POLITECNICO DI TORINO

Master's Degree in Computer Engineering

Master's Degree Thesis

Skill learning and task composition from human demonstrations for a collaborative manipulator

Supervisors

Prof. Marina INDRI

PhD. Pangcheng David

CEN CHENG

Candidate

Carlo MIGLIACCIO

December 2025

Alla mia Famiglia e ai miei Amici,
ai miei Sacrifici,
a chi non ce l'ha fatta.

Acknowledgements

Questa tesi rappresenta la conclusione di un percorso lungo e importante, che non avrei potuto affrontare senza il supporto e la vicinanza di tante persone.

Ringrazio di cuore i miei relatori, **Marina Indri** e **Pangcheng David Cen Cheng**, e i cari dottorandi **Cesare e Rosario**, per la disponibilità, le chiacchierate, la fiducia e i suggerimenti che hanno reso possibile lo svolgimento di questo lavoro.

Grazie ai miei amici **Pietro, Pierfrancesco, Mattia, Samuele e Federico**, per aver condiviso con me giornate di studio, risate e tanti bellissimi ricordi. Quei momenti saranno per sempre parte di me. Siete tra i primi che ho conosciuto nell'ormai lontano 2021, e farei fatica a immaginare il mio percorso universitario — e di vita — senza di voi.

Federico, carissimo collega e amico, incontrato all'inizio del percorso magistrale: solo tu sai quanto abbiamo patito per certi esami, e quanto per me sia stato importante supportarci a vicenda, cercando di evadere con il sorriso da quella pesantezza che le sfide da affrontare ci imponevano. Grazie di cuore!

Grazie a tutti i **ragazzi della magica "Aula Studio 3"**, quella che chiamo "la stanza dello spirito e del tempo". Chi da tanto, chi da poco: nutro per voi tanto affetto. Avete trasformato un luogo di impegno e fatica in un posto in cui ci si sente sempre a casa. Una cosa impagabile!

Stefano, grazie! Non avevo mai conosciuto un turbine di ironia, simpatia e intelligenza come te. Grazie per la stima e il bene che mi hai dimostrato fin dall'inizio; per tutte le volte che mi hai ascoltato e donato con pazienza il tuo tempo aiutandomi a liberarmi dalle ragnatele dei momenti di insicurezza; per tutte le risate che mi hai regalato (Piccè si nu "Pizzarrone"!) portandomi leggerezza anche quando non ce n'era affatto. Con te ho capito, ancora di più, quanto sia importante "volare alto", senza mai arrendersi, restando resilienti. A tutti i costi. In pochissimo tempo sei riuscito a darmi tantissimo — più di quanto avrei mai immaginato. Hai lasciato un

segno profondo, fatto di autenticità, leggerezza e forza, che porterò sempre con me.

Edoardo, Fabio e Giuseppe. Grazie, perché tutti insieme mi siete stati vicino in un periodo che definire burrascoso è poco.Quando — proprio alla fine del mio percorso — mille fantasmi sono venuti a trovarmi, voi eravate lì, insieme a me!Mi avete appoggiato, ascoltato, aiutato, consigliato, rassicurato, spronato.Vi sarò per sempre grato per tutto questo: lo porterò per sempre nel mio cuore.

Un grazie speciale a voi, **mamma e papà**, non ci sono parole adatte per raccontare quanti sacrifici avete fatto per me. Il mio amore e la mia gratitudine per voi sono incommensurabili. Quante sono le volte in cui mi avete detto: «Carlo, devi stare sereno, non devi preoccuparti di niente: l'unica cosa veramente importante è che tu stia bene»? Tantissime! Mi sono sempre sentito al sicuro, protetto. Tanto, se avevo voi dalla mia parte, per me andava sempre tutto bene. Potesse crollare il mondo. Senza di voi non sarei nemmeno un granellino di ciò che sono oggi.

Al mio amato fratello **Antonio**, mio mentore, mio sangue, mia ancora, mio tutto. Ci siamo sempre amati dal primo giorno, da quando — appena nato — con una chiave finta di fortuna mi facesti sobbalzare dalla culletta. Ti ricordi? Quante volte mamma e papà ce l'hanno raccontato e io chiaramente ogni volta, mi sciolgo. Sei sempre stato per me l'esempio da seguire, il mio porto sicuro. Sei stato, in questi anni, una delle cose di cui ho sentito di più la mancanza nonostante ci sentissimo tante volte al giorno, tutti i giorni, in tutti i modi possibili. Qualcuno lo sa bene quanto io tenga a te, ma su questo non c'è da sorprendersi.

Tutti voi, tutti, chi in un modo chi in un altro, chi da tanto, chi da poco, chi da sempre, siete parte della mia vita. Ci siamo incrociati nei modi e nelle occasioni più disparate, ma con ognuno di voi condivido uno, più ricordi o tutti i ricordi.

A tutti voi, **infinitamente, grazie**.
Vostro, Carlo.

Abstract

Human-robot collaboration (HRC) in the modern industry requires the employment of manipulators that can acquire and reuse skills in a easy way and without domain-specific knowledge. Learning from Demonstration (LfD) offers a practical way to do so, however, real deployments still face some caveats; such as turning raw demonstrations into reliable low-level controllers on hardware, re-parameterizing skills to new object/goal poses.

This thesis presents a unified LfD pipeline implementation for the UFACTORY xArm6, a 6DOF collaborative robot (cobot) that allows learning of reusable motor skills from human kinesthetic demonstrations. Such demonstrations can be used to plan more complex manipulation task.

The pipeline follows the canonical stages – demonstration data acquisition, motion encoding, execution, and refinement – and is developed with three learning methods for low-level skills: Behavioral Cloning (BC), Dynamic Movement Primitives (DMP), and Gaussian Mixture Models with Gaussian Mixture Regression (GMM-GMR). The Task-dependent parameters are retrieved from a vision subsystem based on RGB-D RealSense D435 camera, enabling skill adaptation to unseen situations without retraining. On the real robot the execution uses the vendor SDK for fine-grained control which in turn allows to tackle the Sim2Real gap; in simulation, trajectories are executed through MoveIt environment for rapid checking.

The pipeline is validated on four low-level skills: PICK, PLACE, POUR, and SHAKE. Moreover, it is also tested on high-level tasks that organize them into plans, including pick-and-place, object collection, single-drink pouring, and multi-ingredient mixing. Furthermore, we assume that skills (eventually conditioned) sequencing is user-defined.

The overall performance is assessed with geometry and objective fulfillment metrics (Cartesian/orientation RMSE, Hausdorff distance, endpoint error), motion quality (jerk), and task-level success rates, highlighting trade-offs between learning approaches (e.g., GMM-GMR's accuracy on discrete motions and DMPs' convenient goal re-targeting).

The contributions are: (i) a modular, end-to-end LfD pipeline for the cobot that goes from kinesthetic data capture to real-robot execution; (ii) a comparative

implementation of BC, DMP, and GMM-GMR for skill learning; (iii) a vision-guided parameterization layer for skill reuse across poses; and (iv) task-level controllers that include learned primitives into reliable plans. Collectively, these results demonstrate that few-demo LfD can deliver accurate, adaptable, and maintainable behaviors for collaborative manipulation, reducing integration effort while preserving generality.

This page was left intentionally blank

Table of Contents

List of Tables

List of Figures

Acronyms

CPS
Cyber-Physical System

HRC
Human-Robot Collaboration

LfD
Learning from Demonstration

RL
Reinforcement Learning

IRL
Inverse Reinforcement Learning

HRI
Human-Robot Interaction

BC
Behavior Cloning or Behavioral Cloning

GMM
Gaussian Mixture Model

GMR
Gaussian Mixture Regression

TP-GMM
Task-Parameterized Gaussian Mixture Model

MP

Movement Primitive

DMP

Dynamic Movement Primitives

ProMPs

Probabilistic Movement Primitives

MDP

Markov Decision Process

HMM

Hidden Markov Model

DAgger

Dataset Aggregation

IL

Imitation Learning

IIL

Interactive Imitation Learning

POMDP

Partially Observable Markov Decision Process

TAMER

Training an Agent Manually via Evaluative Reinforcement

HG-DAgger

Human-Gated Dataset Aggregation

COACH

COrrective Advice Communicated by Humans

SRMS

Speed and Separation Monitoring

HG

Hand Guiding

SSM

Speed and Separation Monitoring

PFL

Power and Force Limiting

RNN

Recurrent Neural Network

LSTM

Long Short-Term Memory

RRT

Rapidly-exploring Random Tree

Chapter 1

Introduction

Recent advances in machine learning (ML) techniques, edge/cloud computing passing through smart sensors technologies have drastically changed the way to conceive the Industry. All of these elements can collaborate and being integrated together to build up Cyber-Physical systems (CPS), which are nowadays playing an important role while making possible both technologies and methods that once were considered only in science fictions.
There has been a shift from mass production to a more *adaptive* and *flexible* paradigm enabled by the **co-existence of human and robots**, with the objective of taking the best from each one. This is what you need for the *Industry 4.0* to enable the so-called *smart-manifacturing*. With this aim a novel and innovative discipline, which has been catching the attention from the scientific community, is the *Human-Robot collaboration* (HRC) to which is dedicated the next paragraph.

1.1 Human-Robot collaboration

The main focus of HRC is to combine the **flexibility** of human beings with the **automation** of which is equipped a robotic system. Note that, such robots in a collaborative context are called **collaborative robots** (**cobots** for short). In particular they – with respect to the traditional ones – work in a shared workspace with humans and in order to ease the reciprocal interaction are provided with sensors, cameras and other advanced technologies. Relevant applications of cobots in manufacturing and main differences with traditional robots are presented more deeply in the work by Javaid et al. [1]. The transition from a traditional to a collaborative industrial scenario is not straightforward at all, since:

- You have to install cobots instead of robots, this requires a redesign of working cells and places within the factory, note that this is affecting the plant layout of the organization;

- They must be programmed in an efficient and flexible way which could be context-aware and easily customizable;
- The fact that their workspace is shared with humans raises up *safety issues*, that have to be properly addressed and are the leading factor in implementing a smart factory. A detailed analysis of such aspects can be found in Sections 2.5 and 2.6.

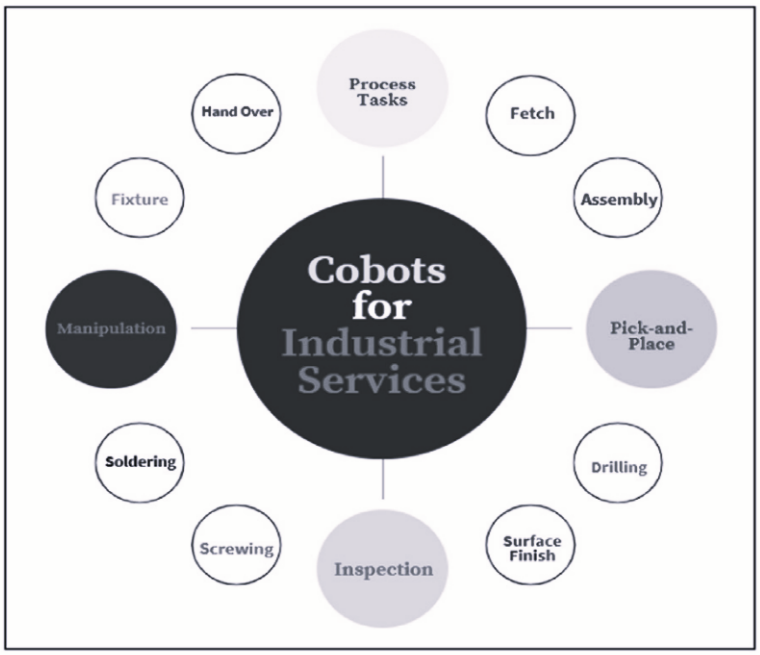


Figure 1.1: Typical uses of a collaborative robots [1]

1.2 The Learning from demonstration paradigm

Generally speaking, one of the most important and enabling functions in HRC is **intuitive learning** which allows both humans and robots to exchange knowledge and notions. One of the most used approach to implement it, is the *Learning from Demonstration (LfD)*, which is the

> *"the paradigm in which robots acquire new skills by learning to imitate an expert" [2].*

This is closing more and more the existing gap between research and practical applications in robotics. Such a technique allows the *programming of robotic systems*

in a unusual way that skip the traditional motion/task planning procedures.
In general, programming a robot using traditional methods is quite hard. Indeed, not only good programming skills are needed but also a medium to low-level knowledge about robot kinematics and dynamics is often required. Furthermore, the traditional way by which a robot is programmed requires the single actions to be specified by an expert "by hand"! Even worse, little changes either in the environemnt or in the procedures are associated with a complete reprogramming of the system.
The LfD paradigm overcomes these issues:

- Allowing non expert to teach the robot without any mechanical or mathematical knowledge;
- Most LfD methods learn and **generalize** the learnt skills to novel task and requirements;
- Defining a task in such a framework results in less effort and a quicker way to serve the *manufacturing requirements*.

LfD takes inspiration from neuroscience, especially in terms of imitation, observation, and feedback phases. Furthermore, according to [3]

> "Starting at an early ages, children use the information around them to learn from observation, experience, and instruction, striving to imitate the adults around them".

A practical roadmap for an industry to adopt and deploy LfD methods is shown in [4] by Barekatain et al. with a specific focus on **manipulation tasks**.

1.3 Overview and Thesis structure

Overall, the objective of this thesis is to design and implement an end-to-end Learning from Demonstration pipeline for a 6DOF collaborative manipulator (the `UFACTORY xArm6`), capable of acquiring reusable manipulation skills from human kinesthetic demonstrations and adapting them to new task configurations. To achieve this, the work combines a conceptual and methodological analysis of HRC and LfD with a comparative study of three motion encoders (Behavioral Cloning, DMP, GMM-GMR), the integration of a vision-based parametrization layer, and an experimental validation on both low-level skills and their composition into higher-level tasks.

The remaining part of this thesis is organized as follows: **Chapter 1** introduces the industrial context, Human–Robot Collaboration and the LfD paradigm; **Chapter 2** presents the state of the art on LfD, interactive learning, robotic assembly and

safety aspects in HRC; **Chapter 3** provides a theoretical overview of the selected LfD methods (BC, GMM–GMR, DMP); **Chapter 4** details the implementation of the proposed LfD pipeline including demonstration collection, learning and execution; **Chapter 5** reports and discusses the experimental results for both low-level skills and high-level tasks; **Chapter 6** concludes the work, highlighting the main contributions and limitations and outlining possible directions for future developments.

Chapter 2

State of the art

The LfD paradigm stays at the intersection between Supervised Learning (SL) and Reinforcement Learning (RL). In such a framework, the robot can be seen as an agent that is put into an environment. Coming back to the paradigms, we can say that, in particular, from the former we inherit the fact that some learning/training data are used (what we are going to call *demonstrations*), from the latter the fact that – at the end of the learning phase – we obtain a function that *for each state of the environment associates an action for the robot (agent).* This function, in first approximation, is called a **policy**:

$$\pi : S \to A \tag{2.1}$$

where S is the *state space* and A is the *action space.* As in RL, also in the LfD framework the *robot learning process* can be modeled through a *Markov Decision Process (MDP)* by which the main results and proofs can be provided also for LfD. How it is explained in [5], the main difference between RL and LfD is that, at least in the *vanilla version*, **during the training the agent is not interacting with the environment since in the learning data there is a guiding policy $\pi_{teacher}$ to imitate**, that is the one showed by the human expert. The importance of MDP theory and the relationship between LfD and RL motivated us in providing a brief theoretical overview in the Appendix A.

2.1 The LfD pipeline: theoretical foundations

Entering more in details here, we want to find some (family of) methods by which the collaborative robots can learn some **low-level skills** or a **high-level task** relying on a natural demonstration of an expert teacher that in some way is meant to show to the robot the way to execute a certain skill or a certain tipology of task. According to [5] and other inherent surveys, such a learning process can be split

into **five stages**: (i) Demonstrator selection, (ii) Data acquisition (demonstrations), (iii) Data modeling (learning), (iv) Task/skill execution, (v) Learning refinement, as shown in Figure 2.1:

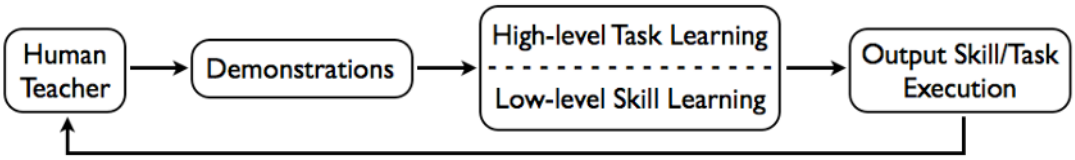


Figure 2.1: LfD pipeline. A *Human teacher* shows the skill/task to be executed by giving *Demonstrations*, on such data some *High-level Task Learning* or *Low-level Skill Learning* is performed; learned skill/task can be *Executed* and, eventually refined. [3]

2.1.1 Demonstration method

There are mainly three modalities by which demonstrations can be performed as illustrated in Figure 2.2:

1. *Kinestetic approach.* Here, there is a human that physically guides the cobot through a desired trajectory. The key aspect here is the **direct interaction** between the learner (robot) and the teacher. Due to this fact safety concerns arise;
2. *Teleoperation.* In this case the human remotely control the robot by using an **external mechanism** (eg. joysticks). This is adviceable for those operations for which a direct interaction should be avoided for safety reasons.
3. *Passive observation.* Here, there is not a direct interaction or explicit guidance of the learner, since the robot is a passive observer. The demonstrations are provided by means of **images from a camera** of some type.

The choice of the demonstration method implies the choice of the demonstrator; in the first two cases it is the learner itself, since the **demonstrated task** is executed in the *configuration space* (which is the joint space), while in the third case, the demonstrator is someone different than the robot, since there is the human that performs the demonstration.
Here, a first issue arises in the demonstration phase the so-called **correspondence issue**, which is related to answering to the following question:

How can we map the state and action spaces of the teacher in the state and action spaces of the learner (agent/robot)?

(a) Kinesthetic Teaching (b) Teleoperation (c) Passive Observation

Figure 2.2: Different demonstration methods: (a) *kinesthetic*: the end-effector is manually guided in the task space; (b) *teleoperation*: an external device is used to move joints; (c) *passive observation* demonstration are provided in the human state-space [4]

In the worst case, there are two correspondence functions to apply [5]: from the teacher space to the record (**Record Mapping**) and from the record to the learner (**Embodiment Mapping**). How far the first two methods are concerned, no mapping is required since the teacher acting directly on the robot (by using its hands or an external mechanism) provides data which are already in its configuration space.

More formally:

$$(s_{record}, a_{record}) = m_R(s_{teacher}, a_{teacher}) \quad \text{Record Mapping} \tag{2.2}$$

$$(s_{learner}, a_{learner}) = m_E(s_{record}, a_{record}) \quad \text{Embodiment Mapping} \tag{2.3}$$

where m_R and m_E are, respectively, the Record and Embodiment mapping functions. The main aspects of the just presented demonstration techniques are given in the table in Figure 2.3 from [4].

2.1.2 Demonstration data and learning space

The Demonstration data must be provided in a structured way, so that learning method can use them in a straightforward manner. In general, whatever is the shape of the demonstration data they have:

1. The information related to the **state** which is called *feature vector*. It may contain also redundant/useless information that must be thrown away;
2. The **action** performed by the teacher to a particular state (in vector sense).

The survey in [5] provides a clear distinction of the features in: (i) **raw features** which are the ones obtained directly from the robot sensors; (ii) **handcrafted**

	Kinesthetic Teaching	Teleoperation	Passive Observation
Concept	Physically guiding robot	Remotely guiding robot	Observing human actions
Advantages	Demonstrate complex motion Minimal setup Intuitive interaction Precise manipulator control	Safe demonstration Isolation of teaching	Safe demonstration Ease of demonstration
Limitations	Safety concerns Physically demanding	Complex setup Requires skills to use	Complex setup Inefficient for complex tasks
Recommended Use	Full-task demonstration Subtask demonstration Motion demonstration	Contact-based demonstration Iterative refinement	Full-task demonstration Large-scale data collection

Figure 2.3: Main demonstration methods and their features concerning *Concept*, *Advantages*, *Limitations* and *Recommended use* [4]

features they are usually obtained by applying manually designed functions giving a rich and minimal representation; (iii) **extracted features**, which are the one obtained by passing the demonstration dataset through a deep-learning architecture (e.g., a Convolutional Neural Network (CNN)).

Another simpler classification for the spaces in which can lie the demonstration data is given in [4]. This is more suitable for *robotic manipulators*. The two individuated spaces are:

Joint Space (Configuration Space) This offers a low-level representation for the *motor commands*. At the end of the learning process, the integration of the policy into the controller is straightforward. The main drawback of such a method for picking the data is that there is high probability to overfit the training data with the related *loss of generalization*.

Cartesian Space (Operational or Task Space) In robotics we use the cartesian space to describe the attitude of the end-effector. The main advantage of such a method is the *adaptability* of such a description to other robotic platforms. This is a human-oriented description, indeed. It facilitates the explainability of the learning process.

2.1.3 Issues related to demonstration data

In the LfD paradigm, as in SL, the quality of the policy that one obtains at the end of the learning process is strongly dependent on the learning data, which in our case, are the ones related to the demonstrations. Note that the choice of a demonstration method instead of another can influence the choice of the learning method, too. The main issues concerning the data quality are:

Incomplete Data The demonstration data that are collected represent a subset of the full MDP. The lower the quantity of collected learning data, the higher

the probabilty of having a state that has not been sampled. In that case, the policy is diverging more and more to the desired actions in a way that the final objective will not be reached. This is the well-known problem of the **distributional shift**. As reported in [5], the ways for improving this issue are: (i) collecting more data, (ii) obtaining new data from the existing ones[1]; (iii) Using transfer learning exploiting the skills learned for other types of tasks.

Inadequate Data Demonstration data are collected by sensors or cameras. Whatever is the demonstration technique, there is uncertainty in the measurement process: a non-expert teacher, an inaccurate sensor, a camera affected by occlusion.

The presented issues give in output policies that achieves *sub-optimality*. This is the reason why the obtained policy has to be tuned using some refinement technique to which we dedicated a section, but first, some other theoretical details about learning methods and outcomes are needed.

2.1.4 Learning methods: introduction

In the field of manifacturing, there are many tasks a robot can learn (e.g., *pick and place*, *peg-in-hole* and so on). Each one of this tasks (also called **High-level tasks**) can be in turn split in several subtask (called also **Low-level skills**). As it is stated in [4], it is possible to achieve better performances when a single subtask at a time is learned and during a second stage a conditional hierarchy is superimposed. This will be clearer in the section dedicated to manipulation tasks in robotics (see Section 2.4). Learning methods can be classified according to the task they allow to learn.

Low-level skill learning

Low-level skills are the *basic movement* a robot can learn (pushing, moving, grasping, and so on). They can be put together in order to form more complex tasks. There are cases in which is more convenient to have a learning space with respect to another. In this family of methods we have a further distinction in:

- *One-shot or deterministic methods*: they are based on a **single skill demonstration**. It is the simplest strategy considering such an aspect, however, it is vulnerable to the noise introduced in the demonstration phase. The simplest method of this type is the Behaviour cloning, which consists of obtaining

[1]This is what in supervised learning is called *Data augmentation*

a deterministic policy[2] $\pi_\theta(s) = a$ in a Supervised Learning fashion. At the opposite the most used one-shot method is called Dynamic Movement Primitive (DMP);

- *Multi-shot or probabilistic methods*: Here, the data coming from demonstrations are modelled with a probability density function (pdf), this is the same to say that the (2.1) will be of the following type

$$\pi = \pi(s_t|a_t) \tag{2.4}$$

The state-of-the-art probabilistic method is the combination of GMM (Gaussian Mixture Model) and GMR (Gaussian Mixed Regression).

Both State-Of-the-Art (SOA) methods are well-presented and explained in [6], where in general an overview of trajectory generation for robotics is given.

High-level task learning

A *High-level task* is a compound of a set of primitive motions (low-level skills). The demonstrations, in this case, are sequences of primitive motions. In order to learn a task, there are three main methods:

1. *Policy learning* The task is reproducing the demonstrator policy by generalizing a set of demonstrations (a set of multiple compounds of elementary tasks); here, the demonstration data is a sequence of state-action pairs;
2. *Reward Learning* The task to be executed is represented by a **reward function**. In order to obtain a reward function, Inverse Reinforcement Learning (IRL) applied to demonstration data can be used.
3. *Semantic learning* They are based on extracting which are the correct parameters related to *task features*, *object affordance* and *task frames*.

Overall, the most used one-shot learning method is DMP, GMM-GMR for multi-shot learning. Finally, the most used methods for high-level task is Semantic Learning. An exhaustive list of methods for both skills and tasks learning is presented in [7] and more deeply discussed in the next subsection.

2.1.5 Learning outcomes

In this section, the objective is to give more details about the learning phase outcome answering to the question: *what are we going to learn from demonstration data*? According to [2], there are mainly three possibilities:

[2]For example, a Multi-Layer Perceptron can be used, it can rely on *Universal Approximation Theorem* by Barron. The obtained policy will result in a deterministic function parametrized by the parameters θ.

1. **Policy learning (associated to low-level skills)** is the mapping function π we have introduced so far, providing the correspondence between a state space variable and an action that applied to the manipulator will generate similar trajectories to the ones showed into the demonstration dataset given by an expert. The most used methods learn policies that provide a mapping into the trajectory space. More specifically they are: (i) dynamical systems-based (e.g., DMP, ProDMP), where the demonstrated trajectories are supposed to be solutions of the dynamical system itself; (ii) probabilistic inference, where the demonstration are compressed in a unsupervised fashion using multivariate models. The output trajectories, moreover, are obtained by **sampling the learned multivariate distribution** after eventually having conditioned on initial and final point (examples of such models are the GMM-GMR based). Finally, the generated trajectories can be either in the joint space or in the operational space (clearly there are pros and cons related to generalization and singularity handling).

2. **Reward or Cost function learning** In this context, IRL is used in order to obtain a reward function R_t from demonstration data. This is one of the most difficult task in Reinforcement Learning. As reported in A, IRL is usually used in learning refinement. A strong assumption on the demonstration data is made, which is the demonstrated policy $\pi_{teacher}$ to be optimal (this is an approximation since it is never the case).

3. **Plan learning (associated to high-level task)** This learning strategy is devoted to complex tasks (eg. folding a towel, assembling a device). These are usually sets of *primitive actions*, which are often demonstrated in continuous sessions. Now, whether the demonstrated trajectories are of the full complex task, which is composed of a sequence of actions, the first problem to be solved is the **segmentation of the trajectories** in subtasks. Within this family of approaches, a further distinction can be introduced among methods performing a learning of a *primitive sequence* or a *primitive hierarchy* (see 2.4).

The main features of the approaches we have just presented are shown in Figure 2.4.

2.1.6 Refinement Learning: introduction

We have seen that due to the distributional shift and the presence of noise in the data, the obtained $\pi(s)$ can be a sub-optimal one. Such a policy, then, could require some **refinement**.
The most common way to do so, is using **Reinforcement learning (RL)** in order

Learning outcome	Low-level control	Action space continuity	Compact representation	Long-horizon planning	Multistep tasks
Policy	✓	✓	✓		
Cost or reward	✓	✓		✓	
Plan			✓	✓	✓

Figure 2.4: Main features of LfD methods according to different learning outcomes: Policy, Cost or Reward, Plan [2]

to let the agent interact with the environment in a way to uncover the unknown part of the MDP. At this aim a reward function can be obtained by using the demonstrations, using such a reward the founded policy can be refined.
One can wonder, *why not using RL from the beginning?* In real-world applications exposing the robotic system to the exploration of the environment starting from a randomic policy could be dangerous for both humans and the robot itself, since a state of the environment may lead to physical constraints violation. On the contrary, using RL only for fine-tuning the policy obtained from LfD, this results in a safer procedure.

In order to further enhance the LfD methods **active** and **interactive learning** can be employed [8]. The approaches are similar. In the first case, when you are out from the distribution of the demonstration data, the learner actively requires further demonstration to the teacher. On the other hand, in interactive learning method the teacher can act during the task execution to correct some wrong behaviours. The next section expands such concepts.

2.2 Learning refinement and Interactive learning

There some other methods, different from IRL, which can be used during the **Learning refinement**. We mean *Active* and *Interactive Learning* that are at the intersection between **Imitation Learning**[3] and **Interactive Machine Learning**, in a field which in the literature is called Interactive Imitation Learning[4]. We have talked about Imitation learning in the latest reports, while – citing the definition from Dudley and Kristensonn (2018) [9]:

[3]In the literature you can find: Learning From Demonstration (LfD), Learning by Demonstration (LbD), Programming by Demonstration (PbF) or Imitation Learning (IL). These are used indistinctly when one wants to enabling robot deriving controllers from human demonstrations. In the present report and the following we will also use such terms interchangebly.

[4]Initially we will analyze only Interactive Learning which is the most discussed, leaving for further research *Active Learning*, the process by which is the learner that ask for help.

> *"Interactive Machine Learning is distinct from classical machine learning in that human intelligence is applied through iterative teaching and model refinement in a relaively tight loop of set-and-check. In other words, the user provides additional information to the system to update the model, and the change in the model is reviewed against the user's design objective."*

Methods like Inverse Reinforcement Learning and Imitation Learning are not interactive, despite the fact they use human demonstrations. These are used in a supervised-learning fashion as a *training dataset* and data are collected before the learning process starts. At the opposite, **Interactive Imitation Learning** (IIL) being at the intersection between IL and IML, exploits the presence of a Human-In-The-Loop (HITL) **during the training of the policy**.

2.2.1 Motivation

The introduction of IIL comes from the idea of instructing a robot "naturally", taking inspiration on how humans learn complex skills. Indeed, while a small set of demonstration is sufficient for learning simple skills (e.g., open the door), complex skills – for example playing a sport – require a **loop of interaction** in which the teacher explains directly what are the issues to fix or evaluates the actions. Similar concepts can be tailored for robotics with the difference that the learner is not a human but a robotic system. Furthermore, there are several advantages in emplyoing **interactions** in the LfD pipeline feedback:

1. At the end of the training, you will earn more accurate datasets which includes, by means of rating feedback or corrections, situations that are not normally present when using traditional demonstration datasets;
2. The teacher is allowed to transfer the knowledge to the robot not only with the demonstrations. On the contrary, other types of interactions can be used, such as: reinforcements, preferences, corrections;
3. The correspondence problem we mentioned at the beginning can be solved in a effective way where the teacher is directly involved.

Without any doubt, the presence of the human teacher in the learning loop is not negligible: humans make mistakes and can be inconsistent. This, at a higher theoretical level, may lead the underlying MDP (or Partially Observable MDP) to diverge. This is the main reason why, in practice, several approaches are used in order to encapsulate the presence of the teacher in a way that the whole LfD process will obtain benefits.
In the following, we are analyzing:

- How the teacher is supposed to intervene giving feedback about either *how good* the work has been carried out or *how to do* to improve its behavior;
- What are the available channels by which the feedback can be conveyed?

The main reference for analyzing and exploring the IIL field can be found in [10] from which we are taking the main aspects. Clear and detailed theoretical insigths are also provided.

It is remarkable that we are going to mention some RL-specific terms, since, as stated at the beginning, IL/IIL and RL are strictly related fields.

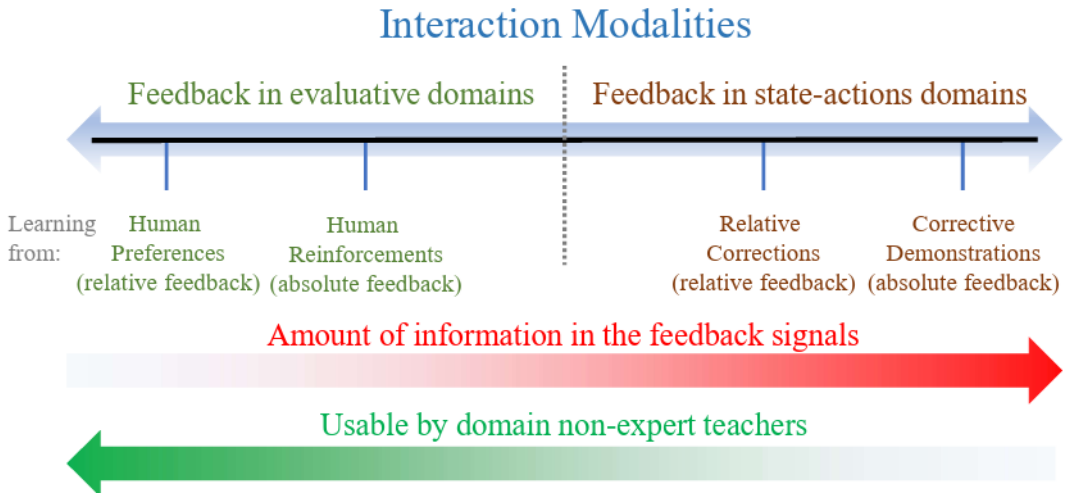


Figure 2.5: Interaction modalities and informations contained in the feedback signal: the more information to be provided the less usable the approach by non-expert teachers [10]

2.2.2 Modalities of interaction

At this stage, we want to analyze possible modalities that a human demonstrator can employ to give feedback on the robot behavior/performance. At first, we need a definition for **feedback**. This is "the signal containing information that human teachers explicitly communcicate to the learning agent through a human-robot (or human-computer) interface" [10]. We are able to choose, according to diverse factors such as the complexity of the task we are dealing with, the expertise of the human, available interfaces, one method with respect to the others, etc
Shortly speaking, we can say that all of the modalities can be grouped into two classes, in particular, feedback can be either in the *evaluative space* (the signal contains an **evaluation** about **how well** the agent has learned) or in the *transition (state-action) space*, where the teacher directly gives hints directly on **how to do**

better the task. Within the two categories *critics*[5] can be provided in an *absolute* or in a *relative* way. The former requires the teacher to know what is the optimal behavior while also requiring a non-negligible *cognitive load*.

2.2.3 Human feedback in the evaluative space

An **evaluative feedback** is nothing but a scalar value indicating the **quality** of the agent's behavior. The most recent papers about interactive learning adopted such a kind of feedback strategy taking inspiration on how domestic pets learn what is a good or bad behaviour.
More technically, the main idea is inspired by RL with the only difference that the reward/penalty are received from a human instead of having a *reward function*, the design of which is not trivial at all. However, sometimes it can happen that inconsistent feedback are provided leading to convergence issues.

Learning from Human reinforcements (absolute)

When one wants to give feedback in an absolute way, the **robot interprets the human signal as a reward or penalty with respect to an optimal policy that is implicitly known by the teacher**. Such a type of feedback are called **Human reinforcement**.
A possible way to implement such feedback is training the agent in an RL fashion, skipping the design of the reward function and letting the human to give judgements *at each time step* leading to the *Interactive RL* fields. Another implementation of absolute feedback in the evaluative space is within a framework called TAMER. Overall, human reinforcements are used for conveying to the robot what is wrong or right. The teacher is supposed to have complete knowledge about the tasks we are teaching. However, he/she does not have any idea on how the robot could improve by means of actions to be executed. On the other hand, the teacher has to be careful, since is not simple to revert a mistaken penalty.

Learning from Human Preference (relative)

In principle, using human reinforcements in some situations could reveal great results in the learning outcome, however, giving a reward or punishment on an absolute scale is not simple and sometimes is not feasible. To solve this issue, there are other methods in the evaluative space that **act by comparing two or more sequences and provide a preference score**. The teacher workload is reduced without any doubt. In particular, the direction toward which the policy should be

[5]In RL field is the way to say 'corrections'

shifted is given implicitly, since a sort of rating of different behaviors is given.
Lastly, an implementation of such concepts can be found in **Preference-based Policy Learning** (PPL) approaches [11], which explored how preferences and RL can be combined. An extension of such method to high-dimensional state spaces is possible by using some function approximators such as neural networks. In summary, learning from preferences reduce the cognitive load on the teacher since a general performance comparison constitutes the feedback. Similarly to reinforcements, mistakes in the teacher judgements have a negative impact on the policy convergence.

2.2.4 The TAMER framework

We have just seen that absolute feedback given in the evaluative space is something similar to what is done in RL: an agent interacting with the environment will gain a (positive/negative) **reward** from it, that is nothing but an evaluation on how good was its action. Here, in this framework, the objective is training the agent using **human rewards**. Neverthless, they cannot be treated in the same way as environment reward. This fact can be summarized as: "MDP reward is informationally poor yet flawless and human reinforcement is rich yet flawed" (Knox, Stone in [12]). In other words, if human rewards are used in the same fashion with respect to MDP reward, the richness related to the fact that the teacher does have some priors on the long-term consequences is lost. The so-called *shaping approach* allows to take into account both long-term consequences and optimality of the robot actions.
The framework **TAMER** (Training an Angent Manually via Evaluative Reinforcement), introduced by Knox and Stone, in 2008 [13] is seminal in this field, since it addresses the problem on how human rewards can be used in RL problems with discrete action spaces. Some other works manage the problem of taking into account both human and MDP rewards, in a framework they call TAMER+RL. In 2013 Knox and Stone implemented TAMER on a real robot, while in 2018 Deep TAMER (D-TAMER) were presented, considering an high-dimensional state space.

2.2.5 Human feedback in Transition (State-Action) space

In this context, the feedback signal contains insights on *how to do* to improve the task, explaining hot to perform a certain set of transitions. No quality assessment is considered here, it is assumed that the teacher has good insights about the task execution. Even in this context we have the split between *absolute* and *relative* feedbacks. In the former case, the demonstrator is expected to show what is the **optimal transition** for the state that the robot is visiting, the latter case expects that feedback are provided with respect to the trajectories the robot is executing

in that time step. The correct action is achieved after some iterations since the correction is not considered to be an optimal action, only an hint to proceed in a certain direction.

Corrective demonstrations (absolute corrections)

While using absolute feedback, the teacher can intervene: every time step, occasionally according to the decision of the human. Another aspect to be considered is that, the **corrections** could be only recorded or both recorded and exectuted. While absolute feedback in the evaluative space was closer to RL, absolute corrections are closer to LfD methods. The most important algorithms included in this bunch take inspiration or belong to a family of methodologies based on *DAgger* [15], which interactively records the corrective demonstrations. As in the case of TAMER, in the following we are dedicating a whole subsection to DAgger. Even if these methods are the most similar to our LfD approaches, they require the teacher to have expertise at solving the task. However, among the advantages, they reduce the errors since corrections are for improving current policy deviations; the workload is reduced since some methods require only occasional intervention.

Learning from human relative intervention

As far as **relative corrections** are concerned, the teacher is not supposed to know which are the actions for each state. On the contrary, the teacher is required to have sensibility about the consequences of changing the magnitude of a certain state-action transitions. That is, **What is the impact of slightly changing the policy?** Relative corrections can be either discrete or continuous. The methods employing relative corrections are: **Advice-Operator Policy Improvement (A-OPI)**, **Physical advice** and **COACH**. While the last have a dedicated subsection, for the others we briefly give some insights:

- **A-OPI** At each iteration, the policy is rolled out online, then in an offline manner. The human teacher individuates what is the fragment of the trajectory to be modified by using an associated **advice operator**, which changes the actions for each pair in the chosen trajectory part. Finally, there is a re-derivation of the policy which is based on the updated dataset. The *advice operator* stands for implementing the relativeness of corrections. For example, for a navigation task, a sample advice *acceleration* could lead to a multiplication of the current velocity by 1.1, which results in increasing the current action magnitude. This is the essentials of the method presented in a work from 2009 [16].
- **Physical advice** Sometimes, especially for manipulation tasks, it is useful to provide relative corrections by using the kinesthetic approach. Here, the

corrections are detected and computed by measuring the difference between the desired trajectory and the one which has been deviated by the teacher.

To finish the development of this part, we can say that relative corrections can be used whether the desired behaviour is not a-priori known or a expert user is not present.

2.2.6 DAgger: a framework employing absolute corrections

DAgger stands for **Dataset Aggregation**; it is an algorithm introduced by Ross et al. [15] that created a family of methods with this idea: instead of training only on the expert's demonstrations, DAgger iteratively collects new data from the learner policy and asks the expert to label it. The overview of the algorithm is the following:

1. **Initialization**

   - Collect a dataset $\mathcal{D}_1$ of expert demonstrations;
   - Train a policy π_1 on this dataset

2. **Iteration** For each iteration i

   - Rollout the current policy π_i to collect the trajectories;
   - The expert observes and labels the visited states with correct actions;
   - Aggregate the new pairs into the dataset
   - Train a new policy π_{i+1} using the aggregated dataset

3. **Repeat until performance convergence**

The objective is to minimize the difference between the agent and expert's actions fixing a certain state. Note that the original DAgger algorithm requires a correction for every state, which is infeasible for robotic tasks where state and action spaces are continuous ones. However, there are some variants that are more suitable, such as the **HG-DAgger** (Human-Gated DAgger) [17], where the most relevant difference is that the human does not label each state-action pair, but the intervention is limited to the cases in which the robot is clearly wrong. More specifically, there is always the presence of a novice policy obtained by demonstration data. Such a policy is used for executing the task, whether the taken actions are wrong, the human teacher takes control and executes a **corrective action**. In this case, the portion of corrected trajectory is recorded and this data is added to the dataset. This is the moment when the policy is retrained using the **aggregated dataset** which – again – contains both human demonstrations and human-corrected generated examples.

2.2.7 COACH: a framework employing relative corrections

We have introduced A-OPI and we have seen that actions can increase/decrease according to the given Advice-Operator. There is a framework called **COACH** (COrrective Advice Communicated by Humans) [18] that exploits binary feedback to encode the direction in which the change should occur, so the change is provided in relative terms where the magnitude/percentage is a hyperparameter to tune. This feedback signal is interpreted as an **Advantage Signal** $A(s, a)$. The obtained policy is then updated using policy-gradient methods such as REINFORCE or Actor-Critic. It is used in all those situations when the teacher can observe but not physically correct the wrong behaviors.
A summary table of the methods is given in Table 2.1:

	Absolute feedback	**Relative feedback**
Evaluative Space	Interactive RL TAMER, TAMER+RL Deep-TAMER	PPL
Transition space	DAgger HG-DAgger	A-OPI Physical Advice COACH, D-COACH

Table 2.1: Most important interaction methodologies grouped according types of *Space* and *Feedback*

2.2.8 Human-robot interfaces

Now, our main focus here is on *Which channel can be used for exchanging feedback-related information?* In principle, such a channel can be bidirectional, even if for IIL the direction is mostly from the human-teacher to the learning agent, being the learner passive.

Physical Contact with the Robot Embodiment

Here, we are essentially referring to kinesthetic guiding approach, which is the most important and simple, to enable the use of robot to non-expert users. It is widely used for robotic arms; however, this approach can be unfeasible for safety or mechanical reasons. A possible scenario is the one in which the robot size is outside the *human manipulability spectrum*; also for high-speed robot such a method is not applicable, other interfaces are for sure more suitable.

Physical Contact with External Devices

The **keys** are the most used external devices to give feedback to a robotic system. They are particularly useful when the information to convey are not so much as in the case of feedback provided in the evaluative space (reinforcements/preferences). Other external devices are the **joysticks** that can be used to encode more information in a single input, for instance, in navigation tasks. Moreover there are **6DOF interfaces** (e.g., space mouses) that can be used for manipulation tasks and provides correction of the end-effector position/velocity.

Contact-free interfaces

The types of interfaces we have presented so far provide an output type that is directly usable for learning methods. In other words, we do not have to solve the correspondence problem. If we want to use other **contact-free** interfaces, a *mapping problem* must be solved. Very often, some pattern recognition methods are used in order to extract from the output of the sensors useful information for the learning process.

Video

The videos of human executing tasks allow the user to perform a task with minimal hardware requirements and effort. There is a non negligible *caveat*: learning from videos is not simple due to the large state space, since they can be treated as a sequence of RGB images. Moreover, due to the correspondence problem, sometimes only the hand keypoints are detected and directly mapped to the end-effector position.
There are related works in which some pretrained models are used for *gesture recognition* for using COACH or facial expression for implementing the human feedback in TAMER, just for giving some examples.

Voice

The voice interface is an even-more complex contact-free interface. The most simple way to use the human voice is setting up a set of predefined words, which are mapped into specific actions, according to the adopted method. More complicated, but state-of-the-art, approches imply the use of *Large Language Models* (LLM) in order to exploit the richness of human languages.

2.3 Challenges in LfD

The main objective of the research in the field of LfD is enabling the robot to learn in an efficient and fast manner from humans, while operating in environments whose characteristics are not priors. There are some challenges that are not *only* related to robotic assembly, but in general to the use of the LfD framework.

2.3.1 Generalization

Generalization is a concept which comes from *cognitive psychology* [4]. It can be summarized as the possibility for a human to react to new stimuli according to the similar ones coming from previous experiences. Many reaserchers have studied the concept of *generalization for artificial systems*. Most ML models are based on this fundamental idea. Sometimes reasonable assumptions are made in order to obtain it in ML. For example, in a supervised learning we use the *iid* one (data are assumed to be independent and identically distributed).
As stated before, in LfD this will not hold anymore due to the intrinsically correlated structure under the hood of demonstrations. Here we want to add something more about **intratask** and **intertask** generalization. The former characterizes the ability of the algorithm to adapt itself to new unseen conditions (e.g., new initial and final location for pick-and-place); the former is also known as *skill transfer* and is related to the use of the learning outcomes for novel similar tasks.

2.3.2 Simultaneous learning of low and high-level behaviors

We have seen that different learning strategies operate at different level of abstraction while having different learning outcomes. In the literature, there are some approaches that attempt to pursue multiple outcomes simultaneously, which is at different level of abstraction. This is related to the issue that low-level and high-learning methods usually reciprocally ignore themselves.

2.4 Manipulation tasks in robotics

After having introduced the LfD framework, analyzed its pipeline and illustrated the main aspects about context-awareness and safety in HRI, now we want to analyze more in details fundamental manipulation tasks (namely *peg insertion*, *pick-and-place*) and conceptually put them together for carrying out robotic assembly tasks.
A top-down approach has been followed: from robotic assembly and related problems to elementary subtasks of which it is made up of. Ad-hoc references have been made

to the LfD pipeline in order to better formalize the notions of **task hierarchy** (with related learning methods) and **motion compliance**.

2.4.1 An overview of manipulation tasks

Generally speaking, a **robotic assembly operation** requires that – following a certain logical/semantic reasoning – two or more parts (workpieces) are put together to form a final artifact, which is the so-called **assembly** [19]. Actually, this can involve many manipulation tasks together, to be performed according to a predefined order. The work *«Obstacles and Opportunities for Learning from Demonstration in Practical Industrial Assembly: A Systematic Literature Review»* investigates the main aspects behind 77 articles on *LfD for manipulation tasks* and splits the basic learnable skills into two categories, which are both essential, even if, only the former have greater attention from the scientific community. They are:

1. **Mating skills**: They refer to the fact that two individual pieces are *positioned* and *aligned* in a way that they can be joined later.
2. **Joining skills**: They refers to the fact of putting together the parts that formerly were aligned in a *permanent* or *semi-permanet way* using specialized grippers.

With the aim of making more effective the comparison between the classes from a research interest point of view, from [20] the following statistics can be retrieved: among 77 sampled inherent papers only 21% were related to joining skills while, as far as the mating skills are concerned the two most studied ones are **peg-in-hole (or peg insertion)** (46%) and **pick-and-place** task (23.4%). Motivated by such a result, in the following we are going to analyze more in details the most explored mating and joining skills.

2.4.2 Mating skills

Peg-in-hole

It could seem trivial, but the ability of inserting an object (called the **peg**) into a specific hole is an essential task in manufacturing. This is used as a benchmark case study for automated assembly.
Even if it may seem very simple, carrying out such a task by using a manipulator is not easy at all. Substancially, the challenge stays in the fact that the object must be inserted in the required hole with a relatively high precision to respect given and *small tolerances* (sometimes less than 1mm). In such a case, particular attention must be devoted to the *motion control*, since it is required to adjust the motion itself by taking into account *contact forces*. A dedicated section in which

the different motion control desiderata are explained follows this introductory part. See [20] for a detailed summarizing table with papers treating peg insertion task. The paper [21] treats a practical example of a toy-car robotic assembly which requires multiple peg-in-hole operations.

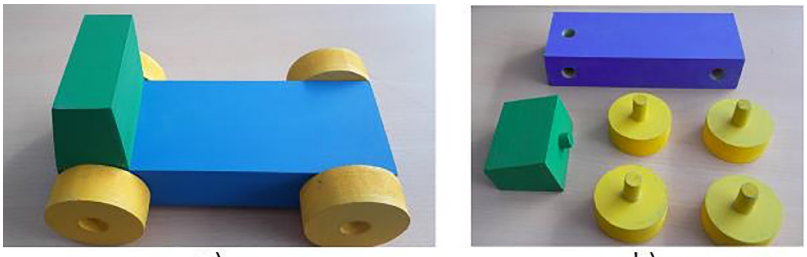


Figure 2.6: Parts of the car and assembled car. In this case the assembly task can be structured in a sequence of *peg-in-hole* subtasks. [21]

Pick-and-place

Another (apparently) simple task is the one about ***picking*** *up an object from a location and* ***placing*** *it into another location.* This is among the most used skills for manipulation tasks. The fact that an assembly task is correctly performed is highly dependent on how well the pick-and-place is performed, since multiple objects must be handled. Related to the pick and place task, there are other elementary skills at a lower layer: for example *reaching*, *grasping* and *stacking.* Such skills are different from peg-insertion for two reasons:

1. A higher uncertainty is allowed since less strict tolerances requirements have to be fulfilled;
2. Contrary to peg-insertion, the final outcome does not have a deterministic success, since a valid final result can be achieved following different path alternatives.

Such differences need different control requirements, and – roughly speaking – more attention on the trajectory can be paid instead of the contact which, in this case, is not stricly relevant. Hereafter, we will see that a complete pick-and-place task requires a hierarchical learning process: low-level skills are, in some way, combined to fulfill the task.

2.4.3 Joining skills

Whether there were the necessity to robustly connect the assembly components in a either permanent or non-permanent way, joining skills are needed. However this

category of low-level skills including *screwing, gluing, soldering and hammering*, are the least studied ones in the literature. Despite this fact, there is a small number of papers which focus on how to fix the parts of an assembly together. The reasons behind the lack of attention to joining skills are explained as follows:

- **Process-Specific** They are less generalizable and industry-specific tasks.
- **Requires more specific validation** Think about on how complex can be evaluating that two parts are well-fixed together. On the other hand verifying alignment and/or insertion is a much easier task.
- **Tool complexity** Each joining skill requires specific tools, the presence of which makes more and more complex the overall system.

2.4.4 Research problems in Robotic assembly

Previously, we introduced the robotic assembly and related skills to learn in a LfD scenario. Instead, in this paragraph we will focus on some problems that occur when one decided to learn robotic assembly from demonstrations. In doing this we are going to follow in part the structure proposed in [19].
In such a context, the **auto-pilot software** of which the learner robotic arm is equipped has to: (i) convert the sequences of tasks into individual movements, (ii) estimate the pose of assembly parts, (iii) calculate forces and torques according to the generated trajectories.

2.4.5 Pose estimation

Pose estimation is related to determine the **position** and the **orientation** of the parts to be assembled. It is a key process to guide the motion planning of the robot through the learned model from human demonstrations.More important, this is essential to generalize the motion when the parts change their positions. Different methods can be used to fulfill such a task. The most comnon method (used also in the work [21] about the toy-car assembly) is the employment of RGB-D cameras (e.g., RealSense, Kinect); moreover, very often deep-learning methods are also used (e.g., PoseCNN).

2.4.6 Force estimation

Some papers also refer to this problem as *Force/Torque sensing*, which is related to the measurement of the contact forces/torques that are generated during the interaction with the environment. This issue especially holds for tasks such as peg insertion. *Force/Torque sensing* is the process which allows the correction of motion commands during the execution of uncertain task. Usually, in this case,

wrist-mounted force/torque sensors are employed. The presence of triggering events and thresholds allows a conditional execution of the manipulation task.

2.4.7 Assembly sequences

This is the essence of what in the next paragraph will be called **high-level task learning or symbolic plan learning**. Here, the order and also the logic of elementary actions, naturally required by a multi-step assembly task is decided. Knowing a sequence of operations is useful for several reasons:

1. A sequence is a sort of encoding of dependencies (for example the insertion of B, must be performed before the insertion of A);
2. Having such a dependency scheme allows the management of **failures and recovery** enabling context-aware robot operations.

Often, this requires task segmentation before any structure encoding.

2.4.8 Robotic assembly and LfD pipeline

At the beginning we have presented for the first time the LfD pipeline also by giving a graphical representation for it. Figure 2.1 has an important role through these research reports due to the fact that effectively summarizes which are the building blocks of our work. Therefore, the objective here is highlighting some aspects, **specific to manipulation tasks**, which can be directly mapped on that guiding scheme.
In [20], the statistics about the used demonstration methods and learning outcome are given. In particular, it appears evident that the most used method in LfD manipulation task is the kinestetic approach, followed by passive observation mehtods. On the other hand, the great majority of the considered papers had as a learning outcome a **policy trajectory based** one.

2.4.9 Robotic assembly, subtasks hierarchy, task learning

As stated before, a full robotic assembly task can be divided into smaller steps which in turn are called subtasks. Such subtasks are logically organized into a hierarchy (see Figure 2.7). This concept is crucial for robotic assembly.
In the literature sometimes full-demonstration task are used, otherwise single demonstrations are provided. One method could be more suitable than the other.

In this context the learning process (whether full-task demonstrations are provided) can be split in two phases:

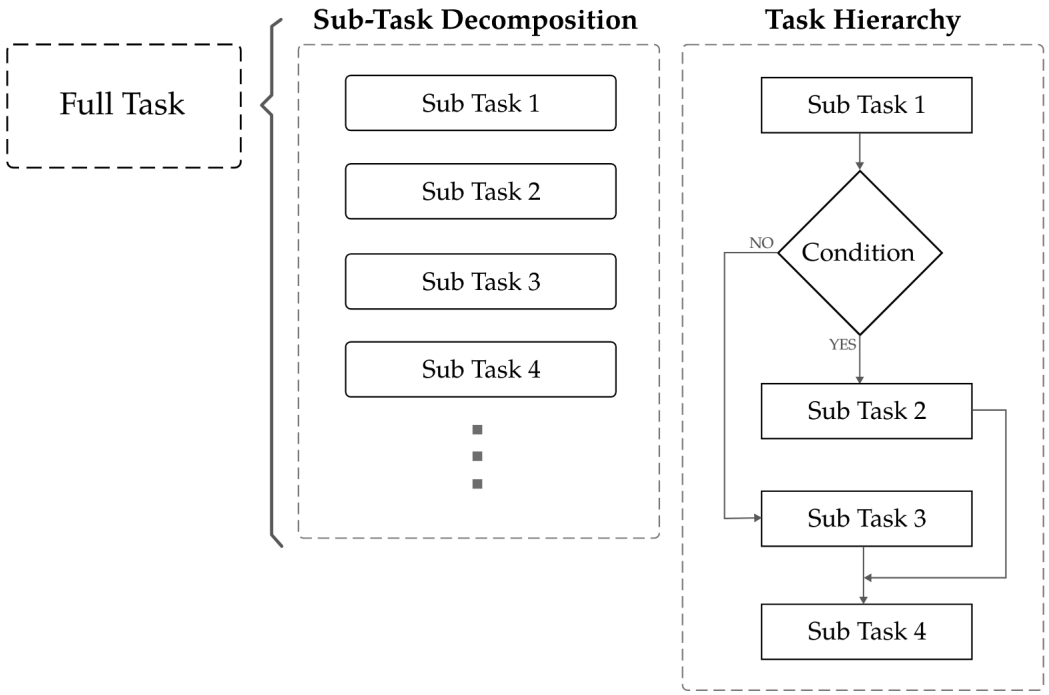


Figure 2.7: Any Full task can be seen as a sequence of Subtasks. After having been decomposed/segmented, subtasks can be used to structure a plan according to conditions [4]

1. **Phase #1: Task segmentation and hierarchy**: the given demonstration is split into smaller *subtask chuncks*, which are then learned separately in the second phase. Moreover, such subtasks are organized into a *task hierarchy*. The segmentation is performed using both spatial and temporal reasoning: the former is useful to uncover the underlying subtasks, the latter to reveal the temporal dependencies among the individuated chunks. The most used methods for task segementation are GMM for spatial segmentation and Hidden Markov Models (HMM) for temporal segmentation [19].
2. **Phase #2: Skill learning**: using the segmented trajectories, lower level tasks (reaching, grasping) can be learned using different methods. We have seen that for skill learning the most used methods are DMP and GMM/GMR.

It is remarkable that the hierarchy obtained by demonstrations analysis can be either linear or conditional. For explaining this aspect we can use the two most common mating skills:

- In the **pick-and-place case** the object must be detected, reached, moved and then placed. In absence of particular requirements the action to be performed

can be arranged in a *linear fashion*.

- In the **peg insertion** case study such a reasoning does not hold. We are in presence of tight tolerances, therefore the first time we are executing the task we can fail. In this case a repositioning behaviour must be learned in order to execute again the task. You can easily imagine that such a sequence is a *conditional one* with respect to the Figure 2.7

Other high-level learning methods are available whether single subtask demonstrations are available. Behavior tree (BT), Decision tree (DT) and Finite State machine (FSM) can be exploited, having the prior knowledge of the task structure and then avoiding to segment the demonstration to uncover it. One of the main differences with respect to HMM is that the transitions are deterministic, and hence they are not robust to uncertain demonstrations.

2.4.10 Mating skills issues: motion-based vs contact-based demonstrations

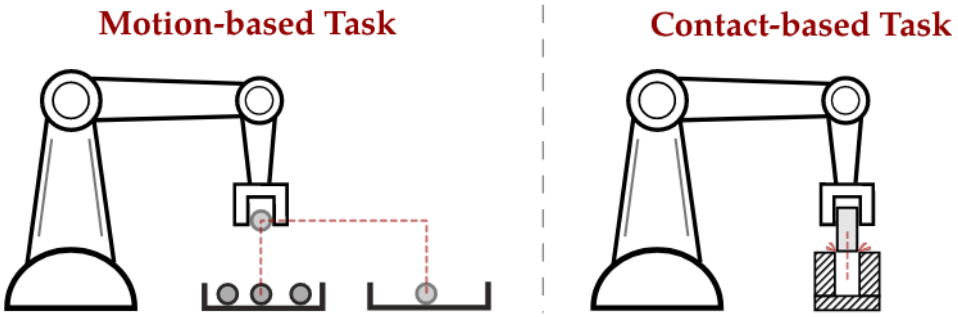


Figure 2.8: Motion-based and **contact-based** demonstrations: while in the former case the the learning of interaction is less important than in the latter. [4]

According to the type of task, demonstrations can be divided into **motion-based** and **contact-based**. They are very different in the way they are able to interact with the environment. Pick-and-place demonstration are in the first category and the contact with the environment is limited. The most important thing is that the given trajectory from a kinematic point of view is coherent with the prescribed task. On the other hand, for tasks such as insertion, replicating the motion does not suffice, since the robot has to learn **how to interact with the object** in order to respect the tolerances and not to fail.
A fundamental notion underlying the difference between motion-based and contact-based demonstrations is the **compliance**. We say that a robot is **compliant** if it is able to regulate its movement in response to contact forces. For modeling

purposes the compliance can be represented using a very well known system: the mass-spring-damper. This is a proxy for modelling how much the trajectory of the robot is adapted in response to the contact with the environment. Indeed, we talk about low-compliance and high-compliance.
To sum up, we want to describe what is the impact on **what is learned** from the robot side. In the case a low-compliance is considered, the main objective is to replicate the behavior at a kinematic level, at the opposite when high-compliance is considered, the manipulator learns how to interact with the objects.
The works in [22] and [23] could be references for further extension of the presented issues.

2.5 Safety aspects in HRC

So far, we have introduced the main aspects related to Human-Robot collaboration (HRC), we have seen the potentialities and the things to be taken into account whether any industry decided to operate in such a setting. In addition, we have only mentioned the fact that the most important aspect to be taken into account is **safety**. Such an aspect is one of the main topic treated in the current report.
In particular, the objective now is to better analyze all of the issues related to safety, in a context where the dynamic and, sometimes, highly unstructured nature of the environment makes it more and more challenging. We will briefly give a classification of **HRC frameworks** allowing a better and deeper comprehension. Next, a brief overview of the **safety modes** dictated by the ISO/TS 15066 standard is given. Finally, a summary for possible **safety strategies** which are standard-compliant has been made.

2.5.1 Three different levels for HRC

Inspired from the work of Arents et al. [24] a possible classification for HRC frameworks is the one based on *Coexistence*, *Cooperation*, *Collaboration*. In particular:

Coexistence Here we find humans working in a partially or completely shared space with a robot, however there are no shared goals.

Cooperation Human works in a shared workspace with a robot sharing a common goal.

Collaboration This is the strongest modality of HRC, since a human and a robot share not only the workspace and the goal for which they are operating, but also the object/artifact on which the work is performed.

It is remarkable that safety must be the highest priority in the setup of any HRC workcell. All the HRI methods have to be developed in accordance with safety standards.

In Figure 2.9, yellow and green circles represent the robot workspace and the human workspace respectively , while the intersection is the human-robot shared workspace.

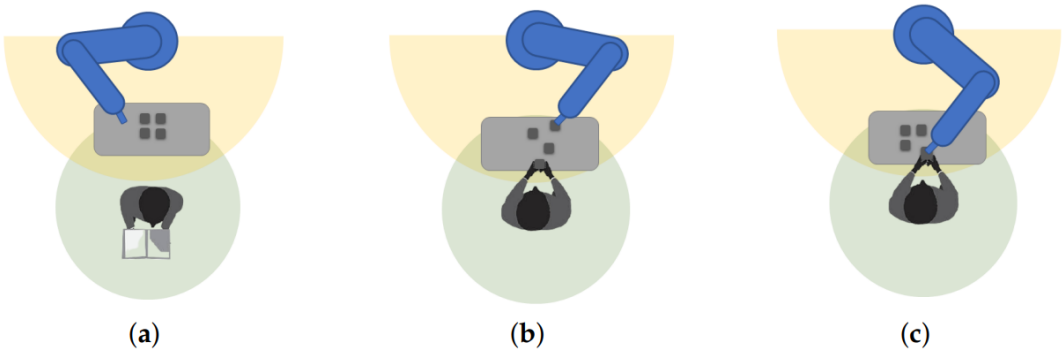


Figure 2.9: Collaboration levels adapted from [24]: (a) Coexistence, (b) Cooperation, (c) Collaboration

2.5.2 Safety standards for collaborative robots and HRC

Generally speaking, *safety assurance* is a crucial prerequisite in designing any type of machinery and machinery systems covering both industrial robots and collaborative robots (cobots). International standards from the *International Organization for Standardisation (ISO)* can be divided into three categories or types. While Type A and Type C are for basic and general machinery design, Type C standards are for *specific machinery design* which embraces also cobots. The guidelines provided by the ISO/TS 15066 [25] standard include some primary safety modes which represent a reference point in the implementation of HRI methods:

1. **Safety-Rated Monitored Stop (SRMS)** Human and robot cannot share the same workspace. More specifically, the robot in its workspace can perform any operation without restrictions. The worker can enter the robot space only if the *stop status* of the robot has been triggered[6].
2. **Hand Guiding (HG)** With respect to the previous mode a manual guidance device enriches the framework. Such a device can be used to cooperate with the robot whithin predefined safety ranges.
3. **Speed and separation monitoring (SSM)** Here, workspaces can be shared with the only restriction that a *protective distance* must be mantained all

[6]Here some cameras or sensors can be used in order to active the stop status whether a human is detected in the robot's range of action.

times[7] The use of tracking devices in this context is widespread. A guiding principle could be to keep a reduced robot operating velocity whether the reciprocal distance reduces.

4. **Power and force limiting (PFL)** The robot and the human can interact with a more reduced distance. Risks have to be minimized adopting suitable safety measures which are employed to keep the contact forces below specific predefined *thresholds* involving passive and/or active measures from the robotic control system.

The operating modes represent general requirements for HRC contexts, however, there are not detailed documentation about the implementation of such requirements in industrial setting. Next, a possible classification for HRC strategies is possible and is presented following the structure adopted in [26] where they are grouped into two general categories: *pre-collision* and *post-collision* safety strategies.

2.6 Safety strategies for HRC

2.6.1 Pre-collision strategies

Pre-collision strategies are adopted to prevent harmful contact between a collaborative robot and a human by predicting and/or detecting the relative movements of both human and robot **in advance**. Here, a further subdivision can be introduced with the aim to distinguish the underlying used technology.

Sensor-based safety strategies: focus on the reciprocal distance

These utilise sensors to observe and optimsise the motion of a collaborative robot in HRC context in order to make the operating phase suitable to the effective distance between the robot and the human collaborator. They are useful to address the SRMS and SSM safety modes. Possbile sensors leaveraging this type of strategies are cameras, radar and AR sensors. Among the others, cameras are the most usable solution in supporting HRC in industries due to their low cost and affordability.

Prediction model-based safety: focus on the human behaviour

The use of models predicting the human behavior can be devisable to better support the safety assurance in HRC frameworks. Both human motion, and in general the human behaviour prediction can be addressed by using stochastic or learning-based models. Example of statistic models are Gaussian Mixture Models

[7]The standard exactly defines how such protective distances are designed.

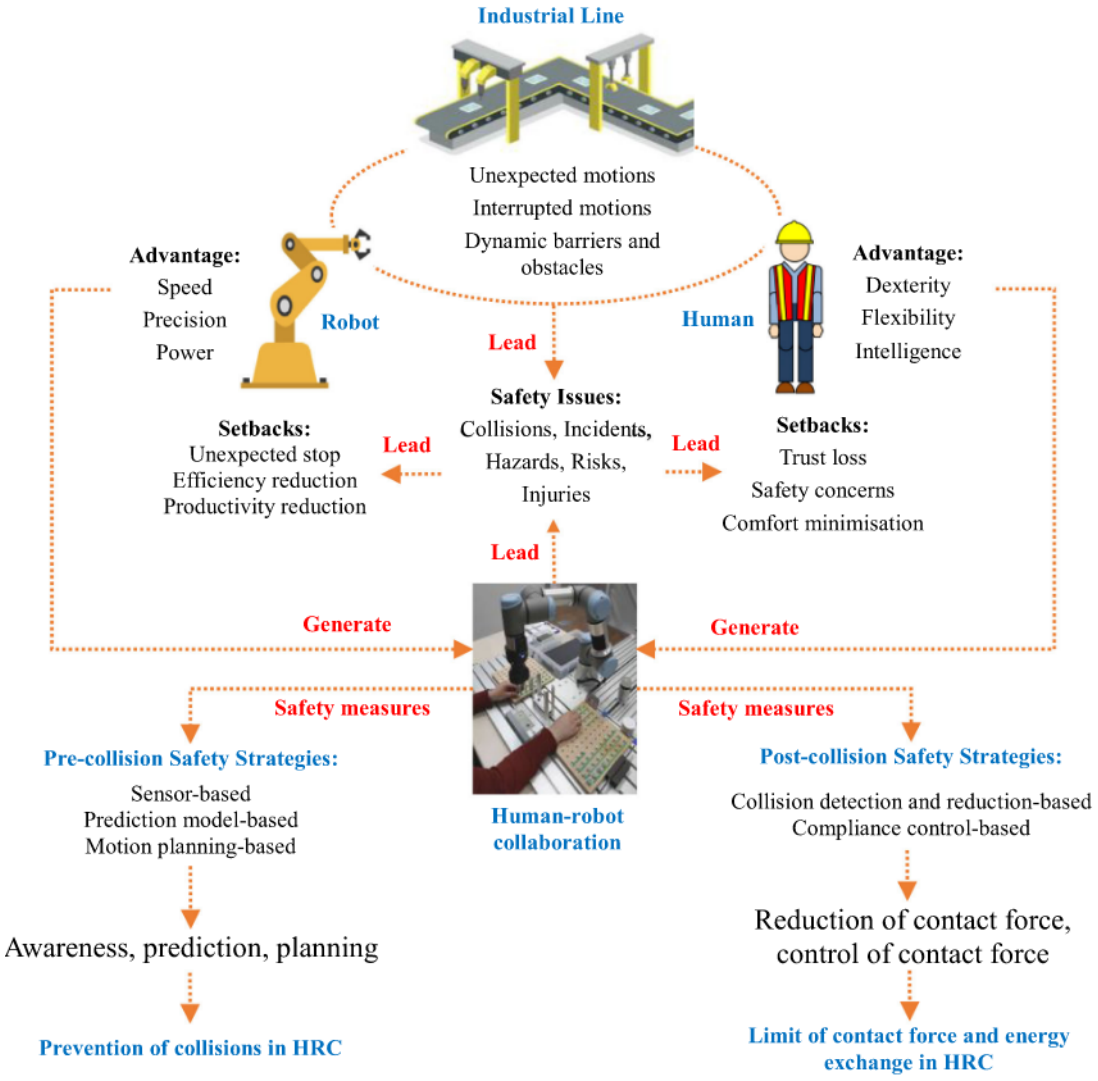


Figure 2.10: Human-Robot collaboration: safety aspects [26]

(GMM), Hidden Markov Models (HMM) and Gaussian Process Regression (GPR). These methods tends to extract patterns from human demonstration data in a given fixed environment. Nonetheless, statistical models suffer of some important issues:

1. They struggle generalizing the learned patterns;
2. If large datasets are available, they do not scale up.

Tackling such problems is possible if alternative ML-based models are exploited. We are talking about, mainly of deep-learning architectures such as MLP, CNN

and recurrent models (e.g., RNN, LSTM), which can extract spatial and temporal dependencies from visual data.
Summarizing, stochastic models act in a unsupervised fashion, so that they do not need of large amount of labelled data to be obtained. More accurate models are the learning-based ones. Leveraging on them complex industrial contexts can be addressed. The main disadvantage here is that the training phase is time consuming, and even more important labelled datasets are needed to be compliant to the supervised setting in which they find place.

Motion planning-based safety: focus on the robot behaviour

While in the previous two approaches the focus was on the relationship between human and robot in term of reciprocal distances or human behaviour, here the main focus is on the robot and in particular to the trajectories it carries out. The common denominator to all of the proposed approaches in this section is the *planning of the trajectory in a way that avoid behavioural interferences between the human, the robot and the manipulated object in one of the HRC levels.*
Heuristic approaches such as *Rapid-exploration Random Tree (RRT)* can be used when the planning of the trajectory occurs in an offline and static manner. The main advantage of such techniques is the existence of robust and efficient algorithms. Also in this context learning-based technique can be employed in dynamic environments with the usual disadvantage of being computationally expensive.

2.6.2 Post-collision strategies

Post-collision strategies are tailored for implementing the last presented PFL safety mode where there is high level of interactivity and vicinity between the human and the robot.
Here, no sensors are needed to avoid possible contacts; so the essence is on the reduction of the contact force and the energy exchange that neither the robot nor the human are damaged. Here, passive strategies are adopted to ensure the HRC safety. A first measure that can be adopted is the realization of lightweight in order to reduce the impact force. This as a drawback, since it forces to a reduction of the accuracy and of the payload.

Collision detection and reduction of contact forces

We have seen for the pre-collision techniques that sensors were used to adapt the robot motion according to the reciprocal proximity with the operator. The idea for post-collision strategies is to reduce the contact forces once the collision has been detected by using **internal sensors** instead of external ones.
In this field, the most common approach is using supervised learning to classify

dangerous trajectories and stimulus leading to human-robot collisions; they are (again) neural-network based methods. Here, a large quantity of collision data is needed for training the model. A further changes which occurs is that the ability of executing a task changes according to the operator, and since collision data requires explicitly an interaction is not so easy to adapt that model to a new human worker.

Compliance control of contact forces

The main aspect that is considered here is the control of the contact force and of the energy exchanged between human and robot. Here, an hybrid control of both force and position is realized. Actually, the force and position trajectory required so that the interact could safely happen must be available this limits the applicability of these strategies to the HRC safety. The use of intelligent learning methods can compensate the presence of noise and uncertainty. On the other hand, the fact that the training process is time-consuming and results in more complex models limits the usability of the aforementioned methods.

Chapter 3

An overview of Learning from Demonstration approaches

The objective here is to provide some theoretical insights about the state-of-the-art methods for Learning from Demonstration. We are indicating in the following the methods with M1, M2 and M3. Main reference for this part is the paper «Learning from demonstration for autonomous generation of robotic trajectory: Status quo and forward-looking overview» by Li et al. [6].

3.1 M1: Behavioral Cloning (BC)

Behavior Cloning or **Behavioral Cloning** (BC) is one of the simplest methods for implementing imitation learning.The goal is to learn a policy that **mimics expert behavior** by treating it as a *supervised learning problem*. In mathematical terms, at the end we are obtaining something similar to:

$$\boxed{\pi_\theta(s_t) \approx a_t} \tag{3.1}$$

where: π_θ is the learned policy parametrized by θ (learned parameters), and s_t, a_t are respectively the state and the action at each time step t.

3.1.1 Mathematical formulation in LfD

Given the dataset $\mathcal{D}$ of expert demonstrations, we want to find the policy parameters θ that minimizes the expected loss. This is a *gradient descent minimization*, defined as:

$$\theta^* = \arg\min_{\theta} \sum_{t=1}^{N} \|\pi_\theta(s_t) - a_t\|_2^2 \tag{3.2}$$

From this you can understand that BC is agnostic to the type of function approximator one wants to use: linear models, radial basis functions, multi-layer perceptron and so on.

3.1.2 Goal Conditioned Behavior Cloning

There is a variant of the method in which a goal g is also added to the input of the policy

$$\pi_\theta(s_t, g) = a_t \tag{3.3}$$

An example of the goal is the case where you want to grasp a certain object which is put in position $g = (x_g, y_g, z_g)$. The policy is trained by using the dataset built as:

$$\begin{aligned} &\text{Input (State)} \qquad [x_t, y_t, z_t, x_g, y_g, z_g] \\ &\text{Output (Action)} \qquad [\Delta x, \Delta y, \Delta z] \end{aligned}$$

The learned model is executed as follows, at each step t:

1. The state s_t is observed; this gives x_t, y_t, z_t
2. A goal g is given
3. The action a_t is computed by using the learned policy π_θ
4. The action is executed; this for a robotic manipulator implies the employment of the Inverse Kinematics to transform Δ_x in Δ_q.

3.2 M2: Gaussian Mixture Model and Gaussian Mixture Regression (GMM-GMR)

A **Gaussian Mixture Model (GMM)** is a *generative probabilistic model* that assumes data is generated from a mixture of several multivariate normal distributions with *unknown parameters*.
In simple terms, GMM assumes that the observed data distribution is composed of *multiple clusters*, each of which can be modeled by a Gaussian distribution, and then the data points are generated by randomly selecting one of these clusters, finally drawing a sample from it. The use of GMM in robotics gives several advantages: (i) multi-modal data can be modeled; (ii) the correlations between different variables can be captured. The use of GMM in LfD were introduced in 2016 by Calinon in «A tutorial on task-parameterized movement learning and retrieval» [27].

3.2.1 Mathematical formulation

Given a D-dimensional continuous variable $x \in \mathbb{R}^D$, the GMM joint probability density function $p(x)$ is defined as:

$$\boxed{p(x) = \sum_{k=1}^{K} \pi_k \, \mathcal{N}(x|\mu_k, \Sigma_k)} \tag{3.4}$$

where:

- K is the number of Gaussian components;
- π_k are the mixture weights, with $\sum_{k=1}^{K} \pi_k = 1$ and $\pi_k \geq 0$;
- $\mathcal{N}(x|\mu_k, \Sigma_k)$ is the multivariate normal distribution with mean μ_k and covariance matrix Σ_k:

3.2.2 Training a GMM

A GMM is typically trained using the **Expectation-Maximization (EM)** algorithm which is composed of two steps:

1. **E-step** (Expectation step): it estimates the probability that each component generated each data point;

2. **M-step** (Maximization step): it updates parameters (means, covariance and mixture weights) to maximize the likelihood;

3.2.3 The Expectation-Maximization (EM) algorithm

The objective of such an algorithm is to maximize the log-likelihood of the observed data given the parameters of the GMM components.

1. Initialization Set initial values for μ_k, Σ_k, π_k via K-mean or randomly;

2. E-step Compute posterior probabilities that component k was responsible for generating the (D-dimensional point) x_i for each data point i:

$$\gamma_{ik} = \frac{\pi_k \, \mathcal{N}(x_i|\mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j \, \mathcal{N}(x_i|\mu_j, \Sigma_j)} \tag{3.5}$$

   we recall that this is quantifying the *responsibility* of component k for data point x_i;

3. M-step Parameters of the clusters are updated according to their responsibilities:

$$N_k = \sum_{i=1}^{N} \gamma_{ik} \tag{3.6}$$

$$\pi_k = \frac{N_k}{N} \tag{3.7}$$

$$\mu_k = \frac{1}{N_k} \sum_{i=1}^{N} \gamma_{ik} x_i \tag{3.8}$$

$$\Sigma_k = \frac{1}{N_k} \sum_{i=1}^{N} \gamma_{ik} (x_i - \mu_k)(x_i - \mu_k)^T \tag{3.9}$$

   At this point the log-likelihood can be computed (see next section);

4. Convergence check If the log-likelihood increase is lower than a certain threshold or the maximum number of iterations is reached, stop; otherwise go back to step 2.

3.2.4 The Log-likelihood for GMM

In the context of GMM, the log-likelihood is a measure of how well the model parameters explain the observed data. Given:

- A dataset $\mathcal{D} = \{x_1, ..., x_N\}$ with $x_i \in \mathbb{R}^D$
- A GMM with parameters $\Theta = \{\pi_k, \mu_k, \Sigma_k\}_{k=1}^{K}$

The **likelihood** of the data under the GMM is given by:

$$L(\Theta|\mathcal{D}) = \prod_{i=1}^{N} p(x_i|\Theta) = \prod_{i=1}^{N} \sum_{k=1}^{K} \pi_k \, \mathcal{N}(x_i|\mu_k, \Sigma_k) \tag{3.10}$$

Then, the **log-likelihood** is simply the natural logarithm of the likelihood:

$$\boxed{\log L(\Theta|\mathcal{D}) = \sum_{i=1}^{N} \log \left(\sum_{k=1}^{K} \pi_k \, \mathcal{N}(x_i|\mu_k, \Sigma_k) \right)} \tag{3.11}$$

This measures how well the current GMM model explains the data: in this context, the higher the better. A feasible stopping criterion is for example:

$$|\log L^{(new)} - \log L^{(old)}| < \epsilon \tag{3.12}$$

3.2.5 Visualization and model selection

Different plots of the log-likelohood with different independent variables can be used to monitor the training process. In particular:

- *Monitor the convergenxe of the EM process* by plotting the log-likelihood against the number of iterations;
- *Select the number of components K* by plotting the log-likelihood against different values of K and looking for the elbow point.

After having trained a GMM, it can be used for extracting the so-called *generalized trajectory* via **Gaussian Mixture Regression (GMR)**.

3.2.6 Gaussian Mixture Regression (GMR)

Applying this method to LfD, GMR is used to generate a *smooth* and *generalizable* representation of the demonstrated trajectories.
In the following, we are giving a practical procedure that summarizes the main steps for using GMM/GMR in LfD context:

1. Demonstration collection Collect multiple trajectories of the type
$$\{(t_i, x_i)\}_{i=1}^{T} \tag{3.13}$$
each demo is a time series of D-dimensional states x_i. At the end, all demonstrations are stacked into a single dataset:
$$\mathcal{D} = \left\{ \begin{bmatrix} t \\ x \end{bmatrix}_1 ..., \begin{bmatrix} t \\ x \end{bmatrix}_N \right\} \tag{3.14}$$
2. GMM training We train a GMM to model the joint distribution over time and states:
$$\xi = \begin{bmatrix} t \\ x \end{bmatrix} \in \mathbb{R}^{D+1} \tag{3.15}$$
where t is the phase variable (independent) and x is the state variable (dependent). The GMM is trained using the EM algorithm as described before. At the end the learned model is:
$$p(t, x) \sim \sum_{k=1}^{K} \pi_k \, \mathcal{N}(\xi | \mu_k, \Sigma_k) \tag{3.16}$$
this can be used to estimate:
$$\boxed{\hat{x}(t^*) = \mathbb{E}[x | t = t^*]} \tag{3.17}$$
with a given uncertainty $\mathrm{Var}[y | t = t^*]$ yielding to a smooth trajectory parametrized by time;

3. Perform GMR (trajectory generation) Given a query time t^*; the following steps are performed for each component k of the GMM:

   - *Split mean and covariance* into contributions due to time and state:

$$\mu_k = \begin{bmatrix} \mu_k^t \\ \mu_k^x \end{bmatrix}, \quad \Sigma_k = \begin{bmatrix} \Sigma_k^{tt} & \Sigma_k^{tx} \\ \Sigma_k^{xt} & \Sigma_k^{xx} \end{bmatrix} \tag{3.18}$$

   - *Compute conditional mean and covariance* of the state given the time t^*:

$$\hat{\mu}_k^x = \mu_k^x + \Sigma_k^{xt}(\Sigma_k^{tt})^{-1}(t^* - \mu_k^t) \tag{3.19}$$

$$\hat{\Sigma}_k^{xx} = \Sigma_k^{xx} - \Sigma_k^{xt}(\Sigma_k^{tt})^{-1}\Sigma_k^{tx} \tag{3.20}$$

   - *Compute responsibilities of component k given t^** in the following way:

$$h_k(t^*) = \frac{\pi_k \, \mathcal{N}(t^* | \mu_k^t, \Sigma_k^{tt})}{\sum_{j=1}^{K} \pi_j \, \mathcal{N}(t^* | \mu_j^t, \Sigma_j^{tt})} \tag{3.21}$$

   - *Fuse results from all components* to get the final output:

$$\hat{x}(t^*) = \sum_{k=1}^{K} h_k(t^*) \; \hat{\mu}_k^x \quad \text{MEAN (point)} \tag{3.22}$$

$$\hat{\Sigma}(t^*) = \sum_{k=1}^{K} h_k^2(t^*) \; \hat{\Sigma}_k^{xx} \quad \text{COVARIANCE (uncertainty)} \tag{3.23}$$

   From this we have a fundamental result. The **generalized trajectory** thus obtained is given by the set of points $\{\hat{x}(t)\}_i$ with uncertainty $\{\hat{\Sigma}(t)\}_i$ for $t \in [0, T]$.

3.3 M3: Dynamic Movement Primitives (DMP)

A **Dynamic Movement Primitive (DMP)** models the motion as a set of nonlinear differential equations that lead the system towards a goal.
Each dimension of motion is modeled using a second-order differential equation inspired by a damped spring-mass system, with a **learned forcing term** to capture complex shapes.
It is important at this stage to understand what is the 1D formulation of a DMP, so that, a posterior extension to multiple dimensions can be done. The reference paper for such a method is in «Dynamical movement primitives: learning attractor models for motor behaviors» by Ijspeert et al. [28].

3.3.1 1D DMP fundamental equations

The fundamental equations of a DMP are the ones related to the *transformation system* and the *canonical system*.

Transformation system

This part of the model generates the actual trajectory. The equations are:

$$\tau \dot{v} = \alpha_z(\beta_z(g - y) - v) + f(x) \tag{3.24}$$

$$\tau \dot{y} = v \tag{3.25}$$

where:

- τ is a temporal scaling factor;
- v is the velocity;
- α_z and β_z are positive constants that determine the system's damping and stiffness;
- y is the position (state variable);
- g is the goal position;
- $f(x)$ is the nonlinear forcing term that modulates the trajectory shape;
- x is the phase variable from the canonical system.

Canonical system

This acts as a time surrogate in the sense that it is decoupled from the actual time, and it is used to drive the transformation system. The equation is:

$$\tau \dot{x} = -\alpha_x x \tag{3.26}$$

where α_x is a positive constant that determines the rate of decay of the phase variable x. Initial condition is $x(0) = 1$ and as $t \to \infty$, $x \to 0$.

Forcing term

This is an important part of the DMP as it allows to encode complex behaviors. The shape of such a term is learned from demonstrations. A common choice is to use a weighted sum of Gaussian basis functions:

$$f(x) = \frac{\sum_{i=1}^{N} \psi_i(x) w_i}{\sum_{i=1}^{N} \psi_i(x)} \cdot x \cdot (g - y_0) \tag{3.27}$$

where:

- N is the number of basis functions;
- $\psi_i(x) = \exp(-h_i(x - c_i)^2)$ is the i-th Gaussian basis function with center c_i and width h_i[1];
- w_i are the weights that need to be learned from demonstrations.

3.3.2 Learning DMP parameters from demonstrations

The steps to follow in order to learn DMP parameters from human demonstrations are:

1. Record $x(t)$, $\dot{x}(t)$, $\ddot{x}(t)$
2. Solve for the desired forcing term

$$f_{target} = \tau\dot{v} - \alpha_z(\beta_z(g - y) - v) \tag{3.28}$$

3. Learn weights w_i using regression. In particular:

$$f_{target} \approx f(x(t)) = \sum_i w_i\phi_i(x) \quad \phi_i(x) = \frac{\psi_i(x)}{\sum \psi_i(x)} \cdot x \cdot g - x_0 \tag{3.29}$$

4. The solution is obtained by integrating the system of differential equations

$$\boxed{\begin{cases} \tau\dot{v} = \alpha_z(\beta_z(g - y) - v) + f(x) \\ \tau\dot{y} = v \\ \tau\dot{x} = -\alpha_x x \end{cases}} \tag{3.30}$$

[1]Note that this is a scalar function.

Chapter 4

Implementation of the LfD pipeline

This part is dedicated to the description of the employed methods for learning low-level and high-level tasks. Details on the implementation are provided along with other useful technical notes for both *simulated* and *real environment*, highlighting also the differences wherever could have eased the comprehension. Most of the software is written in `python` and `C++`, while the used robotic framework is `ROS2 Jazzy`, the client libraries `rclpy` and `rclcpp` constitute a link between the two parts while giving the standard routines for the management of the fundamental components of a ROS ecosystem: nodes, topics, actions, services.... The employed collaborative robot for testing the pipeline and doing the experiments is the `Ufactory xArm6` [29].

4.1 Human demonstrations collection

The datasets on which the learning methods are trained are human demonstrations. Due to its features exposed in Section 2.1.1, we have chosen the *kinestethic approach*: the absence of correspondence issue and the intuitive nature makes it easier to implement. This holds in both real and simulated environment.

Very briefly: the implementation of a proper *subscriber node* having subscriptions to suitable topics guides the human demonstration phase. Then, retrieved end-effector/joints measurements are collected into `csv` files which is used to feed the remaining part of the pipeline.

4.1.1 Demonstrations collection: steps to follow

More in detail, in this part the following steps are carried out:

1. Run the driver on the robot and activate the *free-drive* mode (for real environment only);
2. Run the program from the package `lfd_recorder` with needed parameters;
3. Through the `rviz` interface (simulated) – or manually (real) – guide the robotic arm in order to execute a given movement. Note that – under the hood – the node is spinning for collecting measurements constituting the dataset $\mathcal{D}$ from joints or TCP with a structure of the type:

$$\boxed{\mathcal{D} = \Big\{t, x(t), y(t), z(t), q_x(t), q_y(t), q_z(t), q_w(t)\Big\}_{k=1}^{K} \quad t = 1, ..., T_{\max}} \tag{4.1}$$

   where x, y and z are the *cartesian variables* and $q_{(\cdot)}$ are the *quaternion components*, $T_{\max}$ is the duration (in seconds) of the demonstration and K is the number of collected demonstrations.
4. Terminate the program and (only in case of multiple demonstrations to collect) back to 2;
5. At this point you will have in a given directory (user defined) the set of `csv` files containing **raw demonstration data**.

All of the functionalities we have just given are implementeed in the package `lfd_demonstration`, we dedicate a section to.

☞ `lfd_demonstration` package

This is the package related to the human demonstrations collection. Two major executable programs are provided:

- `tf_to_pose.py` is in order to collect, at a given frequency `rate_hz` (Hz), the end-effector pose (position and orientation) using the transformations between the `base_link` and the `link_tcp` attached to the TCP of the robotic arm. Such information is published by a publisher node on the topic `/ee_pose`
- `demo_recorder.py` this attach to the information of the poses the ones related to the gripper state (angle) from the topic `/joint_states`[1]. A node `demo_recorder` is subscriber of both topics. At the end the information are saved in a `csv` file.

See Figure 4.2 for a clearer schema of the dataflow.

[1] In particular is the first element of the array `float64[] position` of a message of type `sensor_msgs/JointState`

4.1.2 Debug: Visualizing demonstrated trajectories

With the objective of carrying out a phase-by-phase debug of the LfD process, some intermediate plots can be obtained from demonstrated trajectories. This goes toward the direction of making also an a-posteriori comparison between *demonstrated* and *generated generalized trajectory* [6]. Some examples are showed afterwards, where both timeseries and xy sample trajectories have been reported.

A summarizing schema in Figure 4.2 is reported below for the sake of clarity; Figure 4.1 illustrates that human guides the robot kinesthetically and data collected can be associated to TCP poses $\mathbf{x}_e$ or joint variables $\mathbf{q}$ timeseries. The demonstration collection phase is something which is quite general despite the context in which is applied for. Using other terms, even if the task we are demonstrating is different (e.g., Peg-in-hole, assembly), types of steps to be performed are very similar to the ones in 1-5.

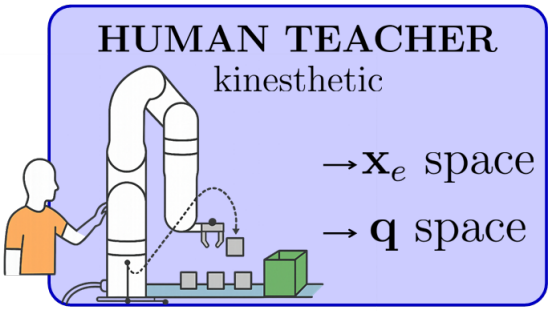


Figure 4.1: Human Kinesthetic demonstration. Both task space ($\mathbf{x}_e$) and joint space ($\mathbf{q}$) data can be collected. In our work only task space data were used

Figure 4.2: Schema of real/simulated demonstration collection

Since demonstrations can have a different duration T_{max}, a way to align them[2] in order to obtain a coherent dataset for obtaining motion encoding models.

[2]In case of multiple demonstrations

4.2 Learning models

Whatever is the approch we are going to use, the learning phase deals with the extraction of an encoding of the robot motion from the dataset $\mathcal{D}$ and chosen hyperparameters θ_{hyper} using a **Machine Learning** (ML) model. The output of this phase is a set of parameters θ that – together with task-specific parameters θ_{task} – will be used in the execution phase for generating generalized trajectories (see Figure 4.3).

Details about three different learning methods that have experimentally implemented are here provided, starting from the main theoretical results, given in Chapter 3. ***Remark***: the results from the learning stage are saved into a numpy file (`npz`), in order to be easily retrieved for the inference (decoding) part of the execution phase.

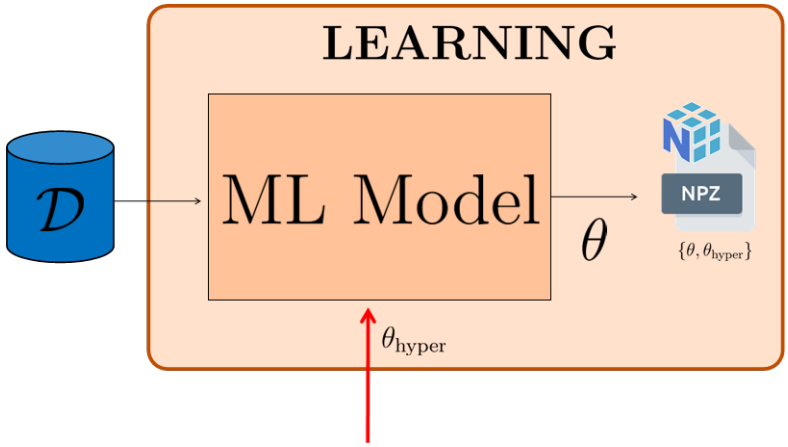


Figure 4.3: Learning phase: general flow for obtaining an encoding from demonstrated trajectories

4.2.1 Behavior Cloning (BC)

Behavior Cloning (BC) is a learning method based on supervised learning and goes toward the direction of modeling the robot behavior in a particular function, which maps robot states to actions to be executed, which is called a **policy** π in the field of MDP-based sequential learning. We recall that being the state and actions respectively s and a the policy π is

$$\pi : \ s \mapsto a$$

More specifically, the objective here is obtaining a parametrization π_θ for such a policy. State and Action spaces are defined as:

- State space $\mathcal{S}$ is made up of current robot position p and rotation r
- Action space $\mathcal{A}$ can be modeled as the one-step differences between each variable, namely Δ_p and Δ_r. This is only one of the possible choices that can be made.

Using such ingredients, we can formulate a supervised learning task for the robot policy where $\mathcal{S}$ is the input space, while $\mathcal{A}$ is the action space. Then:

$$\boxed{\pi : \mathcal{S} \rightarrow \mathcal{A}}$$

The next sections are dedicated to the explanation on: (i) how the demonstration dataset is transformed in something suitable for supervised learning; (ii) what are BC learning steps.

i) Data preprocessing for BC

Imitation dataset $\mathcal{D}$ is preprocessed in the following way:

1. Each demonstration is uniformly resampled in a time grid $\mathbf{t}_{grid}$ with constant time step dt; moreover, quaternions are converted into angle-axis representation (passing through logarithm).
2. For each t_i, state (*input features*) and actions (*output features* or *labels*) are defined as:

$$s_t = \Big(\underbrace{x_t, y_t, z_t}_{p}, \underbrace{r_{x,t}, r_{y,t}, r_{z,t}}_{r} \Big)$$
$$a_t = \Big(\Delta p_t, \Delta r_t \Big), \quad \Delta p_t = p_{t+1} - p_t \quad \Delta r_t = r_{t+1} - r_t$$

3. All of the s_t, a_t are stacked into X and Y (input and output spaces) to form the modified supervised dataset $\mathcal{D}'$

$$\mathcal{D}' = \{X, Y\}, \quad X = [s_t] \in \mathbb{R}^{M \times 6}, \quad Y = [a_t] \in \mathbb{R}^{M \times 6}$$

   on the aggregated datset *z-score* metrics x_{mean}, x_{std}, y_{mean}, y_{std}.

ii) Learning BC model: obtaining a policy parametrization

Before training the model, a normalized datset is obtained by using z-score parameters:

$$X_n = \frac{X - x_{mean}}{x_{std}}, \quad Y_n = \frac{Y - y_{mean}}{y_{std}}$$

An `MLPRegressor` [30] with suitable hyperparameters from scikit-learn library is used. Using such a library the training reduces to the instruction `model.fit(Xn,Yn)` where the method for training it is a standard *gradient-based* method. The used loss function, which guides the training phase, is the standard *Mean-Squared error*.

Trasformation of $\mathcal{D}$ into $\mathcal{D}'$ dataset along with the underlying steps are shown in Figure 4.4.

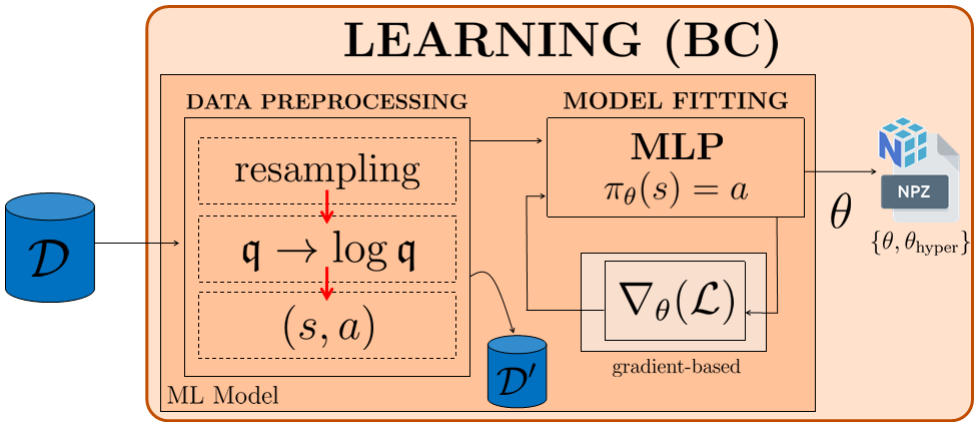


Figure 4.4: Behavior Cloning: learning phase

4.2.2 Dynamical Movement Primitives (DMP)

Here we want to discover what is inside the box named 'ML Model' in Figure 4.3, which – according to the used method – can be arbitrarily complex.
The theory behind DMP has been presented in Section 3.3. Despite the fact that the DMP theory was been formulated for one-dimensional problems, the extension to a more general N-dimensional case is quite straightforward: a DMP for each dimension is fitted, obtained parameters are grouped together and finally a per-dimension trajectory decoding is performed. The main steps are: (i) Data preprocessing, (ii) Fit of a 6-dimensional Dynamic Movement Primitive.

(i) Data preprocessing for training DMP

Three types of operations on the demonstration data are performed:

1. *Quaternion normalization* is useful in order to avoid numerical issues during the training process, this is in order to obtain from the starting quaternion $\mathfrak{q}$ another quaternion $\mathfrak{q}_{\text{norm}}$ such that

$$\mathfrak{q}_{\text{norm}} = [q_x, q_y, q_z, q_w] \quad \|\mathfrak{q}\| = 1$$

2. *Resampling at fixed dt* on (x, y, z): a linear interpolation at fixed dt in order to have a uniform timestep between adjacent samples; on the quaternion components a SLERP[3] is performed.

3. $\mathfrak{q}$ *to* $\log(\mathfrak{q})$ *conversion*. The reason behind this choice is that DMP works for euclidean spaces, quaternion are not[4]. This conversion is also known as *angle-axis representation* [32] and can be used safely for DMP fitting. The inverse transformation can be executed during the trajectory decoding phase.

(ii) 6D DMP fitting

The fitting of a Dynamic Movement Primitive in hyperdimension consists in practice in reusing 1D fit multiple times. The inputs are the demonstrated trajectories $\mathcal{D}$ and the hyperparameters $\theta_{\text{hyper}} = \{\alpha_z, \beta_z, \tau, \alpha_x, N\}$, while the outputs are the DMP parameters $\theta = \{\mathbf{w}, \mathbf{x}_0, \mathbf{g}\}$. Note that $y(t)$ refers to the generic $i-th$ dimension of the demonstrated trajectory, it is not the second cartesian component.
Both phases (i) and (ii) have been graphically represented in Figure 4.5.

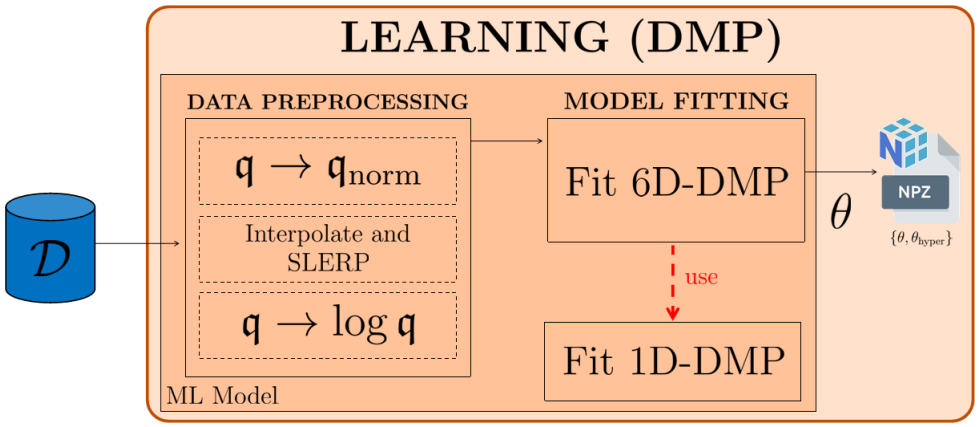


Figure 4.5: Dynamical Movement Primitives: learning phase

4.2.3 Gaussian Mixture Model (GMM)

Differently from the theoretical formulation for Dynamic Movement Primitives that is one-dimensional, the GMM model is for obtaining a succint way to represent robot movements modeling the state-space (task space) variables altogether. That is, a

[3]SLERP stands for Spherical Linear Interpolation, for further details you can see [31]

[4]More in particular, it can be said that quaternions live in a 3D manifold which is embedded in a 4D space

single multidimensional model is obtained for encoding the generalized trajectories. In the remaining part of this paragraph the main *data preprocessing* and *training steps* are explained.

(i) Data preprocessing for training GMM

With the objective of making human demonstration data suitable for the training phase, the following steps are performed:

1. *Constant step resampling*: for each demonstration a uniform time grid $\mathbf{t}_{grid}$ of duration t_{end} with step `dt` is obtained
2. *Emisphere continuity for rotations*: since in the quaternion space $\mathbf{q}$ and $-\mathbf{q}$ are the same rotation, a check is performed on the correct sign in order to avoid abrupt rotations of π
3. *Phase parametrization*: for each demonstration is computed a **phase parameter**, this represents a novelty with respect to the theory presented in Chapter 3, that in a certain way simplifies the problem tractation. For the i-th sample the phase variable is defined as:

$$s_i = \frac{\mathbf{t}_{grid_i} - t_0}{t_{end} - t_0} \in [0,1] \tag{4.2}$$

   the *gesture progress* is delegated to this parameter avoiding the use of the absolute time and allowing the stacking of multiple demonstrations.
4. *Quaternions remapping*: for the same motivations given previously for DMP, here non-minimal representations are converted into something different in which the learning can occur more efficiently; practically:

$$\mathbf{q} \rightarrow \mathbf{r} = [r_x, r_y, r_z] \tag{4.3}$$

5. *Observations building*: for each demo the following matrix is built:

$$Z[s|y] \in \mathbb{R}^{N \times 6} \quad y = [x, y, z, r_x, r_y, r_z] \tag{4.4}$$

   in which there is one row per resampled point.

(ii) Training GMM

Before fitting the Gaussian Mixture with the Expectation-Maximization procedure explained in Chapter 3, all of the available demos are vertically stacked:

$$Z = \begin{bmatrix} Z^{(1)} \\ Z^{(2)} \\ \vdots \\ Z^{(M)} \end{bmatrix}$$

with M being the number of demonstrated trajectories. At this point, in order to approximate the joint distribution of phase and task space variables $p(s, y)$, a GMM with K component is fitted following the EM procedure. After that, a parameter T_{mean} is saved in order to obtain – aposteriori – generalized trajectories. Figure 4.6 illustrates schematically the steps behind *data preprocessing* and *fitting* phases.

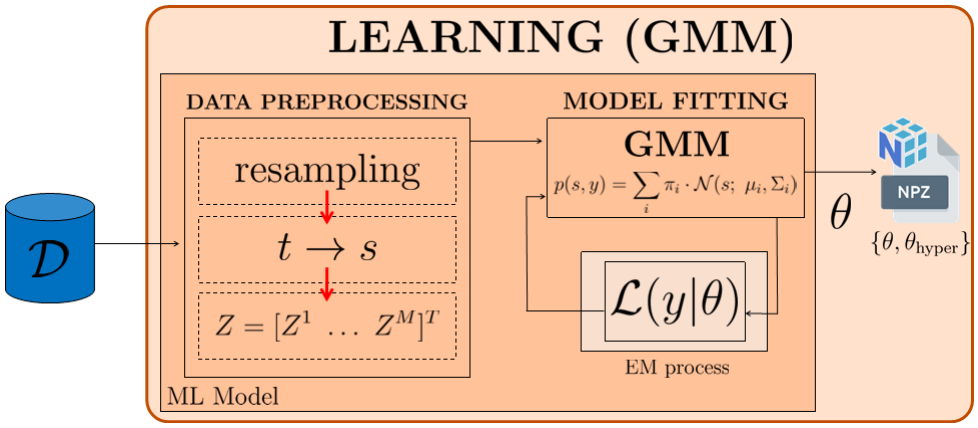


Figure 4.6: Gaussian Mixture Model: learning phase

4.2.4 Trajectory Execution

The *execution phase* deals with: (i) the **decoding of a generalized trajectory** using learned parameters (θ), hyperparameters (θ_{hyper}) and task-specific parameters (θ_{task}); (ii) the **reproduction of the learned movements** in the real/simulated environment. Due to its complexity and importance, we have dedicated a whole section to θ_{task} retrievial given by the cooperation of different hardware and software modules (RealSense camera, object detector, ArUco/Apriltag detector).

Sim2Real issues and execution phase implementation

In order to tackle the gap between the two environments, we use the functions provided by the SDK for the real robot, and the `MoveIt2` framework for the simulated one. The motivation is that the SDK routines provides – under the hood – a fine-grained control of the robot, managing in a stratified way the issues concerning motors, currents and voltages together with velocity and acceleration limits.
On the contrary, passing through the functionalities of MoveIt2 and related high-level routines this is not possible. Indeed, nonidealities neglection leads to vibrations, execution errors and non-smooth trajectories.

This issue management requires a forked implementation of the execution phase whose summarizing schema is given in the Figure 4.7; it must be noted that the execution module can be used also to reproduce the demonstrated trajectories, a problem which is known as **trajectory replay**. This choice has important limitations with respect to the generalization problem, but it is helpful for debugging purposes.

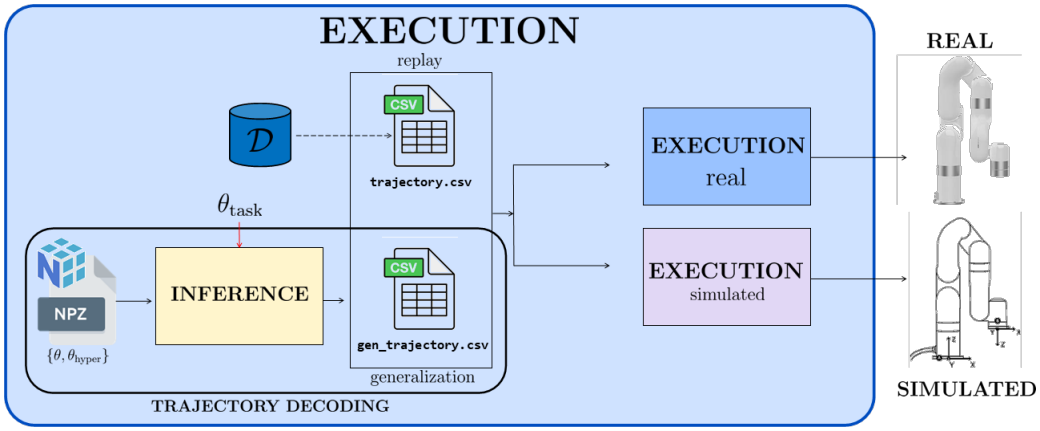


Figure 4.7: Execution phase: real and simulated general flow for execution

A) Executing trajectories in simulated environment

The execution step in the simulated environment is – in a certain sense – simpler. More specifically:

1. The generalized trajectory (or the demonstrated one in case of replay) is, dimension by dimension, interpolated using **cubic splines** in order to obtain a smooth trajectory;

2. The obtained waypoints are then sent to the `MoveIt2` framework that execute the movement on the fake (simulated) robot in the `Rviz` environment.

Experimental evidence confirms that, if the same implemented code is used for controlling the real robot, the execution is quite far to be smooth and precise. This is mainly due to the nonideality issues we have introduced at the beginning of this paragraph. The schematization of such process is given in Figure 4.8.

B) Executing trajectories in real environment

As far as the *real environment* is concerned, we can split the execution phase in two main parts: (i) **generalized trajectory post-processing** and (ii) **robot**

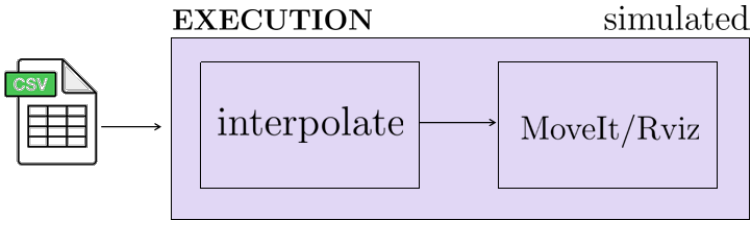


Figure 4.8: Execution phase: simulated environment

control. (Figure 4.9)
The latter is in order to adapt the data to SDK functions and robot hardware[5]; the former takes filtered points from post-processing substage and send them to the robot through the SDK routines for control purposes.
The post-processing phase is, in turn, articulated in three substeps:

1. Quaternion-to-RPY conversion: as already mentioned, the SDK functions require RPY angles for the rotational part of the pose;
2. Downsampling: in order to reduce the number of waypoints to be sent toward the robot, avoiding the saturation of the command buffer, at this stage are filtered out all of the points which has a reciprocal distance that is lower than a given threshold;
3. Singularity check: it is necessary to avoid that the obtained generalized trajectory passes throug singular configurations.

The control of the robot is mainly executed through the method

```
set_position(x, y, z, roll, pitch, yaw, speed, acc,radius, wait)
```

of the class `XArmAPI`, whose documentation can be found in [33]. About the parameters:

- `x,y,z,roll,pitch,yaw` are the 6D-pose waypoint to be reached;
- `speed` and `acc` are the velocity and acceleration limits (in mm/s and mm/s^2);
- `radius` is the blending radius (in mm) for the *blended motion*;
- `wait` is a boolean variable that adds a little pause after the execution of the command, if set to `True`.

[5]As reported in the user manual of the Ufactory `xArm6`, there is a buffer for containing commands that has a limitated capacity

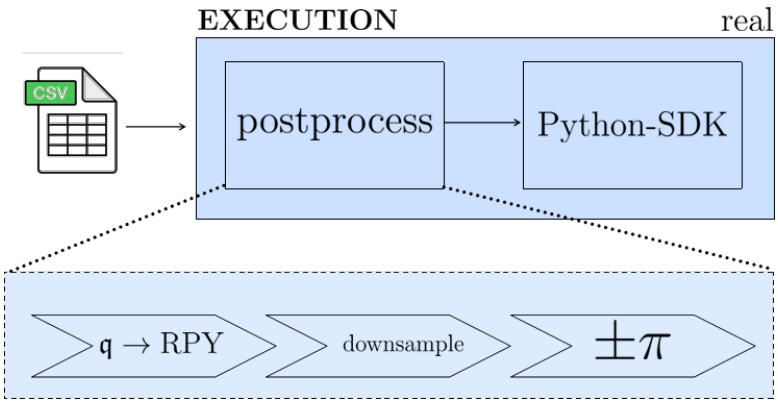


Figure 4.9: Execution phase: real environment

Inference phase for BC model

How to obtain a generalized trajectory from BC model? Since a policy was obtained, the inference is nothing but a policy rollout. That is, starting from an initial state s_0, the function π is used in order to compute the action a_t from which the next state s_{t+1} can be computed. Also in this case we have computed the duration T_{out} as a first step, after that a time grid is created and – finally – for each sample of such a time grid, the following operations are performed:

1. The current state s_i is *normalized* using the z-score metrics computed during the preprocessing steps, s_i^n is obtained;
2. *Forward pass*:here the model MLP is applied to the current state s_i in order to obtain
$$a_i^n = MLP(s_i^n)$$
that in term of scikit-learn code is `model.predict(s_i)`,
3. *Denormalize the action* a_i^n
$$a_i = a_i^n \cdot y_{std} + y_{mean}$$
4. *Splitting of the action*: $a_t = \left(\Delta p_t, \Delta r_t\right)$
5. *Next state* s_{t+1} is computed as
$$s_{t+1} = (p_t, r_t) \qquad p_{t+1} = p_t + \Delta_{p_t} \quad r_{t+1} = r_t + \Delta_{r_t}$$
6. *Quaternion remapping*: angle-axis representation is transformed back to quaternion space.
7. For t the sample
$$x_t,\ y_t,\ z_t,\ q_{xt}\ , q_{yt}\ , q_{zt}\ , q_{wt}$$

The generalized trajectory is nothing but

$$\mathcal{T} = \{x_t,\ y_t,\ z_t,\ q_{xt}\ , q_{yt}\ , q_{zt}\ , q_{wt}\}_t \tag{4.5}$$

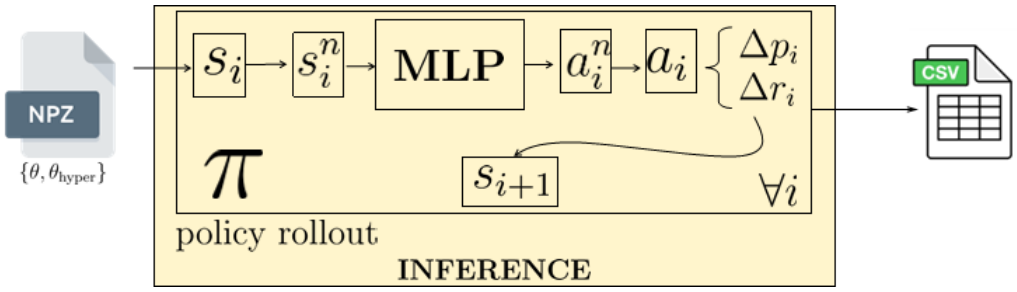


Figure 4.10: Behavior Cloning: inference phase

Inference phase for DMP

This paragraph describes the structure and functionalities of the block named **INFERENCE** in Figure 4.7. We will use interchangebly here the terms *decoding* and *inference*. Very sintetically, a generalized trajectory is obtained by integrating the learned DMP system (for the single dimension), this is done using both hyperparameters and parameters obtained from the learning phase, additionally some task-specific parameters can be provided in order to have different starting and goal points. For the integration, the well-known and simple **Forward Euler** method is used.
The summarizing schema of DMP decoding is given in Figure 4.11.

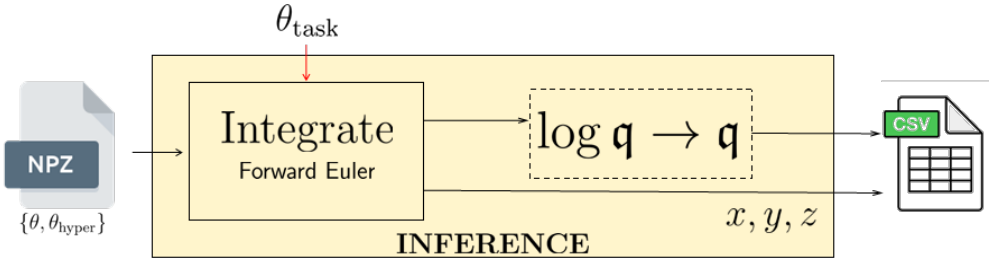


Figure 4.11: Dynamical Movement Primitives: trajectory decoding

Inference phase for GMM (GMR)

Doing trajectory decoding while having a GMM model, is nothing but doing a Gaussian Mixture Regression. Using the parameters T_{mean} and γ previously saved,

the generalized trajectory duration T_{out} is computed as:

$$\boxed{T_{out} = \gamma \cdot T_{mean}}$$

for clarity, for $\gamma = 0.5$ the duration of the generalized trajectory is half the duration of the mean duration of the demonstrations. At this point a uniform time grid is built where, for each i, $t_i = i \cdot dt$; as next step a direct mapping to phase domain can be done

$$s_i = \frac{t_i}{T_{out}} \tag{4.6}$$

After this preparatory stage, for each obtained s_i, Gaussian Mixture Regression can be done as indicated in Chapter 3, the result of which is the trajectory obtained as $\mathbb{E}[y|s]$.
Before achieving the final result the remapping $\log(\mathfrak{q}) \rightarrow \mathfrak{q}$ is needed. It is clear that the information on time t_i is saved, while phase variable s_i is used only for inferring from the mixture model the expected behavior.

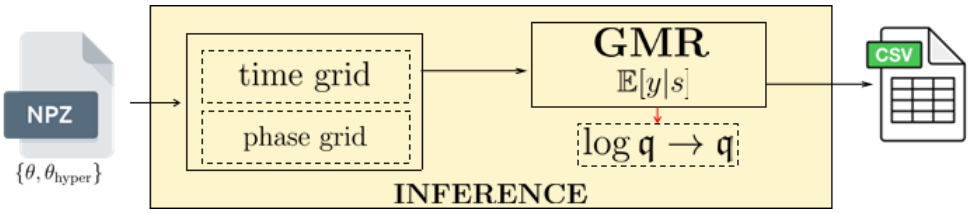


Figure 4.12: Gaussian Mixture Regression: inference phase

We conclude this part giving the summarizing Table 4.1 for the presented learning methods, where the main features can be found in terms of: *theoretical principle*, *training* and *inference* phase

	PRINCIPLE	**TRAINING**	**INFERENCE**
BC	Supervised learning	Gradient-based	policy rollout
DMP	Dynamical systems	Weighted LS (forcing term)	Euler integration
GMM	Probability	EM process	GM Regression

Table 4.1: Learnine methods features

4.3 Evaluation metrics

After having presented the implementation details of the LfD pipeline, it is useful to provide details on the evalutation metrics that will be used for quantifying the performance of the learned models, focusing on different aspects. In particular: *geometric accuracy*, *objective* and *smoothness*.

Evaluating geometric accuracy

- *Root Mean Square Error* ($RMSE$) measures the average euclidean deviation between the two trajectories, instant by instant, and is defined as follows:

$$\boxed{RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N} \|\mathbf{x}^{demo}(t_i) - \mathbf{x}^{gen}(t_i)\|^2}} \tag{4.7}$$

  where $\mathbf{x}^{demo}(t_i)$ and $\mathbf{x}^{gen}(t_i)$ are respectively the demonstrated and generalized trajectory.
- *Hausdorff distance* (d_H) measures the maximum distance of a set to the nearest point in the other set. Given two trajectories $P = \{p_1, p_2, ..., p_N\}$ and $Q = \{q_1, q_2, ..., q_N\}$, it is defined (in discrete form) as:

$$\boxed{d_H(P,Q) = \max\left\{\max_i \min_j \|p_i - q_j\|,\ \max_j \min_i \|q_j - p_i\|\right\}} \tag{4.8}$$

  Such a distance takes into account the global shape of the trajectories without doing reasoning on time.

Evaluating objective fulfilment

Endpoint error (EE) measure is for evaluating the generalized trajectory from the point of view of the objective. This is for quantifying how much the final position of the generated trajectory is distant from the final target of the demonstration. It is defined as:

$$\boxed{EE = \|\mathbf{x}_T^{demo} - \mathbf{x}_T^{gen}\|} \tag{4.9}$$

A low endpoint error indicates that the trajectory is terminated in the correct point. Another way to evaluate whether we have reached or not the objective is computing the rate of successful experiments with respect to the total.

A measure which can be used for evaluating high-level task is the so-called *Task Success Rate*, given by the percentage of total experiments (n_{TOTAL}) of a given

task successfully completed ($n_{SUCCESS}$). It is simply defined as

$$\boxed{TSR = \frac{n_{SUCCESS}}{n_{TOTAL}}} \tag{4.10}$$

This is useful even in those cases in which no high-level algorithms are used since, during the execution, of a certain task some issues can occur whose occurrence is not deterministic.

Evaluating smoothness

In general, when we talk about *smoothness* we refer to the metric used for quantifying how much a certain trajectory is regular and abrupt variations free. Common method for quantifying the smoothness is the **jerk** which is the derivative of the acceleration:

$$\boxed{J = \int_0^T \| \dddot{\mathbf{x}}(t) \|^2 \, dt} \tag{4.11}$$

4.4 Task-specific parameters θ_{task} retrievial

One of the features that some Learning from Demonstration algorithms provide is the **intratask generalization**. As previously seen in Chapter 2, this is the possibility to generate trajectories taking into account user-provided parameters. A common example is to change the *start* and *goal* points for non-periodic trajectories. There are some mehtods which embed this parametrization in the original formulation (e.g., DMP), other that can be extended to take into account them (e.g., GMM finds its generalization in Task Parametrized GMM).
This section describes how a vision subsystem, `realsense2` camera and the OpenCV library, can be used for task-parameters (θ_{task}) retrieval. In particular, the parameters we want to retrieve are:

1. y_0, which denotes the *starting point* for a certain task
2. g which denotes the *goal point*
3. w_{obj}, h_{obj} which are, respectively, the *width* and *height* of the object/objects to be grasped or manipulated. These can be used in order to parametrize the gripper position.

In first approximation, we will assume that objects geometrical dimensions are a-priori known and saved in a *glossary* whose records have the schema

```
(object_name, obj_height, obj_width)
```

Start and goal pose y_0 and g are retrieved by using ArUCO tag detection and identification from the scene. Later, in the next section, we will obtain all the parameters dynamically using a computer vision model which relies on RGB-D camera frames.

4.4.1 Pose estimation by using ArUCO tags

The process of obtaining poses passing through ArUCO tags can be summarized in three different steps:

1. All of the N_{tags} tags are detected together with their ID by using the OpenCV provided tools, in particular the following instructions are used:

```
params   = aruco.DetectorParameters()
detector = aruco.ArucoDetector(dictionary, params)
corners, ids, _ = detector.detectMarkers(gray)
```

   - here in particular an `ArucoDetector` object is created for detecting tags of a certain dictionary (eg. `4x4`), a gray level image (obtained by the realsense2 color stream) is passed to such an object in order to obtain *corners and ids* of the markers that are present into the picture.

2. The 6DOF poses of the detected tags with respect to the camera reference frame are obtained; in particular homogeneous transformation matrices are obtained

$$T_{\text{cam}}^{tag_i} \quad i = 1, ..., N_{\text{tags}}$$

3. This is the most important stage in which tags poses $T_{cam}^{tag_i}$ are converted into the *base reference frame* by doing some matrix multiplications:

$$\boxed{T_{base}^{tag_i} = T_{base}^{ee} \cdot T_{ee}^{cam} \cdot T_{cam}^{tag_i} \quad i = 1, ..., N_{tags}}$$

   note that the matrix T_{ee}^{cam} is obtained by the simulation environment `Rviz` simply: (i) by adding to the launch command a flag for adding both *realsense2 camera* and its support; (ii) using the `tf2_tools` for obtaining the homogeneous transformation matrix between the TCP and the `camera_color_optical_frame`

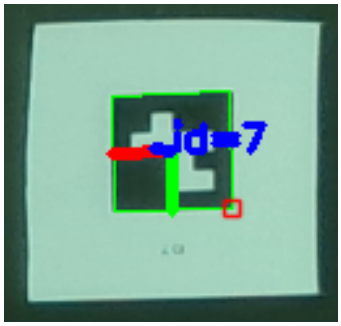


Figure 4.13: ArUCO detection and pose estimation (`realsense2`+OpenCV)

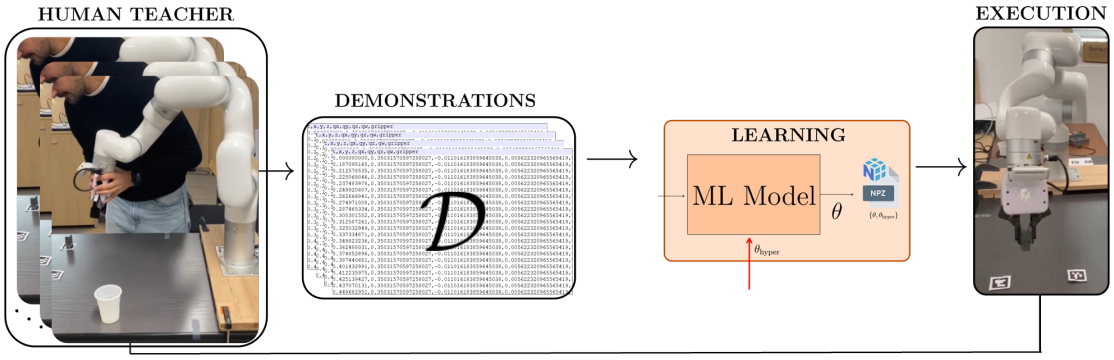


Figure 4.14: LfD pipeline for learning `PICK` low-level skill

The reference frames, topics of this discussion are showed in Figure 4.16 that is from the simulated environment. The tags for the experiments have been generated using the tool in [34] by which the dictionary, the identifier and the tag dimension can be chosen.
The vision pipeline based on ArUCO tags is reported below together with the channel related to camera calibration (see Appendix C). Further details on the process converting RGB images into ArUco tag detections are given in Appendix B.

4.5 Hyperparameters

Explaining the LfD pipeline, we have introduced a certain number of user choices, which are nothing but hyperparameters. They are important for representing the *handles* by which we can adjust the obtained results during the demonstration, learning or execution phase. In the following we are giving, by mean of summarizing tables, a list of the main hyperparameters.

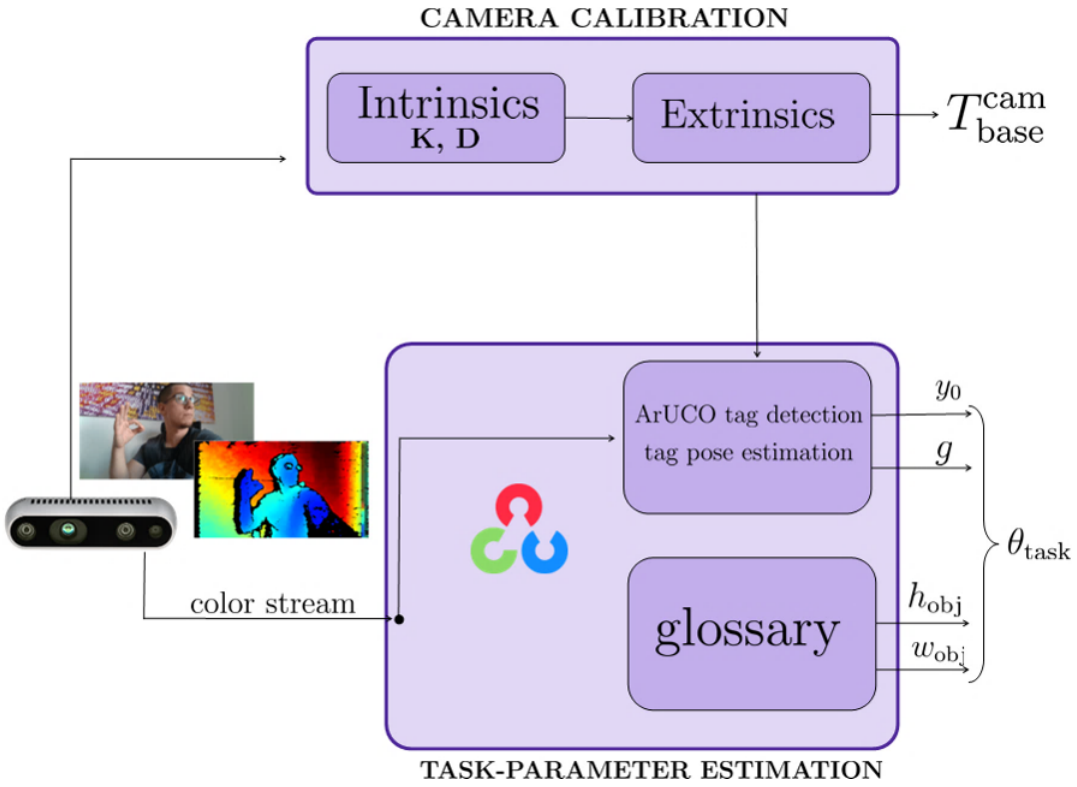


Figure 4.15: Vision pipeline (I): start/goal poses ArUCO-based, object geometry in a static glossary

4.5.1 Demonstration phase hyperparameters

The only relevant *demonstration phase hyperparameter* which is used is `rate_hz` and is related to the nominal frequency at which task space data are collected.

θ_{hyper}	**Description**	**Range**
`rate_hz`	Frequency at which are collected data from topics `ee_pose` and `joint_states`	[50,100]

Table 4.2: Demonstration phase hyperparameters

4.5.2 Learning phase hyperparameters

The learning phase hyperparameters are different according to the used learning method. For BC, DMP and GMM they are reported in Table 4.3, Table 4.4and Table 4.5 respectively.

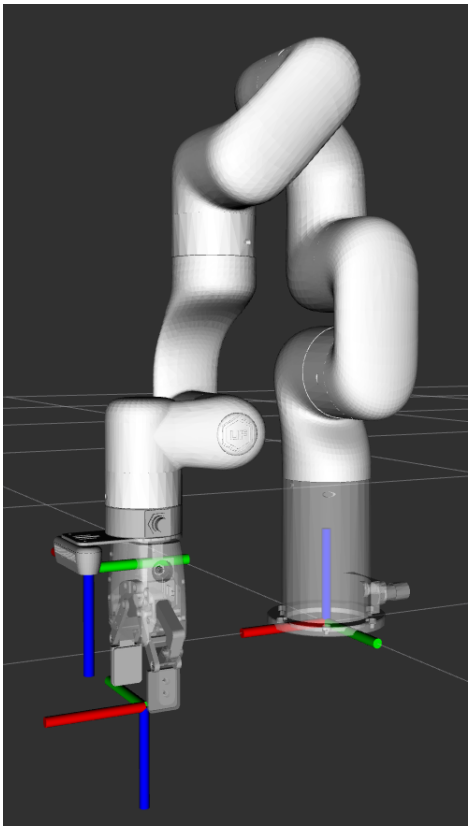


Figure 4.16: Base (*base*), TCP and camera (*cam*) frames of the robot model

θ_{hyper}	**Description**	**Range**
`hidden`	MLP structure $(l_1, l_2, ..., l_N)$	$l_i \in$ [32,256] $N \in$ [1,4]
`LR`	Learning rate	[1e-3,1e-2]
`MAX_ITER`	Maximum number of EM iterations	[100,450]
`activation`	Employed activation function	[50,200]

Table 4.3: Learning phase hyperparameters (**BC**)

θ_{hyper}	**Description**	**Range**
τ	Temporal scaling factor	[1.0,2.0]
α_z, β_z	Related to system damping and stiffness	[12.0,25.0]
α_s	Phase decaying factor	[1.0,4.0]
N (n_{BFS})	Number of Gaussian basis for forcing term $f(x)$	[50,200]

Table 4.4: Learning phase hyperparameters (**DMP**)

θ_{hyper}	**Description**	**Range**
K	Number of Gaussian components	[3,10]
γ	Execution rescaling	[0.5, 2]
`reg_covar`	Regularization for covariance	[1e-6, 1e-5]
`max_iter`	Maximum number of iterations	[100, 300]
`random_state`	Random seed for the parameter initialization	[1, 50]

Table 4.5: Learning phase hyperparameters (**GMM**)

4.5.3 Execution phase hyperparameters

The real execution using the robot's SDK, requires to make some choices. In addition, it is desirable that such parameters are not constant, since *different tasks* can require *different execution parameters*. Table 4.6 sums up such information together with their description and per-parameter range of variation.

θ_{hyper}	**Description**	**Range**
`SPEED`	Limit on joint speed (mm/s)	[70.0,300.0]
`ACC`	Limit on joint acceleration (mm/s^2)	[300.0,1000.0]
`RADIUS`	Blending radius between segments (mm) (see description above)	[10.0,30.0]
`DIST_MIN_MM`	Points whose distance (in position) is under such a threshold are filtered out	[10.0,30.0]
`DIST_MIN_RAD`	Points whose distance (in orientation) is under such a threshold are filtered out	[1.0,3.0]

Table 4.6: Execution phase hyperparameters

4.6 Low-level skills

Former sections were devoted to the explanation of techniques and details behind the implementation of the Learning from Demonstration pipeline. At this point, such a general structure can be used for *learning motor primitives that imitate human behavior*. In particular, in our work, this is used for learning *low-level skills*, which in some way are reused later in order to build *task plans* in which condtions and loops are likely to be included, too. A valid reference for this part of *Low-level skill learning* is the Chapter 4 from [3].

For the sake of completeness and with the aim of doing a brief recap before going on, the main steps behind LfD process are listed hereafter:

1. The human **demonstrates** the low-level skill and a dataset $\mathcal{D}_{\text{skill}}$ is obtained;
2. The Machine learning module **extracts** the motion encoding model parameters θ_{skill} from the dataset, provided the task-specific parameters that are leading to trajectory generalization;
3. A generalized trajectory is obtained for the skill, which is followed by the **model execution**.

In the following we are giving more details on the fundamental building blocks (motor skills) for the robot movement. i.e., `PICK`, `PLACE`, `POUR` and `SHAKE` skills.

4.6.1 `PICK`

This is one of the most used motion primitives for a robotic manipulator. Dealing with demonstration and learning of such a movement is less challenging with respect to other tasks (e.g., the insertion of a peg into an hole), since there are not strict

requirements on contact control.
In general, `PICK` movement is a naturally hierarchical task, although in this case we consider it as a fundamental *atomic* block. Despite this assumption, knowing something more about the breakdown the literature gives of it, can be useful for human demonstration purposes. Phases which are usually individuated are:

1. *Approach*: here you pass from the current position to a *safe* position located above the target. Given the best grasp candidate (user-defined or vision-guided) the hand is put in a certain position with a suitable offset $+z$ with respect to the TCP origin (we call it T_{pg})
2. *Pre-Grasp*: here the pre-grasp pose is eventually refined using local information (eg. proximity to the table and so on)
3. *Grasp execution* is the crucial part of the movement in which the fingers of the gripper are **closed**[6] (being consistent with the object dimensions)
4. *Lift* is the phase when the scene is cleared moving the robot to a carry position in order to pass to the next task (e.g., *place* is frequently executed). Many times this is considered as an *hand-off phase*.

Phases 1-4 are preceded by a Vision/Sense phase in which the target object is located and some candidate grasp poses are proposed, whereas it is not hard-coded for simplicity. The specialized LfD pipeline for `PICK` is given in Figure 4.14.

4.6.2 `PLACE`

`PLACE` primitive is used – not rarely together with `PICK` – in order to move an object from the carry to the final position. Again, we consider it atomic, however splitting it into sub-skills can be helpful for a better comprehension:

1. *Approach*: Here the manipulator is moved from the carry position to a safe pre-place pose T_{pp} above the goal position
2. *Pre-place*: T_{pp} is refined in order to meet local requirements and obtain a new pose definitive pose T_{pl}
3. *Place and release*: here the gripper is **opened** and the manipulator returns to a safe position.

4.6.3 `POUR`

`POUR` is a manipulation task where the robot transfers a liquid from a **source container** to a **target container** by tilting it and controlling the flow according

[6]The value for the gripper position can be hard-coded or obtained through a vision subsystem.

to some approach. It is a continuous task which is executed in proximity of the target in which the main aim is to transfer a certain volume of a liquid without spilling it or knocking the container [35]. This is a more challenging task due to the fact there is *fluid uncertainty* due to viscosity; in addition, *liquid control* is not trivial since can be executed both in open[7] or closed loop (using for instance visual servoing). In this case the following microphases can be individuated [36]:

1. *Approach upright*: move source near target while keeping it vertical
2. *Align*: position lip above the rim, stabilizing it
3. *Pour*: tilt to initiate and regulate flow; track level/volume
4. *Stop & cut stream*: reverse tilt smoothly
5. *Retract*: return to a safe position

Other details on pouring task together learned by DMP together with parametrization issues are discussed in [35].

Open-loop fluid control

This section illustrates how the liquid is poured within the target recipient subject to constraints of volumes, that is: how to ensure that a certain quantity of a certain liquid is poured within the target recipient? The fact of employing human demonstrations for pouring task avoids the explicit modeling of the map $tilt \rightarrow flow$. Moreover, we work under the following assumptions:

1. The geometry of the bottle together with its neck properties are apriori known;
2. The pose of the target recipient is fixed and apriori known;
3. No sensing of flow or liquid level is adopted;
4. The calibration of *tilt vs flow* is embedded in the demonstration and is refined during execution, so that a timed motion can be adopted.

For the sake of clarity, what we are implicitly demonstrating embed a sort of function of the type

$$q(\theta_i) \approx \frac{\Delta V}{\Delta t} \tag{4.12}$$

where q is the *volume flow*, θ_i is the *tilt angle* and ΔV, Δt are respectively the volume and the time interval.

[7] When executed in closed-loop a map tilt $\rightarrow$ flow is obtained

4.6.4 SHAKE

The SHAKE movement consists of a *periodic* (or *rhythmic*) and *bounded oscillation* of the end-effector which holds a container, finalized to mix the content without spilling or losing the grasp. Microphases can be individuated as:

1. *Pose for shaking*: a certain orientation of the end-effector is chosen for executing the shake movement
2. *Rythmic shake* where the periodic movement is executed a certain number of times
3. *Stop and hold*: shaking ends and a pre-place position is assumed by the end-effector

We can describe the shake movement from two different perspectives, in particular there are a *spatial* and a *temporal* pattern. The former consists of *oscillations* along a chosen axis, while the *latter* is a *periodic motion* having smooth start and stop points in order to avoid shocks.
Rhythmic movements have an important role in many tasks (e.g., polishing, wiping) and they have different properties with respect to *discrete motions* such as PICK or PLACE; in particular, they carry information like *frequency*, *amplitude* and *phase*. Such aspects are well described in [37], where *Fourier Movement Primitives (FMP)* are used to learn oscillating movements from demonstrations in the Fourier frequency domain, this is a good choice for the nature of skills to be learnt.
In the reference paper [37] all of the implementation details are given, however the interesting point we want to highlight is the alternative used approach of learning a GMM over complex weights related to the Discrete Fourier Transform (DFT) of time-domain windows of the human-demonstrated periodic skills.

4.7 High-level tasks

Low-level learned skills can be the basic building blocks to create complex robot behaviors. In particular, they can be combined together with the aim of executing a more complex task. There are tipically two ways to obtain high-level task plans:

1. By using *High-level learning methods* (see Chapter 2) which build task plans from demonstrations;
2. By using a *Programmatic approach*, that is giving explicitly the (eventually conditioned) sequence of actions to be performed.

In the following we are going to follow the second approach that is the most immediate, which uses directly the learned primitives without other learning steps. Any motion skill can be used either in its "original form" imitating exactly what

the human had demonstrated, or in the parametrized form. At this aim, we will use the following **notation**:

- `PRIMITIVE` for indicating the non-parametrized motion primitive
- `PRIMITIVE(par1,...,parN)` in order to indicate that skills whose execution is parametrized by a bunch of parameters {`par1,...,parN`}

where `PRIMITIVE` is one among `PICK`, `PLACE`, `POUR`... This formalism to indicate motion primitives and their parametrized version is similar to the one used in [3].

4.7.1 Pick-and-place

We have introduced the *pick-and-place* task in Section 2.4.2 stating that it constitutes a fundamental subtask for industrial applications, and any other more complicate assembly task. It is one of the most elementary high-level tasks where two low-level skills primitives are executed in cascade, that is `PICK` followed by `PLACE` as shown in Figure 4.17. Using a pseudocode notation:

```
function PICK_AND_PLACE(ID_object, ID_goal){
    PICK(ID_object)
    PLACE(ID_goal)
}
```

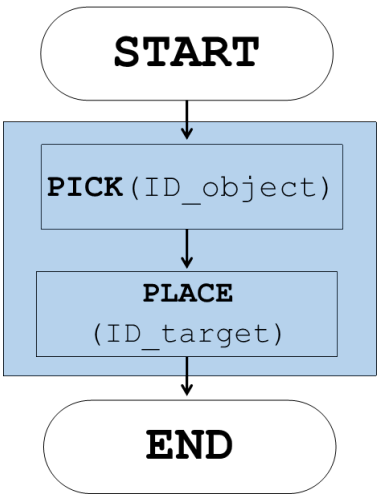


Figure 4.17: `PICK_AND_PLACE()` flowchart

4.7.2 Collecting objects in a recipient/basket

Going a step further, we can reuse the `PICK_AND_PLACE` at a higher level, for executing other plans, for example *bin sorting* or – more simply – *collecting different objects in a recipient/basket*.

```
function COLLECT_OBJECTS(ID_target){
    ID_objects = retrieve_parameters(color_scene)
    for ID_object in ID_objects:
        PICK_AND_PLACE(ID_object,ID_target)
}
```

The routine `retrieve_parameters(color_scene)` is a vision-guided function which dinamically obtains task parameters from the current scene. Figure 4.19 shows the annotaded image with detected ArUco tags and poses.

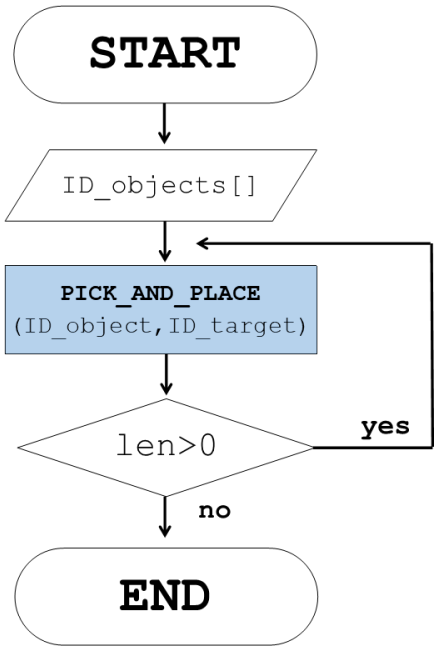


Figure 4.18: `COLLECT_OBJECTS`: flowchart. `len` is assumed to be the *length* of the array containing objects information

4.7.3 Pour a drink in a container

Here we want to exploit the `POUR` skill to issue a more complete task where: (i) a source liquid recipient is picked; (ii) the drink is poured inside the container; (iii) the source recipient is replaced back.

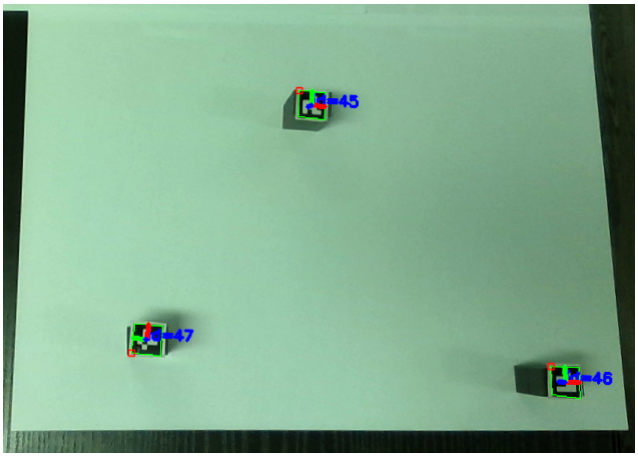


Figure 4.19: Detected objects from the real scene. The image is annoted with detected ArUco IDs and reference frames

Giving a sort of pseudocode the high-level task we want to perform here can be described as:

```
function POUR_DRINK(ID_drink, ID_glass, quantity){
    pick(ID_drink)
    pour_and_align(ID_glass)
    wait(quantity)
    place(ID_drink)
}
```

`quantity` is a recipe-specific parameter and intrinsic of the type of mixture to be prepared; it is important for parametrizing the waiting time which is intrinsically based on the fluid dynamics we described in Section 4.6.3. Another way to visualize the sequence of steps is using the flowchart similar to the one in Figure 4.20.
For the sake of clarity, we assume here that the gripper management (open/close) is embedded in the `PICK()` (close) and `PLACE()` (open) procedure.

4.7.4 Prepare a mixture of drinks

What if we wanted to mix more than one drink? We have to reuse the high-level task `POUR_DRINK()` many times as the number of ingredients constituting the mixture recipe. By using a sort of pseudocode, as before, we can say that preparing the composite drink implies:

```
function PREPARE_MIXTURE(Recipe [], ID_glass){
    for item in Recipe: POUR_DRINK(item.id, item.quantity)
    SHAKE(ID_glass)
    PLACE(ID_glass)
}
```

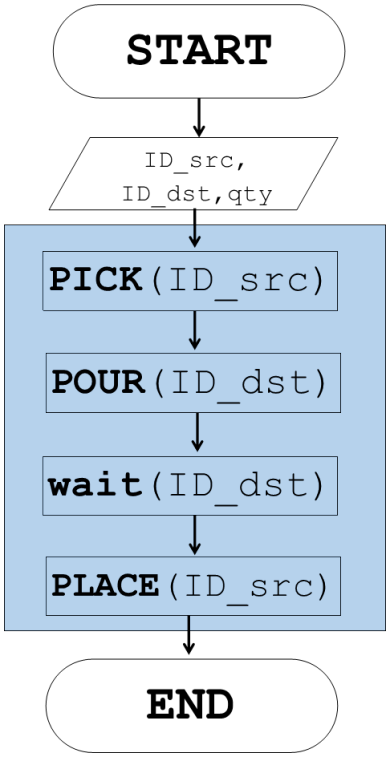


Figure 4.20: `POUR_DRINK()` flowchart

Figure 4.21: Pouring a liquid

We also are giving the flowchart in Figure 4.22, as we have done before. Finally, in order to conclude this section, we can say that the tasks we have executed on

the robot are organized as in a pyramid in which we have low-level and high-level tasks. The interesting fact is that the tasks from an higher level use the ones of a lower level as constituting blocks.

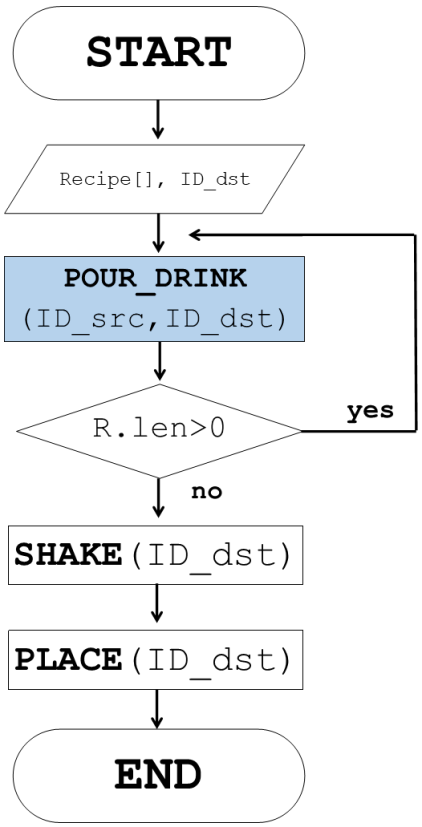


Figure 4.22: `PREPARE_MIXTURE()` flowchart

Preparing the mixture asking for help: introducing Human-Robot Collaboration

We can exploit this last experiment in order to introduce the aspect related to Human-Robot Collaboration (HRC). To this aim, what if the source containers were corked? Moreover, in order to shake the final mixture without spilling any liquid it could be convenient that the destination container/glass was corked. The fact of screwing/unscrewing a cap from a bottle is a *challenging task*, that – in addition – is also quite hard to demonstrate. For this reason, we could ask for help to the human in order to: (i) remove the caps for pouring, (ii) put the cap on the destination container before shaking.
At this point, we need a modfied version of the task `POUR_DRINK` which include human collaboration and the corresponding modified version of `PREPARE_MIXTURE`.

For the sake of clarity, the actions performed by a human are introduced with the notation: `[HUMAN] Action to be performed`.

```
function POUR_DRINK_HRC(ID_drink, ID_glass, quantity){
    pick(ID_drink)
    bring_to_human()
    [HUMAN] Unscrew the cap from the bottle
    pour_and_align(ID_glass)
    wait(quantity)
    give_to_human()
    [HUMAN] Screw the cap and place the bottle
}
```

The modified routine using the just defined task is:

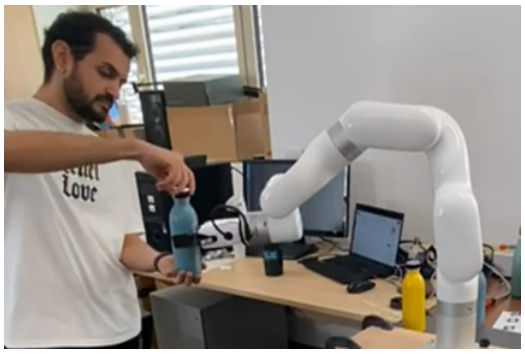
```
function PREPARE_MIXTURE_HRC(Recipe [], ID_glass){
    for item in Recipe:
        POUR_DRINK_HRC(item.id, item.quantity)
    PICK(ID_glass)
    bring_to_human()
    [HUMAN] Screw the cap
    SHAKE(ID_glass)
    give_to_human()
    [HUMAN] Take the container with the prepared mixture
}
```

In order to simplify the experiments, some assumptions have been done: (i) the quantity of liquid to pour is fixed to be 100 mL, thus making easier the interaction of the robot with the environment; (ii) demonstrations are given in order to empty each bottle, this avoids seeking for the minimum tilt θ_0 at which some liquid is poured; (iii) a single recipe is available. Such assumptions do not invalidate the main reasoning which is done – instead – on the robot movement. Two snapshots of the complete experiment are shown in Figure 4.23.

4.8 Knowledge of the robot: motion primitives and tasks

We have: (i) detailed the implementation of the LfD pipeline in each stage; (ii) explained the theoretical and implementative details behind learning methods; (iii) made some examples to specialize the pipeline for chosen motion skills. At this point, it should be clear that through Learning from Demonstration, at the end, we have obtained motor skills knowledge which is splittable in two classes as showed in Figure 4.24:

(a) Robot towards human



(b) Human screws the cap

Figure 4.23: Human screwing the cap on the bottle: the robot goes to the human (a) and asks for help for screwing the cap (b)

1. **Motion Primitives**: the basic building blocks for the robot behavior which can be either discrete or periodic, parametrized or not.
2. **Tasks**: these are the composition of motion primitives and are for defining more complex behavior.

In general, robotic tasks are made up of motion primitives. However, there are not limitations on the fact that a task could be defined in function of another task creating an even higher level of abstraction. For instance, in the following example a three-level hierarchy occurs:

- **3rd level**: `COLLECT_OBJECTS` task. This use repeteadly the `PICK_AND_PLACE` high-level task.
- **2nd level**: `PICK_AND_PLACE` task. This is the composition of `PICK` and `PLACE` low-level skills.
- **1st level**: `PICK` and `PLACE` motion primitives learnt, they are the single pieces demonstrated by the human teacher.

For the sake of simplicity, we have obtained using LfD (low-level) or using a programmatic approach (high-level) only few primitives and tasks, this general reasoning can be extended to expand the knowledge, for example treating the `PEG_IN_HOLE` task, in which another low-level skill is needed (`INSERT`). Another way to visualize this notions, beyond the use of pseudocodes and flowcharts, is the one given in Figure 4.24. On the bottom-right there are the *motion building blocks*, on the left we have high-level tasks, and *top-right* we have an example unrolling of a generic task. Some snapshots of the execution of the learned motion primitives is illustrated in Figure 4.25, where sample generated generalized trajectories are reported in red.

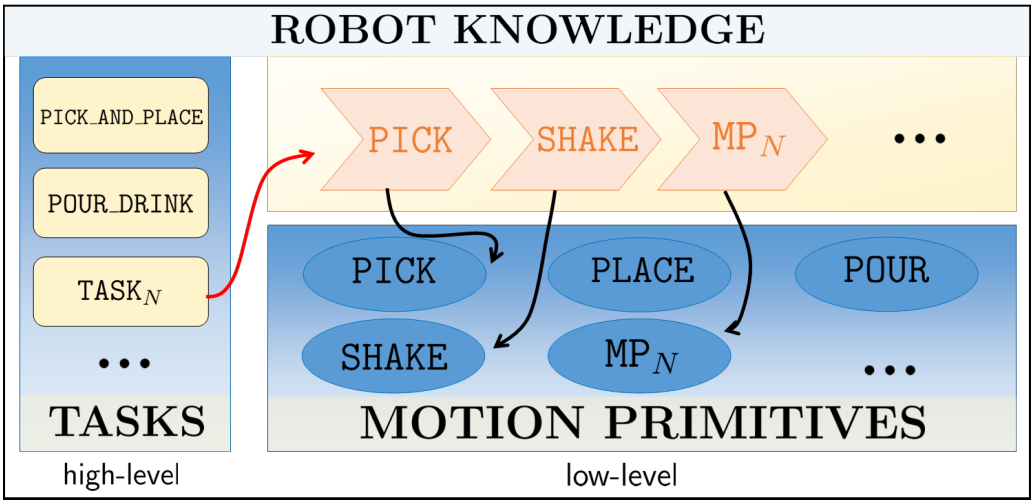


Figure 4.24: Robot knowledge: motion primitives (bottom-right) are the building blocks for tasks (left) which are nothing but a sequence of low-level skills (top-right)

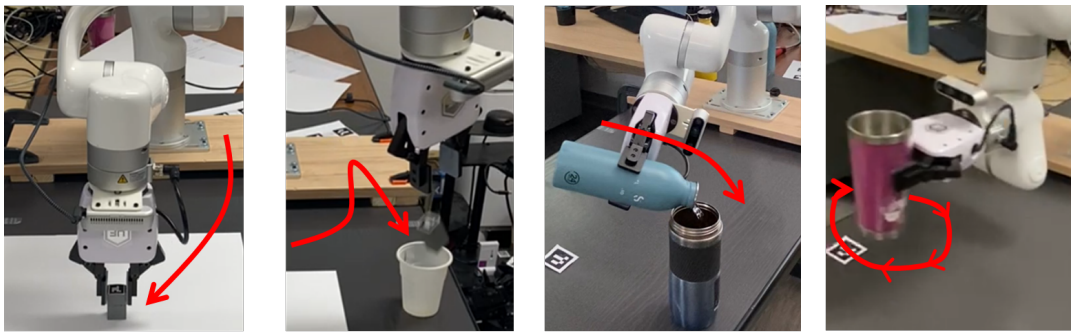


Figure 4.25: Learned low-level skills' snapshots. Sample trajectories are given in red. Note that while for **PICK**, **PLACE** and **POUR** you have point-to-point trajectories, for **SHAKE** there is a periodic one

4.9 Two-finger parallel gripper management

Among all possible tools[8] we use a **two-finger parallel gripper** (Figure 4.26) for executing our manipulation tasks. Till now we have dedicated most of the paragraphs to motion encoding/decoding and related learning method, without further details about the gripper managemenent, that is the aim of this section.

[8]*Vacuum gripper* (for suction-based manipulation) and *Bio gripper* are examples of other tools can be used for our robot (see https://www.ufactory.cc/xarm-collaborative-robot/).

In our LfD pipeline, we decouple *grasp synthesis* from *gripper actuation*. More specifically, demonstrations provide the end-effector approach and grasp-pose to be imitated and generalized (continuous) while the gripper is commanded through a discrete event (open/close). The gripper position (how much we have to close/open the jaws) is determined:

1. *Statically*, whether the employed vision subsystem does provide only start/goal poses and object geometrical insights are stored in a static glossary (see Figure 4.15);
2. *Dinamically*, in the situation where a more sophisticated ad-hoc Computer Vision architecture (eg. *GraspNet-1billion* [38]) is employed. How we mentioned in the dedicated section, we can obtain by using it not only many candidate grasp poses, but also the object geometrical dimensions.

Whatever is the way we retrieve such an information, it must be a value between 0mm (totally closed fingers) and 850mm (totally opened fingers). Then:

$$p_{grip} \in [0,\ 850]$$

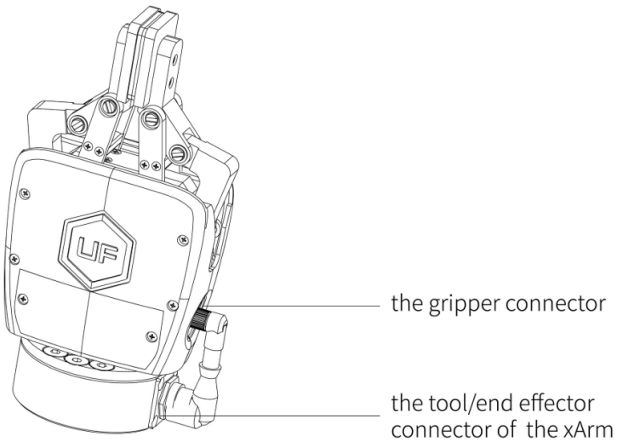


Figure 4.26: `xArm` gripper

From a practical point of view the gripper actuation is interleaved between the different phases of low-level skill execution. For example for `PICK_AND_PLACE` the pseudocode which includes gripper actuation is:

```
const p_APERTO=850
function PICK_AND_PLACE(ID_object, ID_goal){
    PICK(ID_object)
```

```
    # --- M A N A G E  G R I P P E R ---
    p_grip = lookup(ID_object)
    set_gripper_position(p_grip)
    # -------------------------
    PLACE(ID_goal)
    set_gripper_position(p_APERTO)
}
```

Note that here `lookup(ID_object)` is a call to the subsytem which retrieves, either statically or dinamically, gripper position for objects to be manipulated. The block-diagram representation of the same concepts is given in Figure 4.27.

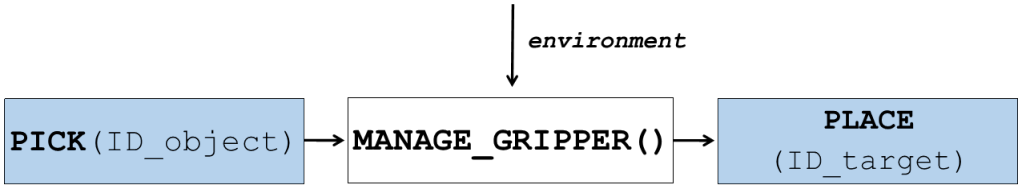


Figure 4.27: Pick and place with gripper management

Both factors of handling separately the gripper actuation and demonstrating singularly subtasks, impact the way the human teacher has to demonstrate motion. There are some works in which the information on the gripper position is used to segment subtasks. For instance, if you decide that for a `pick-and-place` task until the target pose is not reached, gripper jaws are opened, you can use such an insight for dividing `pick` from `place` part. This trick is used, for example, in [39] where multidimensional DMP is trained for pick and place, after that a gripper position-guided segmentation is performed.
Moreover, in this implementation the gripper position is considered as an additional state variables and then a generalized trajectory is generated for the gripper fingers, treating it as an additional joint to control, in a continuous fashion. The demonstration dataset, under this assumption, is defined as:

$$\mathcal{D} = \left\{ t, x(t), y(t), z(t), q_x(t), q_y(t), q_z(t), q_w(t), p_{grip}(t) \right\}_{k=1}^{K} \quad t = 1, ..., T_{\max}$$

In demonstrated motions there are two common approaches for the finger of the gripper: vertical and horizontal, as shown in Figure 4.28.

(a) Vertical approach **(b)** Horizontal approach

Figure 4.28: Horizontal and Vertical approaches

Chapter 5

Experimental results

In this chapter we provide the main results for all the learning methods for low and high-level motor skills learned through the process of LfD. In particular, we provide *quantitative results*, by means of evaluation metrics and *qualitative results* giving some significant plots, using Table 5.1 as a reference for measurement units.

Metrics	$RMSE_{\text{pos}}$	$RMSE_{\text{ori}}$	EE_{pos}	EE_{ori}	d_H	J	t_{train}
Unit	m	deg	m	deg	m	m/s^3	s

Table 5.1: Units for used metrics

5.1 Low-level skills

5.1.1 `PICK`

As shown in Table 5.2, the best performing model for $RMSE$ metrics is GMM; DMP is the best as far as the EE is concerned. GMM achieves best result also for geometric accuracy (d_H) and smoothness (J) metrics. Figure 5.1 shows the YZ plane projection of the generated trajectories on which the given results on Hausdorff's distance d_H can be justified. A graphical interpretation of the EE_{pos} error can be found in quaternion components timeseries plots illustrated in Figure 5.2.

5.1.2 `PLACE`

As far as the `PLACE` primitive is concerned, results are slightly different. In particular, best performing model for $RMSE$ is BC, even if the difference with GMM is almost

	$RMSE_{\text{pos}}$	$RMSE_{\text{ori}}$	EE_{pos}	EE_{ori}	d_H	J	t_{train}
BC	0.22	10.29	9.45E-03	0.33	0.15	2348.81	34.52
DMP	0.15	6.55	**2.74E-07**	**0.0002**	0.18	335.11	**0.47**
GMM	**0.063**	**5.0047**	1.76E-02	1.63	**0.051**	**89.011**	2.03

Table 5.2: Quantitative results (PICK)

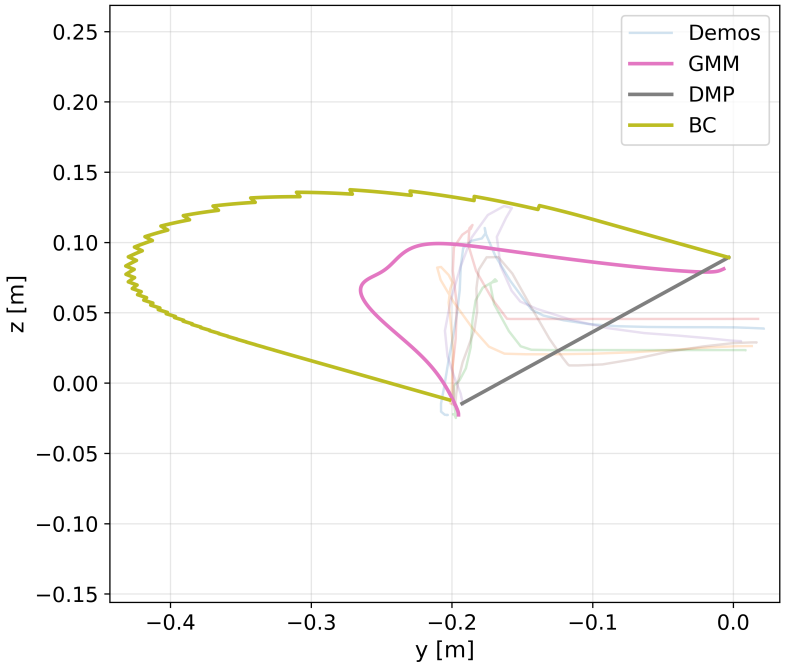


Figure 5.1: PICK - Projection on YZ plane

negligible. As before, DMP confirm its dominance with respect to other models when considering the precision in object fulfillment EE. GMM holds, also in this case, best performances on d_H and J. As an example, the projection on the XY-plane is shown in Figure 5.3.

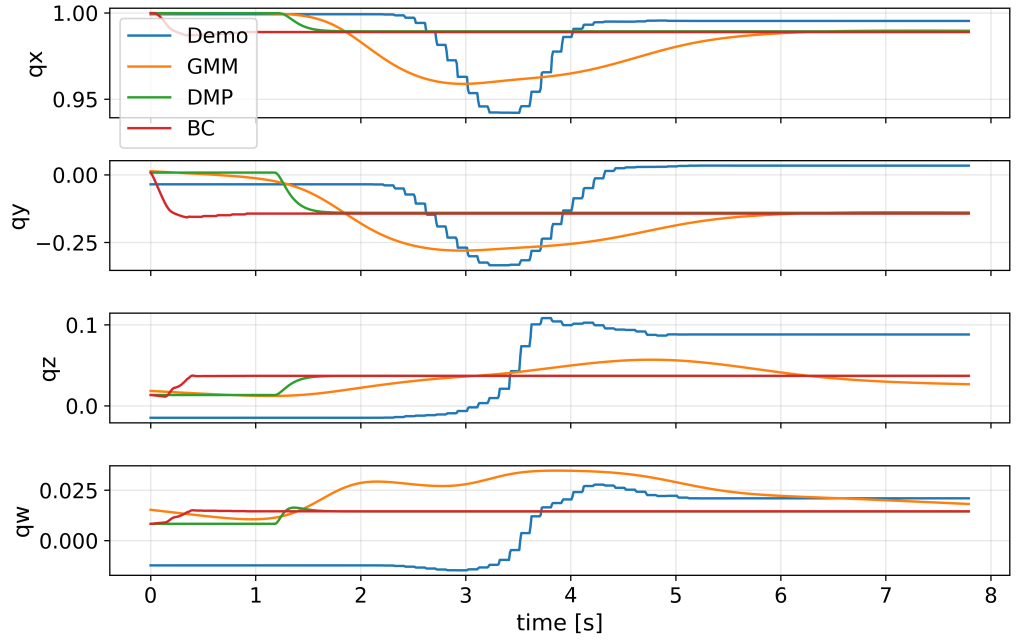


Figure 5.2: `PICK` - Quaternion components timeseries

	$RMSE_{\text{pos}}$	$RMSE_{\text{ori}}$	EE_{pos}	EE_{ori}	d_H	J	t_{train}
BC	**0.14**	**36.02**	7.95E-04	0.41	**0.11**	755.22	35.21
DMP	0.25	47.13	**3.35E-07**	**0.0002**	0.21	469.69	**0.45**
GMM	0.17	39.18	2.12E-01	11.27	0.20	**162.69**	2.10

Table 5.3: Quantitative results (`PLACE`)

5.1.3 `POUR`

Considering the `POUR` skill, the best models as far as the *accuracy* is concerned are BC, having best $RMSE_{pos}$ and d_H and GMM having the best $RMSE_{ori}$ index. DMP, instead is the best for training time t_{train} and object fulfillment. GMM remains the best model for smoothness of the generated trajectories. XZ-plane projection of the generated trajectories and position timeseries are given respectively in Figure 5.4 and Figure 5.5.

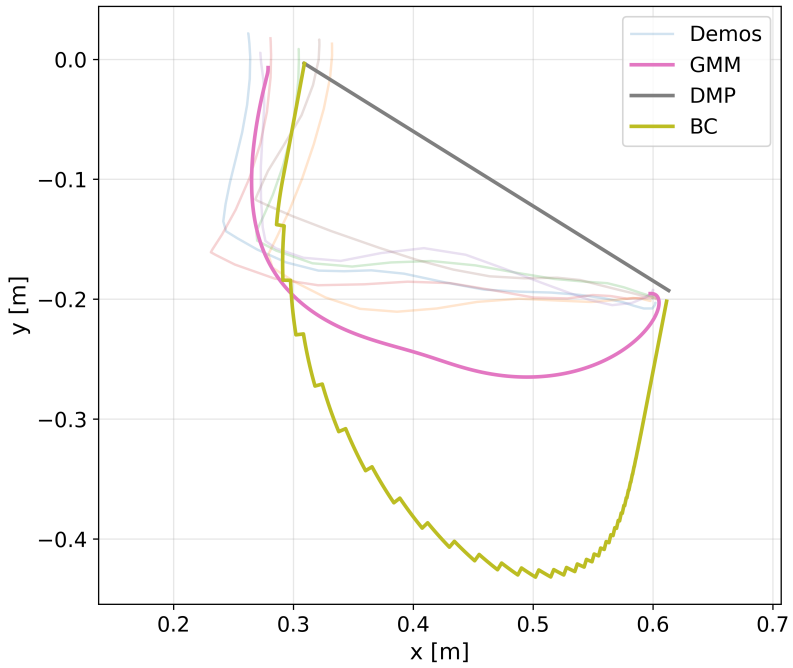


Figure 5.3: `PLACE` - Projection on XY plane

	$RMSE_{\text{pos}}$	$RMSE_{\text{ori}}$	EE_{pos}	EE_{ori}	d_H	J	t_{train}
BC	**0.09**	59.27	5.45e-04	0.35	**0.14**	892.38	51.47
DMP	0.15	61.13	**2.8e-07**	**2.3E-04**	0.31	809.05	**0.67**
GMM	0.09	**17.38**	4.3e-02	24.07	0.22	**473.60**	3.06

Table 5.4: Quantitative results (`POUR`)

5.1.4 `SHAKE`

Differently than `PICK`, `PLACE` and `POUR` that are discrete (point-to-point) movements, `SHAKE` is a ripetitive and hence periodic movement, as we said before. The learning methods we used so far, are not suitable fot this purpose, at least in the original formulation. In particular, some other theoretical aspects must be included which go into the direction of handling *frequency* and *phase* terms.
Despite this fact, we remind that there is another strategy to reproduce demonstrated movement skipping the learning phase: this is what we call *trajectory replaying*. This is similar to the fact we had learned the identity function $f(\mathbf{x}) = \mathbf{x}$.

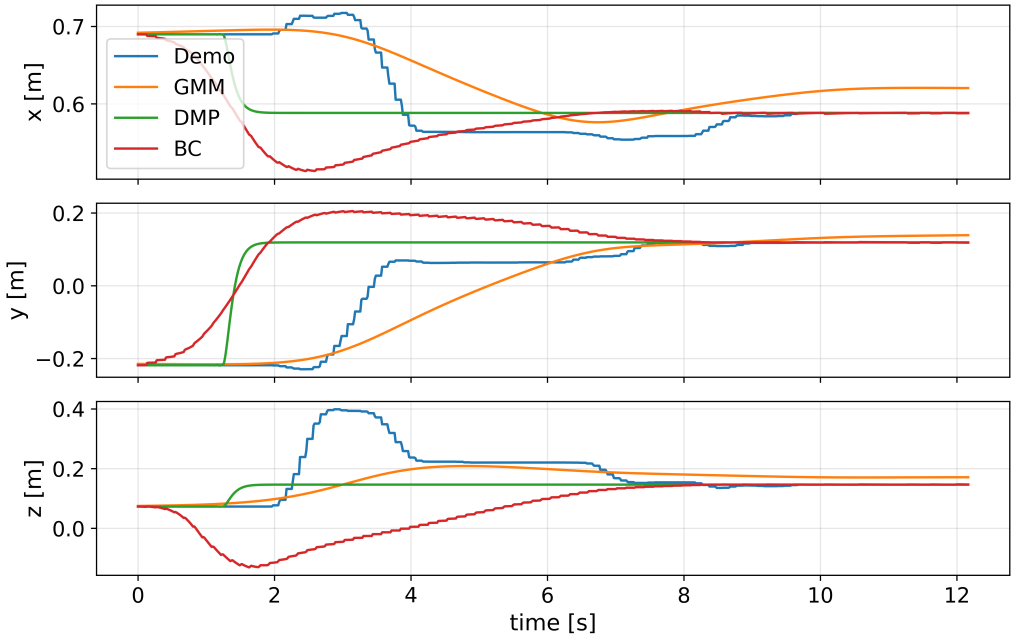


Figure 5.4: POUR - Position (x, y, z) timeseries

It is quite clear – at this point – that all of the advantages a learning method can offer are lost: generalization, re-parametrization and so on. In the work [37], there is an illustration of demonstrated `WIPING` skills. These movements are very similar to our demonstrated `SHAKE` primitives. For this reason, we give it for the sake of completeness in Figure 5.6.

5.2 High-level tasks

With the objective of evaluating high-level tasks, a certain number of experiments, for each task, have been executed and for each of them we monitor: the **Task Success Rate (TSR)** and the **Subtask Success Rate (SSR)**. Since high-level movements are aposteriori composed, one can choose different models for different parts according to what is more important for that specific task. Summarizing tables for obtained success metrics and used learning models are reported hereafter for each high-level executed task.
Remark: The number of experiment N_{exp} to be executed has been chosen according to the task complexity and success variability.

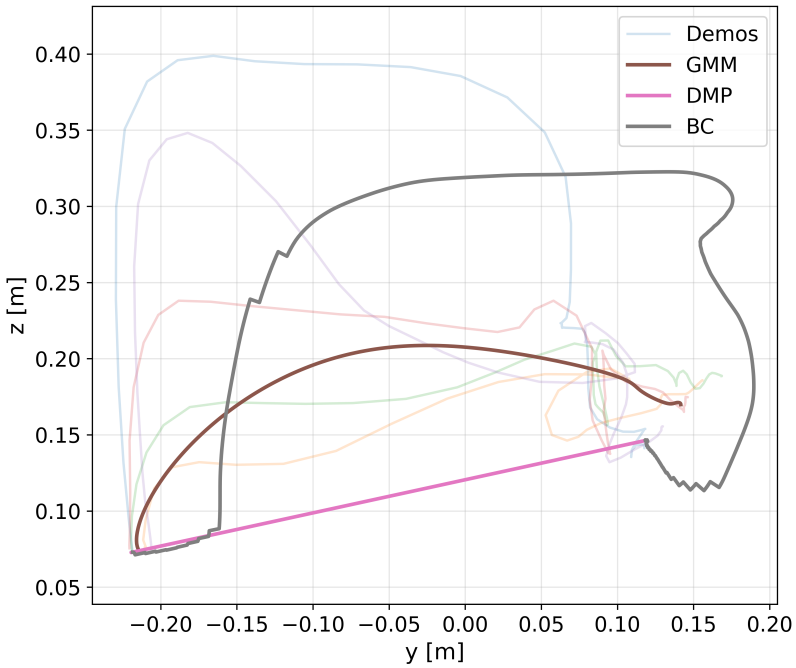


Figure 5.5: POUR - Projection on XZ plane

5.2.1 Pick-and-place, collecting objects

`PICK_AND_PLACE` task has been executed in its "standard form", that is without using vision and, then, task parametrization. For both `PICK` and `PLACE` phases, GMM is used as motion encoder. This leads to excellent TSR and SSR as reported in Table 5.5.

Differently, `COLLECT_OBJECTS` task is a more challenging one due to several factors:(i) the `PICK_AND_PLACE` task is repeated and the impact of random events is higher, (ii) the vision subsystem is used, wrong or faulting tags detection imply a partially/completely wrong execution of the task. Overall, we have good results (further details can be found in Table 5.5), the phase causing the experiment fail is `PICK` mainly due to incorrect tag's detection and pose estimation.
Table 5.6 illustrates the choices of the model for each subtask for both presented high-level tasks.

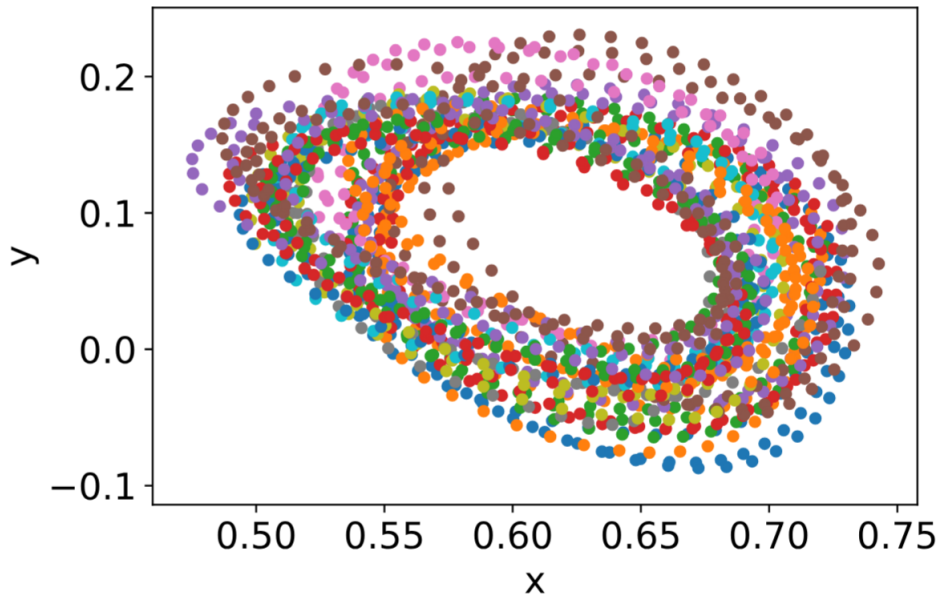


Figure 5.6: Demonstrations in the xy plane (different colors represent different demonstrations) [37]

	N_{exp}	TSR	$SSR_{\texttt{PICK}}$	$SSR_{\texttt{PLACE}}$
PICK_AND_PLACE	5	100%	100%	100%
COLLECT_OBJECTS	15	73.3%	73.3%	100%

Table 5.5: High-level task evaluation: PICK_AND_PLACE, COLLECT_OBJECTS

	PICK	PLACE
PICK_AND_PLACE	GMM	GMM
COLLECT_OBJECTS	DMP	DMP

Table 5.6: Learning models used for each subtask

5.2.2 Pour a drink, prepare a mixture

With the objective of focusing the attention on the movement, POUR skill has been executed with fixed *start* and *goal* positions and without liquid inside the bottle. Overall, 9/10 of the experiments were successufully executed; the error was during

the PLACE phase.

The final PREPARE_MIXTURE experiment has more variability due to: i) multiple bottles to handle, ii) liquid inside the bottle, iii) parametrization of the obtained trajectories in order to reuse skills without re-training. Despite this fact, we have obtained very good success rates as in previous tasks.
Similarly than before, task and subtask success rates and per-subtask used motion encoder are reported in Table 5.7 and Table 5.8.

	N_{exp}	TSR	$SSR_{\texttt{PICK}}$	$SSR_{\texttt{PLACE}}$	$SSR_{\texttt{POUR}}$
POUR_DRINK	10	90%	100%	90%	90%
PREPARE_MIXTURE	10	80%	100%	100%	80%

Table 5.7: Success rates (task and subtask) for POUR_DRINK

	PICK	PLACE	POUR	SHAKE
POUR_DRINK	GMM	GMM	DMP	trajectory replay
PREPARE_MIXTURE	DMP	DMP	DMP	trajectory replay

Table 5.8: Learning models for each subtask

Chapter 6

Conclusion

In this work we have explored how to make collaborative manipulation **learnable** and **reusable** by implementing an end-to-end Learning from Demonstration pipeline tailored for the `UFACTORY xArm6` cobot: from kinesthetic demonstration collection to real and simulated robot execution. In particular, the pipeline is used for low-level skill learning, which can be arranged in customized high-level plans. In addition, the use of a **vision subsystem** (Intel RealSense + ArUCO) allows to parametrize the skills to newly dynamically obtained goal poses. This is reducing per-task motion engineering while retaining generality.

6.1 Methodology

From the methodological point of view:

1. Three complementary *motion encoders* were implemented and trained: **BC** for policy imitation, **DMP** for retargetable movement primitives, **GMM-GMR** for probabilistic trajectory synthesis;

2. Decoded generated trajectories were executed with a fine-grained controller for the real robot, while **MoveIt** was used for simulation purposes; thus solving the Sim2Real gap;

3. Evaluation criteria covered geometric accuracy, objective fulfillment and motion quality. These naturally expose a thread-off between methods: GMM-GMR tends to give – overall – more accurate results, while DMP simplifies the goal re-targeting.

6.2 Contributions

The main contributions of this work are:

- A modular **LfD pipeline** from demonstration to real-robot execution;
- A comparative implementation of **three learning methods** on the same platform and task.
- A vision-based parametrization layer that retrieves task frames to **reuse skills** without retraining.
- Programmatic **task-level controllers** that are able to compose learnable primitives into reliable plans.

6.3 Limitations

Since some assumptions were done, there are some limits on what we have presented so far. In particular:

1. The vision front-end uses ArUco-based pose retrievial and a static geometric glossary, that is implicitly constraining the grasp synthesis;
2. `SHAKE` primitive was replayed rather than learned with a tailored learning method for rhythmic movements.
3. **High-level plans** were composed in a programmatic way instead of relying on a *learned symbolic model*

6.4 Future works

In Chapter 2 we have seen that for each step of the Learning from demonstration pipeline there is a certain number of choices that can be made. Developing the present work, we chose a *particular path* in a big tree of possibilities, making a set of assumptions while taking into account available time and resources.
Different skills have been tested – `PICK, PLACE, POUR, SHAKE` which – then – have been composed into higher-level tasks (pick-and-place, collect objects, single and multiple drink pouring).

Some extensions are possible, involving both practical and theoretical aspects about learning methods and LfD pipeline implementation.

Integrate vision subsystem with `GraspNet-1B` We retrieved task parameters such as ArUCO tag poses by using the `OpenCV` library on RGB images from the Realsense. The depth stream was not used at all; there are some interesting papers [38] that – relying on both color and depth channels – retrieve from the scene, for each object the so-called *grasp groups*, data structures containing

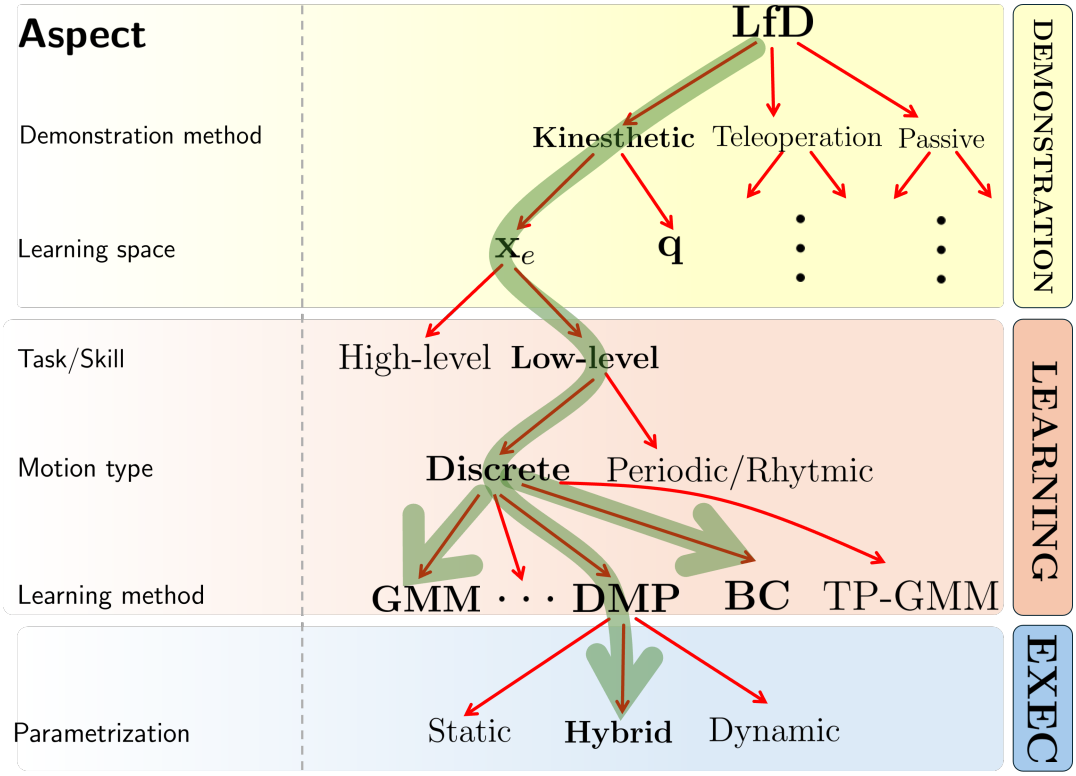


Figure 6.1: LfD pipeline and related choices for each aspect

information about the pose of the object together with their geometrical properties. In other words, such a network solves one-shot the problem we have split into two parts: one devoted to poses (dynamically solved), the other to object dimensions (statically obtained). The schema of such a modified vision system is showed in Figure 6.2.

TP-GMM A fundamental feature of LfD-based learned skills is the parametrization, that is not naturally embedded for each method like in DMP. We explored parametrization only for DMP, however an extension of GMM is possible, introducing the so-called *Task Parametrized GMM* (TP-GMM) [6]

High-level learning methods We adopted *programmatic task composition* for obtaining high-level tasks. Integrating high-level learning methods based on Hidden Markov Models (HMM), Behavior Trees (BT) and Finite State Machine (FSM) allows to complete the impleted pipeline for learning both low and high level. The use of tailored evaluation metrics would be required.

Introducing rhythmic movement Our work was mainly devoted to the low-level learning of *discrete* (or *point-to-point*) movements. Another category of motions which often can occur in practical applications is *periodic or rhytmic motions*. Different learning method have been explored. Just for giving two examples, *Rhytmic DMP* [40] and *Fourier Movement Primitives* [37].

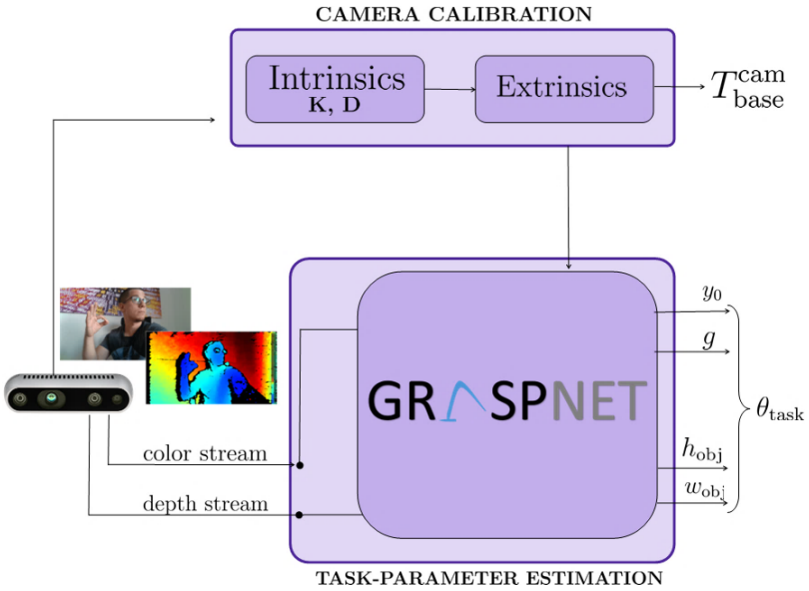


Figure 6.2: Modified vision subsystem relying on GraspNet-1B [38]: static glossary is replaced by a dedicated *grasp proposal network*

These further steps would improve the overall system making it more autonomous, since *manual assumptions* are reduced. However, collectively the obtained results show that few LfD demonstrations can deliver accurate, adaptable and maintanable behavior while having low integration efforts.

Appendix A

Basics of Markov Decision Processes in RL and LfD

Markovian Decision Process is the mathematical tool by which we can provide the main theoretical results and formal proofs for both RL and LfD. In general, a MDP is defined as the tuple:

$$< S, A, \Delta_0, P, R, \lambda > \tag{A.1}$$

where S is the set of the possible states $s_t \in S$, A the set of the possible actions $a_t \in S$, Δ_0 is the probability distribution of the initial state s_0; $P(s_{t+1}|s_t, a_t)$ is the *probability transition function*; given the state s_t an action is chosen using a certain **policy** $\pi : S \to A$. Finally, after the interaction with the environment at time t a certain reward $r = R(s_t, a_t, s_{t+1})$ is obtained where R is the **reward function** and λ is the discount factor. Now, in RL the objective is maximizing the cumulative reward given by

$$G_t = \mathbb{E}[\sum_{t=0}^{\infty} \lambda^t R(s_t, a_t, s_{t+1})] \tag{A.2}$$

A trajectory τ is defined as the set of $H + 1$ states and H actions and reward, that is

$$\tau = (s_0, a_0, r_0, s_1, a_1, r_1, ..., s_H) \tag{A.3}$$

where H is the trajectory length or *epidsode horizon*. In LfD, a demonstration dataset D_{demo} of N samples is defined as:

$$D_{demo} = \{\tau_i, i \in [0, N)\} \tag{A.4}$$

where the single demonstration is

$$\tau_i = (s_t, a_t, s_{t+1}, t \in [0, L)) \tag{A.5}$$

It appears quite clear that a_t here is chosen by the demonstrator according its own policy $\pi_{teacher}$. Finally, considerіding the (A.5), its probability under the policy π is given by:

$$p_\pi(\tau) = \Delta_0(s_9) \cdot \prod_{t=0}^{H-1} \pi(a_t|s_t) \cdot P(s_{t+1}|s_t, a_t) \tag{A.6}$$

Appendix B

Fiducial markers in Robotics

Fiducial markers are simple patterns with known geometry that you can attach to objects in order to identify them and *detect the 6DOF pose* in a precise way. Some examples are reported in Figure B.1. A more detailed explanation can be found in the article by Jurado-Rodriguez et al. [41].

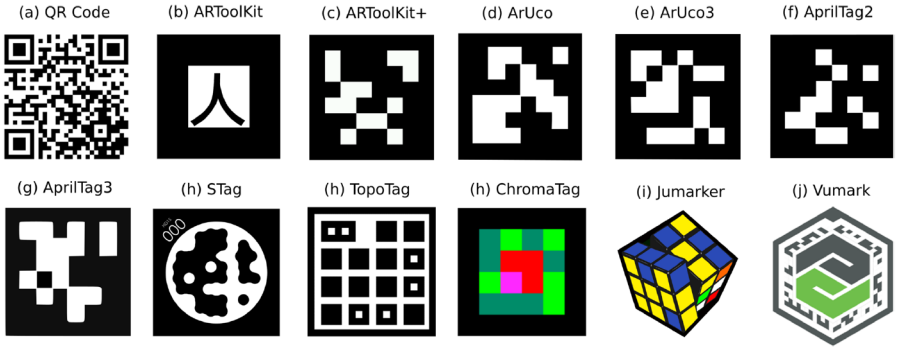


Figure B.1: Examples of fiducial markers [41]

In the following there is a synthesis of the core ideas behind such type of identifiers:

1. An **engineered pattern** makes them *easy to find*. In particular they have: (i) black and white borders; (ii) a square shape, in this way some thresholding technique can be used in order to detect them;

2. They have **encoded bits** so that they can have a unique identifier which is related in particular to the grid into the square. In this way

$$\text{UNIQUE TAG} \rightarrow \text{UNIQUE CODE} \rightarrow \text{UNIQUE OBJECT}.$$

3. The **known geometry** enable the pose recognition from a single view. More in details, the square constituting the code is a known-size one, whose corners can be detected and made an homography. At this point a PnP (Pin-hole) problem is solved to retrieve the object pose by an homogeneous transformation matrix.

4. They allow an **object recognition via association** in a way that each identifier can be mapped to an object/class.

B.1 ArUco marker detection process

In our work we used *ArUco*[1] fiducial tags, whose implementation can be found in the paper [42] from 2014. Hereafter, we are going through the fundamental steps, illustrated in Figure B.2, that allows the tag's code retrievial starting the RGB image. Given a certain tag dictionary (e.g., 4x4, 5x5, etc.)

1. *Image segmentation*: the RGB image (Figure B.2(a)) is converted in a gray-scale one, from this an intermediate representation is extracted using some thresholding approach (e.g., Canny or some other more robust method) (Figure B.2(b)).

2. *Contour extraction and filtering*: Next, a contour extraction procedure is carried out. Most of the obtained contours are irrelevant for our purposes, for this reason most of them are discarded (Figure B.2(c));

3. *Marker Code extraction*: for each of the obtained contours the internal area is analyzed, removing the perspective, with the objective of extracting the associated code. An useful step for doing so is obtaining a binary image, which is afterwards dividd into a regular grid whose elements are associated, by using a *majority voting technique*, with 0 or 1 (see Figure B.2(d)-(f)).

The reference paper [42] gives, also, more details for generating marker dictionaries, and other technicalities that go outside the purposes of this Appendix.

[1] *ArUco* stands for *Augmented Reality University of Cordoba*

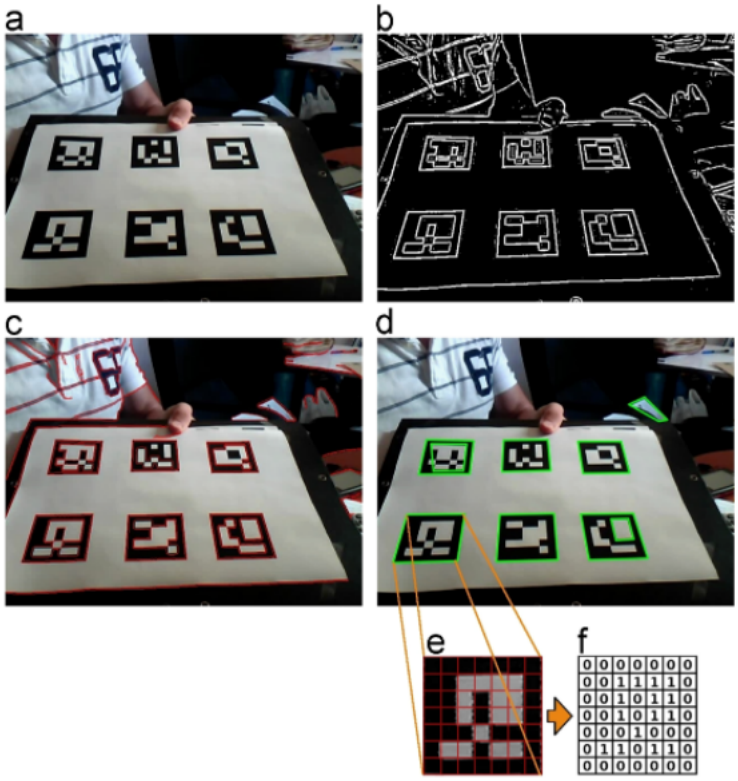


Figure B.2: Image process for automatic marker detection. (a) Original image. (b) Result of applying local thresholding. (c) Contour detection. (d) Polygonal approximation and removal of irrelevant contours. (e) Example of marker after perspective transformation. (f) Bit assignment for each cell. [42]

Figure B.3: Examples of generated ArUco of different sizes n

Appendix C

Robotic vision system, Intel RealSense D435 camera

C.1 Configuration of the visual system

"Vision allows a robotic system to obtain geometrical and qualitative information on the surrounding environment to be used both for motion planning and control" [43]. For a robotic system, the *vision subsystem* consist of one or more cameras. As far as robotic manipulators are concerned, the main feature of the vision layer is the *camera placement*. In the fixed configuration there are two alternatives:

1. *eye-to-hand*: in this case the camera assumes a fixed pose with respect to the base frame of the robotic arm. The advantage of such a choice is that the vision sensor observes always the same scene, however this is a disadvantage in some complex applications (e.g., assembly) where the occlusion problem is present;
2. *eye-in-hand*: is the configuration in which the camera is mounted on the manipulator's wrist. Now, the scena changes for the camera, however, we can say that occlusions are absent. Finally, the accuracy is in general higher with respect to the other approach.

The two configurations are illustrated in Figure C.1 from the paper [44].

C.2 The Intel Realsense D435 camera

The Intel RealSense D435 is a depth camera equipped with stereo vision and an integrated IMU (Inertial Measurement Unit). It provides RGB and depth data, enabling at the same time 3D percpetion which can be useful for robotics and

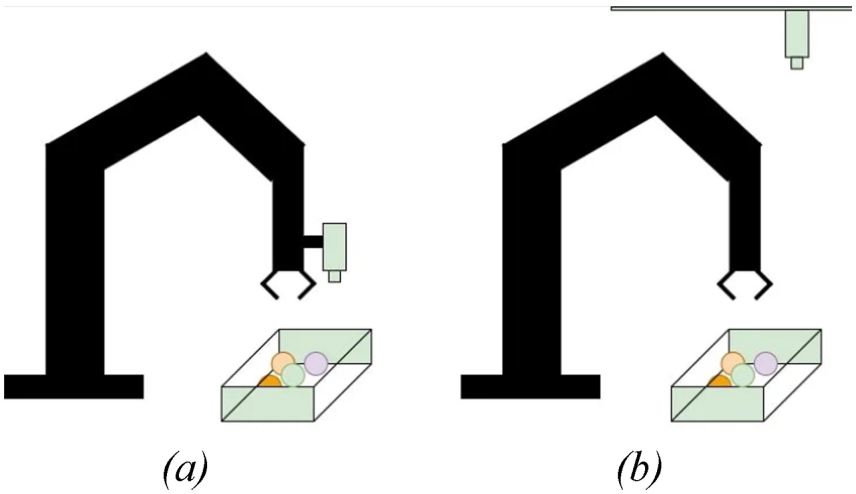


Figure C.1: *Eye-in-hand*(a) and *Eye-to-hand* (b) configurations [44]

SLAM applications. Its compact design and USB connectivity makes it suitable for real-time spatial sensing tasks. In the following there is a section dedicated to specifications.

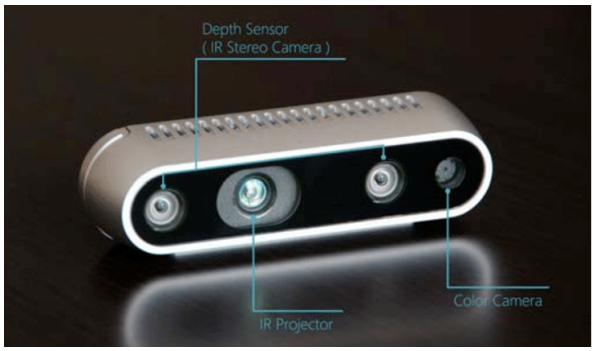


Figure C.2: Intel RealSense d435 camera

C.2.1 Specifications

The Intel RealSense D435 camera features:

- **Depth Technology:** Stereo vision
- **RGB Sensor:** 1920 × 1080 resolution
- **Depth Resolution:** Up to 1280 × 720 at 30 FPS
- **Field of View:** 86° × 57° (depth), 69° × 42° (RGB)
- **IMU:** 6-axis (accelerometer and gyroscope)

- **Range:** 0.1 m to 10 m (optimal up to 3 m)
- **Interface:** USB 3.0
- **Dimensions:** 90 mm × 25 mm × 25 mm
- **Weight:** 72 g

C.2.2 Camera Calibration Procedure

The **camera calibration procedure** allows the *image measurements* to be consistently related to the *robot coordinate frames*. Main reference for this part is the book [45] in which a detailed description of related topics (e.g., perspective transformation, camera model) is provided. Hereafter, we are providing some insights on main parameters that are needed for the calibration procedure:

- *Camera Intrinsics*: they are the ones allowing the transformation between camera coordinates (X_C, Y_C, Z_C) and the 2D pixels' image representation:

$$(X_C, Y_C, Z_C) \rightarrow (u, v) \tag{C.1}$$

- *Camera Extrinsics*: they allows the conversion between the *world and camera coordinates*:

$$(X_W, Y_W, Z_W) \rightarrow (X_C, Y_C, Z_C) \tag{C.2}$$

A schema illustrating the flow from world to image is shown in Figure C.3, which adds more details to the previously provided schemas.

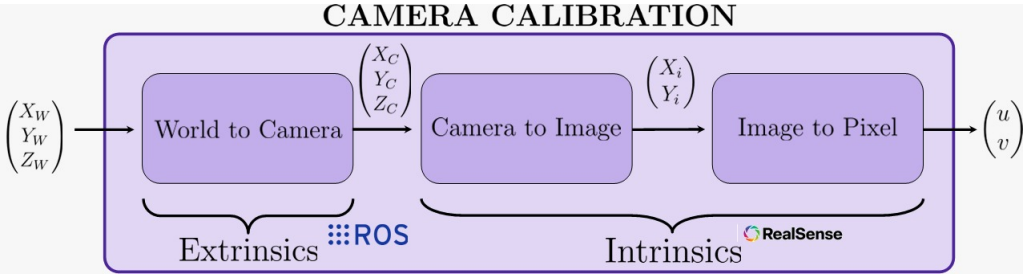


Figure C.3: Camera Calibration. From world coordinates to pixels and viceversa

Intrinsics retrieval

In this work, we used the factory-provided camera intrinsics. Alternatively, some procedure based on a *chessboard* could have been used. More in details, the intrinsics parameters are:

- **Focal length and principal point** which are for providing a model of the *projective geometry*;
- **Distortion coefficients** are used so that straight lines in the scene could remain straight in the obtained image.

Such a set of parameters can be used for both camera → world and world → camera transformations.

Extrinsics retrieval

Camera coordinates, in order to be effectively used must be transformed into world coordinates. This task is carried out introducing another set of parameters, which are called **Camera Extrinsics**. To be more specific, here we are dealing with a *static transformation matrix* that describes what is the relationship between the camera and the end-effector (ee) reference frames. Such a matrix has the following structure:

$$T_{cam}^{ee} = \begin{bmatrix} \mathbf{R}_{cam}^{ee} & \mathbf{t}_{cam}^{ee} \\ 0^{1\times 3} & 1 \end{bmatrix} \tag{C.3}$$

where $\mathbf{R}_{cam}^{ee}$ and $\mathbf{t}_{cam}^{ee}$ are respectively the *rotation* and *translation* between the two reference frames:

$$\mathcal{R}_{cam} = \left\{ O_{cam}, \mathbf{x}_{cam}, \mathbf{y}_{cam}, \mathbf{z}_{cam} \right\} \tag{C.4}$$

and

$$\mathcal{R}_{ee} = \left\{ O_{ee}, \mathbf{x}_{ee}, \mathbf{y}_{ee}, \mathbf{z}_{ee} \right\} \tag{C.5}$$

adopting the formalism used in [46]. Once the end-effector coordinates have been obtained, another transformation matrix T_{world}^{ee} is used for obtaining *world coordinates*.

Bibliography

[1] Mohd Javaid, Abid Haleem, Ravi Pratap Singh, Shanay Rab, and Rajiv Suman. «Significant applications of Cobots in the field of manufacturing». In: *Cognitive Robotics* 2 (Jan. 2022), pp. 222–233. ISSN: 2667-2413. DOI: 10.1016/j.cogr.2022.10.001. URL: https://www.sciencedirect.com/science/article/pii/S2667241322000209 (visited on 03/28/2025) (cit. on pp. 1, 2).

[2] Harish Ravichandar, Athanasios S Polydoros, Sonia Chernova, and Aude Billard. «Recent advances in robot learning from demonstration». In: *Annual review of control, robotics, and autonomous systems* 3.1 (2020), pp. 297–330 (cit. on pp. 2, 10, 12).

[3] Sonia Chernova and Andrea L Thomaz. *Robot learning from human teachers*. Springer Nature, 2022 (cit. on pp. 3, 6, 63, 67).

[4] Alireza Barekatain, Hamed Habibi, and Holger Voos. «A practical roadmap to learning from demonstration for robotic manipulators in manufacturing». In: *Robotics* 13.7 (2024), p. 100 (cit. on pp. 3, 7–9, 21, 26, 27).

[5] Andr Correia and A. Alexandre. «A survey of demonstration learning». In: *Robotics and Autonomous Systems* 182 (Dec. 2024), p. 104812. ISSN: 0921-8890. DOI: 10.1016/j.robot.2024.104812. URL: https://www.sciencedirect.com/science/article/pii/S0921889024001969 (visited on 03/28/2025) (cit. on pp. 5, 7, 9).

[6] Weidong Li, Yuqi Wang, Yuchen Liang, and Duc Truong Pham. «Learning from demonstration for autonomous generation of robotic trajectory: Status quo and forward-looking overview». In: *Advanced Engineering Informatics* 62 (2024), p. 102625 (cit. on pp. 10, 34, 44, 88).

[7] Arturo Daniel Sosa-Ceron, Hugo Gustavo Gonzalez-Hernandez, and Jorge Antonio Reyes-Avendaño. «Learning from demonstrations in human–robot collaborative scenarios: A survey». In: *Robotics* 11.6 (2022), p. 126 (cit. on p. 10).

[8] Ruchik Thaker. «Human-Robot Interaction: Designing Robots That Can Naturally Interact and Collaborate With Humans». In: 6 (Aug. 2020). DOI: 10.5281/zenodo.14001622 (cit. on p. 12).

[9] John J Dudley and Per Ola Kristensson. «A review of user interface design for interactive machine learning». In: *ACM Transactions on Interactive Intelligent Systems (TiiS)* 8.2 (2018), pp. 1–37 (cit. on p. 12).

[10] Carlos Celemin et al. «Interactive imitation learning in robotics: A survey». In: *Foundations and Trends® in Robotics* 10.1-2 (2022), pp. 1–197 (cit. on p. 14).

[11] Riad Akrour, Marc Schoenauer, and Michèle Sebag. «Preference-Based Policy Learning». In: Sept. 2011, pp. 12–27. ISBN: 978-3-642-23779-9. DOI: 10.1007/978-3-642-23780-5_11 (cit. on p. 16).

[12] W Bradley Knox and Peter Stone. «Combining manual feedback with subsequent MDP reward signals for reinforcement learning.» In: *AAMAS*. Vol. 10. 2010, pp. 5–12 (cit. on p. 16).

[13] W Bradley Knox and Peter Stone. «TAMER: Training an agent manually via evaluative reinforcement». In: *2008 7th IEEE international conference on development and learning*. IEEE. 2008, pp. 292–297 (cit. on p. 16).

[14] W Bradley Knox and Peter Stone. «Learning non-myopically from human-generated reward». In: *Proceedings of the 2013 international conference on Intelligent user interfaces*. 2013, pp. 191–202 (cit. on p. 16).

[15] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. «A reduction of imitation learning and structured prediction to no-regret online learning». In: *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings. 2011, pp. 627–635 (cit. on pp. 17, 18).

[16] Brenna D Argall. *Learning mobile robot motion control from demonstration and corrective feedback*. Carnegie Mellon University, 2009 (cit. on p. 17).

[17] Michael Kelly, Chelsea Sidrane, Katherine Rose Driggs-Campbell, and Mykel J. Kochenderfer. «HG-DAgger: Interactive Imitation Learning with Human Experts». In: *CoRR* abs/1810.02890 (2018). arXiv: 1810.02890. URL: http://arxiv.org/abs/1810.02890 (cit. on p. 18).

[18] Carlos Celemin and Javier Ruiz-del-Solar. «Coach: Learning continuous actions from corrective advice communicated by humans». In: *2015 International Conference on Advanced Robotics (ICAR)*. IEEE. 2015, pp. 581–586 (cit. on p. 19).

[19] Zuyuan Zhu and Huosheng Hu. «Robot Learning from Demonstration in Robotic Assembly: A Survey». In: *Robotics* 7.2 (2018), p. 17. DOI: 10.3390/robotics7020017. URL: https://www.mdpi.com/2218-6581/7/2/17 (cit. on pp. 22, 24, 26).

[20] Victor Hernandez Moreno, Steffen Jansing, Mikhail Polikarpov, Marc G. Carmichael, and Jochen Deuse. «Obstacles and Opportunities for Learning from Demonstration in Practical Industrial Assembly: A Systematic Literature Review». In: *arXiv preprint arXiv:2310.00276* (2023). URL: https://arxiv.org/abs/2310.00276 (cit. on pp. 22, 23, 25).

[21] D.A. Duque, F.A. Prieto, and J.G. Hoyos. «Trajectory generation for robotic assembly operations using learning by demonstration». In: *Robotics and Computer-Integrated Manufacturing* 57 (2019), pp. 292–302. DOI: 10.1016/j.rcim.2018.12.007. URL: https://doi.org/10.1016/j.rcim.2018.12.007 (cit. on pp. 23, 24).

[22] ZongWu Xie, Qi Zhang, ZaiNan Jiang, and Hong Liu. «Robot learning from demonstration for path planning: A review». In: *Science China Technological Sciences* 63.8 (2020), pp. 1325–1334 (cit. on p. 28).

[23] Cristian C Beltran-Hernandez, Damien Petit, Ixchel G Ramirez-Alpizar, and Kensuke Harada. «Accelerating robot learning of contact-rich manipulations: A curriculum learning study». In: *arXiv preprint arXiv:2204.12844* (2022) (cit. on p. 28).

[24] Janis Arents, Valters Abolins, Janis Judvaitis, Oskars Vismanis, Aly Oraby, and Kaspars Ozols. «Human–robot collaboration trends and safety aspects: A systematic review». In: *Journal of Sensor and Actuator Networks* 10.3 (2021), p. 48 (cit. on pp. 28, 29).

[25] *ISO/TS 15066:2016 Robots and robotic devices – Collaborative robots*. Geneva, Switzerland: International Organization for Standardization, 2016. URL: https://www.iso.org/obp/ui/#iso:std:iso:ts:15066:ed-1:v1:en (cit. on p. 29).

[26] Weidong Li, Yudie Hu, Yong Zhou, and Duc Truong Pham. «Safe human–robot collaboration for industrial settings: a survey». In: *Journal of Intelligent Manufacturing* 35.5 (2024), pp. 2235–2261 (cit. on pp. 30, 31).

[27] Sylvain Calinon. «A tutorial on task-parameterized movement learning and retrieval». In: *Intelligent service robotics* 9.1 (2016), pp. 1–29 (cit. on p. 35).

[28] Auke Jan Ijspeert, Jun Nakanishi, Heiko Hoffmann, Peter Pastor, and Stefan Schaal. «Dynamical movement primitives: learning attractor models for motor behaviors». In: *Neural computation* 25.2 (2013), pp. 328–373 (cit. on p. 39).

[29] *UFACTORY xArm Collaborative Robot*. Product page. UFACTORY. 2023. URL: https://www.ufactory.cc/xarm-collaborative-robot/ (visited on 11/24/2025) (cit. on p. 42).

[30] The scikit-learn developers. *MLPRegressor — scikit-learn 1.7.2 documentation*. Version 1.7.2. scikit-learn. 2025. URL: https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPRegressor.html (visited on 11/05/2025) (cit. on p. 47).

[31] Wikipedia contributors. *Slerp — Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/wiki/Slerp. Accessed: 2025-10-07. 2025 (cit. on p. 48).

[32] Wikipedia contributors. *Quaternions and spatial rotation — Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/wiki/Quaternions_and_spatial_rotation. Accessed: 2025-10-07. 2025 (cit. on p. 48).

[33] xArm-Developer. *xArm-Python-SDK: Python SDK for UFACTORY robots*. https://github.com/xArm-Developer/xArm-Python-SDK. Accessed: 2025-10-07. 2025 (cit. on p. 52).

[34] Oleg Kalachev. *arucogen: Online ArUco markers generator*. https://github.com/okalachev/arucogen. GitHub repository. 2025 (cit. on p. 59).

[35] Bojan Nemec and Aleš Ude. «Action sequencing using dynamic movement primitives». In: *Robotica* 30.5 (Sept. 2012), pp. 837–846. ISSN: 0263-5747. DOI: 10.1017/S0263574711001056. URL: https://doi.org/10.1017/S0263574711001056 (cit. on p. 65).

[36] Mingshan Chi, Yufeng Yao, Yaxin Liu, Yiqian Teng, and Ming Zhong. «Learning motion primitives from demonstration». In: *Advances in Mechanical Engineering* 9.12 (2017). Article ID: 1687814017737260, pp. 1–13. DOI: 10.1177/1687814017737260. URL: https://journals.sagepub.com/doi/10.1177/1687814017737260 (cit. on p. 65).

[37] Thibaut Kulak, Joao Silverio, and Sylvain Calinon. «Fourier movement primitives: an approach for learning rhythmic robot skills from demonstrations». In: *Proceedings of Robotics: Science and Systems*. Corvalis, Oregon, USA, July 2020. DOI: 10.15607/RSS.2020.XVI.056 (cit. on pp. 66, 82, 84, 89).

[38] Hao-Shu Fang, Chenxi Wang, Minghao Gou, and Cewu Lu. «Graspnet-1billion: A large-scale benchmark for general object grasping». In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 11444–11453 (cit. on pp. 75, 87, 89).

[39] Cheems_JH (mouse826612011). *DMP_Simulation: A simulation project on Dynamic Movement Primitives (DMP)*. `https://github.com/mouse826612011/DMP_simulation`. GitHub repository. Latest commit: 3643e94 (2025-02-27). 2023. (Visited on 10/29/2025) (cit. on p. 76).

[40] Matteo Saveriano, Fares J Abu-Dakka, Aljaž Kramberger, and Luka Peternel. «Dynamic movement primitives in robotics: A tutorial survey». In: *The International Journal of Robotics Research* 42.13 (2023), pp. 1133–1184 (cit. on p. 89).

[41] David Jurado-Rodriguez, Rafael Muñoz-Salinas, Sergio Garrido-Jurado, and Rafael Medina-Carnicer. «Planar fiducial markers: A comparative study». In: *Virtual Reality* 27.3 (2023), pp. 1733–1749 (cit. on p. 92).

[42] Sergio Garrido-Jurado, Rafael Muñoz-Salinas, Francisco J. Madrid-Cuevas, and Manuel J. Marín-Jiménez. «Automatic generation and detection of highly reliable fiducial markers under occlusion». In: *Pattern Recognition* 47.6 (2014), pp. 2280–2292. DOI: `10.1016/j.patcog.2014.01.005` (cit. on pp. 93, 94).

[43] Bruno Siciliano, Lorenzo Sciavicco, Luigi Villani, and Giuseppe Oriolo. *Robotics: Modelling, Planning and Control*. Springer Publishing Company, Incorporated, 2010. ISBN: 1849966346 (cit. on p. 95).

[44] Suhas Kadalagere Sampath, Ning Wang, Hao Wu, and Chenguang Yang. «Review on human-like robot manipulation using dexterous hands». In: *Cognitive Computation and Systems* 5.1 (2023), pp. 14–29 (cit. on pp. 95, 96).

[45] King Sun Fu, R. C. Gonzalez, and C. S. G. Lee, eds. *Robotics: control, sensing, vision, and intelligence*. USA: McGraw-Hill, Inc., 1987. ISBN: 0070226253 (cit. on p. 97).

[46] Basilio Bona. *Dynamic Modelling of Mechatronic Systems*. Torino: CELID, 2018. ISBN: 9788867890118 (cit. on p. 98).