-box predictions, which can lead to error propagation.

To address this issue, the architecture of the Double Control Module evolved through incremental design iterations and validation steps. Specifically, the process started by observing that the positional information given by the bouding-box is usefull only during the reaching phase, while it becomes useless after it. For this reason, the training of the MOSAIC-CTOD has been changed, setting to zero the bounding-box after the picking. Then, observing the obtained results and the error cases, the Double Control Module has been realized.

Table 3.9: The multi-task performance of the proposed MOSAIC-CTOD module is compared to the MOSAIC and MOSAIC-GT-BB baselines. MOSAIC-GT-BB refers to the MOSAIC model, where the Control Module receives the ground-truth target bounding box as input.

Task	Method	Reaching [%]	Picking [%]	Success [%]	Reaching Wrong [%]	Picking Wrong [%]	Success with Wrong Object [%]
Pick-Place	MOSAIC	25.83±1.30	23.96±0.72	22.71±0.72	65.63±2.87	63.96±1.90	67.92±1.30
	MOSAIC-GT-BB	100.00±0.00	89.58±2.19	**58.33±0.90**	0.00±0.00	0.00±0.00	0.00±0.00
	MOSAIC-CTOD	80.21±1.44	67.50±0.62	53.33±1.90	4.16±0.71	3.54±0.36	0.37±0.64
Nut-Assembly	MOSAIC	32.96±1.28	30.74±2.31	28.53±2.31	56.30±4.49	48.15±4.20	41.67±1.66
	MOSAIC-GT-BB	99.63±0.64	91.48±2.31	**37.04±5.70**	0.00±0.00	0.00±0.00	0.00±0.00
	MOSAIC-CTOD	68.69±1.11	49.63±3.30	33.33±4.00	11.48±1.69	5.55±2.94	0.00±0.00
Stack-Block	MOSAIC	57.78±1.92	57.78±1.92	55.56±2.54	42.22±1.92	42.22±1.93	41.11±1.67
	MOSAIC-GT-BB	94.44±5.85	91.11±6.73	73.89±5.00	10.00±2.88	0.00±0.00	0.00±0.00
	MOSAIC-CTOD	97.92±2.09	96.67±2.35	**87.92±2.00**	1.25±0.83	0.41±0.83	0.00±0.00
Press-Button	MOSAIC	78.33±1.66	-	77.22±2.54	22.78±2.54	-	22.78±2.54
	MOSAIC-GT-BB	100.00±0.00	-	**98.33±0.00**	0.55±0.96	-	0.55±0.96
	MOSAIC-CTOD	92.78±0.96	-	91.11±2.54	3.89±4.19	-	3.88±4.19

Table 3.10 summarizes the obtained results, from which several important considerations can be made.

Firstly, it is noteworthy that removing the bounding box after picking reduces the error percentage for the *Place Wrong Correct Object* category to 0.00%, confirming the earlier observation regarding the noisy nature of the predicted bounding box after the picking phase. However, this modification alone does not lead to a general improvement in the overall success rate. In fact, the *MOSAIC-CTOD (no-bb after pick)* model has a lower picking rate. This occurs because the control module issues the closing command too early, creating an out-of-distribution sample, which drives the robot into a novel state that was not encountered during training, making it unable to complete the task (Figure 3.27).

Regarding the performance of the proposed *MOSAIC-COD* module, it is notable that it achieves the highest success rate in 3 out of 4 tasks. Specifically, for tasks prone to placing the correct object in the wrong position, such as Pick-Place and Nut-Assembly, this type of error is eliminated, resulting in an overall improvement in the success rate.

Moreover, a novel variant of the MOSAIC baseline, *MOSAIC-DP*, has been implemented. This model follows the classic MOSAIC architecture proposed in [8] but is equipped with two control modules: one for the reaching primitive and another

Table 3.10: MOSAIC-COD results obtained in the single-task setting. The model is compared to baselines such as MOSAIC-CTOD (Section 3.4.2.1), a modified version of MOSAIC-CTOD that does not receive the predicted bounding box after picking, and MOSAIC-DP, which is the MOSAIC architecture proposed in [8], but with two control modules.

Task	Model	Reaching [%]	Picking [%]	Success [%]	Place Wrong Correct Object [%]
Pick-Place	MOSAIC-CTOD	98.12±0.62	91.88±4.88	77.11±5.60	**11.25±1.08**
	MOSAIC-CTOD (no-bb after pick)	95.69±1.88	83.89±3.45	82.36±3.13	0.00±0.00
	MOSAIC-DP	36.60±0.32	28.82±1.26	27.36±1.59	0.00±0.00
	MOSAIC-COD	100.00±0.00	93.75±0.62	**93.33±0.72**	0.00±0.00
Nut-Assembly	MOSAIC-CTOD	98.89±1.11	97.41±2.31	64.07±0.64	**22.22±2.93**
	MOSAIC-CTOD (no-bb after pick)	81.76±0.58	64.44±5.09	58.15±0.15	0.00±0.00
	MOSAIC-DP	30.62±0.57	25.06±0.57	23.21±1.13	0.00±0.00
	MOSAIC-COD	99.63±0.64	85.19±4.60	**81.11±3.84**	0.00±0.00
Stack-Block	MOSAIC-CTOD	100.00±0.00	100.00±0.00	91.67±2.88	-
	MOSAIC-CTOD (no-bb after pick)	92.59±2.51	85.74±0.85	76.30±5.04	-
	MOSAIC-DP	67.22±2.55	54.26±0.85	51.48±1.70	-
	MOSAIC-COD	100.00±0.00	100.00±0.00	**95.00±1.66**	-
Press-Button	MOSAIC-CTOD	97.22±1.92	-	**95.56±1.92**	-
	MOSAIC-CTOD (no-bb after pick)	81.85±3.35	-	77.04±3.39	-
	MOSAIC-DP	88.70±3.39		82.96±5.89	-
	MOSAIC-COD	100.00±0.00	-	91.11±1.92	-

for the primitive used in the final part of the task (i.e., placing, assembly, stacking, or pushing, respectively).

It is important to note that the performance of this variation is very similar to the original MOSAIC baseline, even though the control problem is simplified by splitting it into two phases corresponding to different primitives. Specifically, the most significant error in this case also stems from manipulating the wrong object. The success rates with the wrong object are 54.38%, 46.67%, 46.67%, and 7.59% for the Pick-Place, Nut-Assembly, Stack-Block, and Press-Button tasks, respectively.

This further supports the central thesis, which suggests that the end-to-end architecture struggles to create an optimal embedding that addresses both cognitive and control tasks effectively.

Multi-task multi-variation scenario

Given the results and observations from the previous section, the MOSAIC-COD module is directly compared to the base-

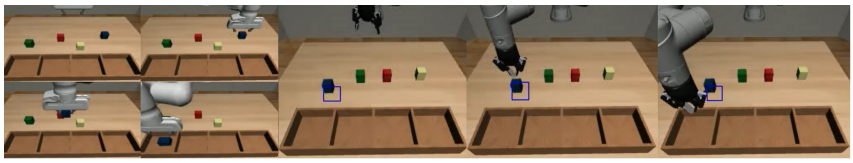


Figure 3.27: Example of an unsuccessful Pick-Place operation using the "no-bb after pick" variant. In this scenario, the robot successfully reaches the target box based on the predicted bounding box (blue). However, the single-control module prematurely predicts the closing command, preventing the robot from correctly picking up the object.

Table 3.12: MOSAIC-COD results obtained in the multi-task setting. The model is compared with MOSAIC and MOSAIC-CTOD models.

Task	**Model**	**Reaching** [%]	**Picking** [%]	**Success** [%]
Pick-Place	MOSAIC	25.83±1.30	23.96±0.72	22.71±0.72
	MOSAIC-CTOD	80.21±1.44	67.50±0.62	53.33±1.90
	MOSAIC-COD	100.00±0.00	94.79±0.90	**89.58±3.55**
Nut-Assembly	MOSAIC	32.96±1.28	30.74±2.31	28.53±2.31
	MOSAIC-CTOD	68.69±1.11	49.63±3.30	33.33±4.00
	MOSAIC-COD	99.63±0.64	78.15±2.31	**70.74±4.49**
Stack-Block	MOSAIC	57.78±1.92	57.78±1.92	55.56±2.54
	MOSAIC-CTOD	97.92±2.09	96.67±2.35	**87.92±2.00**
	MOSAIC-COD	98.33±0.00	98.33±0.00	85.00±5.00
Press-Button	MOSAIC	78.33±1.66	-	77.22±2.54
	MOSAIC-CTOD	92.78±0.96	-	**91.11±2.54**
	MOSAIC-COD	83.89±2.55	-	71.67±2.88

lines MOSAIC and MOSAIC-CTOD in the Multi-Task setting. Table 2.4 summarizes the results. As can be seen, the use of specialized models trained for the two task phases, reaching and placing, leads to a system that consistently picks the target object. This improvement is particularly evident in the more complex tasks, such as Pick-Place and Nut-Assembly. The enhanced picking rate demonstrates that having accurate information about the target object position enables correct target acquisition, while the presence of two control modules allows for training specific controllers for simpler primitives.

However, despite the improvements observed in Pick-Place and Nut-Assembly, MOSAIC-COD has the lowest overall suc-

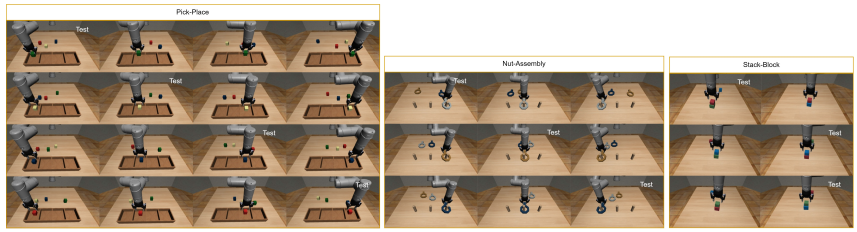


Figure 3.28: The dataset used for generalization tests removes one variation from each set of variations for a given target object.

cess rate. This can be attributed to two main factors. First, the Press-Button task is substantially different from the other three tasks. Unlike the others, it does not involve pick-and-place primitives, making it an out-of-distribution task. Second, the instability of the bounding boxes produced can lead to undesirable behaviors, such as sudden movements or freezing.

3.4.2.3 Proprioceptive state and Generalization tests

In this section, additional experiments are described, focusing on two key objectives.

The first set of experiments aims to determine whether incorporating proprioceptive information can enhance the system's robustness. This is particularly important for tasks requiring fine manipulation, such as the Nut-Assembly task, where the goal is to reduce errors during the assembly phase, specifically avoiding instances where the nut hits the peg.

The second set of experiments explores the generalization capabilities of the proposed system. Specifically, the goal is to assess whether the system can generalize to previously unseen task variations. To evaluate this, the system is trained on a subset of task variations selected as follows: for a set of variations involving a specific target object (e.g., a green box), one variation is excluded (e.g., placing the box into the first bin). However, in other sets involving different objects, the variation that requires placing the object into the first bin is retained. This setup allows for testing how well the system can transfer knowledge across different variations. By doing so, it becomes

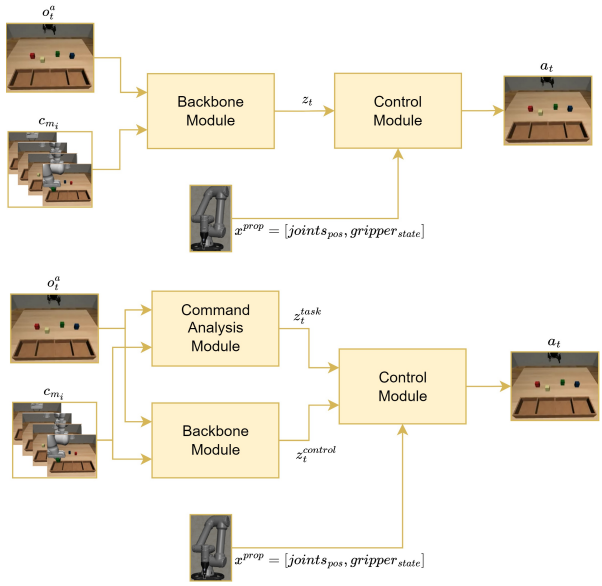


Figure 3.29: Proprioceptive information is integrated in both the end-to-end architecture (top) and the modular architecture (bottom). The proprioceptive vector, x^{prop}, is constructed from the robot's six continuous joint positions and the binary gripper state.

possible to evaluate whether the system can achieve robust performance with less training data, eliminating the need to collect every possible variation for each object (Figure 3.28).

Proprioceptive information

The proprioceptive information selected for the four tasks considered includes the joint positions, $joints_{pos} \in \mathcal{R}^6$ and the gripper state, $gripper_{state} \in [0, 1]$. This vector of elements, x^{prop}, is provided as input directly to the Control Module, as shown in Figure 3.29.

Specifically, for each baseline method, *MOSAIC*, *MOSAIC-CTOD*, and *MOSAIC-COD*, a version of the model incorporating proprioceptive information was trained. Table 3.13 presents the success rates for both the Single-Task and Multi-Task scenarios.

It is important to note that for 3 out of 4 tasks, the best

Table 3.13: The results obtained by integrating proprioceptive information in both Single-Task and Multi-Task scenarios. For each baseline model, the corresponding version that includes the proprioceptive state (P) was trained and tested.

Task	**Model**	**Success (Single-Task)** [%]	**Success (Multi-Task)** [%]
Pick-Place	MOSAIC	58.75±1.87	22.71±0.72
	MOSAIC-P	12.84±0.31	7.26±1.75
	MOSAIC-CTOD	77.11±5.60	53.33±1.90
	MOSAIC-CTOD-P	87.84±2.43	68.29±3.80
	MOSAIC-COD	93.33±0.72	**89.58±3.55**
	MOSAIC-COD-P	**96.04±0.56**	58.93±8.76
Nut-Assembly	MOSAIC	33.33±1.11	28.53±2.31
	MOSAIC-P	13.83±0.57	9.63±0.64
	MOSAIC-CTOD	64.07±0.64	33.33±4.00
	MOSAIC-CTOD-P	95.06±0.56	38.14±3.40
	MOSAIC-COD	81.11±3.84	**70.74±4.49**
	MOSAIC-COD-P	**95.06±0.57**	53.83±1.44
Stack-Block	MOSAIC	53.33±1.66	55.56±2.54
	MOSAIC-P	29.07±2.50	13.33±5.77
	MOSAIC-CTOD	91.67±2.88	**87.92±2.00**
	MOSAIC-CTOD-P	91.11±6.90	72.22±2.55
	MOSAIC-COD	95.00±1.66	85.00±5.00
	MOSAIC-COD-P	**96.67±1.66**	22.77±1.93
Press-Button	MOSAIC	**100.00±0.00**	77.22±2.54
	MOSAIC-P	65.00±3.33	42.00±2.65
	MOSAIC-CTOD	95.56±1.92	**91.11±2.54**
	MOSAIC-CTOD-P	86.66±6.66	50.33±0.58
	MOSAIC-COD	91.11±1.92	71.67±2.88
	MOSAIC-COD-P	72.03±3.39	79.33±5.13

performance is achieved by combining the Double-Policy approach with proprioceptive information, effectively solving the manipulation tasks of Pick-Place, Nut-Assembly, and Stack-Block in a robust manner. The greatest improvement is seen in the Nut-Assembly task, where the proprioceptive information resolves issues related to collisions and robot freezing.

However, this improvement is not observed in the Press-Button task, where a 35% drop in success rate is seen, even with the *MOSAIC* baseline. In this task, the main failure cases are related to unstable robot behavior. Additionally, in variations focused on object orientation, the robot often gets stuck when correctly approaching the button.

Notably, the same behavior is not observed in the Multi-Task scenario. In fact, the highest performance is achieved by methods that exclude proprioceptive information. Specifically, the introduction of proprioceptive data appears to exacerbate the imbalance between tasks, hindering the system from effectively utilizing this additional information during testing, which leads to instability.

For example, in the case of the MOSAIC-CTOD-P module, the robot successfully picks the target object only 58.87% of the time. This occurs because the closing command is sent too early, relative to the object position. Furthermore, the success rate drops further to 38.14%, as the robot, despite reaching the target peg, collides with it, an issue where proprioceptive information should theoretically provide assistance, especially since the pegs are fixed.

This instability is most evident in the Stack-Block task, where the success rate falls to 22.77% for the MOSAIC-COD-P module, despite the robot picking the target object with an average rate of 80.00%. The errors in this case are due to the object being dropped during stacking, caused by imprecise placement, as well as instances where the robot becomes completely stuck.

These results highlight the need for a balanced learning procedure in the context of Multi-Task Learning, as discussed earlier in Section 3.4.2.1.

Table 3.15: The results obtained by testing the models on unseen variations.

Task	Model	Reaching (Single-Task) [%]	Success (Single-Task) [%]	Reaching (Multi-Task) [%]	Success (Multi-Task) [%]
Pick-Place	MOSAIC	35.83±2.82	31.67±3.82	32.50±2.50	20.83±5.70
	MOSAIC-CTOD	100.00±0.00	87.50±4.30	100.00±0.00	63.33±5.20
	MOSAIC-COD	100.00±0.00	**89.17±2.89**	100.00±0.00	**94.17±2.88**
Nut-Assembly	MOSAIC	22.22±1.93	13.33±3.33	27.78±3.85	18.88±3.84
	MOSAIC-CTOD	100.00±0.00	38.89±6.94	100.00±0.00	51.11±1.92
	MOSAIC-COD	98.89±1.93	**57.78±5.09**	100.00±0.00	**90.00±3.33**
Stack-Block	MOSAIC	58.89±1.93	53.33±3.33	60.00±5.77	56.66±3.33
	MOSAIC-CTOD	100.00±0.00	94.44±6.94	100.00±0.00	**88.88±5.00**
	MOSAIC-COD	100.00±0.00	**97.78±1.93**	100.00±0.00	72.20±5.09

Generalization

Regarding the generalization tests, we evaluated only the three tasks where the previously explained variation selection was applicable. Specifically, the following variations were removed for each task (Figure 3.28):

- **Pick-Place**: Variations 1, 6, 11, and 16 were excluded.
- **Nut-Assembly**: Variations 1, 5, and 9 were excluded.
- **Stack-Block**: Variations 1, 4, and 6 were excluded.

The models were trained on the remaining variations and tested on the excluded ones following the same procedure.

Table 3.15 summarizes the success rates obtained for both the Single-Task and Multi-Task settings. Notably, with the introduction of the CTOD and COD modules, the system consistently reaches the correct target object in unseen variations. Similarly, the robot is able to pick and complete tasks, outperforming the MOSAIC baseline across all tested tasks.

This is a promising result, as it demonstrates that the system can effectively share knowledge across different variations, enabling robust training with less data.

3.5 Conclusion

In conclusion, this chapter introduced and elaborated on the Object Conditioned Control Policy (OCCP), a modular control architecture designed to integrate the Conditioned Object

Detector. This integration allows the control policy to utilize low-dimensional positional information about the target object location, while the Control Module Backbone generates an embedding containing task-relevant information. This embedding is derived from gradients obtained through the backpropagation of an action-centric loss function, thereby addressing or mitigating the target-object misidentification issues observed in baseline models.

Two specific control architectures were proposed: MOSAIC-CTOD, which integrates the Conditioned Target Object Detector, and MOSAIC-COD, which incorporates the Conditioned Object Detector. The latter generates both the bounding box of the target object and the bounding box of the target position of interest.

These architectures were validated in a simulation environment under both single-task multi-variation and multi-task multi-variation scenarios. Results demonstrated that incorporating object-related information enabled the system to consistently reach and pick the target object, leading to successful task completion.

In the single-task scenario, MOSAIC-COD achieved the highest average success rate (**90.13%**), representing a +28.77% improvement over the MOSAIC baseline. In the multi-task scenario, MOSAIC-COD achieved an average success rate of **79.24%**, improving by +33.23% compared to the baseline. The most significant improvements from object conditioning were observed in tasks involving "pick-place" primitives, such as Pick-Place, Nut-Assembly, and Stack-Block. However, a general performance drop was noted in the Press-Button task, where variations in bounding-box predictions led to unstable robot behaviors.

Additional notable improvements were observed when proprioceptive information (e.g., joint positions and gripper state) was integrated. Specifically, the Nut-Assembly task, which requires precise, contact-rich manipulation, saw the highest gains. For MOSAIC-CTOD-P, the average success rate reached **95.06%**, marking an improvement of +30.99% over the version without proprioceptive data. This success was attributed to the fixed positions of the target pegs, where joint position in-

formation helped the robot avoid collisions and correctly insert the pegs.

Moreover, the proposed architecture demonstrated zero-shot generalization to novel variations, with MOSAIC-COD achieving an average success rate of **81.57%** in the single-task scenario and **85.45%** in the multi-task scenario. This represents improvements of +48.79% and +53.32% over the MOSAIC baseline, respectively.

Overall, the introduction of inferred object priors led to significant improvements over the baseline. However, further refinements in the learning process, particularly in multi-task scenarios, are needed. Specifically, future work should account for the varying difficulties of different tasks in these scenarios.

Chapter 4

Experimental validation in real-world scenario

This chapter details the validation of the proposed methods in a real world scenario. Specifically, Section 4.1 describes the experimental setup. Section 4.2 discusses the dataset used to train the system. Finally, Section 4.3 presents the results obtained.

4.1 Experimental Setting

In this section, the experimental setup is explained and defined. Figure 4.1 illustrates the workspace where the robot operates, and compares it to the corresponding simulation environment. As can be observed, the simulation environment closely resembles the real-world counterpart. Both environments consist of the same robot agent, with identical camera configurations and workspace setup.

Specifically, the experimental setup includes:

- The Universal Robots UR5e robot [137], equipped with the Robotiq 2F-85 gripper [138], which acts as the agent.
- Four Zed-Mini stereo cameras [139]: one camera is mounted on the gripper, while the remaining three are positioned around the robot to ensure complete coverage of the workspace.
- A 100×100 cm working table.

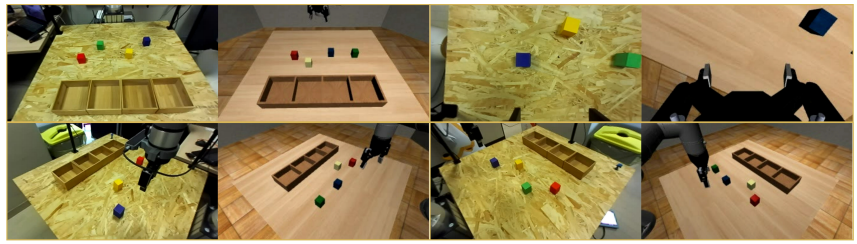


Figure 4.1: Workspace comparison between real-world (left) and simulated (right) scenarios. Images are taken from the frontal camera (Top-Left), the gripper camera (Top-Right), lateral-left camera (Bottom-Left) and lateral-right (Bottom-Right).

The reason for maintaining a high similarity between the real-world and simulation environments is to evaluate the potential for pre-training the model on a large and comprehensive simulation dataset, followed by fine-tuning on a smaller and incomplete real-world dataset. This approach allows leveraging the advantages of simulation for initial training while adapting the model to real-world conditions with minimal additional data.

4.2 Dataset

For the real-world validation, the focus will be on testing the model on the **multi-variation** *Pick-Place* tasks, focusing on a smaller set of variations compared to the complete 16 variations described in Section 2.4.1.

Figure 4.2 represents the 6 variations used in the real-world validation. As can be noted, these are essentially the first six variations of the Pick-Place Task used in the simulation environment.

A preliminary dataset composed of **40 trajectories** for each variation has been collected by teleoperating the robot with a console controller. Regarding the object placement, the same algorithm described in Section 3.4.1 was applied. This means that the set of 4 bins ($15 \times 15 \times 7$ cm) was fixed in position, while the 4 boxes ($4 \times 4 \times 6$ cm) could vary in their

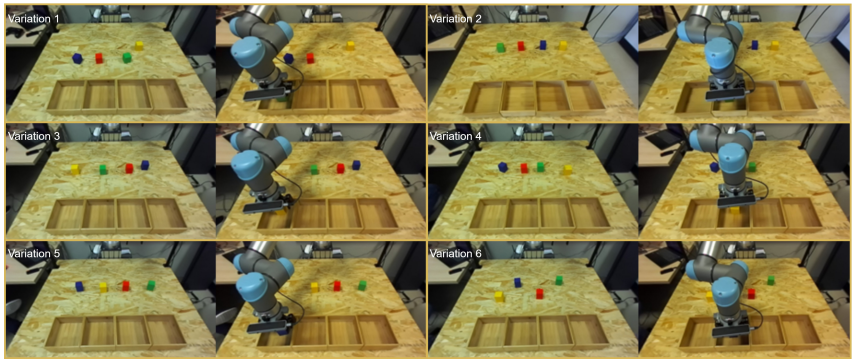


Figure 4.2: Set of variations used in the real-world robot evaluation. For each variation, the first and last frames are provided

position within a region of 60 cm in length and 15 cm in height.

In this extensive placement area, a specific protocol was implemented for collecting the trajectories. The protocol dictates how the objects are positioned within the picking region. The various placements of the target object are illustrated in Figure 4.3. Although the workspace may appear somewhat limited, this dataset is the first real-world dataset in the context of Visual-Conditioned Imitation Learning to encompass such a large operational space. In contrast, other works are either restricted to simulation environments [6, 8] or present very simple scenarios where objects are placed in a much smaller area [5]. Trajectories are collected through teleoperation the robot is controlled in its operational space, with the controller that sends continous velocity commands on a specific axis, this velocity command is defined by the value read by the controller's potentiometers. During teleoperation different informatin are recorded:

- *Images* from the front, laterals and gripper cameras, both RGB and Depth images are recorded.

- *Proprioceptive information* like joints positions and velocities.

- *Trajectory state*, which is manually changed by the human operators based on the task state. The phases are:

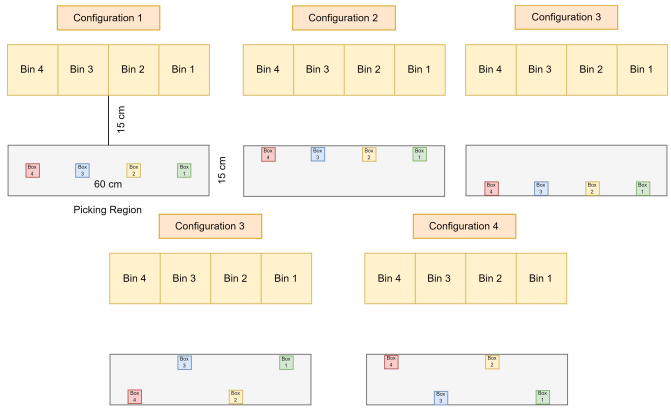


Figure 4.3: Placement configuration used for the trajectory collection. For each placement configuration, 4 trajectories were collected. The target object initially starts at the rightmost position, and in each subsequent trajectory, the box is moved to the adjacent position. This process is then repeated twice, each time with a different object orientation, resulting in a total of 40 trajectories for a given configuration.

1. *Start*, this phase starts at the beginning of the trajectory till the robot gripper is perpendicular to the target object.
2. *Approaching*, this phase starts when the robot approaches the target object with its discending movement.
3. *Picking*, this phase is characterized by the gripper that is ready to be closed to pick the object, and contains the closing command.
4. *Moving*, this phase starts after the robot picks the object and lift it from the table, this phase ends when the gripper is perpendicular to the target bin.
5. *Placing*, this phase starts when the robot is ready to place the object, starting the discending phase towards the target bin, this phase also contains the opening command.

- *Objects bounding boxes*, which are are generated automatically, requiring minimal input from human opera-

tors. At the start of the trajectory, the operator needs to specify the position of each object in the scene, including both boxes and bins. This is done by displaying the first frontal frame of the trajectory and clicking on the objects with the cursor. The positions, initially defined in discrete pixel-space, are then converted into continuous world-space through a series of transformations, using the camera's intrinsic and extrinsic parameters. The extrinsic parameters are obtained via a calibration procedure that uses ARUCO markers.

The overall space coverage of the real-world dataset is shown in Figure 4.4. As can be observed, the coverage is more limited and considerably noisier compared to the simulated dataset, even when restricted to the same variations. This is due to the collection of fewer trajectories and the use of teleoperation without any hand-written control rules, which typically generate smoother and more deterministic robot behaviors. These limitations introduce additional challenges in learning a robust control policy, especially for generalizing to different placements of the target object. This issue will be addressed in this thesis by initially training the policy in the simulation environment, where a complete dataset is available, and then fine-tuning it on the noisy real-world dataset.

4.3 Results

In this section the results obtained with the real-world experiments are going to be discussed. Specifically, the focus is on evaluating the proposed conditioned object detectors, which were described in Chapter 2, and the proposed object conditioned control policies, which were described in Chapter 3.

4.3.1 Conditioned object detector validation

In this section, the validation of the proposed C(T)OD modules is going to be discussed. To validate this model also in

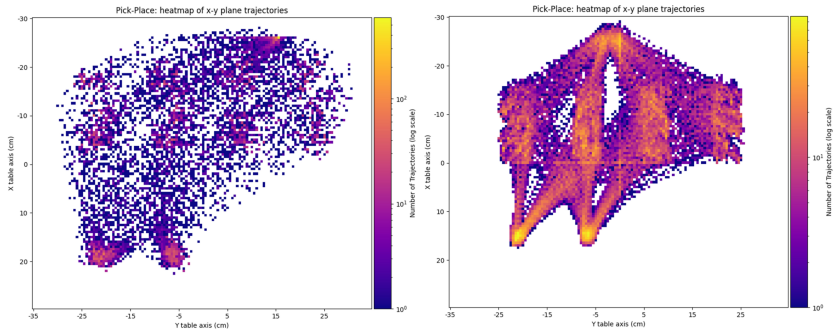


Figure 4.4: (Left) Trajectory distribution along the x-y axis of the real-world dataset. (Right) Trajectory distribution along the x-y axis of the simulated dataset, constrained to the same variations and number of trajectories as the real-world counterpart. It can be observed that the real-world dataset exhibits a much sparser and noisier distribution, due to the fact that the trajectories are collected via teleoperation.

a real-world scenario, the same metric used for the simulated counterpart are used (Section 2.4.2).

Table 4.1 and Table 4.2 summarize the obtained performance of the both the CTOD and the COD modules respectively, trained in two different settings, the first one training the module from scratch, the second one by finetuning the model obtained with the dataset collected in the simulation environment (Section 2.4.1). These models were trained with a dataset divided according to the 90-10 rules, and the results refers to the Precision, Recall and Average IoU obtained by the best model on the Validation set.

As noted, in this case, the system consistently assigns a bounding box to the target classes, with Recall always equal to 1.00. However, for the CTOD module, compared to the simulated model performance (Table 2.1) a drop in performance is observed. Specifically, Precision@0.5 for the CTOD module decreases from 0.770 to 0.640, for the model trained from scratch. Instead, a further drop is observed for the finetuned model, where the Precision@0.5 drops to 0.604.

To better understand the drop in performance, it is crucial to analyze the error distribution. For the CTOD module, out of

Table 4.1: Results of the CTOD module obtained in the real-world scenario. Performances are reported in terms of *Precision* (Prec), *Recall* (Rec) with an IoU threshold of 0.5 and the Average IoU (IoU_{avg}), for both the modal trained from scratch and the finetuned.

Task	**Precision@0.5**	**Recall@0.5**	**IoU$_{avg}$**
Pick-Place (scratch)	0.640	1.00	0.563
Pick-Place (finetuned)	0.604	1.00	0.535

Table 4.2: Results of the COD module obtained in the real-world scenario. Performances are reported in terms of *Precision* (Prec), *Recall* (Rec) with an IoU threshold of 0.5 and the Average IoU (IoU_{avg}), for both the modal trained from scratch and the finetuned.

Task	**Precision@0.5**	**Recall@0.5**	**IoU$_{avg}$** (target)	**IoU$_{avg}$** (target-place)
Pick-Place (scratch)	0.725	1.00	0.444	0.916
Pick-Place (finetuned)	0.758	1.00	0.498	0.919

a total of **24,660** frames, there are **8,867/9,771** (scratch/fine-tuned) false positives. Specifically, **2,599/3,340** occur during the reaching phase, and **6,268/6,431** occur during the placement phase, i.e., after the picking action.

This increase in errors is due to several factors, with **occlusion** being a primary cause. Since the physical robot has a camera mounted on the gripper, the object can become partially occluded during motion, leading to detection difficulties. Furthermore, the automated bounding-box generation procedure can introduce noise, particularly during movement, due to errors in camera calibration and the physical measurement of the robot base link position relative to the center of the table (where the ARUCO marker is placed for calibration).

Conversely, the COD module exhibits a different behavior. Specifically, Precision@0.5 improves for both the fine-tuned

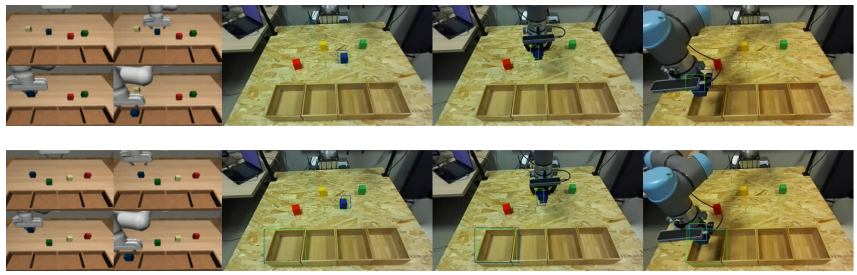


Figure 4.5: (Top Row) Example of CTOD prediction. (Bottom Row) Example of COD prediction. The images illustrate one of the less accurate predictions. While the system successfully identifies the target (blue bounding box), the prediction is less precise compared to the ground truth (green bounding box), particularly for the target object. As shown, the offset error increases during placement, likely due to the object being occluded by the gripper. Additionally, some inaccuracies in the ground truth are caused by the automatic generation process.

model and the model trained from scratch. When comparing the COD module trained in simulation (Table 2.3) with the real-world version, Precision@0.5 increases from 0.667 to 0.725 for the model trained from scratch and to 0.758 for the fine-tuned model.

It is also important to examine the distribution of false positives. With the same number of validation frames (**24,660**), there are a total of **13,577/11,915** false positives for the target-object class in the scratch/fine-tuned models, respectively, and **0/7** false positives for the target-place class. Additionally, the false positives are unevenly distributed between phases: **5,207/4,093** false positives occur during the reaching phase (before the pick), while **8,370/7,822** occur during the placing phase (after the pick). This imbalance in false positives is primarily caused by occlusions and inherent errors in the automated generation of bounding boxes.

Finally, Figure 4.5 shows sample predictions from both the CTOD and COD modules. Although the system consistently detects the target object's location, errors increase during the placement motion, primarily due to the aforementioned challenges.

Table 4.3: Results obtained by the MOSAIC-C(T)OD(-P) models tested in the real-wolrd sceanrio. For each model two variants are proposed, the first one with the model trained from scratch, the second one with the model finetuned from a simulated trained one.

Task	**Method**	**Reaching** [%]	**Picking** [%]	**Success** [%]
Pick-Place (scratch)	MOSAIC	11.29	0.00	0.00
	MOSAIC-CTOD	95.00	11.66	5.00
	MOSAIC-CTOD-P	100.00	30.00	30.00
	MOSAIC-COD	86.66	16.66	16.66
	MOSAIC-COD-P	55.36	21.43	12.50
Pick-Place (finetuned)	MOSAIC	16.95	0.00	0.00
	MOSAIC-CTOD	93.22	44.07	33.90
	MOSAIC-CTOD-P	81.36	47.46	44.07
	MOSAIC-COD	86.67	55.00	**55.00**
	MOSAIC-COD-P	85.00	46.67	46.67

4.3.2 Object conditioned control policy validation

In this section, the validation of the MOSAIC-C(T)OD(-P) modules will be discussed, with a comparison of the performance of the proposed methods against the MOSAIC baseline. The systems were trained under two different settings: first, with the model trained from scratch using only real-world data, and second, with the model fine-tuned from an initial model that was pre-trained in a simulated environment.

Table 4.3 summarizes the obtained results. Specifically, these results are based on 10 rollouts per variation, each starting from different initial configurations. Throughout the tests, the aim was to maintain consistency in the scenarios presented while also introducing new scenarios not encountered during training. The evaluation metrics remain the same as those used in Section 3.4.2, with a focus on the *Reaching*, *Picking*, and *Success* metrics.

Several observations can be drawn from these results. First, for models trained from scratch, the overall system perfor-

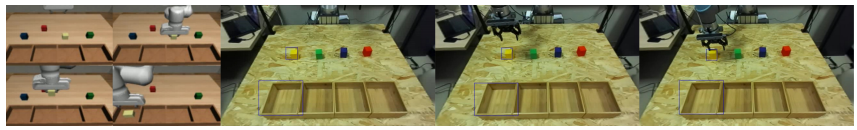


Figure 4.6: An example of a collision with the target object. This represents the most significant and relevant error mode in the proposed real-world system. As observed, the COD module successfully identifies both the target object and its intended placement location. However, the control module fails to complete the pick operation due to a collision.

mance is poor, with a maximum success rate of 30%. However, the robot demonstrates a high *Reaching-Rate*, which is consistently above 80%. This suggests that the object prior is effective in informing the control policy about the object's position, even when trained with limited and noisy data.

The most common failure mode is evident in the drop between the *Reaching* and *Picking* rates, which is primarily due to the gripper colliding with the target object during the picking phase (Figure 4.6). This issue can be attributed to several factors: one major reason is the precision of the predicted bounding box, which may cause slight misalignment (by a few centimeters) of the gripper, resulting in a collision. Furthermore, the limited and noisy trajectories exacerbate the problem. Even with a correct bounding box, the noisy nature of the trajectories, combined with the probabilistic nature of the policy, can lead to high-variance actions, increasing the likelihood of collisions.

To address these limitations, a fine-tuned model was introduced. This model leverages the initialization from a simulation-trained model, where the workspace coverage includes less noisy trajectories. As shown in Table 4.3, the fine-tuned model significantly improves performance, with the highest success rate (**55.00%**) achieved by the MOSAIC-COD module. All fine-tuned models outperform their scratch-trained counterparts. This supports the earlier observation regarding the control module's importance. With proper initialization, the fine-tuned model is better equipped to handle variations in object placement during testing, owing to the improved trajectories

from the simulated environment.

Finally, it is important to note that in the same training scenario, the MOSAIC baseline fails to solve the task, even after fine-tuning. Specifically, the baseline cannot reach any objects, sending the closing command before interacting with any object. This behavior is expected due to the challenges discussed in the training dataset. These results further highlight the importance of object priors in learning an effective control policy, especially when working with low-quality data.

4.4 Conclusion

In conclusion, this chapter presented the experimental validation of the proposed methods in a real-world scenario. Both the proposed detectors and control policies were tested in a single-task multi-variation scenario.

Regarding the conditioned object detectors, it was observed that the system successfully identified the target object even when the video demonstration was provided by a simulated robot, which had a different visual appearance compared to the real robot. This highlights the strong potential of the proposed method to generalize across different demonstrators and environments, such as human demonstrators. Generally, the patterns observed in the simulated environment were replicated in the real-world tests, where the system consistently identified the target object during the reaching phase but generated false positives during the manipulation and placing phases. However, the drop in precision was more pronounced in the real-world setting due to occlusions and inaccuracies in the automated bounding box generation process.

As for the control policies, the results demonstrated that the proposed architectures were able to successfully complete tasks in real-world scenarios. The object-conditioned control policies consistently reached the target object, leading to successful task completion. Across all variations of the proposed method, an average reaching rate of 85.40% was achieved, compared to 14.12% for the MOSAIC baseline. This indicates that object priors can be effectively leveraged to simplify the control

problem, resulting in a policy that consistently reaches the target object, even when using a dataset with low-quality demonstrations. In terms of overall success rate, the MOSAIC-COD module achieved the highest rate of 55.00%. While this may not seem particularly high, it is a promising result given the challenging nature of the dataset, which contains fewer and noisier trajectories compared to the simulated environment. This result becomes even more significant when compared to the MOSAIC baseline's success rate of 0.00%.

Generally speaking, the most critical error observed involves collisions with the target object. This type of error could be significantly reduced by integrating additional exteroceptive modalities, such as depth images from the gripper camera. Incorporating this data would enable the system to better detect the target object and avoid collisions during the picking phase, which is expected to result in a substantial improvement in the overall success rate.

Chapter 5

Graph Neural Network for heuristic estimation

?? This chapter presents an initial exploration of how learning algorithms could enhance the capabilities of the proposed system. The system, as introduced, operates based on a control policy capable of handling single-step tasks, meaning tasks that consist of a single manipulation step. However, in real-world applications, tasks are often more complex, involving multiple steps. For instance, a task may require performing several pick-and-place operations, with the additional constraint that they must follow a specified sequence. In traditional robotics, such problems are classified as *Planning Problems*, where the objective is to produce a sequence of **high-level actions** that move the system from an initial state to a desired goal state.

The primary idea for future work is to integrate planning algorithms into the current framework. This would enable the planning system to generate the high-level action sequence, while the proposed OCCP module executes the corresponding low-level actions, directing the robot to fulfill the tasks derived from the high-level commands.

The purpose of this chapter is twofold: first, to review state-of-the-art methods for solving planning problems, encompassing both classical planning approaches and modern data-driven techniques (Section 5.1); and second, to present preliminary results that demonstrate the applicability of these methods within the specific context of the system under discussion, with

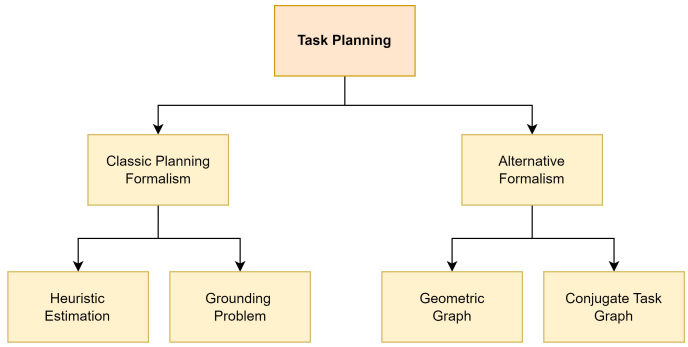


Figure 5.1: Proposed taxonomy for GNNs methods used in the context of robotic learning.

a comparative analysis between these data-driven approaches and traditional planning algorithms (Section 5.2).

5.1 Related Works

This section reviews prior research on the application of Graph Neural Networks (GNNs) to robot learning tasks. As shown in Figure 5.1, GNNs can be applied in various ways; however, they all share a common underlying principle: leveraging GNNs capability to explicitly model dynamic relationships between objects involved in robotic manipulation. This ability to capture and represent interactions between objects is vital for effective robotic perception, planning, and task execution, as these relationships often govern the complexity and success of manipulation tasks.

In general, most research has focused on solving *task planning* problems. According to classical literature [140], a task-planning problem can be defined as a state-transition model $\Pi = (S, A, s_0, G)$, where S is the set of states, A is the set of actions, s_0 is the initial state, and G is the goal state. Here, an action $a \in A$ is a function that maps a state $s \in S$ to a new state (successor) $a(s)$, i.e., $a : S \to S$. Essentially, a plan is a sequence of actions $\tau = a_1, a_2, \ldots, a_n$ that enables the system to transition from the initial state s_0 to the goal state $s_n = a(s_{n-1}) = G$. Various approaches have been proposed to solve the problem of finding a plan given a specific initial

and goal state. These approaches either adhere to the classical PDDL formalism [141], and thus fit within the traditional planning framework, or introduce novel representations that do not rely on PDDL to address the planning problem.

This review begins by examining methods that follow the *Classic Planning Formalism* (Section 5.1.1), followed by a brief discussion of novel approaches that do not use this kind of formalism (Section 5.1.2).

5.1.1 Classic Planning Formalism

In this section, we will describe the methods that follow the classic task-planning formalism. Before introducing the methods, it is crucial to define the formalism with a set of foundational definitions.

Task-planning is a well-known problem that has been studied for many years. One of the most important formalization, still widely used today, is STRIPS (Stanford Research Institute Problem Solver) [142], which was developed in the 70s. STRIPS was initially an automated planning solver, but its primary contribution lies in its definition of a planning problem, which has served as the foundation for much of the research in subsequent decades.

According to the STRIPS formalism, a planning task is defined as a tuple $\Pi = (P, A, s_0, G, c)$, where:

- P is a set of propositions (also called facts).
- A is a set of actions.
- A state s is a subset of predicates, $s \subseteq P$, where s_0 is the initial state,
- $G \subseteq s$ represents the goal conditions.
- $c : A \leftarrow R$ is the cost function, which assign to an action $a \in A$ a real-value, $c(a)$ representing the cost of performing a given action.

An action $a \in A$ is defined as a tuple $a = (pre(a), add(a), del(a))$, where $pre(a), add(a), del(a) \subseteq P$. These represent:

- $pre(a)$: the preconditions (i.e., predicates that must be true to perform the action).
- $add(a)$: the added conditions (i.e., predicates that become true after the action is executed).
- $del(a)$: the deleted conditions (i.e., predicates that become false after the action is executed).

with the constraint that $add(a) \cap del(a) = \emptyset$. According to this formalism, an action is applicable in a state s if $pre(a) \subseteq s$, and it results in the successor state $s' = (s \setminus del(a)) \cup add(a)$.

Building on these foundational concepts, the **Planning Domain Definition Language** (PDDL) was introduced in the 90s [141]. PDDL is a human-readable format for describing automated planning problems. It provides a way to describe, the possible states of the world, the set of available actions, where each action description includes the prerequisites and the effects of the action, a specific initial state of the world and a specific set of desired goals.

PDDL separates the model of a planning problem into two main components:

1. The domain description, which defines the elements that are common across all problems within a given domain.
2. The problem description, which specifies the specific planning problem, including the initial state and the goals to be achieved.

Figure 5.2 provides examples of domain and problem definitions for the Blocks-World environment. As shown, when using the PDDL formalism, it is essential to fully define the elements of the world (e.g., blocks), the available actions (e.g., pick-up), along with their corresponding preconditions (i.e., predicates that must be true for the action to be executed) and postconditions (i.e., the effects of the actions). In the problem definition, specific instances of objects in the environment are specified, along with the initial state and goal state, both of which are defined based on the predicates available in the domain definition.

```
(define (domain blocksworld)
    (:requirements :strips :typing)
    (:types
        block
    )
    (:predicates
        (on ?x - block ?y - block)
        (on-table ?x - block)
        (clear ?x - block)
        (handempty)
        (holding ?x - block)
    )

    (:action pick-up
        :parameters (?x - block)
        :precondition (and (clear ?x) (on-table ?x) (handempty))
        :effect (and (not (on-table ?x))
            (not (clear ?x))
            (not (handempty))]
            (holding ?x))
    )

    (:action put-down
        :parameters (?x - block)
        :precondition (holding ?x)
        :effect (and (not (holding ?x))
            (clear ?x)
            (handempty)
            (on-table ?x))
    )
)
```

```
(define (problem BW-3-42-1)
    (:domain blocksworld)
    (:objects b1 b2 b3 - block)
    (:init
        (handempty)
        (on b1 b3)
        (on b2 b1)
        (on-table b3)
        (clear b2)
    )
    (:goal
        (and
            (on b1 b2)
            (on-table b2)
            (on-table b3)
        )
    )
)
```

Figure 5.2: (Left) Example of a domain definition for the Blocks-World, specifying the predicates and actions available in the environment. (Right) Example of a problem definition for the domain, where a specific instance of the problem is defined, including object instances, the initial state, and the desired goal state.

Once the problem is defined using the PDDL formalism, a complete parameterized instance must be specified to obtain a solution to the planning problem. This involves solving what is known as the **grounding problem**. During this process, all predicate and action variables must be instantiated with the objects defined in the problem. For instance, in the example shown in Figure 5.2, the variable x is replaced with objects $b1$, $b2$, and $b3$ from the problem definition, leading to a complete enumeration of all predicates and actions. The result of this process is known as the *Grounded Planning Problem*, which follows the same formalism as the STRIPS representation described earlier.

The grounded problem then serves as the input for planning algorithms, such as the well-known Fast Downward [143] or LAMA [144]. While delving into the specifics of these algorithms is beyond the scope of this section, it is crucial to note that they are essentially *search algorithms*. Given a state-space represented as a graph, which is derived from the Grounded

STRIPS representation, these algorithms compute the (estimated) the shortest path in the state-space, connecting the source node (representing the initial state) to the target node (representing the goal state).

The limitations of these approaches have been well-known for a long time. Specifically, these algorithms do not scale well as the size of the problem (in terms of actions, predicates, and objects) increases. There are two main areas where higher problem dimensionality can cause performance issues:

1. *Search Phase*, as the dimensionality of the problem increases, the state-space expands correspondingly. This increases the time required to explore the state-space and find a solution path.

2. *Grounding Problem*, as the problem dimensionality grows, the time needed to generate a complete enumeration of all object combinations grows exponentially.

To address the first issue, research has focused on developing methods to accelerate the search phase by introducing *heuristics* that guide the search more efficiently towards the goal state. For the second issue, the literature has proposed methods that perform *partial* grounding, concentrating on the action instances that are most relevant to the specific problem instance. Examples of these methods will be discussed in the following two paragraphs.

Here, a heuristic is a function $h : S \to \mathbb{R}$, which maps a state $s \in S$ to a real value $h(s)$, representing an estimate of the cost to reach the goal state from the state s. The optimal heuristic, denoted $h^*(s)$, is the heuristic that gives the exact cost of the optimal plan to reach a goal state from s. A heuristic is considered *admissible* if it never overestimates the actual cost, i.e., $h(s) \leq h^*(s), \forall s \in S$. By utilizing the heuristic cost, a search algorithm can focus on exploring the most promising states instead of performing an exhaustive search of the entire state space.

In the context of task planning, a common heuristic is computed by solving the **Relaxed-STRIPS** problem. This is a simplified version of the STRIPS problem, where the delete

conditions (del) are ignored, i.e., an action a is represented as $a = (pre(a), add(a), \emptyset)$. The Relaxed-STRIPS problem is easier to solve because removing the del conditions relaxes some of the constraints, allowing the state to include predicates that logically cannot be true simultaneously. By solving this relaxed problem, an estimated cost for reaching the goal from a given state can be computed. This approach has been used in well-known algorithms such as Fast-Downward [143] and combined with novel heuristics in LAMA [144].

Heuristic Estimation

In this paragraph the methods that propose GNN for computing heuristics estimation through the usage of GNNs are discussed. Specifically, the review will focus on two recent works [145, 146].

The first remarkable work towards the use of GNN for heuristic estimation was proposed by authors of [145]. Here, authors started from the idea to leverage data-driven approaches to learn an heuristic function, and explore the possibility of such methods to generalize to different cardinalities and problems, and comparing the performance of such methods with classic heuristic functions.

The first step to solve this problem is related to the definition of the input. Specifically, the authors started by the Relaxed-STRIPS formulation. The STRIPS problem can be easily formulated as a graph if the following observations are done:

- A node, can be described a predicate which can be either a pre-condition or an added condition.
- The edge is represented by the action that connects the pre-conditions to the added-conditions.

Figure 5.3 illustrates the mapping of predicates and actions to graph nodes and edges. As can be observed, an action connects multiple nodes. To model this, the authors use a generalized graph structure known as a *hypergraph*. Formally, a hypergraph is defined as a triple $G = (u, V, E)$, where $V = \{\mathbf{v}_i : i \in \{1, 2, \ldots, N^v\}\}$ is the set of N^v vertices, and $E =$

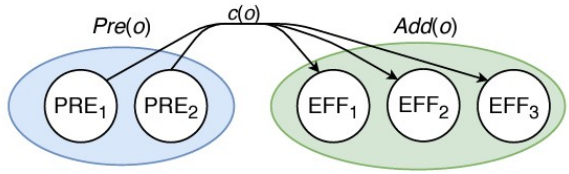


Figure 5.3: Example of nodes and edges definition in [145]. The hyper-edge represents an action that connects multiple nodes. The sending nodes are preconditions, while the receiver nodes are the effects.

$\{(e_k, R_k, S_k) : k \in \{1, 2, \ldots, N^e\}\}$ is the set of N^e hyper-edges. In this structure, R_k is the receiver set, containing the indices of the nodes for which the k-th edge acts as an incoming edge, and S_k is the sender set, containing the indices of the nodes for which the k-th edge acts as an outgoing edge. Then, $\mathbf{v}_i$ is the node embedding, which has been implemented as a binary-vector $v_i = [x_s, x_g]$, where $x_s, x_g \in \{0, 1\}$. $x_s = 1$ iff the i-th predicate is true in the state s and $x_g = 1$ iff the predicate is true in the goal-state. While, e_k is the edge embedding, $e_k = [c(a_k), |Pre(a_k)|, |Add(a_k)|]$, where $c(a_k)$ is the cost of the operator, $|Pre(a_k)|$ is the number of pre-condition and $|Add(a_k)|$ is the number of added condition.

Once the graph structure has been defined, the module that processes this input can be described. Specifically, the authors propose leveraging the concept behind the *Interaction Network* [147]. The architecture, depicted in Figure 5.4, implements the computational flow shown in Figure 5.5, and can be divided into the following three steps.

1. The *Encoding Block* consists of two functions, ϕ^e and ϕ^v, both implemented as Multi-Layer Perceptrons (MLPs). These functions transform the initial binary inputs into higher-dimensional representations, specifically, $\phi^e(e_j)$ produces e^0_{hid} and $\phi^v(v_i)$ produces v^0_{hid}, where both e^0_{hid} and $v^0_{hid} \in \mathbb{R}^{32}$. This step expands the original binary vectors (of size 2 or 3) into 32-dimensional vectors.

2. The *Core Block* implements the Message-Passing Iterative Procedure and is divided into the following steps:

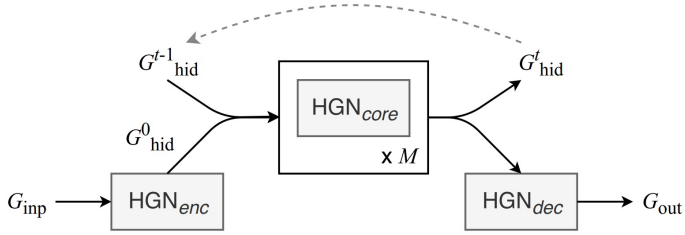


Figure 5.4: Architecture of the STRIPS-HGN model [145]. The model is composed of three main components: the encoder, the hyper-graph network, and the decoder. The encoder processes the input graph, generating a first hidden graph representation. The HGN implements the message passing mechanism, updating the hidden graph representation. The decoder generates the output graph, which global value represents the predicted heuristic.

(a) *Edge Block*: For each edge (representing an action), $e^{t-1}_{hid,j}$ is generated by concatenating the embeddings of the receiver and sender nodes associated with the edge. An MLP encoder is then applied to this concatenated information to create a new edge representation $e^{t}_{hid,j}$.

(b) *Node Block*: Similar to the Interaction Network, only the receiver nodes' states are updated. The edge information influencing each receiver is aggregated by summing the embeddings of the incoming edges using the function $\rho^{e\to v}$. The node input is constructed by concatenating the incoming edge embeddings $e^{t}_{hid,j}$, the initial node embedding $v^{0}_{hid,i}$, and the previous node state $v^{t-1}_{hid,i}$. This concatenated input is then passed through an MLP to generate the updated node representation $v^{t}_{hid,i}$.

(c) *Global Block*: This block updates the global graph representation by aggregating both node and edge embeddings using the functions $\rho^{e\to u}$ and $\rho^{v\to u}$, respectively. The aggregated information is passed through an MLP to produce the new global graph representation.

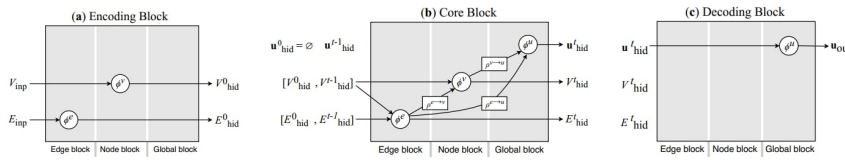


Figure 5.5: Computational flow of the STRIPS-HGN model [145], consisting of three main blocks: Encoding, Core, and Decoding. The Encoding Block generates initial hidden representations for both node and edge embeddings. The Core Block implements the message-passing mechanism, updating the hidden representations of nodes, edges, and global embeddings. Finally, the Decoding Block generates the output graph, where the global embedding represents the predicted heuristic value.

3. The *Decoding Block* acts as a decoder, implemented as an MLP. It takes the updated global graph representation as input and generates a heuristic value for a given state, aiding in the decision-making process.

The operations in the core block are repeated for M iterations, where $M = 10$ in this work. The system is trained by minimizing the loss function defined in Equation 5.1. Here, $h^*(s)$ represents the ground truth heuristic value for state s, obtained by solving the training planning problem with a classical planning algorithm.

$$\mathcal{L}_\theta(\mathcal{B}) = \frac{1}{|\mathcal{B}|} \sum_{(G,h^*(s))\in\mathcal{B}} \frac{1}{M} \sum_{t\in\{1,\ldots,M\}} \left(h_t^\theta(G) - h^*(s)\right)^2 \quad (5.1)$$

Interesting results were obtained during the experiments. Notably, the system demonstrated the ability to learn both domain-dependent heuristics (when tested on the same domain it was trained on) and domain-independent heuristics (when tested on a different domain). This shows that the trained GNN was capable of generating meaningful heuristic values that could be effectively used by search algorithms like A*. However, a general drawback of this approach is the time required to compute the heuristic, particularly when the model

is run on a CPU instead of a GPU, which significantly impacts performance.

The development of such work was proposed by [146]. In this work, the authors conducted a comprehensive analysis of learned heuristics in relation to problem representation. Specifically, they introduced three distinct representations for the planning problem:

- *STRIPS Learning Graph* (SLG): This is a graph constructed from the STRIPS representation. Compared to the representation used in [145], the main improvement lies in the use of labeled edges, which can now represent all edge types, namely, preconditions, added conditions, and deleted conditions.

- *Finite Domain Representation Learning Graph* (FLG): This graph is built from the Finite Domain Representation. In this framework, nodes represent both facts (tuples of state variables v_i and values $d \in D_v$) and actions, while edges represent the relationships between facts and actions, such as preconditions and effects.

- *Lifted Learning Graph* (LLG): This representation avoids the grounding phase entirely. To achieve this, the authors proposed a schema consisting of two main components: the *Action Schema* and the *Instance Subgraph*. The Action Schema encodes domain-specific information, where nodes represent predicates, argument indices, and variables. The Instance Subgraph encodes specific information about the current state and goal state, where nodes represent predicates, states, goal arguments, and objects.

Using these representations, the authors trained a GNN architecture to learn the heuristic function across the three different representations. The results demonstrated that the proposed method was capable of learning both *domain-dependent* and *domain-independent* heuristics. However, when analyzing the coverage performance, i.e., the number of problems solved within a 10-minute timeout, it became clear that performance varied significantly depending on the problem domain. The

classic feed-forward heuristic outperformed the GNN-based heuristic in 3 out of 8 domains, while the GNN-based heuristic outperformed the feed-forward heuristic in 4 out of 8 domains. One domain showed equal performance between the two approaches. Notably, the GNN-based heuristic struggled with the domain-independent test scenarios, meaning that the model was not able to generalize well to unseen domains. This limitation represents a significant drawback, as one of the most important assumptions behind the use of GNNs, or data-driven approaches in general, is their ability to generalize to novel scenarios.

Grounding Problem

As stated earlier, one of the preliminary challenges to address before running the search algorithm is the *Grounding Problem*, which involves computing all valid instantiations that assign objects to the arguments of predicates and action parameters.

It is important to note that the size of the grounded task grows exponentially with the number of arguments in the predicates and action schemas. Although this size remains constant for all tasks within a given domain, and grounding can be computed in polynomial time, it may still become prohibitive when the number of objects is large and/or when some predicates or actions have many parameters. This issue is particularly problematic in open multi-task environments, where many objects exist, but only a few are relevant to the task at hand.

The authors in [148] build upon these observations and propose an algorithm that performs *partial grounding*. The main idea of this approach is to focus on the action instances most relevant to the specific problem instance. In this context, the challenge lies in how to identify and select the most relevant actions for grounding. To address this, a *priority function*, $f : O \to [0, 1]$, is learned, where O is the set of all operators, and $f(o)$ represents the priority of operator o.

This priority function is learned using a supervised learning approach, with the training dataset defined as $\mathcal{D} = \{s_0, G, \Sigma^O, o, \{0, 1\}\}$. Here, s_0 is the initial state, G is the goal state, Σ^O is the set of objects, $o \in O$ is the operator, and $\{0, 1\}$ is the label, where 1 indicates that the operator is part

of an optimal path, and 0 otherwise.

The authors trained a Support Vector Regression (SVR) model, representing the operators with *Relational Rules* (RR). Each operator is associated with a set of rules that are instantiated by grounding the operator's variables. The operator is then represented by a binary feature vector, where each element corresponds to a rule, with a value of 1 if the rule is satisfied, and 0 otherwise. The SVR is trained to predict the priority of an operator based on its binary feature vector.

The system was evaluated on problems where the size of grounded instances grows cubically with respect to the parameters of the generator, resulting in large instances that are difficult to fully ground. The model was trained on smaller problem instances, where full grounding could be computed in a reasonable time. During testing, the model was evaluated on larger instances. Results showed that the learned partial-grounding approach using SVR-RR was able to solve 255 out of 313 problems, outperforming the fully grounded approach, which solved only 156 out of 313 problems.

However, no discussion on the quality of the generated plans was provided. In fact, an evolution of these methods was proposed at the ICAPS 2023 Planning Competition [149]. Results from the competition, however, revealed that these learning-based methods are still significantly behind classical grounding methods, both in terms of optimality and computation time.

5.1.2 Alternative Formalism

In this section, the methods that do not rely on classical planning formalism are going to be discussed. These approaches instead focus on using a graph-based representation of the environment, where nodes represent objects and edges denote relationships between objects or possible actions. This structure enables the application of Graph Neural Networks (GNNs) to process the graph and select the next high-level action the robot should perform. The following paragraphs will describe various ways the environment can be represented and how the network's output is translated into corresponding robot actions.

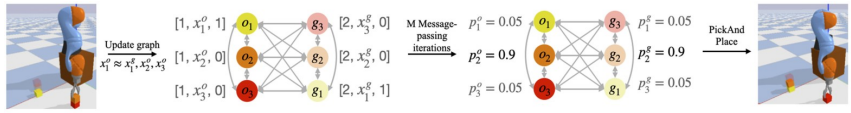


Figure 5.6: Computation flow in a scene represented by a Geometric Graph. Object nodes (o_i) and goal nodes (g_i) are created, with edges connecting them based on spatial and semantic relationships. A GNN classifies the nodes, selecting one object node and one goal node. These nodes are then passed as input to a motion primitive, which generates the necessary actions to move the selected object towards the target goal.

Geometric Graph

This section explores methods based on the concept of the Geometric Graph. In its initial formulation [150], a Geometric Graph is a scene representation where nodes correspond to task-relevant entities, such as objects and their target positions. A fully connected graph is constructed by linking two types of nodes: *object* nodes, which represent current instances of objects with their positions as attributes, and *goal* nodes, which represent the desired final positions of these objects, also characterized by spatial attributes.

Using this representation, a GNN can be trained to perform node classification. At each step, an object node and a goal node are selected, representing the object to move and its target position. With these selections, a pick-and-place primitive can be executed. An example of this inference process is shown in Figure 5.6.

A similar approach is presented in [151], where two GNNs are trained separately to classify an object node and a goal node.

While these methods are noteworthy because they can solve multi-step tasks through supervised learning without relying on a planning algorithm, they have significant limitations. The most notable is their dependency on knowing both the current and target positions of the objects. In these early studies, such information is provided either by the simulation environment or by using special markers on objects in real-world settings to facilitate position estimation.

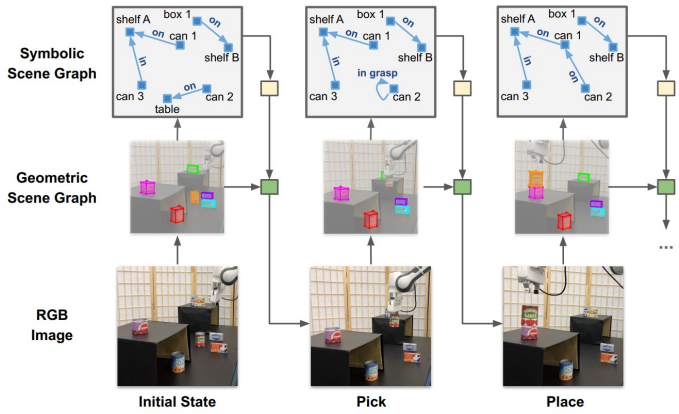


Figure 5.7: Example of a Symbolic Scene Graph. The graph is constructed by extracting objects from the scene and adding edges based on predefined relationships.

To address these limitations, the authors in [152] extended the Geometric Graph concept and proposed a new representation called the *Symbolic Scene Graph*. In this framework, a 3D Object Pose Estimation module is used to extract objects from the scene, constructing a geometric graph. Edges are then added based on predefined relationships (e.g., "can1 on shelf A", as illustrated in Figure 5.7). The next step in the planning process is determined using a Neuro-Symbolic Task Planning module [153], which takes the current Symbolic Scene Graph and the goal (expressed as desired predicates, i.e., relationships between objects) as input. This module outputs the next action, expressed as a subgoal achievable with a single motion primitive.

While this method showed promising results, including reduced computation times compared to the PDDLStream [154] baseline, it has limitations. The primary challenge lies in the cognitive module, which relies on 3D object pose estimation. This technique is only robust and reliable when used with pre-known objects.

Conjugate Task Graph

The Conjugate Task Graph (CTG) is an alternative representation proposed in [155]. This representation is based on the

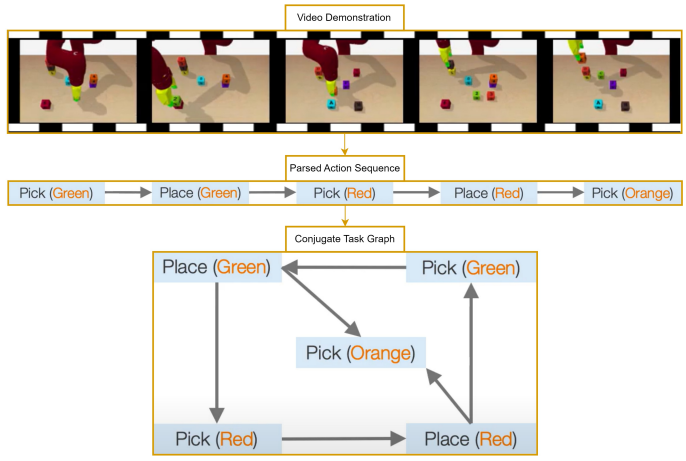


Figure 5.8: Conjugate Task Graph representation. Starting from the video demonstration, the sequence of actions is extracted and used to construct the initial graph. Then the complete Conjugate Task Graph is built by predicting the missing edges, which represents possible actions sequences observed in the training set.

idea that constructing a traditional *Task Graph*, a graph where nodes represent possible states and edges represent valid transitions between those states, can be challenging, especially for novel, unseen tasks. In such cases, mapping the states encountered in a new task to novel nodes is often infeasible. However, in the Conjugate Task Graph, nodes represent actions, and edges implicitly represent states and their preconditions (Figure 5.8). Based on this representation, and assuming all possible actions have been observed in the training set, it is possible to build and train a modular architecture composed of the following components:

- *Demo Interpreter*: A Convolutional Neural Network trained to extract the sequence of actions performed by a robot from a given demonstration. This sequence of actions is used to construct an initial graph.

- *Graph Completion*: A Graph Neural Network trained to complete the graph produced by the Demo Interpreter. This module predicts the missing edges in the graph, rep-

resenting unseen transitions between actions.

- *Node Localization*: A Multi-Layer Perceptron that, given the current observation's encoding and the embeddings of the nodes in the graph, predicts the node corresponding to the action the robot is currently performing.

- *Edge Localization*: Another MLP that, given the current observation's encoding and the node embeddings, predicts the edge representing the next transition and, consequently, the next action the robot should take.

The complete architecture was trained end-to-end, and experimental results demonstrate its effectiveness in multi-step tasks, such as object ordering and stacking. The CTG representation enables the policy to generalize to novel sequences of actions, making it particularly well-suited for tasks like object ordering.

However, this approach fails to generate valid paths when the demonstration includes out-of-distribution observations, produced by sequences of actions that were never encountered in the training set. Additionally, experimental results reveal that this method struggles to perform tasks requiring more than 6 steps. This limitation arises mainly from the Demo Interpreter module, which must generate long action sequences to instantiate the CTG.

In conclusion, with respect to the discussed method, the goal of this chapter is to evaluate and assess the feasibility of using data-driven methods for heuristic estimation within a specific domain of interest. This evaluation aims to explore the potential of leveraging such methods in **future works**, where integrating low-level policies within high-level planning tasks could enable generalization to multi-step tasks.

5.2 Experimental Results

In this section, the experimental results are presented. These results are preliminary within the context of this thesis and primarily aim to evaluate the performance and limitations of

using GNNs for learning heuristics, specifically through the assessment of the STRIPS-HGN method proposed in [145].

The method was tested in the well-known *BlockWorld* domain [156]. This domain consists of a single object type, the **block**. This domain then, contains five predicates, and four actions that are listed below:

- *On*: True if block x is on block y.
- *On-Table*: True if block x is on the table.
- *Clear*: True if block x has no object on top, making it available for manipulation.
- *Handempty*: True if the robotic gripper is not holding any object.
- *Holding*: True if the robotic gripper is holding block x.
- *Pick-Up*: Picks up block x if it is clear, on the table, and the gripper is empty.
- *Put-Down*: Places the block currently held by the gripper onto the table.
- *Stack*: Stacks block x (held by the gripper) onto a clear block y.
- *Unstack*: Removes block x from on top of block y (if block x is clear) and picks it up with the gripper.

For the experiments, 50 problem instances of varying complexity were generated, with complexity measured by the number of blocks, ranging from 3 to 15. The STRIPS-HGN was trained on a subset of problems with block counts between 5 and 10. Testing was conducted in two scenarios:

- *In-Training Distribution*: The model was tested on new problems with complexities within the same range as the training data.
- *Out-of-Training Distribution*: The model was tested on problems with unseen complexities (3, 4, 11, 12, 13, and 15 blocks).

The ground-truth heuristic (h^*) was computed by solving each problem instance using the Scorpion planning solver [157], in a configuration that ensure the optimality of the solution. In this case, the heuristic is represented as the number of actions needed to reach the goal state from the initial state.

The STRIPS-HGN performance was compared against several classic heuristics, including: h_{max}, an admissible heuristic; h_{add}, a generally faster heuristic; $LM - cut$ [144], a heuristic offering a balance between path optimality and speed. The STRIPS-HGN was also compared against itself with different numbers of planning steps (1, 5, and 10).

Moreover, different metrics have been proposed to compare the different algorithms. Specifically, the algorithms are compared with respect to:

- *Number of expanded nodes*, this measures the number of nodes that are traversed during the exploration of the state-space in order to find a given solution. This number is accumulated over the different problems of a given complexity, then the average and standard deviation is computed.

- *Search time*, this measures the time needed for the algorithm to return a valid solution. This number is accumulated over the different problems of a given complexity, then the average and standard deviation is computed.

- *Plan length*, this measures the number of actions that are needed to reach a solution. This number is accumulated over the different problems of a given complexity, then the average and standard deviation is computed.

- *Ratio of solution found*, this measures the ratio of number of times the search algorithm with a given heuristic returns a solution before a timeout of 5 minutes over all the number of problems.

In-Training Distribution Generalization

The first generalization test performed was related to the In-Training Distribution Generalization. This means that the algorithms are tested on 10 novel problem instances for each

Table 5.2: Comparison of "ratio of solution found" on BlocksWorld domain across different complexities. The table presents the results for h_{max}, h_{add}, $LM-cut$, and Strips-HGN with 1, 5, and 10 steps.

Complexity	h_{max}	h_{add}	LM – cut	Strips – HGN_1	Strips – HGN_5	Strips – HGN_{10}
5	1.00	1.00	1.00	1.00	1.00	1.00
6	1.00	1.00	1.00	1.00	1.00	1.00
7	1.00	1.00	1.00	1.00	1.00	1.00
8	1.00	1.00	1.00	1.00	1.00	1.00
9	0.00	1.00	1.00	1.00	1.00	1.00
10	0.10	1.00	0.9	1.00	1.00	1.00

Table 5.3: Mean and standard deviation of "number of expanded nodes" on the BlocksWorld domain across different complexities. The table presents the results for h_{max}, h_{add}, $LM-cut$, and Strips-HGN with 1, 5, and 10 steps.

Complexity	h_{max}	h_{add}	LM – cut	Strips – HGN_1	Strips – HGN_5	Strips – HGN_{10}
5	98.80 ± 97.06	22.00 ± 14.68	24.40 ± 23.30	16.50 ± 10.31	$\mathbf{12.40 \pm 3.29}$	12.90 ± 3.70
6	643.70 ± 430.89	38.70 ± 22.12	35.10 ± 22.56	17.50 ± 5.16	$\mathbf{17.10 \pm 6.77}$	17.30 ± 6.42
7	3918.80 ± 3124.33	54.40 ± 23.86	83.70 ± 100.14	29.80 ± 15.96	27.90 ± 12.29	$\mathbf{27.90 \pm 12.19}$
8	77988.50 ± 53611.24	142.70 ± 64.76	481.60 ± 558.93	$\mathbf{109.90 \pm 98.16}$	126.40 ± 100.73	119.90 ± 94.02
9	154351.20 ± 3673.33	307.70 ± 176.77	758.90 ± 1073.05	99.70 ± 94.47	$\mathbf{44.00 \pm 20.83}$	47.30 ± 24.00
10	103801.70 ± 11155.55	366.20 ± 320.12	2799.70 ± 4595.73	235.90 ± 270.38	150.80 ± 229.64	$\mathbf{95.70 \pm 110.71}$

problem complexity. The first results are related to the number of solution found, reported in Table 5.2. It can be noted how most of the heuristics are able to completely return a solution in the given timeout. This does not happen for the admissible heuristic h_{max}, which starts to not return a valid solution, starting from the complexity of 9 boxes.

To provide more insight into the generalization capabilities of STRIPS-HGN with respect to the training distribution, Tables 5.3, 5.4, and 5.5 report key metrics, including the number of expanded nodes, search time, and plan length, respectively.

The results indicate that STRIPS-HGN can generate a valid path while using a limited number of nodes compared to traditional heuristics. This advantage is particularly pronounced at the complexity of 10 boxes, where $STRIPS-HGN_{10}$ requires an average of only 95.70 nodes to produce a valid path. This represents a significant improvement over other heuristics, which typically need a greater number of nodes to achieve similar results.

Regarding search time, it is noteworthy that $STRIPS-HGN_1$, starting from complexity 7, generates valid paths in

Table 5.4: Mean and standard deviation of "search time" on the BlocksWorld domain across different complexities. The table presents the results for h_{max}, h_{add}, $LM-cut$, and Strips-HGN with 1, 5, and 10 steps.

Complexity	h_{max}	h_{add}	LM − cut	Strips − HGN_1	Strips − HGN_5	Strips − HGN_{10}
5	0.05 ± 0.05	$\mathbf{0.01 \pm 0.01}$	0.05 ± 0.06	0.13 ± 0.08	0.31 ± 0.10	0.57 ± 0.20
6	0.53 ± 0.31	$\mathbf{0.04 \pm 0.02}$	0.18 ± 0.13	0.20 ± 0.07	0.56 ± 0.24	1.05 ± 0.41
7	4.10 ± 2.99	$\mathbf{0.08 \pm 0.03}$	0.58 ± 0.62	0.36 ± 0.18	1.04 ± 0.55	1.74 ± 0.97
8	95.44 ± 53.45	$\mathbf{0.34 \pm 0.19}$	5.94 ± 6.98	1.68 ± 1.55	5.52 ± 4.60	9.73 ± 8.03
9	300.00 ± 0.00	$\mathbf{0.88 \pm 0.47}$	13.10 ± 18.61	1.79 ± 1.44	2.10 ± 1.06	4.11 ± 2.28
10	289.29 ± 32.15	$\mathbf{1.44 \pm 1.14}$	69.89 ± 114.03	4.79 ± 5.35	8.90 ± 13.89	11.05 ± 14.90

Table 5.5: Mean and standard deviation of "plan length" on the BlocksWorld domain across different complexities. The table presents the results for h_{max}, h_{add}, $LM-cut$, and Strips-HGN with 1, 5, and 10 steps.

Complexity	h_{max}	h_{add}	LM − cut	Strips − HGN_1	Strips − HGN_5	Strips − HGN_{10}
5	10.20 ± 2.09	10.60 ± 2.69	10.20 ± 2.09	10.20 ± 2.09	10.20 ± 2.09	10.20 ± 2.09
6	13.00 ± 2.72	14.80 ± 3.25	13.00 ± 2.72	13.00 ± 2.72	13.00 ± 2.72	13.00 ± 2.72
7	15.40 ± 2.69	16.60 ± 2.97	15.40 ± 2.69	15.40 ± 2.69	15.40 ± 2.69	15.40 ± 2.69
8	18.80 ± 2.23	20.60 ± 3.10	18.80 ± 2.23	18.80 ± 2.23	18.80 ± 2.23	18.80 ± 2.23
9	-	26.00 ± 3.69	23.00 ± 2.72	23.00 ± 2.72	23.00 ± 2.72	23.00 ± 2.72
10	-	26.40 ± 3.67	23.11 ± 3.14	23.60 ± 3.32	23.40 ± 3.10	23.40 ± 3.10

less time than the admissible heuristic h_{max} and the well-known trade-off heuristic $LM-cut$. The only heuristic that outperforms STRIPS-HGN in terms of speed is the h_{add} heuristic, recognized for its faster performance relative to the others. However, this speed comes at a cost: all STRIPS-HGN variants produce paths that are shorter on average compared to those generated by the h_{add} heuristic.

Out-of-Training Distribution Generalization

In the second step, we evaluated the generalization capabilities of the STRIPS-HGN on problem instances with complexities outside the training distribution. For each task complexity, we considered 50 problem instances. The results are summarized in Table 5.6.

As shown in Table 5.6, STRIPS-HGN exhibits weaker generalization capabilities compared to the h_{add} heuristic, which consistently returns valid solutions in most cases. Among the STRIPS-HGN variants, only $STRIPS-HGN_1$ consistently produces a reasonable number of valid solutions across all complexities, achieving a success ratio of 0.48 for the complexity

Table 5.6: Comparison of "ratio of solution found" on BlocksWorld domain in out-of-training distribution scenario. The table presents the results for h_{max}, h_{add}, $LM-cut$, and Strips-HGN with 1, 5, and 10 steps.

Complexity	h_{max}	h_{add}	LM – cut	Strips – HGN_1	Strips – HGN_5	Strips – HGN_{10}
3	1.00	1.00	1.00	1.00	1.00	1.00
4	1.00	1.00	1.00	1.00	1.00	1.00
11	0.00	1.00	0.76	0.98	0.98	0.98
12	0.00	1.00	0.46	0.92	0.96	0.90
13	0.00	1.00	0.36	0.84	0.82	0.72
14	0.00	1.00	0.2	0.62	0.48	0.60
15	0.00	0.98	0.10	0.48	0.02	0.08

Table 5.7: Comparison of "ratio of solution found" on BlocksWorld domain in out-of-training distribution scenario. Here the STRIPS-HGN is trained with suboptimal heuristics.

Complexity	Strips – HGN_1	Strips – HGN_5	Strips – HGN_{10}
3	1.00	1.00	1.00
4	1.00	1.00	1.00
11	1.00	1.00	1.00
12	1.00	1.00	0.98
13	1.00	0.82	0.76
14	0.96	0.06	0.00
15	0.82	0.00	0.00

involving 15 boxes. However, this performance is notably lower than the h_{add} heuristic, which achieves a success ratio of 0.98.

To examine whether this behavior results from overfitting the model to optimal solutions, we also trained a STRIPS-HGN variant using suboptimal heuristics obtained by running the Scorpion solver under a configuration that does not guarantee optimality. The results of this analysis are presented in Table 5.7. A comparison of the results for $STRIPS-HGN_1$ between Table 5.6 and Table 5.7 reveals that the model trained with suboptimal heuristics generalizes better to unseen complexities. This is particularly evident for the 15-box complexity, where the success ratio increases to 0.82, compared to 0.48 for the model trained with optimal heuristics.

To further evaluate the quality of STRIPS-HGN, a comparison between average plan length and search time was conducted between h_{add} and $STRIPS-HGN_1$. Specifically, the

Table 5.8: Mean and standard deviation of "search time" on the BlocksWorld domain across different complexities. The table presents the results for h_{add}, $Strips-HGN_1$ trained with optimal-heuristic (Opt) and non-optimal-heuristic (No-Opt).

Complexity	$\mathbf{h_{add}}$	**Strips – HGN$_1$** (Opt)	**Strips – HGN$_1$** (No-Opt)
3	$\mathbf{0.001 \pm 0.0007}$	0.027 ± 0.0152	0.027 ± 0.0152
4	$\mathbf{0.004 \pm 0.0023}$	0.07 ± 0.0333	0.049 ± 0.0336
11	$\mathbf{3.906 \pm 5.1267}$	15.819 ± 31.4679	5.235 ± 19.5074
12	$\mathbf{4.573 \pm 4.0812}$	48.437 ± 70.4687	15.426 ± 42.4102
13	$\mathbf{13.592 \pm 19.1822}$	52.113 ± 77.7223	21.928 ± 52.4085
14	50.439 ± 63.2529	45.355 ± 66.5785	$\mathbf{24.637 \pm 54.7523}$
15	29.362 ± 40.3048	65.405 ± 69.2462	$\mathbf{27.988 \pm 57.1919}$

Table 5.9: Mean and standard deviation of "path length" on the BlocksWorld domain across different complexities. The table presents the results for h_{add}, $Strips-HGN_1$ trained with optimal heuristic (Opt) and non-optimal heuristic (No-Opt).

Complexity	$\mathbf{h_{add}}$	**Strips – HGN$_1$** (Opt)	**Strips – HGN$_1$** (No-Opt)
3	4.72 ± 2.55	4.72 ± 2.55	$\mathbf{4.72 \pm 2.55}$
4	7.84 ± 2.56	7.84 ± 2.56	$\mathbf{6.28 \pm 2.99}$
11	31.06 ± 4.18	25.63 ± 3.95	$\mathbf{12.64 \pm 9.69}$
12	33.48 ± 5.68	28.87 ± 4.29	$\mathbf{16.47 \pm 11.11}$
13	37.86 ± 4.37	30.67 ± 3.40	$\mathbf{18.99 \pm 11.53}$
14	42.90 ± 4.60	33.55 ± 3.89	$\mathbf{20.67 \pm 11.88}$
15	44.17 ± 5.44	35.25 ± 4.16	$\mathbf{21.87 \pm 12.12}$

problems considered are those that the optimally-trained $STRIPS-HGN_1$ was able to solve.

As observed in Table 5.8, the baseline heuristic h_{add} is generally faster than the learned heuristic across 5 out of 7 complexities. However, at higher complexities, specifically with 14 and 15 blocks, the learned heuristic becomes faster than the baseline. It is particularly noteworthy that the heuristic learned with non-optimal heuristics outperforms both the baseline heuristic and the optimally-trained learned heuristic at these higher complexities.

Regarding plan length, as shown in Table 5.9, both learned

heuristics generally produce shorter plans compared to the baseline. Moreover, the heuristic trained on non-optimal heuristics consistently generates shorter paths than the optimally-trained heuristic.

These findings highlight the potential of learned heuristics to generalize better to novel problem complexities and to scale more effectively as the dimensionality of the problem increases. In general, the results align with the literature discussed in Section 5.1. However, further work is necessary to optimize the inference process of these methods, as their primary limitation remains the computational cost of inference, as demonstrated by the results.

5.3 Conclusions

In conclusion, this section has evaluated the potential of using data-driven methods for heuristic estimation within a specific domain of interest. The results indicate that it is feasible to learn effective heuristic values by leveraging learning algorithms and architectures such as Graph Neural Networks (GNNs), which can model relationships between objects more effectively.

However, for these methods to be fully integrated into a framework that generalizes to multi-step tasks, further research is necessary. Specifically, a crucial next step is the development of a robust **perception system** capable of dynamically identifying objects in the environment, thus removing the need for prior knowledge about them.

Once the current state of the environment is obtained, the corresponding PDDL (Planning Domain Definition Language) problem can be formulated, allowing the use of classical planning techniques. However, as noted in Section 5.2, the learned heuristics have shown promise. In particular, within the *in-training distribution* scenario, the learned heuristic outperformed well-established heuristics like h_{max} and $LM-cut$ in terms of nodes expanded and search time. While it did not surpass the faster h_{add} heuristic in search time, it generated shorter plans.

In the *out-of-training distribution* scenario, however, the learned heuristics faced challenges in generating valid plans within the 5-minute timeout, underscoring the need for further research to enhance both the generalization capabilities of these heuristics and the overall performance of the algorithm.

Chapter 6

Conclusions

In conclusion, this thesis is framed in the context of *Learning from Demonstration*, a supervised learning problem that leverages data-driven methods to learn control functions, enabling robots to perform tasks. Section 1.2 explores the formulation of the learning problem, addressing all its aspects, from the mathematical foundations to the techniques used for data collection (Section 1.2.1), and concludes with a comprehensive taxonomy of how this problem has been addressed in the literature (Section 1.2.3). From this analysis, it is clear that the current methodologies do not lead to a "general-purpose" robotic platform. Instead, the learned policies are typically limited to controlling the robot for the specific task on which they were trained, with limited generalization capabilities to different initial configurations of known objects. This limitation prevents a human operator from commanding the robot to perform arbitrary tasks, a highly desirable feature in agile and user-friendly robotic platforms, especially in modern industrial collaborative environments.

For this reason, this thesis addresses the problem of *Multi-Task Imitation Learning* (Section 3.1.1), where a policy is learned to perform multiple variations of a given task, as well as multiple distinct tasks. The policy also incorporates the capability to select, at runtime, the desired task to execute, either through commands based on textual prompts or by using videos of other agents performing the desired task in different configurations. Among these two modalities, the thesis focuses on

Video-Conditioned Multi-Task Imitation Learning, which aims to train control policies where the desired task is specified via video demonstrations of other agents completing the task.

Specifically, in this thesis a different approach to the problem is leveraged. Indeed, started from the consideration that in the Multi-Task scenario there are two main problems to be solved, one related to the **command analysis and understanding**, which can be seen as a *cognitive-problem*, where starting from the current observation of the scene and the command, the system must be able to understand the task intent by localizing for example the objects of interest, and the second related to the **action generation**, which can be seen as a *control-problem*, where starting from the current agent state and the information coming from the command analysis, must generate the correct action towards the task completion. Based on these considerations, an intuitive way to model the problem is through a modular system, however all the state-of-the-art methods proposed in literature for the problem in hand are based on end-to-end architectures (Section 3.1), which means that models starting from high-dimentional inputs (images), are converted into low-level actions, generally the pose of the robot's end-effector with respect to some fixed reference frame.

In the state-of-the-art methods validation (Section 3.4.2) it was observed how the SoTA end-to-end architectures are able to generate valid and reasonable trajectories, however they complete the task by manipulating the wrong object, this means that the Backbone Module is not able to inform the Control Module with the position of the target object. Indeed, it was observed that by performing just 2 ground truth actions (e.g., actions generated through an hand-written control with access to ground state information), the success-rate improved in a considerable way.

For this reason the problem of *Conditioned Object Detection* has been formulated and proposed in Chapter 2, where a conditioned convolutional neural network has been proposed with the ability to generate category agnostic bounding-boxes that given in input the current agent observation and the command is able to localize the objects of interest (e.g., the box to manipulate as well as the bin where place the object). This mod-

ule was experimental tested in a simulated environment both in a Single-Task Multi-Variation and in a Multi-Task Multi-Variation, and it was observed how the proposed module is actually able to identify the locations of the objects of interest with a high precision, considering that the lowest average IoU is equal to **0.563**.

Once the COD module was validated and tested, it was integrated into the SoTA framework to validate the hypothesis that solving the localization problem with a dedicated module can reduce the **object misidentification problem**, thereby improving both the overall success rate and the system's interpretability. The generated bounding box provides information about where the robot is going to move. Specifically, two modules were proposed and tested: MOSAIC-CTOD and MOSAIC-COD. The first module integrates the CTOD module (Section 2.4.2.1), which detects only the target object, while the second module integrates the COD module (Section 2.4.2.2), which detects both the target object and the final location of interest.

These modules were tested in both single-task multi-variation scenarios and multi-task multi-variation scenarios. The results, described in Section 3.4.2.1 and Section 3.4.2.2, show that in the single-task setting, the target object prior effectively facilitates consistent and robust target object reaching and picking, with a picking rate always greater than **80%**.

Regarding the success rate, the highest average value was achieved with the final MOSAIC-COD module, which across four tested tasks showed an average success rate of **90.13%**. This highlights how including information about the object's location improves system robustness.

In the multi-task scenario, it was observed that the use of object priors also resulted in a more successful and robust system. The MOSAIC-COD system achieved an average success rate of **79.24%**, a significant improvement compared to the baseline (46.01%). However, there is still room for improvement, particularly in balancing performance across different tasks.

In conclusion, the proposed system was also validated in a real-world scenario (Chapter 4). Specifically, MOSAIC-CTOD

and MOSAIC-COD were tested in a restricted single-task, multi-variation scenario with two different configurations. The first configuration involved training the system from scratch, while the second used a system finetuned from a model pre-trained in simulation. Results show that the finetuned systems outperformed the systems trained from scratch, demonstrating the potential of leveraging different data domains to initialize the system, which can then be finetuned for the target context.

It was generally observed that the C(T)OD module could predict the location of target objects based on demonstrations from a simulated agent, indicating that the system can interpret task execution from demonstrations by different agents and in different domains. This opens interesting possibilities for future development, including the use of human-based demonstrations.

Regarding the final control policy, the highest success rate was achieved with the MOSAIC-COD module, reaching **55.00%**. While this success rate is lower compared to the corresponding results in simulation, several factors need to be considered, including the characteristics of the dataset and the real-world challenges. Notably, the system consistently reached the target object with a reaching rate of **86.67%**, indicating that the system can reliably inform the control module about the target object's position. The primary source of errors involved collisions with the target object, mostly due to noisy and low-quality data. These issues could be mitigated by enhancing the robot's perceptual capabilities, such as using depth information or a camera mounted on the robot.

Bibliography

[1] S. Bini, G. Percannella, A. Saggese, and M. Vento, "A multitask 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] 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.

[12] T. Yu, C. Finn, S. Dasari, A. Xie, T. Zhang, P. Abbeel, and S. Levine, "One-shot imitation from observing humans via domain-adaptive meta-learning," in *Proceedings of Robotics: Science and Systems*, Pittsburgh, Pennsylvania, June 2018.

[13] 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.

[14] 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

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

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

[26] 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.

[27] 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.

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

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

[30] 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.

[31] 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.

[32] 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.

[33] 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.

[34] 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.

[35] 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

[36] 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.

[37] 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.

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

[39] 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.

[40] 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.

[41] 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.

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

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

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

[45] 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.

[46] 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.

[47] 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.

[48] 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.

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

[50] 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.

[51] 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.

[52] F. Meier, E. Theodorou, F. Stulp, and S. Schaal, "Movement segmentation using a primitive library," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2011, pp. 3407–3412.

[53] A. Agostini, M. Saveriano, D. Lee, and J. Piater, "Manipulation planning using object-centered predicates and hierarchical decomposition of contextual actions," *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 5629–5636, 2020.

[54] N. M. Shafiullah, Z. Cui, A. A. Altanzaya, and L. Pinto, "Behavior transformers: Cloning k modes with one stone," *Advances in neural information processing systems*, vol. 35, pp. 22 955–22 968, 2022.

[55] S. Nair, A. Rajeswaran, V. Kumar, C. Finn, and A. Gupta, "R3m: A universal visual representation for robot manipulation," *arXiv preprint arXiv:2203.12601*, 2022.

[56] C. Chi, S. Feng, Y. Du, Z. Xu, E. Cousineau, B. C. Burchfiel, and S. Song, "Diffusion Policy: Visuomotor Policy Learning via Action Diffusion," in *Proceedings of Robotics: Science and Systems*, Daegu, Republic of Korea, July 2023.

[57] L. X. Shi, A. Sharma, T. Z. Zhao, and C. Finn, "Waypoint-based imitation learning for robotic manipulation," in *Conference on Robot Learning*. PMLR, 2023, pp. 2195–2209.

[58] K. Grauman, A. Westbury, E. Byrne, Z. Chavis, A. Furnari, R. Girdhar, J. Hamburger, H. Jiang, M. Liu, X. Liu *et al.*,

"Ego4d: Around the world in 3,000 hours of egocentric video," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 18 995–19 012.

[59] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[60] E. Perez, F. Strub, H. De Vries, V. Dumoulin, and A. Courville, "Film: Visual reasoning with a general conditioning layer," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 32, 2018.

[61] S. Ross and D. Bagnell, "Efficient reductions for imitation learning," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings, 2010, pp. 661–668.

[62] S. Ross, G. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings, 2011, pp. 627–635.

[63] M. Laskey, C. Chuck, J. Lee, J. Mahler, S. Krishnan, K. Jamieson, A. Dragan, and K. Goldberg, "Comparing human-centric and robot-centric sampling for robot deep learning from demonstrations," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 358–365.

[64] M. Kelly, C. Sidrane, K. Driggs-Campbell, and M. J. Kochenderfer, "Hg-dagger: Interactive imitation learning with human experts," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 8077–8083.

[65] A. Mandlekar, D. Xu, R. Martín-Martín, Y. Zhu, L. Fei-Fei, and S. Savarese, "Human-in-the-loop imitation learning using remote teleoperation," *arXiv preprint arXiv:2012.06733*, 2020.

[66] E. Chisari, T. Welschehold, J. Boedecker, W. Burgard, and A. Valada, "Correct me if i am wrong: Interactive learning for robotic manipulation," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 3695–3702, 2022.

[67] 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.

[68] 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.

[69] 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.

[70] 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.

[71] 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.

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

[73] 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.

[74] 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.

[75] 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.

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

[77] 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.

[78] 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.

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

[80] 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.

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

[82] 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.

[83] 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.

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

[85] 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.

[86] 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.

[87] 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.

[88] 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.

[89] 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.

[90] 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.

[91] 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.

[92] 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.

[93] 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.

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

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

[96] ——, "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.

[97] 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.

[98] 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.

[99] 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.

[100] 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.

[101] 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.

[102] 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.

[103] 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.

[104] 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.

[105] C. Devin, P. Abbeel, T. Darrell, and S. Levine, "Deep object-centric representations for generalizable robot learning," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 7111–7118.

[106] J. Park, Y. Seo, C. Liu, L. Zhao, T. Qin, J. Shin, and T.-Y. Liu, "Object-aware regularization for addressing causal confusion in imitation learning," *Advances in Neural Information Processing Systems*, vol. 34, pp. 3029–3042, 2021.

[107] S. Belkhale and D. Sadigh, "Plato: Predicting latent affordances through object-centric play," in *Conference on Robot Learning*. PMLR, 2023, pp. 1424–1434.

[108] Y. Zhu, A. Joshi, P. Stone, and Y. Zhu, "Viola: Imitation learning for vision-based manipulation with object proposal priors," in *Conference on Robot Learning*. PMLR, 2023, pp. 1199–1210.

[109] Y. Zhu, Z. Jiang, P. Stone, and Y. Zhu, "Learning generalizable manipulation policies with object-centric 3d representations," in *Conference on Robot Learning*. PMLR, 2023, pp. 3418–3433.

[110] Y. Jiang, A. Gupta, Z. Zhang, G. Wang, Y. Dou, Y. Chen, L. Fei-Fei, A. Anandkumar, Y. Zhu, and L. Fan, "Vima: General robot manipulation with multimodal prompts," in *Fortieth International Conference on Machine Learning*, 2023.

[111] R. Girshick, "Fast r-cnn," in *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015, pp. 1440–1448.

[112] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *Computer Vision–ECCV 2014:*

13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13. Springer, 2014, pp. 740–755.

[113] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. Curran Associates, Inc., 2017.

[114] J. Johnson, B. Hariharan, L. Van Der Maaten, L. Fei-Fei, C. Lawrence Zitnick, and R. Girshick, "Clevr: A diagnostic dataset for compositional language and elementary visual reasoning," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 2901–2910.

[115] C. Chen, D. Han, and C.-C. Chang, "Caan: Context-aware attention network for visual question answering," *Pattern Recognition*, vol. 132, p. 108980, 2022.

[116] ——, "Mpcct: Multimodal vision-language learning paradigm with context-based compact transformer," *Pattern Recognition*, vol. 147, p. 110084, 2024.

[117] H. Liu, C. Li, Q. Wu, and Y. J. Lee, "Visual instruction tuning," *Advances in neural information processing systems*, vol. 36, 2024.

[118] D. Tran, H. Wang, L. Torresani, J. Ray, Y. LeCun, and M. Paluri, "A closer look at spatiotemporal convolutions for action recognition," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2018, pp. 6450–6459.

[119] V. Bhutani, A. Majumder, M. Vankadari, S. Dutta, A. Asati, and S. Kumar, "Attentive one-shot meta-imitation learning from visual demonstration," in *2022 International Conference on Robotics and Automation (ICRA)*, 2022, pp. 8584–8590.

[120] B. Ichter, A. Brohan, Y. Chebotar, C. Finn, K. Hausman, A. Herzog, D. Ho, J. Ibarz, A. Irpan, E. Jang, R. Julian, D. Kalashnikov, S. Levine, Y. Lu, C. Parada, K. Rao, P. Sermanet, A. Toshev, V. Vanhoucke, F. Xia, T. Xiao,

P. Xu, M. Yan, N. Brown, M. Ahn, O. Cortes, N. Sievers, C. Tan, S. Xu, D. Reyes, J. Rettinghouse, J. Quiambao, P. Pastor, L. Luu, K. Lee, Y. Kuang, S. Jesmonth, N. J. Joshi, K. Jeffrey, R. J. Ruano, J. Hsu, K. Gopalakrishnan, B. David, A. Zeng, and C. K. Fu, "Do as I can, not as I say: Grounding language in robotic affordances," in *Conference on Robot Learning, CoRL 2022, 14-18 December 2022, Auckland, New Zealand*, ser. Proceedings of Machine Learning Research, K. Liu, D. Kulic, and J. Ichnowski, Eds., vol. 205. PMLR, 2022, pp. 287–318. [Online]. Available: https://proceedings.mlr.press/v205/ichter23a.html

[121] O. Mees, L. Hermann, and W. Burgard, "What matters in language conditioned robotic imitation learning over unstructured data," *IEEE Robotics and Automation Letters (RA-L)*, vol. 7, no. 4, pp. 11 205–11 212, 2022.

[122] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *International conference on machine learning*. PMLR, 2017, pp. 1126–1135.

[123] C. Finn, T. Yu, T. Zhang, P. Abbeel, and S. Levine, "One-shot visual imitation learning via meta-learning," in *Conference on robot learning*. PMLR, 2017, pp. 357–368.

[124] T. Yu, P. Abbeel, S. Levine, and C. Finn, "One-shot hierarchical imitation learning of compound visuomotor tasks," *arXiv preprint arXiv:1810.11043*, 2018.

[125] Y. Duan, M. Andrychowicz, B. Stadie, O. Jonathan Ho, J. Schneider, I. Sutskever, P. Abbeel, and W. Zaremba, "One-shot imitation learning," *Advances in neural information processing systems*, vol. 30, 2017.

[126] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 2014, pp. 1532–1543.

[127] M. Tan and Q. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *International conference on machine learning*. PMLR, 2019, pp. 6105–6114.

[128] D. Cer, Y. Yang, S.-y. Kong, N. Hua, N. Limtiaco, R. S. John, N. Constant, M. Guajardo-Cespedes, S. Yuan, C. Tar *et al.*, "Universal sentence encoder," *arXiv preprint arXiv:1803.11175*, 2018.

[129] M. Ryoo, A. Piergiovanni, A. Arnab, M. Dehghani, and A. Angelova, "Tokenlearner: Adaptive space-time tokenization for videos," *Advances in neural information processing systems*, vol. 34, pp. 12 786–12 797, 2021.

[130] K. Grill-Spector, "The neural basis of object perception," *Current opinion in neurobiology*, vol. 13, no. 2, pp. 159–166, 2003.

[131] O. Mees, J. Borja-Diaz, and W. Burgard, "Grounding language with visual affordances over unstructured data," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 11 576–11 582.

[132] M. Reuss, Ö. E. Yağmurlu, F. Wenzel, and R. Lioutikov, "Multimodal diffusion transformer: Learning versatile behavior from multimodal goals," in *First Workshop on Vision-Language Models for Navigation and Manipulation at ICRA 2024*, 2024.

[133] M. Shridhar, L. Manuelli, and D. Fox, "Cliport: What and where pathways for robotic manipulation," in *Conference on robot learning*. PMLR, 2022, pp. 894–906.

[134] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark *et al.*, "Learning transferable visual models from natural language supervision," in *International conference on machine learning*. PMLR, 2021, pp. 8748–8763.

[135] A. Zeng, P. Florence, J. Tompson, S. Welker, J. Chien, M. Attarian, T. Armstrong, I. Krasin, D. Duong, V. Sindhwani *et al.*, "Transporter networks: Rearranging the visual world for robotic manipulation," in *Conference on Robot Learning*. PMLR, 2021, pp. 726–747.

[136] Z. J. Cui, Y. Wang, N. M. M. Shafiullah, and L. Pinto, "From play to policy: Conditional behavior generation from uncurated robot data," in *The Eleventh International Conference on Learning Representations*, 2023.

[137] U. Robots, "Ur5e cobot per automazione industriale." [Online]. Available: https://www.universal-robots.com/it/prodotti/robot-ur5/

[138] Robotiq, "Pinze modello 2f-85." [Online]. Available: https://robotiq.com/it/prodotti/pinze-modello-2f-85-e-2f-140

[139] ——, "Zed mini - mixed reality." [Online]. Available: https://www.stereolabs.com/zed-mini/

[140] H. Geffner and B. Bonet, *A Concise Introduction to Models and Methods for Automated Planning*. Morgan & Claypool Publishers, 2013.

[141] C. Aeronautiques, A. Howe, C. Knoblock, I. D. McDermott, A. Ram, M. Veloso, D. Weld, D. W. Sri, A. Barrett, D. Christianson *et al.*, "Pddl— the planning domain definition language," *Technical Report, Tech. Rep.*, 1998.

[142] R. E. Fikes and N. J. Nilsson, "Strips: A new approach to the application of theorem proving to problem solving," *Artificial intelligence*, vol. 2, no. 3-4, pp. 189–208, 1971.

[143] M. Helmert, "The fast downward planning system," *Journal of Artificial Intelligence Research*, vol. 26, pp. 191–246, 2006.

[144] S. Richter and M. Westphal, "The lama planner: Guiding cost-based anytime planning with landmarks," *Journal of Artificial Intelligence Research*, vol. 39, pp. 127–177, 2010.

[145] W. Shen, F. Trevizan, and S. Thiébaux, "Learning domain-independent planning heuristics with hypergraph networks," in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 30, 2020, pp. 574–584.

[146] D. Z. Chen, S. Thiébaux, and F. Trevizan, "Learning domain-independent heuristics for grounded and lifted planning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, no. 18, 2024, pp. 20 078–20 086.

[147] P. Battaglia, R. Pascanu, M. Lai, D. Jimenez Rezende *et al.*, "Interaction networks for learning about objects, relations and physics," *Advances in neural information processing systems*, vol. 29, 2016.

[148] D. Gnad, A. Torralba, M. Domínguez, C. Areces, and F. Bustos, "Learning how to ground a plan–partial grounding in classical planning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, 2019, pp. 7602–7609.

[149] P. R. Gzubicki, B. P. Lachowicz, and Á. Torralba, "Huzar: Predicting useful actions with graph neural networks," *Tenth International Planning Competition (IPC-10) Learning Track: Planner Abstracts*, 2023.

[150] Y. Lin, A. S. Wang, E. Undersander, and A. Rai, "Efficient and interpretable robot manipulation with graph neural networks," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 2740–2747, 2022.

[151] F. Di Felice, S. D'Avella, A. Remus, P. Tripicchio, and C. A. Avizzano, "One-shot imitation learning with graph neural networks for pick-and-place manipulation tasks," *IEEE Robotics and Automation Letters*, 2023.

[152] Y. Zhu, J. Tremblay, S. Birchfield, and Y. Zhu, "Hierarchical planning for long-horizon manipulation with geometric and symbolic scene graphs," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 6541–6548.

[153] D. Xu, R. Martín-Martín, D.-A. Huang, Y. Zhu, S. Savarese, and L. F. Fei-Fei, "Regression planning networks," *Advances in neural information processing systems*, vol. 32, 2019.

[154] C. R. Garrett, T. Lozano-Pérez, and L. P. Kaelbling, "Pddlstream: Integrating symbolic planners and blackbox samplers via optimistic adaptive planning," in *Proceedings of the international conference on automated planning and scheduling*, vol. 30, 2020, pp. 440–448.

[155] D.-A. Huang, S. Nair, D. Xu, Y. Zhu, A. Garg, L. Fei-Fei, S. Savarese, and J. C. Niebles, "Neural task graphs: Generalizing to unseen tasks from a single video demonstration," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 8565–8574.

[156] J. Slaney and S. Thiébaux, "Blocks world revisited," *Artificial Intelligence*, vol. 125, no. 1-2, pp. 119–153, 2001.

[157] J. Seipp, T. Keller, and M. Helmert, "Saturated cost partitioning for optimal classical planning," *Journal of Artificial Intelligence Research*, vol. 67, pp. 129–167, 2020.

List of Figures

List of Tables