Mondragon Unibertsitatea

DOCTORAL THESIS

REINFORCEMENT LEARNING FOR COLLABORATIVE ROBOTIC CONTACT-RICH DISASSEMBLY TASKS

ANTONIO SERRANO MUÑOZ | Arrasate-Mondragón, 2023

ANTONIO SERRANO MUÑOZ | Reinforcement learning for collaborative robotic contact-rich disassembly tasks.

MONDRAGON UNIBERTSITATEA

PHD IN APPLIED ENGINEERING (ROBOTICS AND AUTOMATION)

Reinforcement learning for collaborative robotic contact-rich disassembly tasks

Author:

Antonio SERRANO MUÑOZ

Supervisors:

Dr. Nestor ARANA AREXOLALEIBA

Dr. Dimitrios CHRYSOSTOMOU

Robotics and Automation
Electronics and Computer Science

September 12, 2023

MONDRAGON UNIBERTSITATEA

Abstract

Reinforcement learning for collaborative robotic contact-rich disassembly tasks

by Antonio SERRANO MUÑOZ

With the exponential growth of the world's population and the resulting increase in consumption rates, the efficient treatment of end-of-life (EOL) products has become critical to mitigating environmental impacts. Remanufacturing offers an environmentally and economically beneficial approach to counteracting these impacts. While automation has been successful in assembly and manufacturing, manual labor is preferred in remanufacturing, especially disassembly, to cope with operational uncertainties. Reinforcement learning (RL) is presented as an alternative for decision making and control in changing systems, but the extent to which disassembly tasks can be automatically learned and generalized is unknown.

This doctoral dissertation, in Applied Engineering, explores the application of RL techniques for collaborative robot control to generalize disassembly tasks with uncertainties due to the variability of the geometric and physical properties of the manipulated objects. With this, a modular RL library that enables simultaneous training of agents in massively parallel environments is presented to reduce training time while consuming the same amount of resources and increasing the perceived reward. In addition, a control framework for KUKA LBR iiwa cobots that outperforms existing solutions and allows the use of different types of force overlays to reduce contact forces caused by friction and the probability of jamming states when performing disassembly tasks is presented. Furthermore, a collection of ready-to-use packages for rapid prototyping and reducing the development and deployment time of collaborative robotic systems for assembly and disassembly is proposed.

Contents

List of Figures

List of Tables

List of Abbreviations

EOL End-Of-Life
HRI Human-Robot Interaction
AI Artificial intelligence
ML Machine Learning
RL Reinforcement Learning
ROS Robot Operating System
ROS2 Robot Operating System - version 2
API Application Programming Interface
DOF Degree of Freedom
MDP Markov Decision Process
GUI Graphical User Interface
TCP Tool Center Point
ReLU Rectified Linear Unit
ELU Exponential Linear Unit
GPU Graphics Processing Unit

Chapter 1

Introduction

1.1 Introduction and motivation

During the last two centuries, a short time compared with the history of humanity, human and society have experimented significant socio-economic, political, and cultural transformations [1][2] characterized by unprecedented advances in technology. These transformations have been historically labeled by periods called "Industrial Revolutions".

The First Industrial Revolution, between 1760 and 1840, began with the introduction of steam power into manufacturing processes and led to increased mechanization and enabled industrial-scale mass production. It also induced changes in existing social structures, such as labor laws, educational systems, transportation networks, and communication technologies, which had far-reaching effects on society at large. Productivity, trade, and demographics increased.

The second Industrial Revolution took place at the end of the 19th century, when electricity use became widespread throughout Europe. It led to new developments, such as the introduction of assembly lines in factories, which made it possible to increase efficiency and reduce the costs associated with manual labor, as well as consumer prices. In addition, new forms of energy were developed, leading to an improvement in transportation infrastructure, such as the railroad, which enabled long-distance transport and the geographic expansion of commerce. New products appeared and demographics grew again.

The Third Industrial Revolution (also known as "The Digital Revolution") began in the middle of the last century. It saw a gigantic advance in communication technologies, the development and use of the Internet, and new sources of energy production such as natural gas, nuclear, solar and wind power.

Today, there is a trend towards automation [3] and data exchange [4][5] in manufacturing technologies, including advances in Artificial intelligence (AI), Internet of Things and cloud computing. Such a trend is presented as the Fourth Industrial Revolution (or Industry 4.0) and aims to increase efficiency, reduce costs and improve the overall quality of products and services [6][7].

Industrial revolutions have changed how people work, communicate and live [8], but they have also caused serious environment problems. The population explosion has led to overpopulation and, with it, a negative impact on nature [9]. As the world's population grows exponentially, consumption rates and demand for new products also increase dramatically, and with it the waste generation [10]. Environmental consequences include pollution, desertification and drought, depletion

of natural, energy and biological resources, climate change, among others.

In this global scenario, the circular economy is presented as a new and necessary transition or political action to counteract the environmental problems caused by the linear economy system [10][11]. This model proposes to close the production cycle by reusing waste, reducing the use of essential resources, and changing energy sources [12]. In this context, the responsible treatment of End-Of-Life (EOL) products is a key element in achieving the goals of this economic model.

Responsible EOL treatment, which may include reusing, recycling, or remanufacturing [13] products or parts can be beneficial both environmentally [14] and economically [15][16]. Waste is minimized, while valuable components and materials are recovered [17].

1.1.1 Remanufacturing

Remanufacturing plays a fundamental role in the circular economy model, which aims to reduce waste, increase production efficiency and prolong the product's useful life [18]. The essence of remanufacturing is to recover a used or discarded product and return it to a near-original state so it can be used again.

The main steps in the remanufacturing cycle are EOL product (core) acquisition, disassembly, condition assessment, cleaning, repair, assembly, service life and core return [19][20], as shown in Figure 1.1. Of these, disassembly allows the other steps to be carried out.

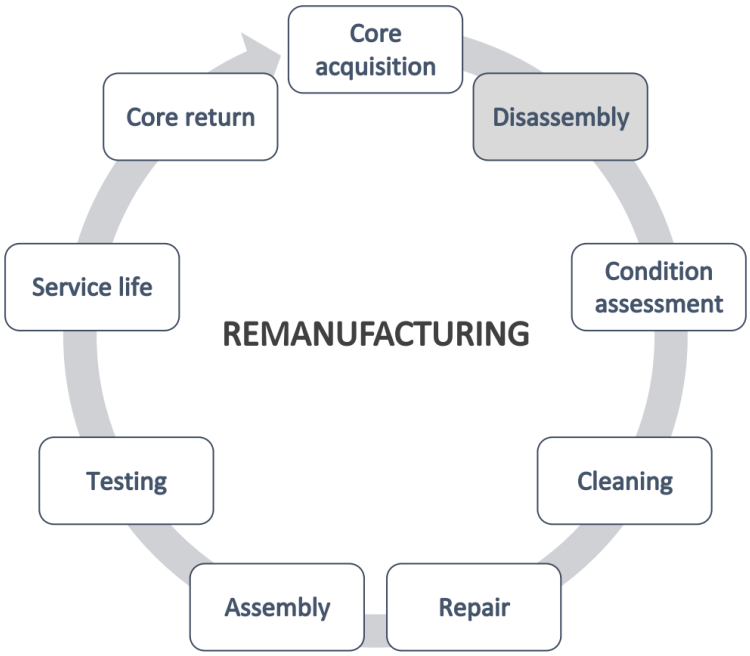

FIGURE 1.1: Remanufacturing cycle.

1.1.1.1 Disassembly

In general, disassembly for remanufacturing involves breaking down a used product into its individual components in order to repair or recondition those components

for reuse in a new product [21]. It also makes it possible to identify the components of a product that can be recycled and those to be disposed of [22].

The disassembly processes are divided into three main stages: disassembly line balancing [23][24][25], disassembly sequence planning [26][27], and disassembly task execution [28].

- Disassembly line balancing refers to the process of determining the optimal allocation of disassembly tasks across multiple workstations on a disassembly line. The goal of disassembly line balancing is to maximize the efficiency of the disassembly process by minimizing the total time required to process incoming EOL products.
- Disassembly sequence planning refers to the process of determining the order in which the various disassembly tasks should be performed. The order in which the tasks are performed can significantly affect the successful completion and efficiency of the disassembly process.
- The disassembly task refers to a specific operation that needs to be performed during the disassembly process. Examples of disassembly tasks might include removing fasteners, separating components, or performing inspections.

Industrial disassembly plants are supplied with a wide range of products from multiple manufacturers, each with its own characteristics, components and mounting/dismounting processes. These products have a high variability in their condition, whether due to storage and/or maintenance, operating conditions, service life, damage and repairs, among others. Such variability in both product range and product condition makes the disassembly process uncertain and complex requiring a high level of adaptability to change, identification and flexibility [28].

Automation has been successfully implemented for assembly and manufacturing [29][30]. However, for remanufacturing, and in particular for disassembly, manual labor is preferred [31]. Humans can adapt to changes in the disassembly process easily [32]. But recent technological advances have introduced collaborative robots that could work alongside humans in disassembly tasks, combining the adaptability of humans with the precision and efficiency of automation [33].

The nature of the disassembly process requires that the autonomous systems involved (robots and their controllers, sensor systems, etc.) be designed to be flexible and robust enough to cope with the operational variability.

1.1.2 Robotic control of uncertain disassembly processes

Robotic control of uncertain disassembly processes is a challenging problem that requires the ability to adapt and learn from the changing environment.

Reinforcement Learning (RL), a sub-field of Machine Learning (ML), is presented as a possible solution for controlling uncertain processes. In RL, an agent learns to make decisions by interacting with its environment [34]. The agent's actions are influenced by rewards or punishments that are used to guide the learning process [35]. The agent's goal is to learn a policy, which is a mapping from states to actions, that maximizes the expected cumulative reward.

In recent years, RL has been successfully applied to a wide range of problems, such as game playing [36][37], recommendation systems [38], and autonomous vehicles [39]. This success is supported by the use of deep artificial neural networks as function approximators (Deep RL). Deep neural networks have led to breakthroughs in areas such as image [40] and text [41][42] generation.

In the field of robot control, RL is becoming an increasingly important technique, notably for robotic manipulation tasks [43][44]. RL has been applied to tasks such as grasping [45], manipulation [46], and object recognition [47], among others. Using RL algorithms, robots can learn to perform these tasks more efficiently and accurately, and adapt to new situations more easily through trial and error in real and dynamic environments.

One of the main advantages of RL is that robots can learn tasks that are difficult to model analytically or physically using traditional programming techniques [48]. In addition, it can be used to learn policies that allow multiple robots to coordinate and perform cooperative tasks that would be difficult or impossible to perform alone [49].

Although RL is an exciting and rapidly evolving area of ML, with many practical applications and ongoing research [50][51], it is still in its infancy in industrial robotics. The field remains in the laboratory and is not yet widely used in industry, where robotic applications may involve uncertain, complex, and safety-critical systems where errors or mistakes can have serious consequences [52][53].

In addition, training can be time consuming and computationally expensive, requiring many iterations to learn an optimal policy. The deployment and integration process can be complex, requiring a high degree of configuration and customization. However, to facilitate the adoption of RL in robotics, several dedicated libraries and frameworks have been created that offer a set of tools and algorithms specifically designed to implement, train, and deploy RL agents in robotic applications.

1.1.3 Frameworks and libraries for reinforcement learning and robotic applications

Robot control frameworks provide a set of tools and methodologies for implementing robot control applications. By providing easy-to-use, well-documented libraries for common robotics-related tasks, these frameworks should reduce the time and effort required to develop control and automation implementations, enabling significant advances and accelerating the development and deployment of robotic systems for various applications [54][55]. This can be especially useful for researchers and developers who are not robotics experts and need to perform complex tasks quickly and accurately.

The importance of control frameworks in robotics lies in their ability to provide a standardized and versatile approach to building robot control systems. By using a common set of tools and libraries, developers can build more reliable and robust systems that perform tasks more efficiently [55]. Examples of this are applications developed on top of Robot Operating System (ROS) [56], a popular robot control framework that offers a comprehensive set of libraries, tools, and communication infrastructure to facilitate the development of complex robotic systems.

In the case of RL, libraries are essential tools for implementing and evaluating

such algorithms. They provide researchers with a high-level framework for designing, training, and evaluating RL models in a standardized and reproducible way [57].

By abstracting away from complex implementation details, the libraries should allow researchers to focus on developing new algorithms and testing their effectiveness in a range of applications. They will also be able to compare their algorithms with existing methods and benchmark their results against state-of-the-art technology. Overall, the implementation of RL libraries in academic and research environments can lead to significant advances in RL algorithms and their application in various fields such as robotics, education, entertainment, and others.

Due to different frameworks and libraries using different Application Programming Interface (API) and syntaxes, it can be difficult for the scientific community, researchers, and engineers to integrate different systems and components [58]. For example, despite the wide adoption of ROS, the lack of a unified framework and standardized development practices in the packages developed makes it difficult to scale and maintain robotics applications [59][60].

This can lead to interoperability issues and make it difficult to grow and evolve the code over time [61]. In addition, the lack of standardization and unification can cause code portability and compatibility issues [62][63], which can limit the adoption of new technologies, slow the development of new applications and use cases, and make it difficult for new researchers to enter the field.

1.1.4 Reinforcement learning for disassembly tasks

At the time of initiating this investigation, search results on specialized online academic sites for the terms *robot AND disassembly*, since 2015, yield the results shown in Figure 1.2.

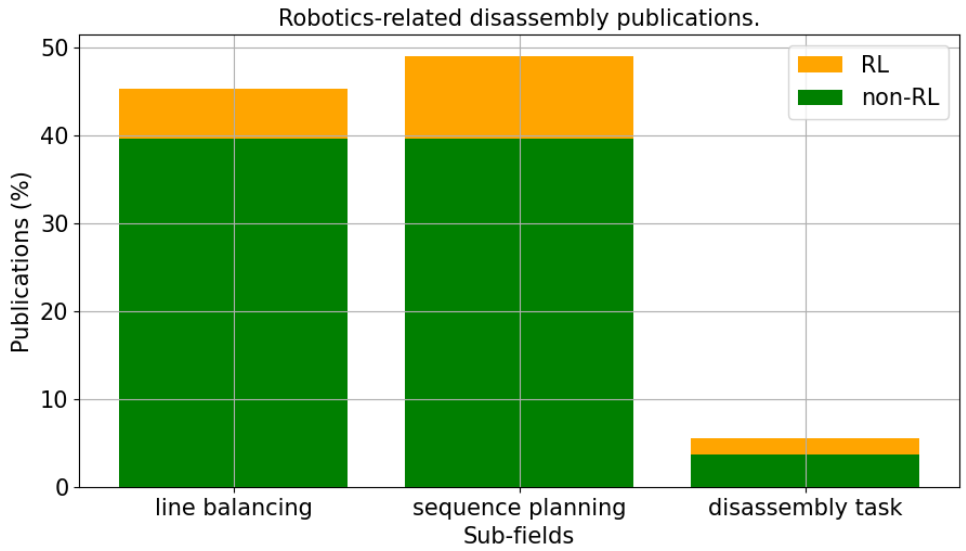

FIGURE 1.2: Robotics-related disassembly publications since 2015 grouped according to their contribution to the disassembly field.

Although RL is presented as an alternative for decision making and control of changing systems with complex and nonlinear dynamics, there is a few works, as

shown in Figure 1.2, on the research and application of RL solutions to execute disassembly tasks.

1.2 Problem

The high degree of variability in the range and condition of EOL products makes disassembly tasks complex, diverse and uncertain (in terms of lack of knowledge about the state of the products prior to performing the task) in nature, requiring generalist systems for automation.

Although RL is presented as a method for decision making and control of changing systems with complex nonlinear dynamics, it is not known to what extent disassembly tasks can be learned automatically and generalizable.

Moreover, since RL systems train an agent by trial and error, the training process is time-consuming and often requires a large number of iterations to achieve optimal performance, as well as extensive computational resources to handle complex environments and high-dimensional state and action spaces.

1.3 Research questions

Research question 1: To what extent can RL algorithms generalize the execution of disassembly tasks using collaborative robots?

Research question 2: How can the RL algorithms' learning process and the capabilities of a collaborative robot for disassembly tasks be leveraged for?

a) reducing training time and resource consumption; and
b) reducing contact forces and jamming states, respectively.

1.4 Document structure

The dissertation structure, outlined in Figure 1.3, is organized as follows: this segment provided an introduction, motivation, and research questions. Chapter 2 provides an overview of the basic RL concepts and techniques. Chapter 3 contains a state-of-the-art review of related work in the field of RL and disassembly, as well as an introductory study on the execution of disassembly tasks using RL. Chapter 4 describes the research carried out to reduce the time and the consumption of computational resources for the training of off-policy algorithms. This research leads to the implementation of a modular and flexible RL library. Chapter 5 presents the research conducted to reduce contact forces and the possibility of jamming states during the execution of the disassembly task, culminating in the development of a unified and scalable control framework for KUKA LBR IIWA collaborative robots. Chapter 6 describes the investigation to execute disassembly tasks with a high degree of variability in terms of geometry and physical properties of the manipulated objects using RL. Chapter 7 extends the previous study to include the human operator's experience to reduce the action space and as a source of disambiguation

during task execution. As an extra (technical) contribution, and for the deployment and integration of robotics applications, Chapter 8 describes the implementation of a collection of ready-to-use packages that unify several components for the creation and execution of collaborative robotic disassembly tasks. Finally, Conclusions and Future Work are drawn.

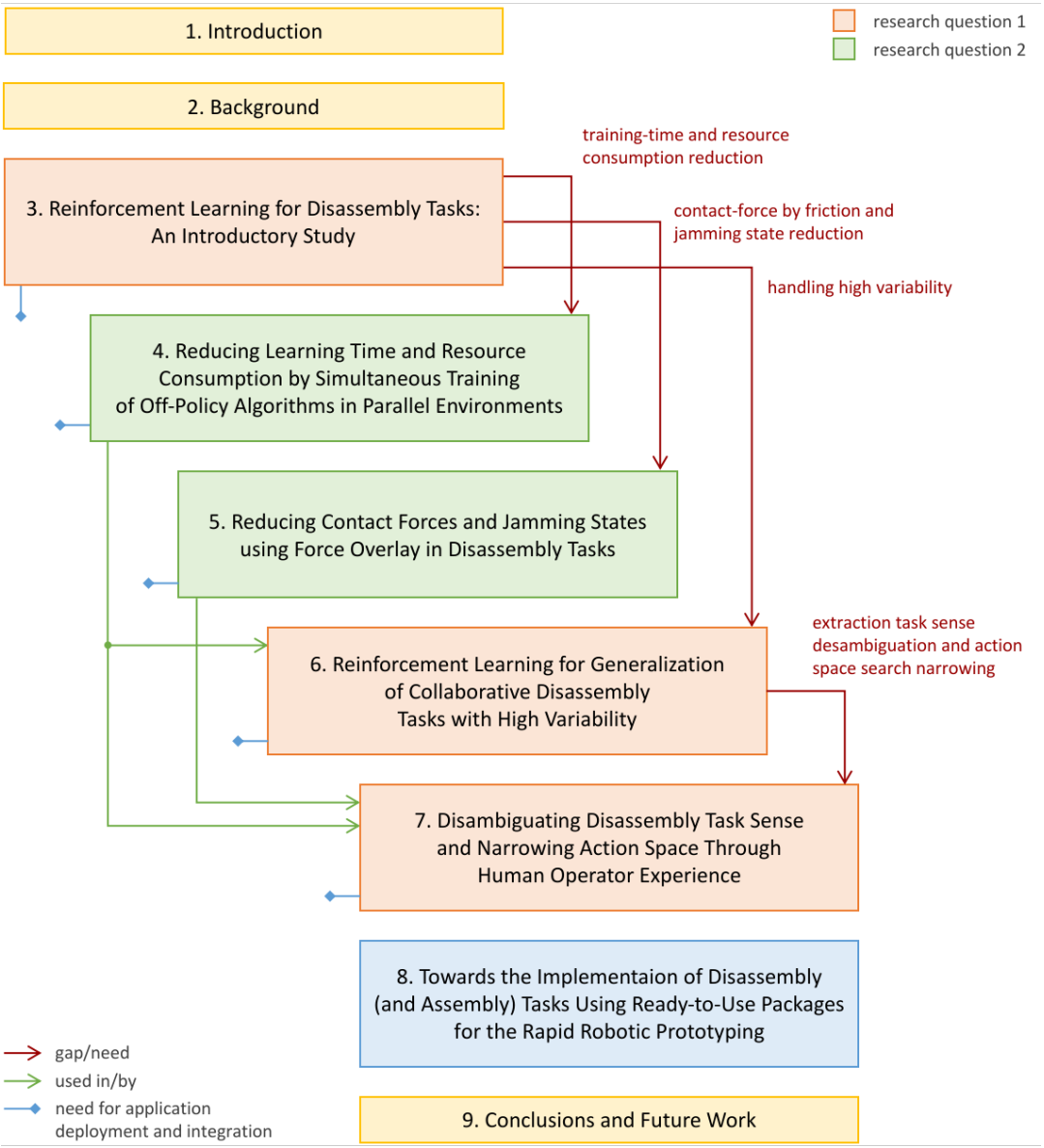

FIGURE 1.3: Outline of the thesis.

1.5 Publications

The Table 1.1 lists, in chronological order, the primary articles submitted/published during the Ph.D. project. It also shows their respective contribution to each chapter of the thesis.

Publications	Chap.
Antonio Serrano-Muñoz, Nestor Arana-Arexolaleiba, Dimitrios Chrysostomou, Simon Bøgh. "Learning and generalising object extraction skill for contact-rich disassembly tasks: an introductory study". In: *The International Journal of Advanced Manufacturing Technology* (2021), pp. 1–13. URL: https://link.springer.com/article/10.1007/s00170-021-08086-z	3
Íñigo Elguea-Aguinaco, Antonio Serrano-Muñoz, Dimitrios Chrysostomou, Ibai Inziarte-Hidalgo, Simon Bøgh, and Nestor Arana-Arexolaleiba. "A review on reinforcement learning for contact-rich robotic manipulation tasks". In: *Robotics and Computer-Integrated Manufacturing* 81 (2023), p. 102517. URL: https://www.sciencedirect.com/science/article/pii/S0736584522001995	
Antonio Serrano-Munoz, Dimitrios Chrysostomou, Simon Bøgh, Nestor Arana-Arexolaleiba. "skrl: Modular and flexible library for reinforcement learning". In: *Journal of Machine Learning Research* 24.254 (2023), pp. 1–9. URL: https://www.jmlr.org/papers/v24/23-0112.html	4
Antonio Serrano-Muñoz, Íñigo Elguea-Aguinaco, Dimitris Chrysostomou, Simon Bøgh, Nestor Arana-Arexolaleiba. "A Scalable and Unified Multi-Control Framework for KUKA LBR iiwa Collaborative Robots". In: *2023 IEEE / SICE International Symposium on System Integration (SII)*. IEEE. 2023, pp. 1–5. URL: https://ieeexplore.ieee.org/abstract/document/10039308	5
Íñigo Elguea-Aguinaco, Antonio Serrano-Muñoz, Dimitrios Chrysostomou, Ibai Inziarte-Hidalgo, Simon Bøgh, and Nestor Arana-Arexolaleiba. "Goal-Conditioned Reinforcement Learning within a Human-Robot Disassembly Environment". In: *Applied Sciences* 12.22 (2022), p. 11610. URL: https://www.mdpi.com/2076-3417/12/22/11610	6, 7

TABLE 1.1: Primary submitted/published articles

In addition, Table 1.2 shows the articles in which the author has collaborated and that contribute in a secondary way to this research.

Publication	Chap.
Idoia Altuna-Galfarsoro, Antonio Serrano-Muñoz, Iker Castro Alfaro, Jon Aurrekoetxea, Nestor Arana-Arexolaleiba. "Generative design of 3D printed grippers for robot/human collaborative environments". In: *2021 IEEE International Workshop of Electronics, Control, Measurement, Signals and their application to Mechatronics (ECMSM)*. IEEE. 2021, pp. 1–5. URL: https://ieeexplore.ieee.org/abstract/document/9468855	8
Ainhoa Apraiz Iriarte, Ganix Lasa Erle, Antonio Serrano-Muñoz, Íñigo Elguea-Aguinaco, Nestor Arana-Arexolaleiba. "Evaluation of the User Experience of an industrial robotic environment in Virtual Reality". In: *International Congress on Project Management and Engineering*. Vol. 7. 3. AEIPRO, 2022. URL: http://dspace.aeipro.com/xmlui/handle/123456789/3288	

Ainhoa Apraiz Iriarte, Ganix Lasa, Maitane Mazmela, Nestor Arana-Arexolaleiba, Antonio Serrano-Muñoz, Íñigo Elguea-Aguinaco, et al. "Evaluating the Effect of Speed and Acceleration on Human Factors During an Assembly Task in Human-Robot Collaboration". In: *Available at SSRN 4371145* ()	

TABLE 1.2: Secondary submitted/published articles.

Chapter 2

Background

This chapter provides an overview of the basic concepts and techniques used in RL. If you are already familiar with RL, you can skip this chapter and proceed to the other chapters in this document.

2.1 Reinforcement learning

RL is a sub-field of ML, along with supervised and unsupervised learning. It describes a specific type of problem in which autonomous agents interact with the environment, make sequential decisions, and receive a reward as a measure of how good a decision was made. RL algorithms attempt to learn a decision-making policy that maximizes the reward received over time.

Figure 2.1 shows basic schema of a RL problem. There are two main components: the agent and the environment.

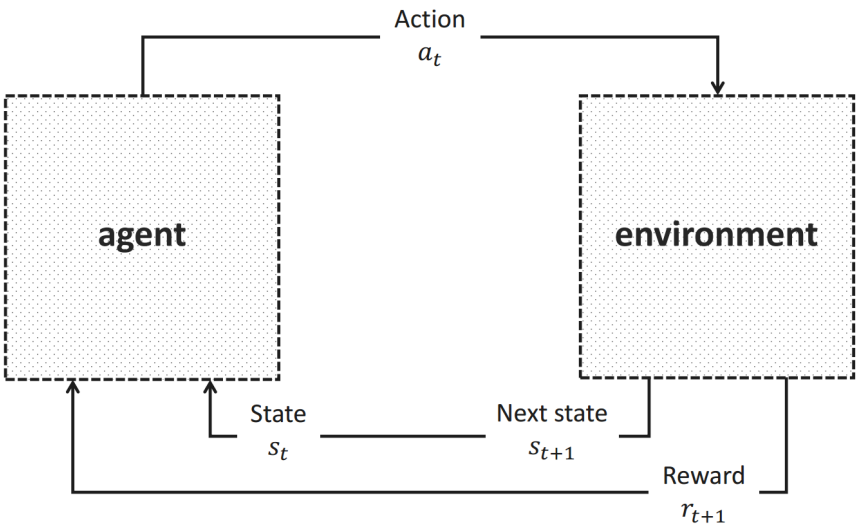

FIGURE 2.1: Reinforcement learning schema (adapted from Wikimedia Commons).

Deep reinforcement learning

Deep RL combines concepts from RL (goal optimization) with artificial neural networks (function approximation) to create much more versatile autonomous

agents. This combination allows learning to solve problems in complex environments where the state has large dimensions and information may be incomplete.

2.1.1 Components and terminology

2.1.1.1 Basic terminology

The basic terminology and components related to RL are defined below.

State and observation

The state $s \in S$ is a complete description of the state of the environment S. There is no information about the environment which is hidden from the state. The state must have the Markov property (knowing the state implies that everything that could determine the response of the environment to a particular action of the agent is known).

On the other hand, an observation o is a partial description of a state, which may omit information.

According to the above definition, an environment in which the agent can observe the complete state is called *fully observed*, while an environment in which the agent can only see a partial observation is called *partially observed*.

Action

The action $a \in A$ is what the agent can do in the environment. The set of all valid actions in a given environment is called the action space A.

Depending on the nature of the action space, the latter can be classified into *discrete action space* or *continuous action space*. In discrete action spaces, the agent can decide on a finite set of actions. In the continuous action space, the actions are real-valued vectors.

Policy

A policy is a rule used by an agent to decide what actions to take based on a given observation or state. The policy can be *deterministic* or *stochastic*.

A deterministic policy, denoted as $a_t = \mu(s_t)$, is a function from the set of states S of the environment to the set of actions A.

A stochastic policy, denoted as $a_t \approx \pi(\cdot|s_t)$, is a family of conditional probability distributions from the set of states S of the environment to the set of actions A. A probability distribution is a function that assigns a probability to each action given a state, where the sum of all probabilities is 1. In that sense, a deterministic policy can be interpreted as a stochastic policy that gives a probability of 1 to one and only one of the available actions for a given state.

Additionally, a policy (deterministic or stochastic) can be parameterized. The output of a parameterized policy is a computable function that depend on a set of

parameters that can be adjusted to change its behavior. Those parameters are denoted as θ or ϕ. For example, a parameterized deterministic policy $a_t = \mu_\theta(s_t)$ and a parameterized stochastic policy $a_t \approx \pi_\theta(\cdot|s_t)$.

Reward

The reward r (or instantaneous reward) is a scalar signal or feedback provided by the environment through a deterministic or stochastic reward function $R()$ as a measure of how good the agent's action is in a given state. It depends on the current state of the environment, the taken action, and the next state of the environment: $r = R(s_t, a_t, s_{t+1})$. However, it can be simplified to just a dependence on the current state $R(s_t)$ or the state-action pair $R(s_t, a_t)$.

The reward function can be *dense* or *sparse*. A dense reward function gives value to most transitions, so the agent gets feedback at almost or all times. On the other hand, a sparse reward function is zero over most of its domain, and only gives values to very few transitions. Such a definition implies that the agent will get no feedback on whether the instantaneous actions it performs are good or bad, which makes learning more challenging.

2.1.1.2 Complementary terminology

A complementary terminology related to RL is defined below.

Trajectory

A trajectory τ is a sequence of states and actions $\tau = (s_0, a_0, s_1, a_1, \ldots)$.

Return

The return $R(\tau)$ is a function of reward sequence over a trajectory.

Most popular return functions are the *finite-horizon undiscounted return* (sum of the rewards obtained in a fixed window: $R(\tau) = \sum_{t=0}^{T} r_t$) and the *infinite-horizon discounted return* (sum of all rewards discounted by how far off in the future they are obtained: $R(\tau) = \sum_{t=0}^{\infty} \gamma^t r_t$ where γ is a discount factor $\gamma \in (0, 1]$).

Transition function

The transition function (also called state transition function) is a probability distribution function $P(s_{t+1}|s_t, a_t)$ that gives the probability that, in state s_t, action a will lead to the next state s_{t+1}.

Value functions

The value function measures the goodness of a state or a state-action pair by predicting the expected return.

- Value function (state-value function): Give the expected return if the agent starts in a state s and always acts according to the policy π

$$V^{\pi}(s) = \underset{\pi \sim \tau}{E}[R(\tau)|s_0 = s]$$

- Action-value function (Q-value function): Give the expected return if the agent starts in a state s, take an arbitrary action a (that may not come from the policy) and always acts according to the policy π

$$Q^{\pi}(s,a) = \underset{\pi \sim \tau}{E}[R(\tau)|s_0 = s, a_0 = a]$$

Optimal value functions

The optimal value function measures the goodness of a state or a state-action pair by predicting the expected return when the policy is optimal. The optimal value functions produce the maximum return.

- Optimal value function: Give the expected return if the agent starts in a state s and always acts according to the optimal policy π

$$V^{*}\pi(s) = \max_{\pi} \underset{\pi \sim \tau}{E}[R(\tau)|s_0 = s]$$

- Optimal action-value function (optimal Q-value function): Give the expected return if the agent starts in a state s, take an arbitrary action a (that may not come from the policy) and always acts according to the optimal policy π

$$Q^{*}(s,a) = \max_{\pi} \underset{\pi \sim \tau}{E}[R(\tau)|s_0 = s, a_0 = a]$$

The optimal policy achieves optimal value functions:

$$\pi_{*} = \arg V^{*}(s) = \arg \max_{a} Q^{*}(s,a)$$

then:

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

Advantage function

The advantage function (A-value) is the difference between the action-value and state-value

$$A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s)$$

2.1.2 Taxonomy of reinforcement learning algorithms

Elaborating a taxonomy of RL algorithms is a challenging task due to the wide modularity of the algorithms. One of the most widely used taxonomies is the one proposed by OpenAI[1]. It classifies algorithms into *model-based* or *model-free* depending on the agent's accessibility to learn or use a model of the environment.

Figure 2.2 shows an adapted version of the indicated taxonomy according to the access to the model of the environment.

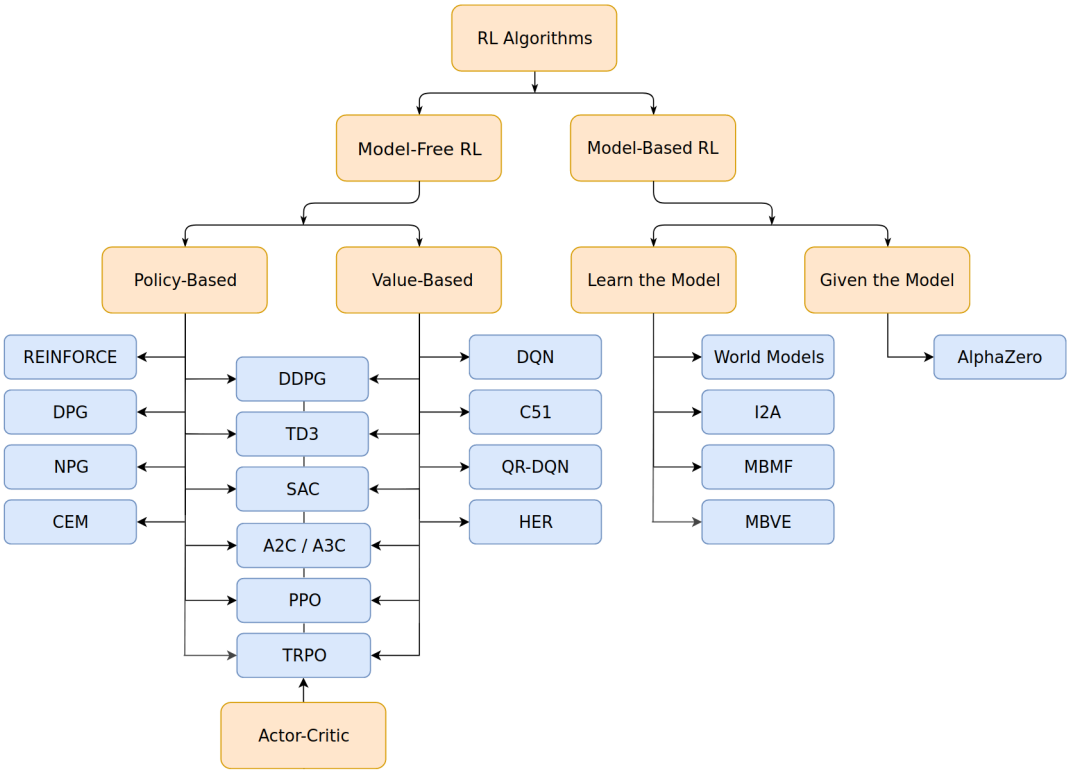

FIGURE 2.2: A non-exhaustive reinforcement learning taxonomy (adapted from the OpenAI Spinning Up taxonomy).

2.1.2.1 Model-based algorithms

Model-based algorithms are algorithms that use the transition function (and the reward function) to estimate an optimal policy by thinking ahead. There are two main approaches: learning the model or learning given the model.

- Learn the model: Supervised learning is used to minimize the error between the model and the trajectory data observed by a random or educated base policy.
- Learn given the model: It is planned, through the given model, to choose the actions

Model-based RL is highly efficient because it requires a reduced number of interactions between the robot and the environment (real-world samples are expensive)

[1]OpenAI Spinning Up: https://spinningup.openai.com

and allows faster convergence to the optimal solution. However, model-based RL has the following disadvantages [72]:

- Additional calculations are required for model training as well as for the planning operation.
- Sincerity and model approximation errors have a great impact on learning tasks.
- They usually have many hyperparameters for estimating uncertainty, balancing planning and real data collection among others.

2.1.2.2 Model-free algorithms

Meanwhile, model-free algorithms can be classified by what they learn and how they learn.

What to learn

- **Value-based:** Value-based algorithms compute the optimal state-value function of each state or state-action pair of the optimal policy by iteratively improving the Q-value and/or the value function values until they converge. Then, the optimal policy is found by acting greedily over the optimal computed function.
- **Policy-based:** Policy-based algorithms are model-free algorithms that improve the policy parameters directly without considering the Q-value or the value function. These algorithms will map from each state to the best corresponding action at that state.
- **Actor-critic:** Actor-critic algorithms combine the best from both policy-based algorithms and value-based algorithms. They use the learned value function as a baseline to update the actor's policy.

How to learn

- **On-policy:** On-policy reinforcement learning algorithms update the policy by interacting with the environment using the exact same policy. This implies that the agent collects data for learning as it explores and takes actions based on its current policy. The main characteristic of on-policy methods is that they directly optimize the policy that generates the data.

  In general, on-policy algorithms are known to be more stable during the learning process because they learn from the policy that generates the data. They also have a greater likelihood of converging to a locally optimal policy because they directly optimize the policy itself.

  However, these algorithms face challenges in balancing exploration and exploitation, as they struggle to effectively use past experience or external data

generated by different policies. In addition, they require a significant amount of interaction with the environment to gather sufficient data for learning, which can be both computationally expensive and time consuming.

- **Off-policy:** Off-policy learning algorithms decouple the learning policy from the data collection policy. These algorithms learn a policy using data generated by the current policy, other policies, or external sources. This characteristic allows them to leverage existing data and experience, potentially improving sampling efficiency and exploration.

  In addition to being able to reuse data collected from different sources, on-policy algorithms can better balance the exploration/exploitation tradeoff because they can explore using one policy while learning and exploiting another.

  However, their susceptibility to instability during the learning process is greater, compared to on-policy methods, due to the mismatch between the policy that generates the data and the policy being learned. In addition, the use of data generated by significantly different policies in the learning phase may lead to bias-related challenges.

2.2 Chapter conclusions

This chapter has presented an overview of RL, providing a foundation for understanding the basic concepts and techniques used in the field. With this overview, the reader is prepared to enter the following chapters, where the terms and concepts described earlier are common.

Chapter 3

Reinforcement Learning for Disassembly Tasks: An Introductory Study

The content of this chapter addresses, in part, research question 1: To what extent can RL algorithms generalize the execution of disassembly tasks using collaborative robots?

The work presented in this chapter is partially published in the papers: Íñigo Elguea-Aguinaco, Antonio Serrano-Muñoz, Dimitrios Chrysostomou, Ibai Inziarte-Hidalgo, Simon Bøgh, and Nestor Arana-Arexolaleiba. "A review on reinforcement learning for contact-rich robotic manipulation tasks". In: Robotics and Computer-Integrated Manufacturing 81 (2023), p. 102517. URL: *https://www.sciencedirect.com/science/article/pii/S0736584522001995*; *and Antonio Serrano-Muñoz, Nestor Arana-Arexolaleiba, Dimitrios Chrysostomou, Simon Bøgh. "Learning and generalising object extraction skill for contact-rich disassembly tasks: an introductory study". In: The International Journal of Advanced Manufacturing Technology (2021), pp. 1–13.* URL: *https://link.springer.com/article/10.1007/s00170-021-08086-z*.

This chapter reviews the state-of-the-art of related work in the domain of RL and disassembly (and assembly). In addition, an introductory study is conducted to explore the use of RL in the execution of disassembly tasks.

3.1 A review on the execution of disassembly tasks

Disassembly tasks can vary depending on the specific object or system being disassembled. Some common types of disassembly tasks are listed below:

- Unscrewing: Removing screws or bolts using a screwdriver, wrench, or other tool designed for this purpose.
- Press/fit removal: Using tools or methods to press or push out tightly fitted components, such as press-fit connectors or bearings.
- Contact-rich extraction: Removing components or parts that have numerous contact points or connections.
- Adhesive dissolution: Using solvents or solutions to dissolve adhesives or glues that hold components together.

- Cutting: Employing cutting tools, such as a knife, scissors, or wire cutters, to sever or trim materials like plastic, fabric, or wires.
- Unsoldering: Removing solder from electrical or electronic components.

The following section provides an overview of the state-of-the-art of research and application in disassembly tasks up to the time of the study.

3.1.1 Disassembly tasks

Relatively few works have explored the execution of disassembly tasks. A snapshot of these works is shown in Figure 3.1.

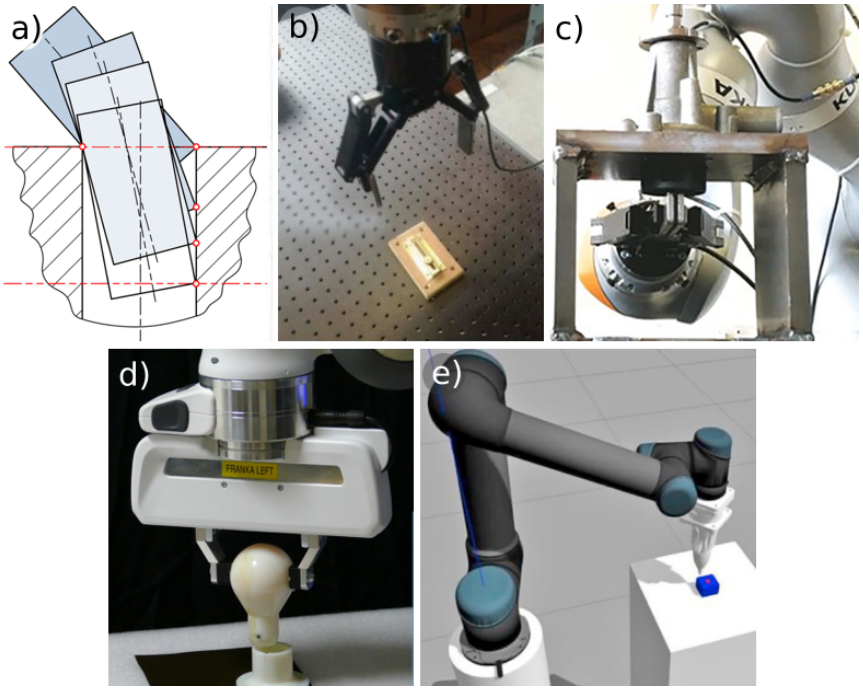

FIGURE 3.1: Snapshot of the publications of a) Zhang et al. [73], b) Herold et al. [74], c) Huang et al. [75], d) Simonič et al. [76], and e) Kristensen et al. [76].

Zhang et al. [73] explore generic pin-hole disassembly processes with a compliant robotic manipulator to identify the effects of parameters such as the manipulator stiffness configuration, compliance center location, and pin initial position errors. As a result of this exploration, they propose a theoretical model, based on a quasi-static analysis, for the prediction of the range and position of the two-point contact region for the task. However, because the model is designed with specific assumptions and simplifications, it is not flexible enough to adapt to highly variable environments where objects, their physical properties, and geometries (and thus the number of contact points) can differ greatly.

Herold et al. [74] conduct a study that experimentally evaluates four strategies for separating a bolt along a door chain. The first three strategies adjust the position and rotation around the motion axes in response to the force measured at the end effector during task execution. In the fourth, they evaluate the strategy of applying a periodic oscillating motion at the manipulator's last joint (a feature of the KUKA

LBR iiwa cobots) for separation. The results show that the latter strategy reduces the task execution time and resistance forces, making the disassembly process more effective. However, they do not report data on the role of oscillatory motion under different parameters in improving the performance of the disassembly task. In addition, the execution of the task is manually programmed to go from one point to another of the slotted base and always with the same setup and under the same initial conditions.

Huang et al. [75] presented a case study of disassembling press-fitted components based on active compliance in the context of collaborative Human-Robot Interaction (HRI). In their work, a compliant robotic manipulator is used to hold the part to be extracted. While the operator applies a force to the component using a press, the robot grips the component and follows its movement during the extraction process. Robot control is performed via a sequential state machine with fixed operations. Although the results demonstrate the feasibility of the proposed cell for disassembling a water pump, the system can only handle tasks that require the same basic operations, and it is the operator who carries out the disassembly work.

Although it is not specifically research focused on disassembly, Simonič et al. [77] used the information obtained during disassembly tasks to perform assembly tasks. They implemented a hierarchical RL algorithm and a graph representation under the criterion that an assembly task is nothing more than a reverse execution of the corresponding multi-stage disassembly task. Disassembly is complete when the motion is unconstrained in the desired degrees of freedom, and the constructed graph is used to perform the corresponding assembly task. Such an implementation is used to remove a car bulb from its socket. However, the implementation of the network is based on the discretization of the disassembly action space into multiple predefined moves. Moreover, for the correct execution of the tasks, the search is explicitly limited to the Z-axis, both in rotation and translation.

Kristensen et al. [76] developed a framework for training and testing RL algorithms to specifically support unscrewing operations, involving one rotational Degree of Freedom (DOF), in the robotic disassembly of electronic waste in simulation environments. The framework was implemented on top of Gazebo and uses ROS as middleware. Although the implementation provides a framework for training and evaluating RL algorithms, it is not possible to evaluate different unscrewing tasks or to use observation information beyond that provided. Furthermore, to demonstrate the functionality of the framework, only results with Q-learning, an algorithm restricted to discrete observation and action spaces, are presented.

3.1.1.1 Analysis

An overview of the disassembly tasks presented in the papers of the literature review is shown in Table 3.1. The table summarizes the use or not of RL and the different disassembly tasks.

Research that does not use RL for the execution of disassembly tasks emphasizes the use of compliant robotic manipulators. Their use is justified by the variability of product geometries and conditions, as well as the presence of unstructured and uncertain environments typical of disassembly processes. The compliant robotic manipulators can provide an additional level of safety by minimizing the risk of damage to the product, the environment, the robot itself, and the operators involved in the process.

Ref.	RL	Disassembly task
Zhang et al. [73]	No	Contact-rich extraction
Herold et al. [74]	No	Contact-rich extraction
Huang et al. [75]	No	Press/fit
Simonič et al. [77]	Yes	Unscrewing
Kristensen et al. [76]	Yes	Unscrewing

TABLE 3.1: A summary of the articles reviewed for solving disassembly tasks up to the time of the study.

The research on RL for disassembly tasks has been conducted for one DOF: the rotational one. The unscrewing task around the Z-axis by Kristensen et al. [76] and the removal of a light bulb from its socket by Simonič et al. [77]. Work with more DOFs has not been reported.

Observation spaces incorporate information in the absolute reference systems. Furthermore, the tasks start from the same initial spatial position. Kristensen et al. [76] uses the absolute spatial Cartesian pose of the manipulator's end-effector. These particularities make it difficult to generalize the learning to new spatial positions or to different configurations of the robot manipulator. In disassembly, the ability to generalize is critical because the state of EOL products can be extremely diverse as opposed to assembly.

There are very few works on RL for disassembly tasks. A review by Poschmann, Brueggemann, and Goldmann [28] summarizes current research on disassembly and robotics. The authors show that the use of robots can bring enormous benefits in the field of disassembly, where they intend to advance the development of hybrid and fully autonomous robotic cells, as well as the improvement of HRI conditions and processes. They also confirm the statement that disassembly is not necessarily reverse assembly: assembly technologies or specific geometric characteristics prevent reverse assembly, especially if this objective was not considered in the design phase.In addition, the need for flexible systems and autonomous agents is emphasized. Despite this, the review only identifies one article that uses RL to perform disassembly tasks.

Due to the lack of publications related to RL for disassembly tasks, the literature review is extended from the perspective of contact-rich manipulation tasks and assembly. It aims to identify key factors and implementations of RL that can be applied to disassembly.

3.1.2 (Contact-rich) manipulations tasks

RL methods have been researched and applied to a greater extent in contact-rich robotic manipulation tasks than in disassembly tasks. This section summarizes about 140 research articles published since 2015 in this field.

It is relevant to note that pick-and-place and grasping tasks have their own field and research community, with promising results and lines of work [78][79][80]. In fact, these tasks are not included by the scientific community in the context of contact-rich manipulation, and many of the studies assume that the object to be assembled/manipulated is already picked up and grasped. In the same vein, tasks related to picking and grasping will not be addressed in this document.

3.1.2.1 Implemented tasks

In contact-rich manipulation for assembly, there are basically two types of tasks depending on the physical structure of the manipulated objects: manipulation of rigid solid objects and manipulation of deformable objects.

Within rigid object manipulation, the insertion task stands out in comparison to other tasks such as surface contact or door opening. In the insertion task, the most common use case is peg-in-hole (in which a peg is inserted into a fixed hole with a small clearance) [81][82] followed by ring-in-hole [83]. For both use cases, there are a series of works that particularize the geometry of the manipulated objects, such as inserting a USB connector [84], fitting gears [85], or staking Lego pieces [86].

When manipulating deformable objects, it is difficult to represent their state because they typically lack a clear and consistent shape throughout the whole manipulation. In an effort to characterize the deformable object configuration spaces, Sanchez et al. [87] proposed a geometry-based classification, identifying fundamentally three types of deformable objects: linear or uniparametric, planar or biparametric, and volumetric/solid or triparametric. According to the proposed classification, most of the publications related to deformable objects refer to manipulations of one-dimensional objects (ropes) [88] and two-dimensional objects (clothing, fabrics and tissues) [89]. The folding task [90] predominates over other tasks such as tensioning [91], cutting [92] and wrapping [93].

3.1.2.2 Reinforcement learning components

Exploring the RL components (bbservation space, action space, reward function, etc.) of the different publications is relevant to take advantage of existing knowledge and experience. This allows to build on established concepts, to understand the advantages and disadvantages of such components, to adapt or extend existing approaches, to keep up with developments, and to inspire creativity and innovation.

The RL components of the state-of-the-art in contact-rich manipulation tasks are examined below.

Observation spaces

The Figure 3.2 shows the usage, among all the analyzed articles, of different sensors or data to build the observation space when performing contact-rich tasks for both rigid and deformable objects. This usage provides an overview of the information or sensors that are significant for the execution of these tasks.

Regarding the data used to construct the observation space, the position and orientation of the end effector in Cartesian space, as well as the force and torque measured at this link, are the most commonly used data types. This is followed, to a lesser extent, by the position of the robot's joints and images captured by digital cameras.

The force/torque measurement is carried out by external sensors attached to the robot wrist [94] or by sensors integrated with the robot itself [95]. In the tasks in which it is used, such information directly indicates the magnitude of contact during assembly, as long as there is effective contact between the parts involved. It is not the case for the manipulation of deformable objects since the configurations

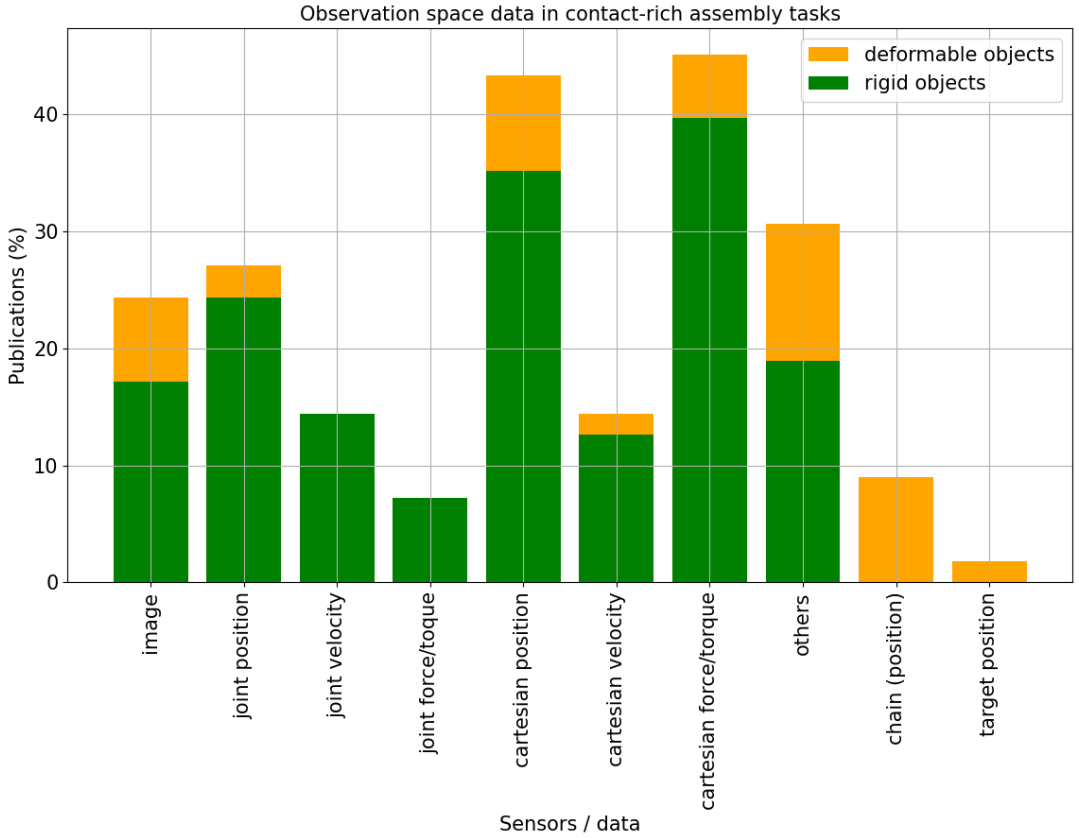

FIGURE 3.2: Observation space data in contact-rich manipulation tasks. For deformable objects, the label *chain* groups the positions or location of point or grid markers on the manipulated object. Within the label *other* are grouped data such as tactile sensors, gripper aperture, inclination and object depth, among others.

used (linear and planar deformable objects) do not offer movement restrictions that generate significant forces [96].

Another type of relevant information in the manipulation tasks is the Cartesian position of the end-effector, the Tool Center Point (TCP), or the object to be manipulated, as well as the position of the robot's joints. Since the task space is typically a three-dimensional environment, there is a greater tendency for researchers to use the Cartesian position as it results in simpler and more intuitive observation spaces [97].

In the case of manipulation of deformable objects, the use of cameras is quite relevant [96]. It is justified for tracking or estimating the positions or localization of indicators, fiducial trajectories, features or nodes of chains (in linear deformable objects) [98], such as ropes, or meshes (in planar deformable objects) [99], such as clothes, fabrics and textiles.

In addition, the use of cameras is reflected in the inclusion of images, both in RBG color space and grayscale, as well as depth sensors both to construct the observation space and to specify targets to the agent. Relatively low resolutions (between 48x48 pixels and 224x224 pixels) are used to reduce the dimension of the observation space [100].

Action spaces

The Figure 3.2 shows the usage, among all the analyzed articles, of different actuators or data to build the action space to control the robotic manipulator when performing contact-rich tasks for both rigid and deformable objects.

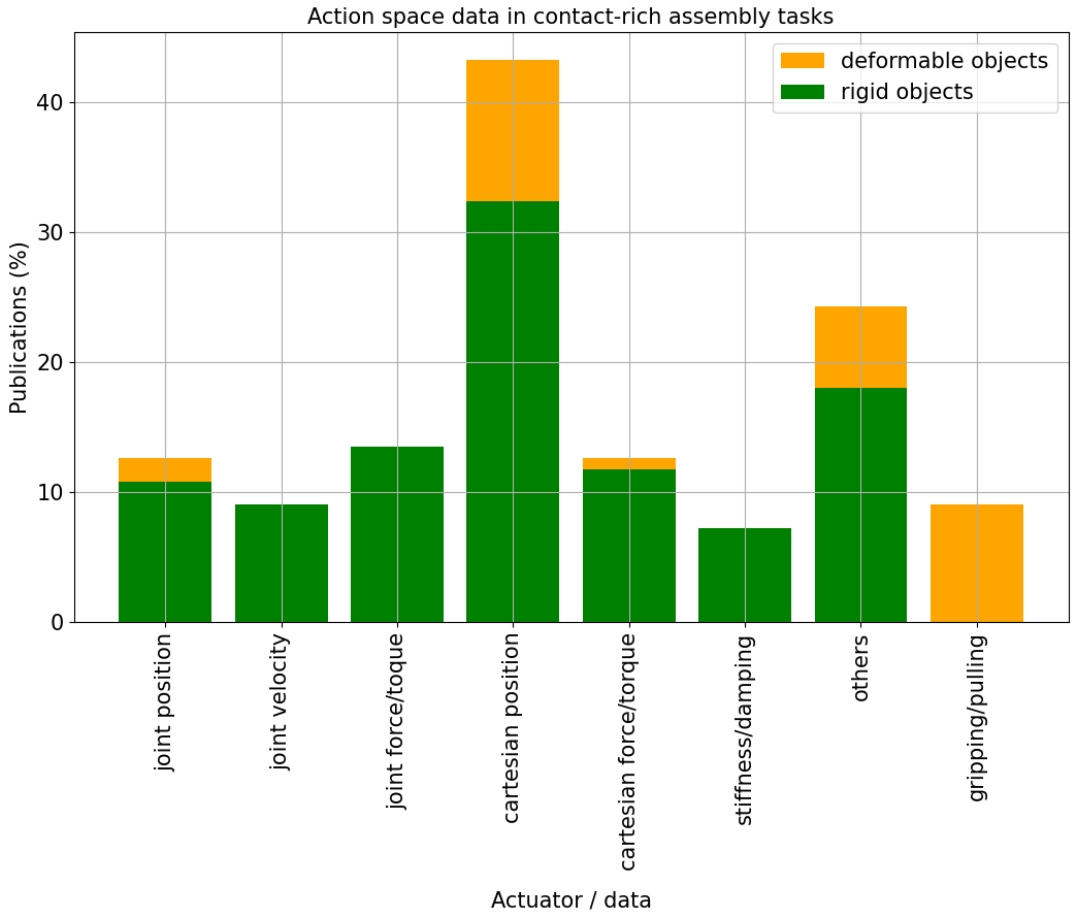

FIGURE 3.3: Action space data in contact-rich manipulation tasks. Within the label *other* grouped actions such as the adjustment of controller parameters, among others.

For rigid object manipulation, the information of the action space covers position, velocity, and force/torque actions, both for the joint space and the Cartesian space, with the Cartesian position being the most used.

When manipulating deformable objects, the action spaces are most often applied through both Cartesian and joint position actions. As discussed above, this type of object offers practically no resistance to manipulation, so this type of action is sufficient to provide adequate control.

Regarding the type of control used, the Figure 3.4 illustrates the use of different variants by the reviewed works to manage the action space in tasks involving complex and contact-rich interactions.

As in the manipulation of deformable objects, position and velocity controls are also used to manipulate rigid objects [101][102], although there is a trend toward the use of impedance control and other force-regulation control methods. Impedance control methods allow the robot to adapt to changes in the interactions with objects

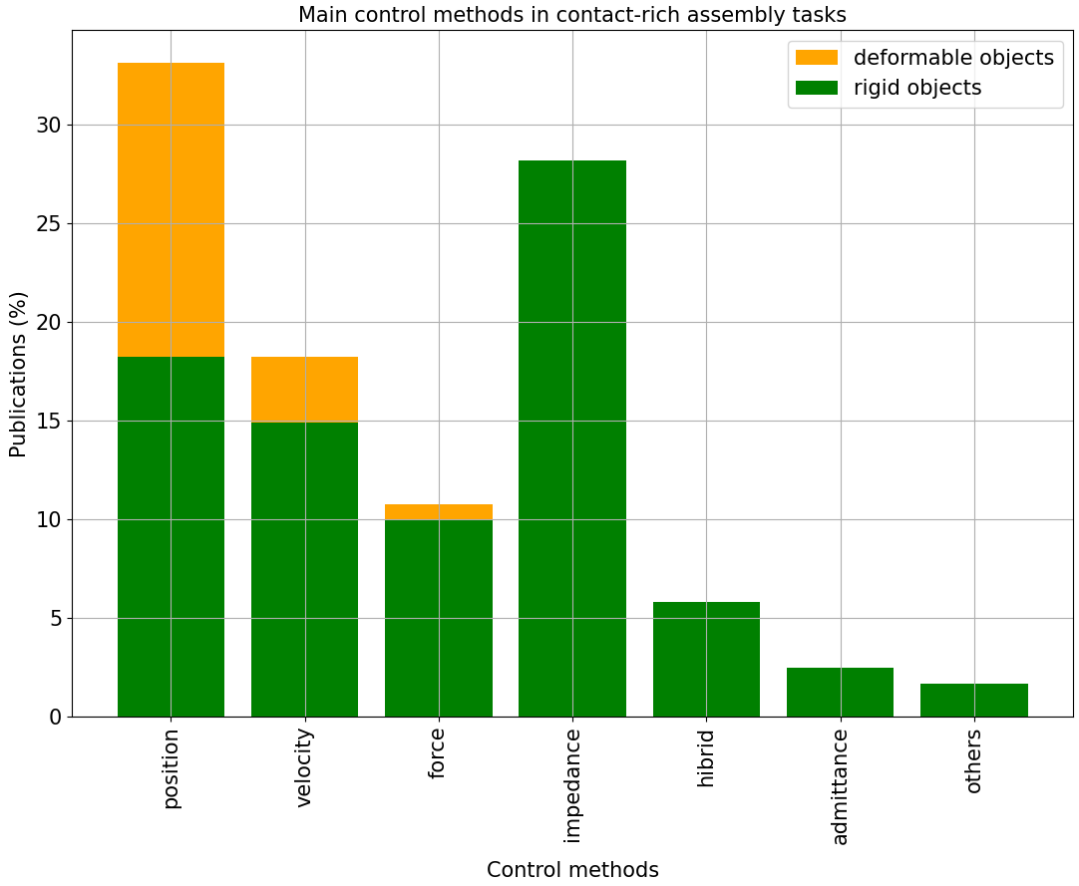

FIGURE 3.4: Control methods used to apply the action space in contact-rich manipulation tasks.

and to the external forces from the environment by regulating the robot's stiffness and damping [103][104][105].

They also help to reduce the risk of damage to the robot or the environment by limiting the forces and torques applied to the manipulator [106]. However, such control methods require the presence of more sophisticated sensors and control algorithms not available in all industrial manipulator robots [101].

Rewards

The use of the dense reward function predominates over the sparse reward function. The dense reward function relies in many cases on Euclidean distance to guide the task due to its spatial nature, but the sparse reward function is widely used even when RL problems may converge slowly or not at all [107].

Furthermore, to avoid the complex engineering of reward functions, some works tend to learn this function from multimodal observations [108][109] or expert demonstrations [110][100].

Reinforcement learning algorithms

Model-based RL algorithms are used by some authors arguing their high sampling efficiency compared with model-free algorithms [111][112][113][114].

However, the choice of model-free algorithms prevails over model-based algorithms due to the difficulty of generating accurate models. The reason is that the physics and dynamics of the environment are highly variable, and the forces resulting from collisions, friction, and jams during manipulation are nonlinear and complex to model [115][116]. Additionally, model-free RL algorithms are preferred because of their ease of implementation and reprogrammability [100][117].

Within the model-free, actor-critic algorithms (which combine the advantages of value-based and policy-based methods) [118] enjoy great popularity among authors. DDPG, SAC, and PPO, all actor-critic algorithms, are among the most popular algorithms used in the reviewed articles.

Environments

The most common practice among the authors is to conduct the training in simulation and then validate or deploy the trained policy in real-world scenarios [119].

Transferring learned policies from simulation to the real world (also known as *Sim2Real*) can be challenging due to differences between both scenarios. Because simulations are simplified models of the real world, they may not capture all of the complexities, dynamics, and uncertainties that exist in reality.

Domain randomization [120] is a widely used technique in the reviewed literature to address the *Sim2Real* gap. This technique involves randomizing different simulation parameters, including both, visual data (such as textures and lighting [121][110]), and physical variables (such as object mass, friction, and geometry [83][94][113]).

3.1.2.3 Analysis

RL has been used to a large extent for learning and executing contact-rich manipulation tasks for assembly. However, those research and applications do not go beyond laboratory prototyping based primarily on insertion, such as peg-in-hole, ring insertion, Lego stacking, among others.

Based on the content of the articles and their publication dates, there is a trend towards the development of applications that use impedance control as an alternative to position control in both, Cartesian or joint space. Impedance control is intended to ensure the safety of the robotic manipulator itself, its environment and the operators, at least in tasks involving rigid objects.

In contrast to the manipulation of rigid objects, the manipulation of deformable objects has received less attention in the robotics community, with emphasis only on linear or planar deformable objects. One of the challenges in this area is the computational cost, the difficulty and complexity of modeling the physics of deformation, and the handling of contact and friction between such objects and other surfaces in simulation environments. Therefore, they will not be covered in this document.

Overall, RL has shown promising results for contact-rich manipulation, but it still faces several challenges and limitations. Among them are:

- The ability to generalize and perform tasks with more complex dynamics remains limited. In particular, as the problem scales, it becomes difficult to identify appropriate states, actions, and reward functions, leading to slower convergence and weaker performance.
- The design of reward functions can be complex due to lack of specifications, the presence of multiple goals, or sensitivity to distractors.
- The limited fidelity and scope of simulation platforms limits learning and makes transfer to reality difficult.
- Real-world interaction is challenging due to unstructured realities and constraints.

3.1.3 Summary of state-of-the-art review

In the last years, RL has gained importance in decision-making and control tasks for adaptive and flexible autonomous systems, mainly due to the use of artificial neural networks as function approximators.

The majority of research in robotic manipulation and RL focuses on manufacturing and contact-rich assembly tasks. Within this field, the manipulation of rigid objects is extensively investigated while the manipulation of deformable objects (in particular three-dimensional deformable objects) remains unexplored.

There is a trend in contact-rich assembly tasks, already presented in disassembly, towards the development of applications using compliance control, as an alternative to traditional Cartesian or joint space control, to ensure the safety of the robotic manipulator itself, its environment and human operators. However, research and applications for contact-rich tasks are addressed in relatively simple tasks and remain laboratory prototypes.

As for disassembly, there are only a few works that use RL to perform the tasks, particularly those requiring contact-rich manipulation skills. In disassembly, which is not necessarily the reverse of assembly, the ability to generalize is critical since the state of EOL products can be extremely diverse. This process requires agents that can handle different physical and geometric properties and uncertainties associated with the state of the product. However, the extent to which RL algorithms can perform such tasks is unknown.

Both the lack of work in disassembly tasks and the need for generalizable systems open the door to a field of research and application of RL algorithms for disassembly processes.

3.2 Reinforcement Learning for Disassembly Tasks

To disassemble EOL products, autonomous systems must be generalist enough to overcome challenges such as the wide range of product conditions and uncertainties related to their properties, as well as the complexity of their operation.

This section describes the introductory study performed to determine the extent to which RL algorithms can generalize disassembly tasks. It also aims to evaluate some of the most popular libraries and solutions for RL application development and collaborative robot control.

To evaluate the generalizability of the RL algorithms and some of their components identified in the previous state-of-the-art review, the extraction of a rigid object that is inside and in contact with a rigid slotted part is proposed as a use case.

3.2.1 Reinforcement learning experimental setup

The experimental setup consists of a robotic manipulator mounted on a table and an object composed of two rigid bodies in contact with each other. The objective is, with RL algorithms, to learn how to control a robotic manipulator to remove the detached object from the fixed slotted base while avoiding actions that generate undesired forces as a result of the mechanical interaction between the two parts.

3.2.1.1 Real-world setup

The real-world setup consists of the following items:

- The robotic manipulator is a highly sensitive, flexible, and lightweight KUKA LBR Iiwa 14 R820 collaborative robot with a payload capacity of 14 kg.
- The target to be disassembled contains two rigid solid objects, synthesized using additive manufacturing, as shown in Figure 3.5.

  The object that serves as the base, attached to the table by screws, has dimensions 0.1 x 0.1 x 0.03 meters. This object has a slot of dimensions 0.1 x 0.02 x 0.01 meters positioned 0.01 meters from its top surface and centered with respect to a pair of opposite sides.

  The object to be extracted has a dimension of 0.1 x 0.02 x 0.01 meters. As part of this work and for future work, different objects to be extracted were synthesized with a gap with the base ranging from 1.0 millimeters to 0.1 millimeters. Synthesized with different plastic materials and in different printing directions, they also have varying physical properties such as friction and rigidity.

  The combined object is placed 0.65 meters in front of the robot, at the same level as the robot's base.

Real-world robot control was performed using the *iiwa_stack* library [122], a native ROS Java node for controlling the LBR iiwa robots. The selection of this library was based on the analysis and comparison of different implementations. The *iiwa_stack* was the most versatile and popular implementation for controlling the robotic manipulator. Despite this, its scope is limited to ROS and any changes or adaptations depend on a dense code programmed in JAVA (the native programming language of the LBR iiwa). Other popular libraries, such as *KUKA-IIWA-API* or *iiwa_ros*, are more limited in scope or depend on specific hardware that is not available.

3.2.1.2 Simulated setup

Using simulations to train RL algorithms is less expensive, more efficient, and safer than their real-world counterparts.

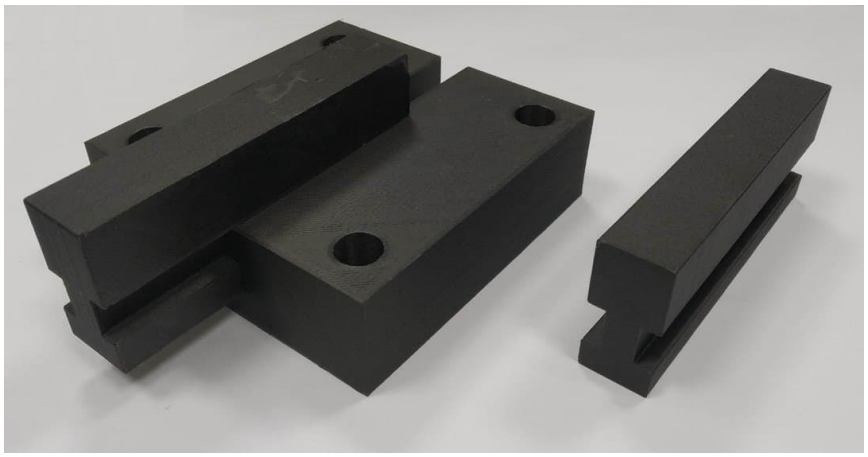

FIGURE 3.5: Target composed object to be disassembled synthesized using additive manufacturing.

At the time of this particular research, a replica of the real task was implemented on the NVIDIA Isaac Sim simulator version 2020.2.2: a photorealistic, real-time, physically accurate robotics platform that had recently been released in the evaluation form. This implementation simulated only one environment (a single robot and a single object to be disassembled). As in the real world, during the evaluation the robot was controlled using ROS Melodic through the modules integrated with the simulator.

3.2.2 Reinforcement learning formulation

A Markov Decision Process (MDP) with a finite-horizon discounted return was used to frame the problem. The agent will learn how to move the robotic manipulator's end-effector on the Cartesian plane parallel to the table to perform the contact-rich extraction of two rigid objects (object extraction skill).

During each timestep of interaction with the environment, the agent is presented with an observation o of the state $s \in S$ of the environment, which is not fully observable. The agent then selects an action $a \in A$ from the action space using a parameterized policy π_θ. The environment, which changes according to the agent's action, provides a reward signal $r_t = R(s_t, a_t, s_{t+1})$ to the agent, indicating how good or bad the new state is. The agent's objective is to optimize the cumulative reward, discounted by a factor $\gamma \in (0, 1]$, by adjusting the policy's behavior through some optimization algorithm.

The following subsections describe the RL formulation:

3.2.2.1 Reinforcement learning elements

Observation space

To evaluate the agents' generalization capabilities, an observation based on the object or manipulator pose was designed.

The observation space is made up of several components, including the normalized length of the episode, which is represented as the ratio between the

current timestep and the maximum allowed timestep (max_steps); the relative position of the robot's end-effector in the XY-plane concerning the initial grasping pose (pos_{XY}), and the rotation of the object as shown in Equation 3.1. The rotation is measured between 0 and 180 degrees and is scaled to the $[-1, 1]$ interval it from being dominant over the others.

$$o_t = [\ \frac{step}{max\_steps},\ pos_{XY},\ rot_{object}\ ] \tag{3.1}$$

Action space

The action space is a 2-dimensional vector. Each component maps to the respective translation (in centimeters) of the robot's end-effector in the $[-1, 1]$ continuous interval on the XY-plane as shown in Equation 3.2.

The decision for an action space in relative coordinates responds to the generalization capabilities in robotic manipulation since they allow the robot to perform the same task in a variety of initial positions without requiring explicit knowledge of the absolute positions of the targets.

$$a_t = [\ \Delta x,\ \Delta y\ ] \tag{3.2}$$

Reward

A sparse was used to determine how agents learn and generalize the extraction skill. It is important to note that the sparse reward can make the problem more difficult to solve, instead of guiding the agents to the goal.

At the end of each episode, a sparse-reward function provides the reward as described by Equation 3.3, . In the case of successful extractions, the agent receives a positive reward equivalent to the maximum timestep (max_steps) provided by the environment. However, if the extraction fails due to the detection of an undesired critical force or if the maximum timestep is reached, the agent receives a reward from a continuous function described by Equation 3.4. This function penalizes extractions with short displacement heavily in time as shown in Figure 3.6. During task execution, the instantaneous reward remains zero.

$$r_t = \begin{cases} 0 & \text{if } t < max\_steps \\ r_{fail} & \text{if } t >= max\_steps \text{ or extraction fails} \\ max\_steps & \text{if successful extraction} \end{cases} \tag{3.3}$$

where

$$r_{fail} = -\frac{max\_steps}{2} + \exp\left(ln(1.5\, max\_steps + 1)\ , ||\frac{pos_{XY}}{0.1}||\right) - 1 \tag{3.4}$$

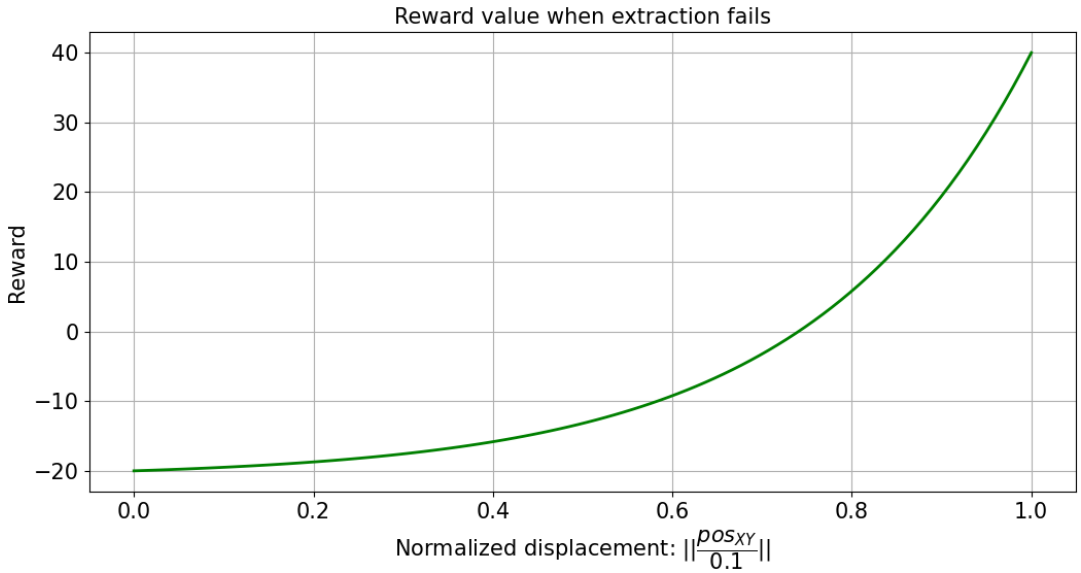

FIGURE 3.6: Graphical representation of the reward function when extraction fails.

Episode termination

The episodes conclude under two conditions: firstly, when the agent successfully completes the disassembly task of extracting the fixed object from the slotted one; secondly, when a critic force is identified, or the maximum timestep (*max_steps*) is reached, regardless of the successful execution of the task.

3.2.2.2 Reinforcement learning algorithms

Agents

Model-based RL algorithms depend on the existence of an accurate model of the environment that is capable of predicting the outcomes of different actions in different situations. When there is a wide variety of objects with different physical, mechanical, and geometric properties, as in the case of EOL products, it is difficult to build an accurate model that can generalize across all this variability. In these cases, the model may need to be updated frequently to account for new or changing conditions, and even then, the model may not be able to capture all the nuances of the environment, which can lead to inaccuracies in its predictions and suboptimal decision-making.

In contrast, model-free RL algorithms do not require an accurate model of the environment but learn directly from experience through trial and error. This makes them more suitable for environments with a wide variety of conditions and properties, as they can adapt and learn from experience without relying on a precise model.

Only model-free actor-critic algorithms were considered for this study. Actor-critic algorithms combine the best of policy optimization algorithms and value-based algorithms [123]. Within the model-free actor-critic algorithms, off-policy

algorithms were chosen (in particular DDPG and TD3). Compared to on-policy algorithms, off-policy algorithms can learn from previous experience or from any data collected by other policies, not only the current one, which leads to more sample-efficient learning, at least in one environment.

- **Deep Deterministic Policy Gradient (DDPG)**: a model-free deterministic off-policy actor-critic algorithm. It uses deep function approximators to learn the policy (and to estimate the action-value function) in high-dimensional, continuous action spaces [124].
- **Twin Delayed Deep Deterministic policy gradient (TD3)**: an actor-critic algorithm based on DDPG. This algorithm relies on double Q-learning, target policy smoothing, and delayed policy updates to address the problems introduced by overestimation bias, leading to estimates and suboptimal policies in actor-critic algorithms [125].

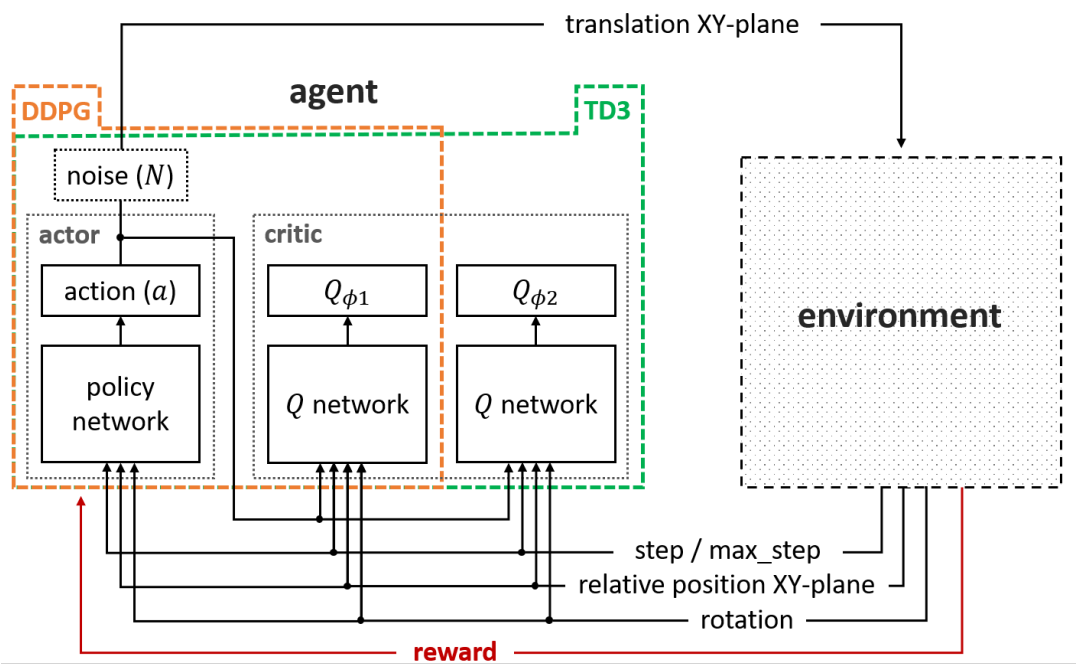

FIGURE 3.7: Reinforcement learning schema.

Model architectures

The architecture of the Policy and Q networks used by both DDPG and TD3 agents is the same. The networks receive the observation space as input for the Policy, and the concatenated components of the observation space and action space for Q networks as input.

They are followed by two hidden dense layers of 32 neurons each with Rectified Linear Unit (ReLU) activation functions. The use of two dense hidden layers with 32 neurons each helps to learn a more complex representation of the observation space. The ReLU activation function is commonly used in deep neural networks because of its ability to accelerate the convergence of the learning process by reducing the vanishing gradient problem.

The Policy network has two output neurons that use the hyperbolic tangent function to fit the action space to the expected interval $[-1, 1]$, while the Q networks use a linear activation function for their output neuron to estimate an unbounded Q-value.

3.2.3 Experiments and results

The extraction process for DDPG and TD3 in both simulation and the real world is depicted in Figure 3.8.

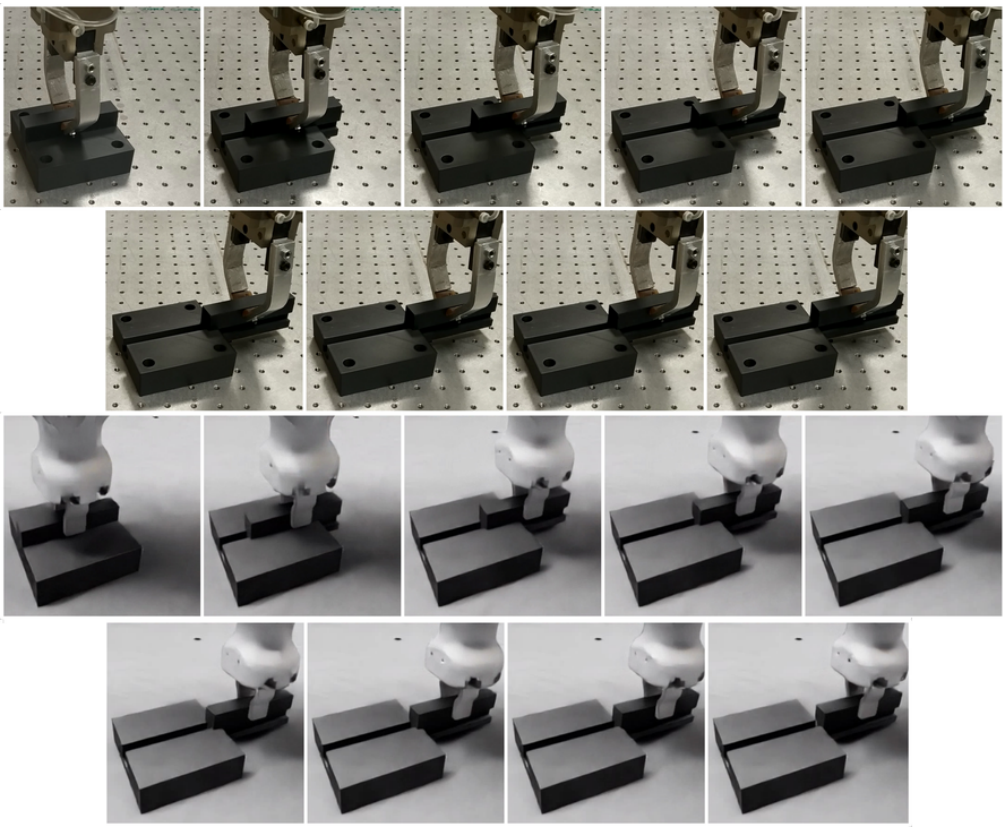

FIGURE 3.8: The optimal policies learned were used to perform the extraction task in both the real world and the simulation. The first and second rows of the figure show the sequence for DDPG. The third and fourth rows show the sequence for TD3. The frames for DDPG were taken every 3 timesteps, while the frames for TD3 were taken at each timestep.

3.2.3.1 Experiment implementation

The RL environment was defined using the OpenAI Gym interface in Python. for both, learning and evaluation the *RLlib* v1.0.0 library was used. The selection was based on the comparison of features and the level of implementation of the state-of-the-art RL algorithms. This library, despite its scalability, has a higher complexity of use compared to other libraries. However, other implementations analyzed prior to the study have an encapsulated and rigid code around the models that is difficult to customize.

In the Table 3.2 are listed the hyperparameter values configured and allowed by the *RLlib* library. The architecture for both agents was implemented using the Keras API from TensorFlow.

Hyperparameters	**Value**
Optimizer	Adam
Actor learning rate	10^{-3}
Critic learning rate	10^{-3}
Experience replay buffer size	10^{5}
Batch size	256
Discount factor	0.99
Ornstein-Uhlenbeck noise (DDPG)	θ: 0.15, σ: 0.2, base scale: 0.1
Gaussian noise (TD3)	mean: 0.0, std: 0.1
Exploration decay (type)	linear
Exploration decay (initial scale)	1.0
Exploration decay (final scale)	10^{-3}
Target policy smoothing (Gaussian)	mean: 0.0, std: 0.1
Target policy smoothing (clip)	[-0.5, 0.5]
Maximum timesteps per episode	150

TABLE 3.2: Hyperparameter configured and allowed by *RLlib* library.

The training was done using a remote workstation with an Intel(R) Xeon(R) Silver 4114 CPU @ 2.20 GHz, 125 GiB of RAM, and a Graphics Processing Unit (GPU) GeForce RTX 2080 Ti. Each training session, lasting approximately 18 hours each, ended in 375 thousand timesteps.

The algorithms were trained 10 times, with different seeds for the random number generator, to identify repetitive behaviors. The policies with the highest mean reward during training were selected for both simulation and real-world evaluation.

3.2.3.2 Training

Figure 3.9 shows the mean and standard deviation of both, the perceived reward (upper) and the Q-value (bottom), for DDPG and TD3, in 10 different training sessions.

DDPG achieves a value of a reward close to the optimal one (40.0) while TD3, although in the first third of the training shows an increasing trend, falls towards negative values. In addition, DDPG showed increasing learning with a steep slope during the exploration phase (first third), but the results in the exploitation phase, although slightly upward trending, were slow. Considering those results, in this particular case, DDPG outperforms TD3 in the acquisition of the extraction skill.

Even when the DDPG Q-function overestimated the Q-value (a known DDPG behavior [125]) during the exploitation phase, it remained relatively stable around the maximum expectation, as shown in the Figure 3.9 (down). In contrast, TD3 started to learn the skill, but towards the end of the exploration phase, its learning performance deteriorated and did not recover. This can be attributed to the inappropriate estimation of the Q-function used as a basis for updating its actor policy, as shown in the Figure 3.9 (bottom).

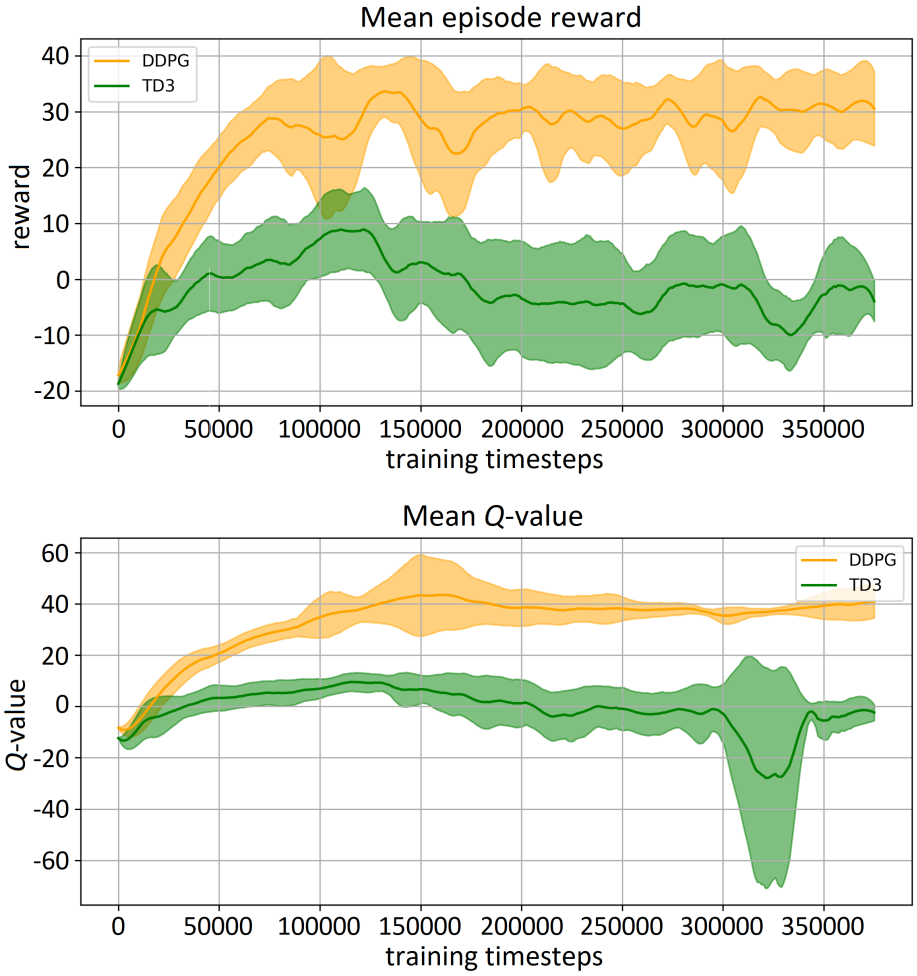

FIGURE 3.9: Simulation training results: Mean reward and standard deviation during training (upper). Mean estimated Q-value and standard deviation returned by Q networks (bottom). For TD3, Q-value corresponds to $Q_{\phi 1}$ (used to optimize the policy).

Surprisingly, there is a notable difference in the behavior of the two policies. According to the definition of the action space, the initial object position and geometry, a successful extraction requires at least 10 timesteps. TD3 learns the fastest way to perform the disassembly task (assuming it can successfully complete an episode), while DDPG takes more timesteps (about 24 timesteps), as shown in Figure 3.10.

Figure 3.11 illustrates such behavior by mapping the actions taken by both policies. The "extreme" actions of TD3 (right) are located near the boundaries of the action space which is $[-1, 1]$ centimeters. In contrast, the scatter of DDPG actions (left) is concentrated around a central perimeter, resulting in shorter displacements and slower extraction speed.

3.2.3.3 Evaluation of the generalization capabilities

As stated above, for both simulation and real-world evaluation, the best policies were selected based on the mean reward value during training.

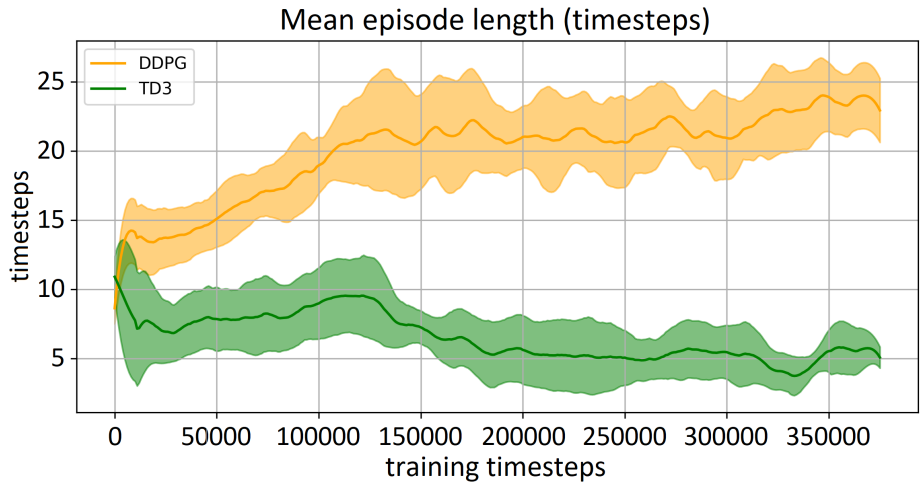

FIGURE 3.10: Mean and standard deviation of the length of the episodes (in timesteps) during the training.

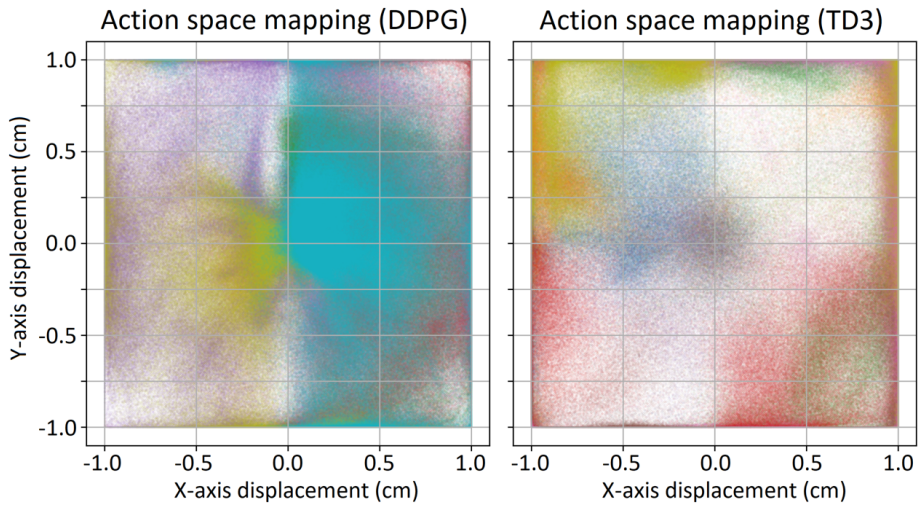

FIGURE 3.11: DDPG (left) and TD3 (right) mapping of the action $a \in [-1, 1]$ performed during training.

Simulation

Two tests were performed to evaluate the generalization of the learned skill to different initial conditions: rotation and location (position). The agents completed 30 episodes for each case, and the evaluations were run with the exploration behavior turned off.

- **Initial object rotation**: The rotation of the target object was sampled discretely between 0 and 180 degrees, with an interval of one degree. The Figure 3.12 shows the mean and the standard deviation of the reward obtained by the best trained policies as well as their mean episode length.

  In 87.29% of the sampled rotations, the policy trained with DDPG was able to complete the task with more than 15 successful episodes per angle. If the episode runs for more timesteps (than the maximum allowed) for the rotation interval between 35 and 55 degrees, the proportion of completed tasks for DDPG could be higher.

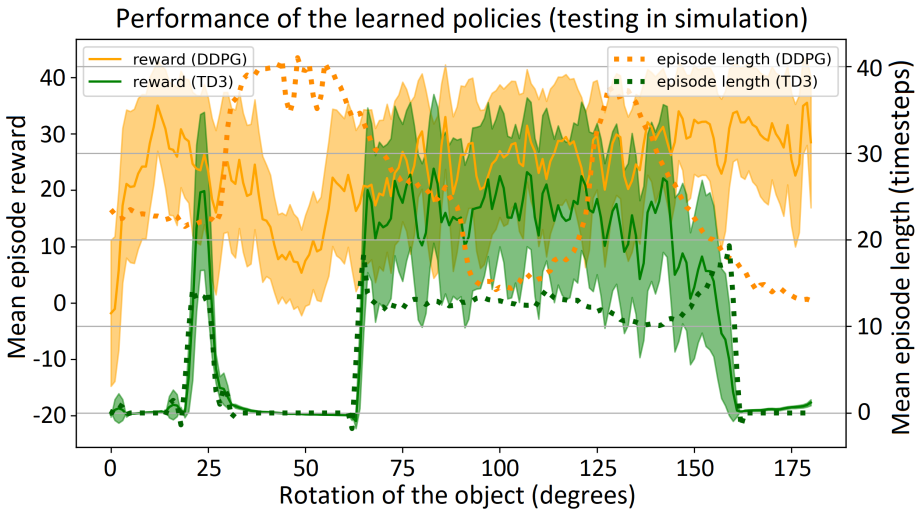

FIGURE 3.12: Mean reward and standard deviation (on the left axis scale) and mean episode length (on the right axis scale) obtained during the evaluation of the learned policies in the simulated environment at different initial rotations.

On the other hand, the policy trained using TD3 was only able to successfully complete the task in 35.35% of the sampled rotations. It had more than 15 successful episodes per angle. As expected from the training process, DDPG outperformed TD3 in the disassembly task, although TD3 performed faster than DDPG.

- **Initial object position**: The initial position of the target object on the table was sampled to cover a rectangular region of 0.1 x 0.25 meters with a step of 2.5 millimeters.

  For this scenario, the object maintains a fixed rotation (of 50 and 130 degrees) selected from the regions where policy performance is quite diverse according to the Figure 3.12. The center of the region of interest is located at 0.6 meters in front of the robot. The mean reward achieved by both agents, at the end of the task, is shown in Figure 3.13.

  The results show that the learned skill is generalizable for a wide range of different initial positions. The policy trained with DDPG is able to perform the extraction for different initial positions, even for the worst performing rotation. In the case of TD3, its negative performance for a rotation of 50 degrees is maintained for different initial positions.

  The use of position information relative to its initial value, as part of the observation space, is key to achieve such behavior.

Transference to the real world

In addition, the study evaluated the generalization of the learned skill in the real world. For this purpose, the tests were performed for different initial object rotations. The rotation of the target object was sampled discretely between 0 and 180 degrees with a 10 degree interval to reduce the number of episodes executed. The agents performed five episodes for each particular case.

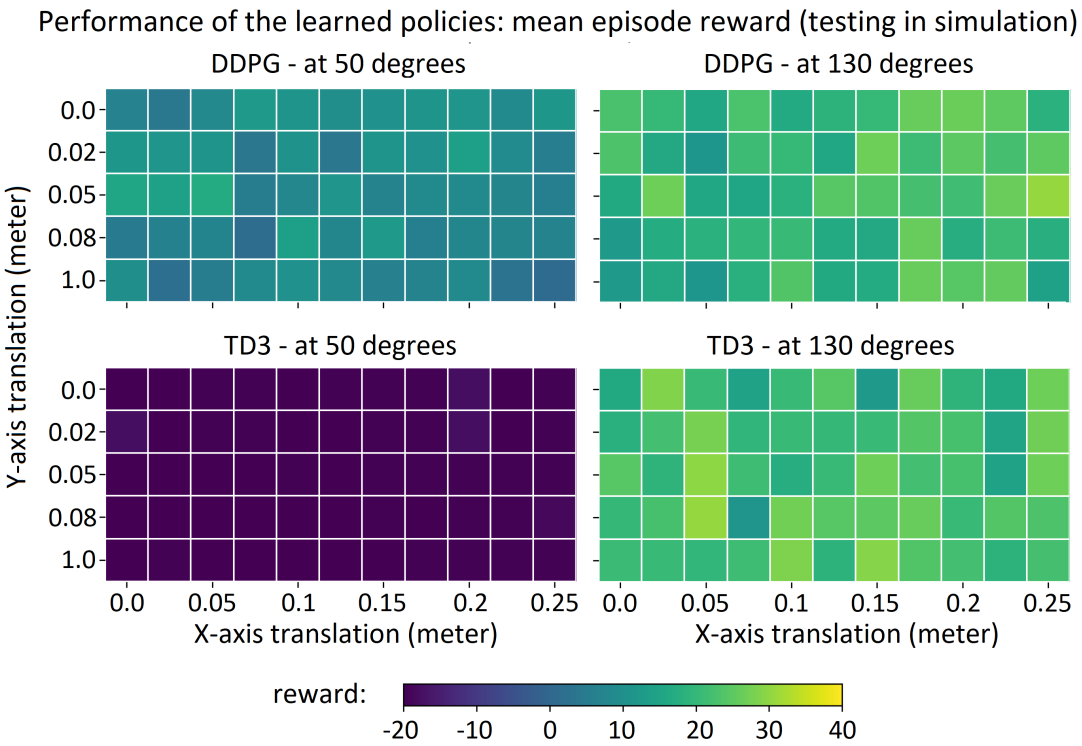

FIGURE 3.13: Performance of the learned policies (DDPG at the top and TD3 at the bottom), without exploration, in the simulated environment evaluated at different starting positions.

Both the number of angles sampled and the number of episodes evaluated correspond to the difficulty and time required to perform real-world experiments. The results of the evaluation are presented in Figure 3.14, which shows the mean reward obtained by the best-trained policies and their mean episode length.

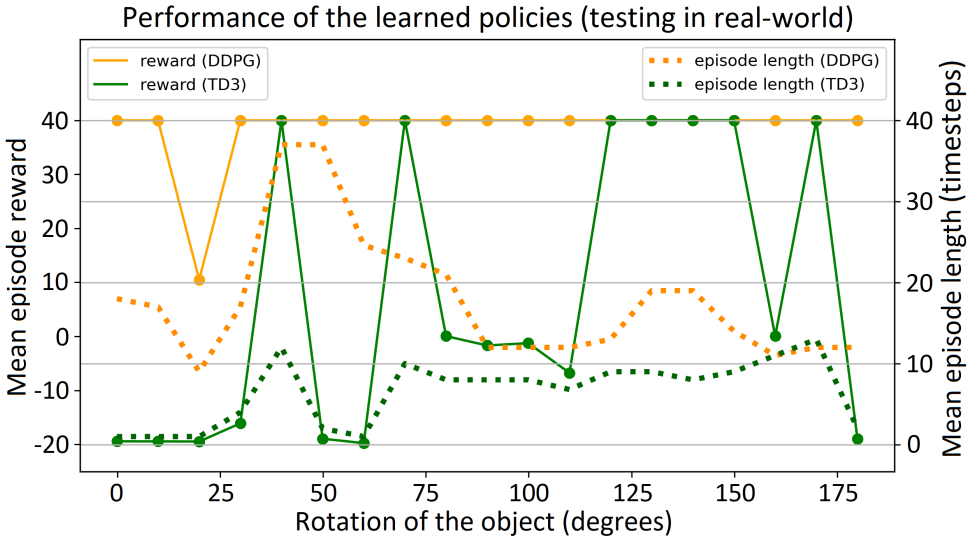

FIGURE 3.14: Mean reward and standard deviation (on the left axis scale) and mean episode length (on the right axis scale) obtained during the evaluation of the learned policies in the real world environment at different initial rotations.

In the real-world evaluations, the DDPG policy showed very similar behavior

to that observed in the simulation, successfully performing the disassembly task for all angles sampled. On the other hand, TD3 continued to show poor performance, similar to what was observed in the simulation.

In fact, the success rate for the real-world execution was higher than that in the simulation, attributed to the slight differences in the critical force threshold used for both environments. The training and evaluation performed in the simulated environment used a critical force threshold of 30 Newtons, slightly lower than the 60 Newtons established in reality. The measured force during episode execution in the real world is shown in Figure 3.15.

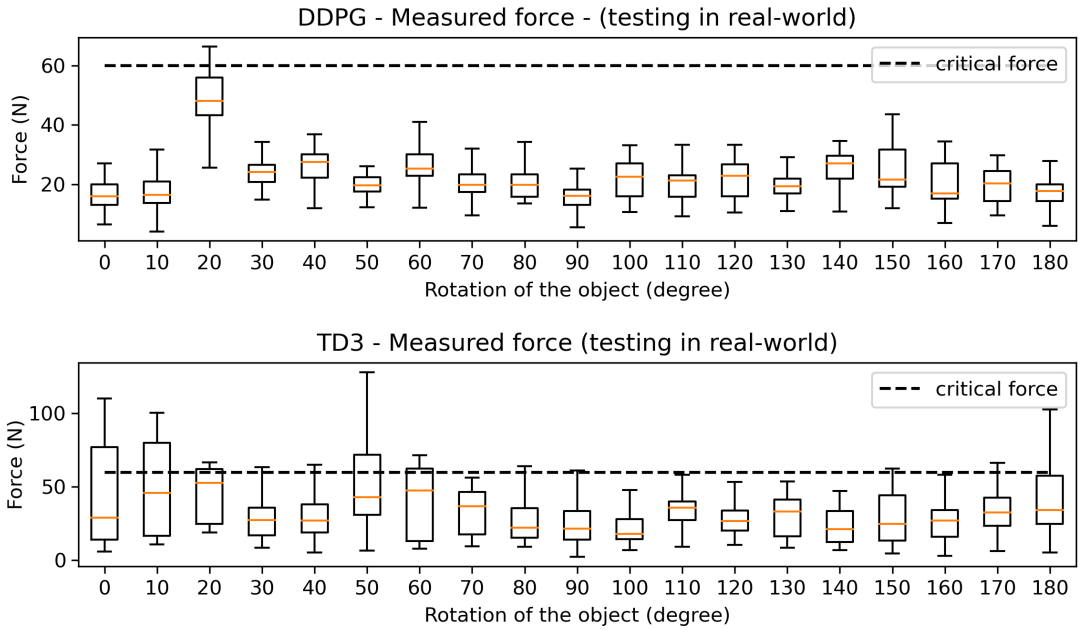

FIGURE 3.15: Force measurements obtained during the real-world evaluation of the DDPG and TD3 policies. The dashed line represents the critical force threshold used as the episode termination condition.

3.2.4 Summary of study results and limitations

DDPG and TD3, the selected model-free actor-critic algorithms, can learn the ability to extract objects by interacting with the environment. In addition, they can generalize learning across multiple initial conditions, such as positions and rotations. For this particular use case, TD3 shows poor performance, attributed to the learning of extreme actions at the boundaries of the action space. This phenomena, which was also reported in [84], attempts against the learning robustness.

Although the results of this initial study show promising results, its scope is limited to robot end-effector motions in only 2 DOFs for extraction on the XY-plane, as well as for the geometry described for the test case. The high degree of variability that can be found in EOL products, such as the physical properties and geometries of the objects manipulated, was not addressed.

In RL, training times are long, at least in a single environment, slowing or preventing iterative development and experimentation with different algorithms, architectures, or hyperparameters by waiting for training to complete. In addition, the solutions available to control the KUKA LBR iiwa robot do not allow access to most

of its features and do not offer integration to be able to use, from a single implementation, different workflows such as direct control from Python (used to define the environment) or via ROS.

3.3 Chapter conclusions

Although RL algorithms have been successful in learning robotic skills for manipulation tasks in manufacturing, particularly assembly tasks, their application to disassembly tasks (which are not necessarily the reverse of assembly tasks) has not been fully explored.

The research conducted responds, in part, to research question 1: The initial study on the use of RL algorithms for disassembly tasks shows that some RL algorithms can learn object extraction skills through interaction with the environment, and they can generalize these skills to various initial conditions, such as positions and rotations. However, the study's scope is limited to particular extraction motions and geometry. To perform disassembly tasks, agents must also be able to handle the high degree of variability of EOL product states.

Furthermore, on one side, training RL algorithms in a single environment can be time consuming and existing libraries hinder the customization of such applications as well as to train and evaluate over massive parallel environments.

Chapter 4

Reducing Learning Time and Resource Consumption by Simultaneous Training of Off-Policy Algorithms in Parallel Environments

The content of this chapter addresses research question 2 (a): How can the RL algorithms' learning process be leveraged for reducing training time and resource consumption?

The work presented in this chapter is partially published in the paper: Antonio Serrano-Munoz, Dimitrios Chrysostomou, Simon Bøgh, Nestor Arana-Arexolaleiba. "skrl: Modular and flexible library for reinforcement learning". In: Journal of Machine Learning Research 24.254 (2023), pp. 1–9. URL: https://www.jmlr.org/papers/v24/23-0112.html.

The time-consuming training and evaluation of RL algorithms not only limits the pace of scientific research, but also hinders the design, implementation, and deployment of related applications. Therefore, any effort to optimize those algorithms, including the use of computational techniques as well as software and hardware resources, is crucial to accelerate further research and application.

This chapter presents the research conducted to reduce the learning time and the amount of software and hardware resources required to train (off-policy) RL algorithms in parallel environments. It also describes a new RL library that implements and enables such a reduction, while providing a flexible and modular API that allows for further customization of research and application solutions.

4.1 Reinforcement learning in parallel environments

Due to the scientific community's need and interest in RL, several libraries and frameworks have been developed. Figure 4.1 shows how, starting in 2016, there is a significant increase in the development of RL libraries. Such growth can be explained by 3 fundamental milestones that mark the rise of interest in RL in our era:

- The development of new learning algorithms, especially those using artificial neural networks as approximation functions (Deep RL).

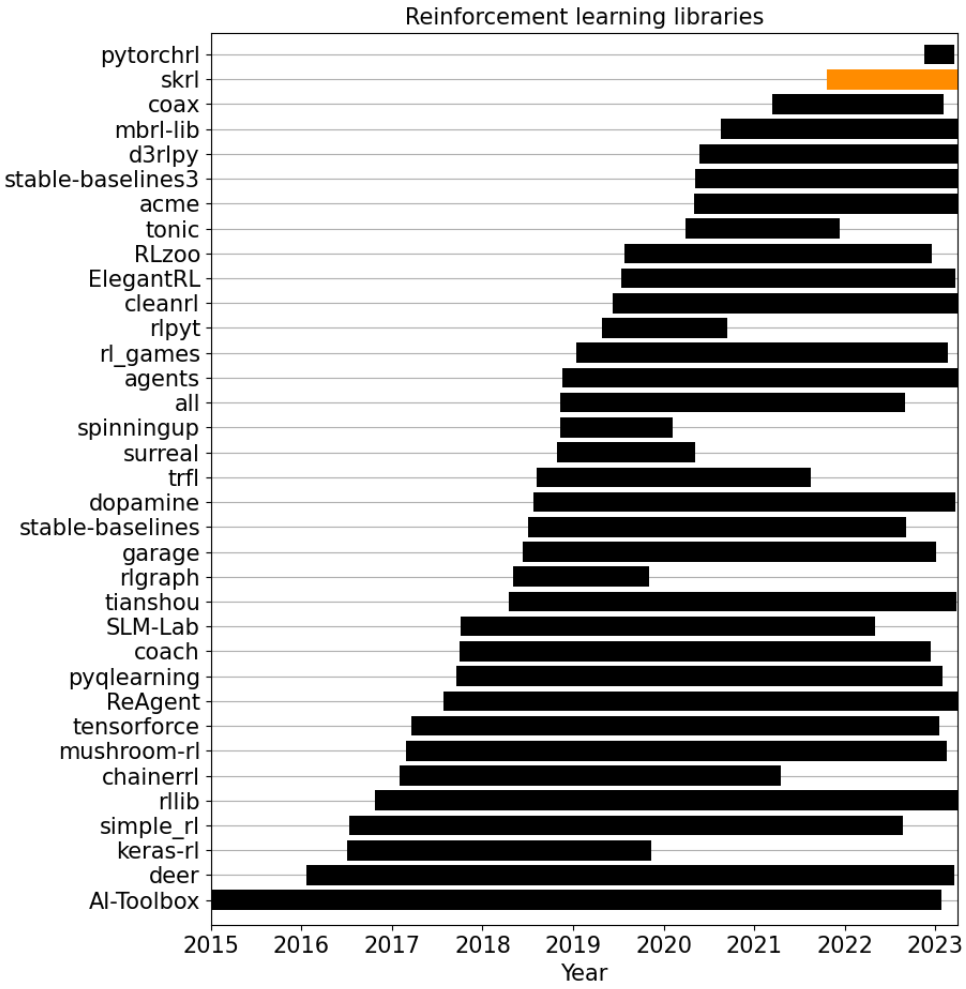

FIGURE 4.1: Main RL libraries' lifecycle. The lifecycle is computed using the repository's creation date and the last commit message retrieved from GitHub.

- The development of Gym by OpenAI (2016). It provides a common interface for the design and standardization of single and vectorized environments.
- The development of single-environment benchmarking scenarios in areas such as video games and gaming, autonomous navigation, and robotics. These benchmarks are widely accepted by the scientific community and allow to compare results between different implementations.

More recently, with the release of Isaac Gym preview (and later Omniverse Isaac Gym and Isaac Orbit), a GPU-based physics simulation platform by NVIDIA, a new generation of robotics simulation has emerged with tens of thousands of parallel environments running on a single GPU [126]. These platforms allow researchers to easily run massive experiments using an API similar to OpenAI Gym, offloading both physics simulation and neural network training to the GPU.

Massive parallelization offers on-policy algorithms, such as PPO and TRPO, the advantage of diverse exploration, allowing such agents to concurrently experience a wider range of states and actions. This increased diversity contributes to overcoming suboptimal solutions and significantly reducing training time. In contrast, off-policy algorithms such as DDPG, SAC or TD3, do not yield benefits in massively

parallel environments, requiring a much longer training time and a smaller number of environments even though these algorithms are more sample efficient than on-policy.

On-policy high performance increase in massive parallel environments is reported in the NVIDIA family of physics simulators (Isaac Gym, Isaac Orbit and Omniverse Isaac Gym), where PPO is presented as the de facto algorithm for performing training and evaluation [126][127][128][129]. For example, for the Omniverse Isaac Gym's Ant environment PPO training (in 4096 parallel environments) runs 16 times faster than SAC training (in 64 parallel environments).

This research proposes a simultaneous learning method in parallel environments that takes advantage of the capabilities of off-policy algorithms to learn using previously generated data in order to reduce the overall training time and save computing resources. This method is embedded in a new RL library designed with special attention to handle massively parallel environments, mainly because existing libraries have a structure, even for modular implementations, and rigid code that is difficult to scale-up and adapt.

The following section describes the implemented library. In the subsequent section, and with an understanding of the implementation of simultaneous learning, the experiments and results are presented.

4.2 Modular and flexible reinforcement learning library

This section describes the design and implementation of a RL library (*skrl*) that enables simultaneous training in massively parallel environments to reduce training time and resource consumption. As an added benefit, the implementation focuses on the following principles: modularity, leaving room for each component to be interchangeable and allowing the creation of more complex systems; readability, simplicity and transparency of algorithm implementations, reducing the learning curve; and support for different interfaces, among others.

4.2.1 Library architecture and components

skrl is an open-source modular library for RL written in Python (using PyTorch) and designed with a focus on readability, simplicity, and transparency of algorithm implementation. In addition to supporting the OpenAI Gym / Farama Gymnasium, DeepMind and other environment interfaces, it allows loading and configuring NVIDIA Isaac Gym, NVIDIA Isaac Orbit and NVIDIA Omniverse Isaac Gym environments, enabling agents' simultaneous training by scopes (subsets of environments among all available environments), which may or may not share resources, in the same run.

The library consists of several components that can be integrated under a single execution to create experiments for training and testing RL applications. These components, illustrated in Figure 4.2, are: agents, environments (named as *envs*), memories, models, resources, trainers and utilities (named as *utils*). In turn, the resources are divided into three sub-components: noises, input data pre-processors (named as *preprocessors*) and learning rate schedulers (named as *schedulers*).

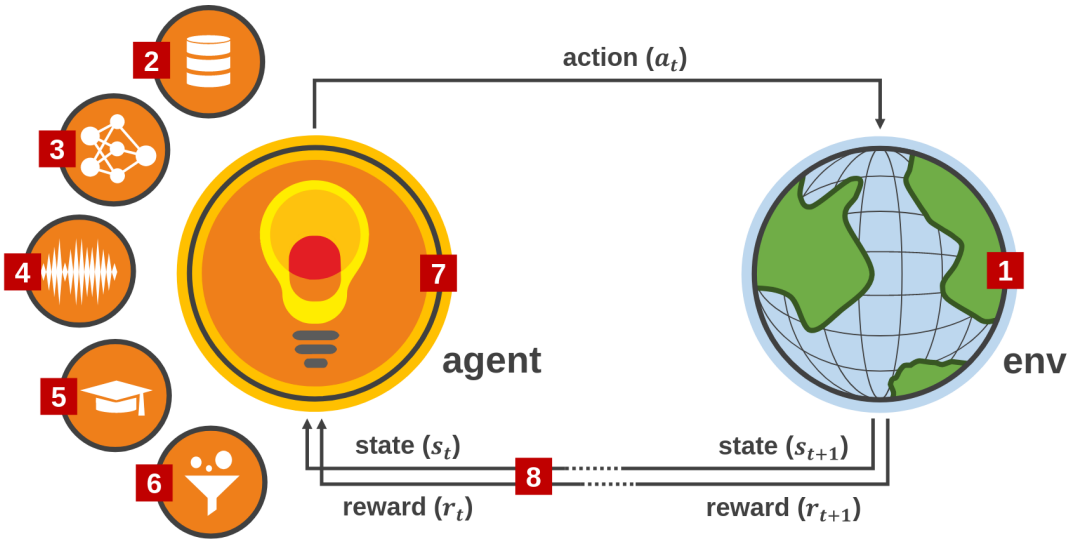

FIGURE 4.2: Reinforcement learning schema.

4.2.1.1 (1) Environments

The environment plays a fundamental and crucial role in the definition of the RL application. It is the place where the agent interacts, and it is responsible for providing the agent with information about the current state of the system, as well as the rewards and penalties associated with each action. The nature of the environment shapes the problem that the agent must solve, and thus has a profound impact on the selection of the agent and its design. Because observation and action spaces can be continuous or discrete in nature, and high or low dimensional, not all problems can be solved with the same types of agents, policy representations, and value functions or algorithms. For example, a continuous observation and action space will require a different approach than a discrete one.

As mentioned above, there are several interfaces to interact with the environments such as OpenAI Gym / Farama Gymnasium or DeepMind. However, each of them has a different API and works with non-compatible data types.

skrl provides a function to wrap the environment described above and return a common interface on which the defined agents can operate. This interface is based on the most updated Gym/Gymnasium[1] API in terms of method names, arguments and return values. In addition, it defines some properties to support parallel environments and multi-device execution, as shown in Figure 4.3.

Properties:

- `num_envs`: Number of parallel environments. If the wrapped environment does not have this property, it will be set to 1.
- `device`: The device used by the environment. If the wrapped environment does not have this property, the value of this property will be `cuda:0` or `cpu` depending on the device availability.
- `observation_space`: Observation space as a Gym space object.
- `state_space`: Alias for `observation_space`.

[1] https://www.gymlibrary.dev/api/spaces/#gym.spaces.Space

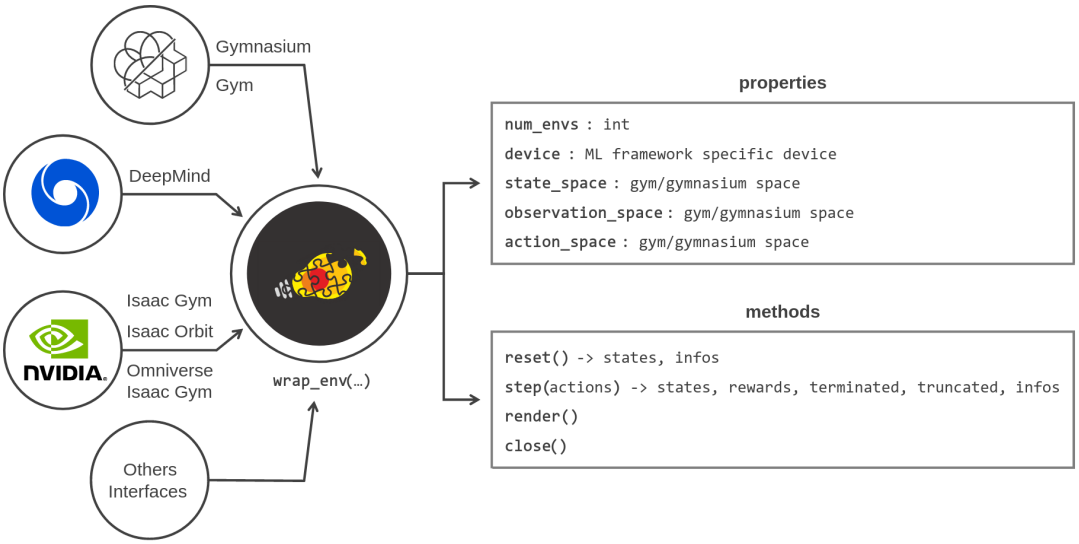

FIGURE 4.3: Supported environment interfaces and wrapping

- `action_space`: Action space as a Gym space object.

Methods:

- `reset()`: Resets the environment to an initial state and returns the initial observation. This method returns the state of the environment after a restart and optionally additional information that may be useful for the learning algorithms.
- `step()`: Run one timestep of the environment's dynamics. This method returns the next state of the environment due to the agent's actions, the instantaneous reward as a result of taking the action, whether the environment reaches its final state, whether the truncation condition outside the scope of the MDP is satisfied, and optionally additional information that may be useful for the learning algorithms.
- `render()`: Display the current state of the environment to the user. The specific behavior of this method will depend on the particular RL environment being wrapped.
- `close()`: Perform any necessary cleanup.

At the time of writing, the next environment interfaces have been supported:

- OpenAI Gym [130]: A set of tools for developing and comparing RL algorithms. OpenAI Gym provides a standardized Python interface for interacting with different environments, such as games, robotic simulators, and control systems. Of all the environment interfaces, this is the most popular in the RL community.
- Farama Gymnasium [131]: A fork of the OpenAI Gym library maintained by the Farama[2] foundation.

[2] Farama is a nonprofit organization working to develop and maintain open source RL tools: https://farama.org/

- DeepMind [132][133]: A set of continuous control tasks from DeepMind, with a standardized structure and interpretable rewards, designed to serve as performance benchmarks for RL agents.
- robosuite [134]: A simulation framework based on the MuJoCo physics engine for robot learning that provides a set of reference environments for reproducible research.
- NVIDIA Isaac Gym preview [127][126]: A physics simulation platform from NVIDIA for GPU-accelerated RL research.
- NVIDIA Omniverse Isaac Gym [127][126]: An extension that provides an interface for performing RL training and inference in Isaac Sim for GPU-accelerated RL research.
- NVIDIA Isaac Orbit [128]: An open-source unified and modular framework for robot learning in Isaac Sim.

4.2.1.2 (2) Memories

Memories are storage components that allow agents to collect and use/reuse recent or past experiences of their interaction with the environment, or other types of information. They can be large to help to break the temporal correlation between experiences (such as replay buffers used by off-policy algorithms like DDPG, TD3, or SAC) or small (such as rollout buffers used by on-policy algorithms like PPO or TRPO to store batches that are discarded after use).

skrl provides generic memory definitions that are not tied to the agent implementation and can be used for any role, such as rollout or replay buffers. They are empty shells when they are instantiated and the agents are in charge of defining the tensors according to their needs. The total space occupied is the product of the memory size (`memory_size`), the number of environments (`num_envs`) obtained from the wrapped environment, and the data size for each defined tensor as shown in Figure 4.4.

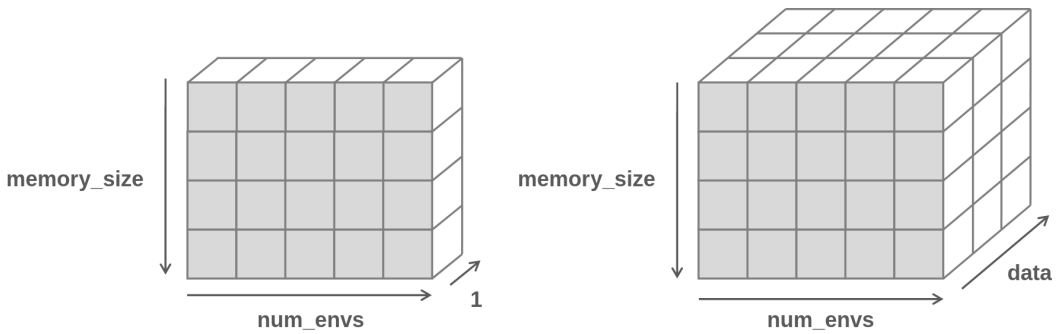

FIGURE 4.4: Generic definition of tensors in memory

Memories are passed directly to the agent constructor during its instantiation if needed (not all agents need memory, such as Q-learning or SARSA).

At the time of writing, the following memories have been implemented:

- Random sampling memory: Memory that randomly samples, with or without replacement, sample batches of transitions.

4.2.1.3 (3) Models

Models (or agent models) refer to a representation of the agent's policy or value function that the agent uses to make decisions. Agents can have one or several models and their parameters are adjusted via the optimization algorithms.

skrl offers two main types of agent models: tabular models and function approximation models. A tabular model represents the agent's policy or value function as a table of values, where each entry in the table corresponds to a particular state-action pair and contains the corresponding value or probability. Tabular models are only feasible for problems with a small or discrete state and action space. In this case, the agent can learn the optimal policy or value function by looking up the values in the table. A function approximation model represents the agent's policy or value function as a mathematical function, such as a neural network, that can be used to approximate the true values or probabilities for any state-action pair. This allows the agent to generalize from its experience and make decisions even for unseen states, making it more sample efficient and able to handle problems with larger state and action spaces.

In contrast to other implementations, this library does not provide predefined models or fixed templates (this practice tends to hide and reduce the flexibility of the system, forcing developers to deeply inspect the code to make even small changes). Nevertheless, helper classes/mixins are provided to create discrete and continuous (stochastic or deterministic) models with the library. In this way, the user/researcher should only be concerned with the definition of the approximation functions (tables or artificial neural networks), having all the control in his hands.

At the time of writing, the following models have been implemented:

- Tabular model: Models that represent the agent's policy or value function as a table of values (for discrete domain observation and action spaces).
- Categorical model: Models that represent the agent's policy or value function as function approximators (for discrete or continuous domain observation spaces and discrete action spaces) that populate a Categorical distribution. Figure 4.5 depicts the model as it is used in the *skrl* library.

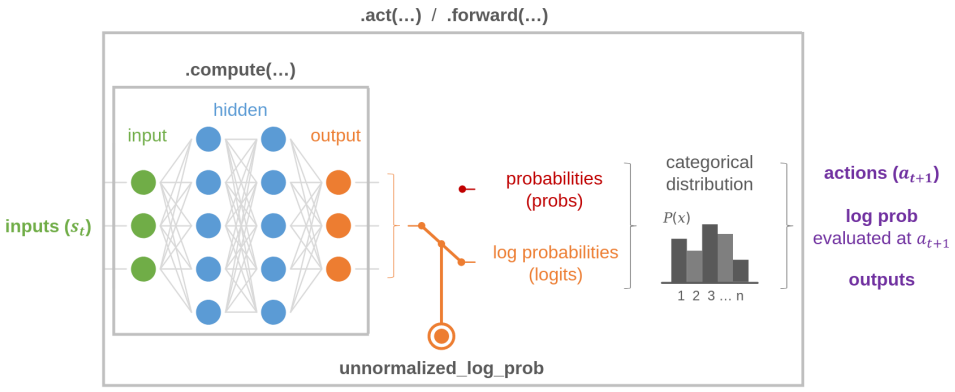

FIGURE 4.5: Categorical model.

- Deterministic model: Models that represent the agent's policy or value function as function approximators (for discrete or continuous domain observation

spaces and continuous action spaces) that deterministically map states to actions. Figure 4.6 depicts the model as it is used in the *skrl* library.

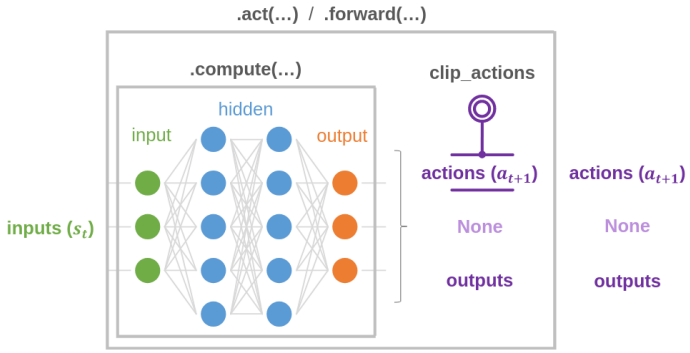

FIGURE 4.6: Deterministic model.

- Gaussian model: Models that represent the agent's policy or value function as function approximators (for discrete or continuous domain observation spaces and continuous action spaces) that populate a Gaussian distribution. Figure 4.7 depicts the model as it is used in the *skrl* library.

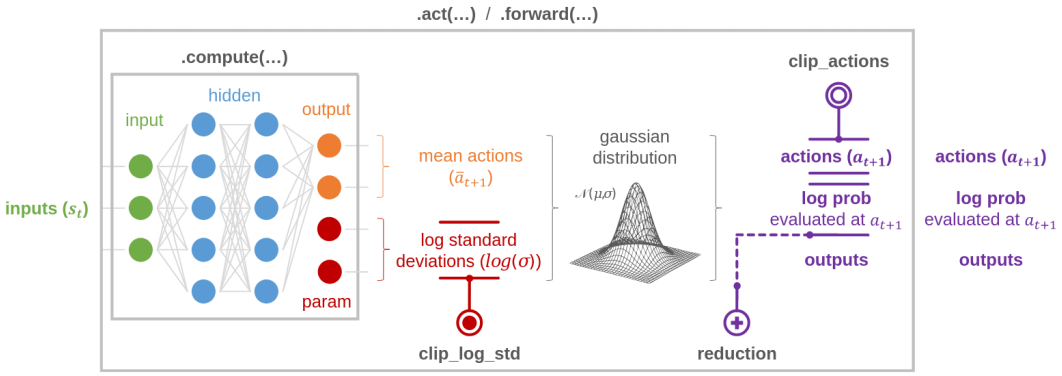

FIGURE 4.7: Gaussian model.

- Multivariate Gaussian model: Models that represent the agent's policy or value function as function approximators (for discrete or continuous domain observation spaces and continuous action spaces) that populate a multivariate Gaussian distribution. Figure 4.8 depicts the model as it is used in the *skrl* library.

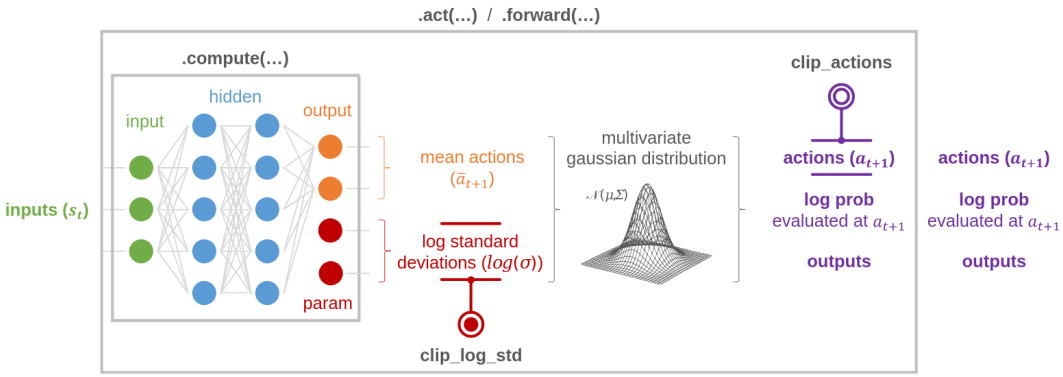

FIGURE 4.8: Multivariate Gaussian model.

Models must be collected in a Python dictionary and passed to the agent constructor during its instantiation. The number of models is specific to each agent.

4.2.1.4 (4) Noises

Noise refers to a way of adding randomness to the agent's actions during the exploration phase to encourage the agent to try different actions and explore the state space. The goal of adding exploration noise is to balance the use of the agent's current knowledge with the exploration of new states and actions that may lead to better rewards. In agents of a deterministic nature, such as DDPG or TD3, it plays a fundamental role.

As part of its resources, *skrl* provides classes for instantiating noises. Noise instances must be passed to the agents in their respective configuration dictionaries.

At the time of writing, the following noises have been implemented:

- Ornstein-Uhlenbeck noise: Noise generated by a stochastic process that is characterized by its mean-reverting behavior.
- Gaussian noise: Noise generated by a normal distribution, parameterized by the mean value and standard deviation.

4.2.1.5 (5) Schedulers

A learning rate scheduler is a technique used in RL to adjust the agent's learning rate during training to improve the agent's performance. The learning rate is a hyperparameter that controls the step size of the model parameter update. Learning rate schedulers help RL systems converge faster and improve accuracy.

skrl supports all PyTorch learning rate schedulers and provides additional schedulers as part of its resources. The learning rate schedulers' classes and their respective arguments (except the optimizer argument) are passed to the agents in their respective configuration dictionaries.

At the time of writing, the following schedulers have been implemented:

- KL Adaptive: Adjust the learning rate according to the value of the Kullback-Leibler (KL) divergence between two versions of the distribution used by stochastic policies for decision-making.

4.2.1.6 (6) Preprocessors

Input preprocessors are techniques used to process and transform input observations from the environment before they are used as input to the agent. The goal of an input preprocessor is to clean or make the input data more suitable for the agent to learn from and to improve its performance.

skrl provides preprocessor classes as part of its resources. Preprocessor classes and their respective arguments are passed to the agents in their respective configuration dictionaries.

At the time of writing, the following input preprocessors have been implemented:

- Running standard scaler: Standardize input features by removing the mean and scaling to unit variance.

4.2.1.7 (7) Agents

An agent is an autonomous entity that interacts with an environment to learn and improve its behavior. The agent performs actions in the environment based on its current state and receives information in the form of a reward signal indicating the appropriateness of the action performed. The agent's goal is to learn an optimal policy, which is a correspondence between states and actions that maximizes the cumulative reward over time. Such agents use various learning algorithms, such as Q-learning or policy gradient methods, to update their policy or value function.

The learning and optimization algorithm is implemented within a function under the same name (*_update*) in all cases.

At the time of writing, the following state-of-the-art agents, alphabetically arranged, have been implemented:

- Advantage Actor-Critic (A2C) [135]: a model-free, stochastic on-policy policy gradient algorithm that uses the advantage function to update the policy with the goal of maximizing the expected cumulative reward. It can be parallelized, allowing for faster training times.
- Adversarial Motion Priors (AMP) [136]: a model-free, stochastic on-policy policy gradient algorithm (trained using a combination of GAIL and PPO) for adversarial learning of physics-based character animation. It enables characters to imitate diverse behaviors from large unstructured datasets, without the need for motion planners or other mechanisms for clip selection.
- Cross-Entropy Method (CEM) [137]: a model-free algorithm that selects elite states and actions based on a performance metric, such as the average reward received over a fixed number of timesteps.
- Deep Deterministic Policy Gradient (DDPG) [124]: a model-free, deterministic off-policy actor-critic algorithm that uses deep function approximators to learn a policy (and to estimate the action-value function) in high-dimensional, continuous action spaces.
- Double Deep Q-Network (DDQN) [138]: a model-free, off-policy algorithm that relies on double Q-learning to avoid the overestimation of action-values introduced by DQN.
- Deep Q-Network (DQN) [139]: a model-free, off-policy algorithm that trains control policies directly from high-dimensional sensory using a deep function approximator to represent the Q-value function.
- Proximal Policy Optimization (PPO) [140]: a model-free, stochastic on-policy policy gradient algorithm that alternates between sampling data through interaction with the environment, and optimizing a surrogate objective function while avoiding the new policy does not move too far away from the old one.
- Q-learning (Q-learning) [141]: a model-free off-policy algorithm that uses a tabular Q-function to handle discrete observations and action spaces.

- Robust Policy Optimization (RPO) [142]: a model-free, stochastic on-policy policy gradient algorithm that adds a uniform random perturbation to a base parameterized distribution to help the agent maintain a certain level of stochasticity throughout the training process.
- Soft Actor-Critic (SAC) [143]: a model-free, stochastic off-policy actor-critic algorithm that uses double Q-learning (like TD3) and entropy regularization to maximize a trade-off between exploration and exploitation.
- State Action Reward State Action (SARSA) [144]: a model-free on-policy algorithm that uses a tabular Q-function to handle discrete observations and action spaces.
- Twin-Delayed DDPG (TD3) [125]: a model-free, deterministic off-policy actor-critic algorithm (based on DDPG) that relies on double Q-learning, target policy smoothing and delayed policy updates to address the problems introduced by overestimation bias in actor-critic algorithms.
- Trust Region Policy Optimization (TRPO) [145]: a model-free, stochastic on-policy policy gradient algorithm that deploys an iterative procedure to optimize the policy, with guaranteed monotonic improvement.

4.2.1.8 (8) Trainers

Trainers are responsible for the management of agent training/evaluation and their interaction with the environment as shown in Figure 4.9. They orchestrate the training/evaluation process in a loop by:

1. Pre-interaction: Execute any agent logic prior to interaction with the environment.
2. Compute actions: Forward the current state of the environment to the agent to make a decision based on its policy.
3. Interact with the environments: Executes a timestep of the dynamics of the environment using the actions taken by the agent.
4. Render scene: Represent on the screen or terminal the current state of the environment, usually for human consumption.
5. Record transitions: Store an environment transition in memory for use during training, or store data persistently for future use.
6. Post-interaction: Execute any agent logic after interacting with the environment. This step runs the agent optimization algorithm.
7. Reset environments: Restores the environment to an initial internal state when the program is started, or when the environment is terminated or truncated.

They are also responsible for initializing the agents and, unless otherwise specified, closing the environments at the end of the training or evaluation execution.

At the time of writing, the following trainers have been implemented:

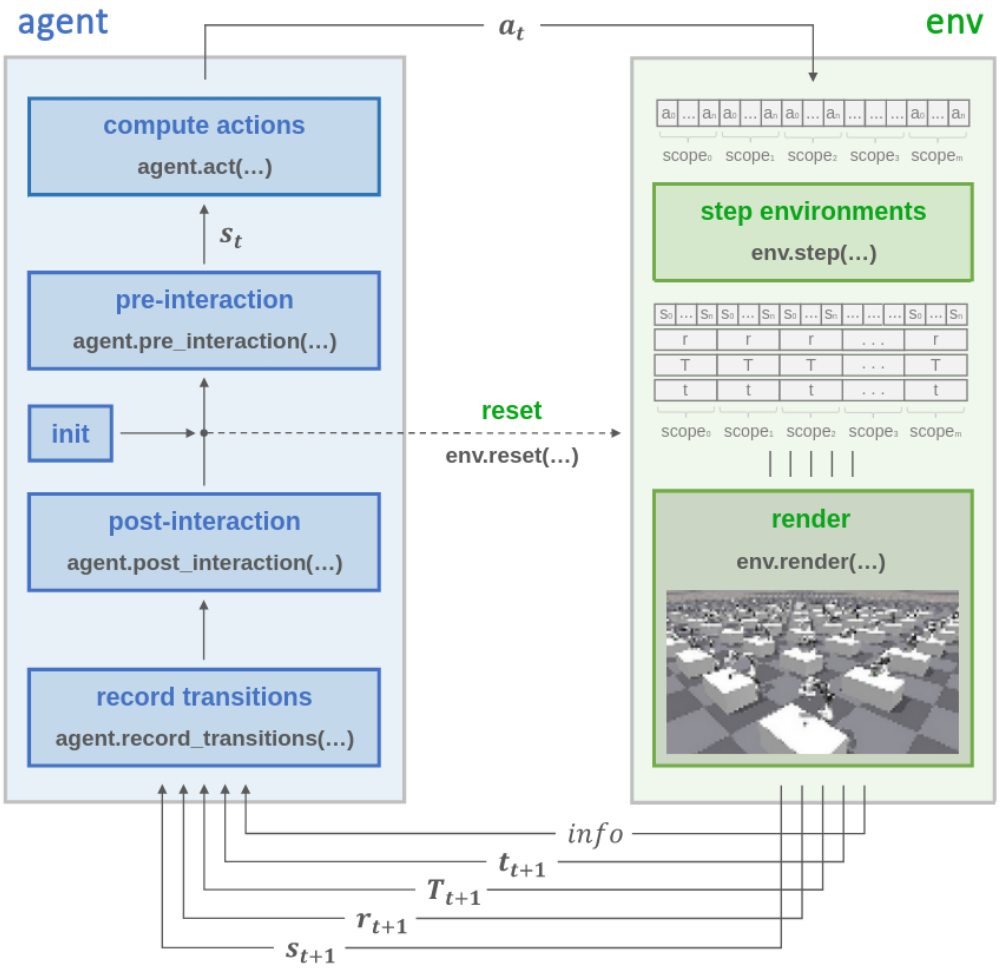

FIGURE 4.9: Training/evaluation pipeline.

- Sequential trainer: Train agents sequentially (i.e., one after the other in each interaction with the environment) as shown in Figure 4.10 (left).
- Parallel trainer: Train agents in parallel using multiple processes as shown in Figure 4.10 (right).
- Manual trainer: Train agents by manually controlling the training/evaluation loop. If there are simulated agents, they are trained/evaluated sequentially.

These definitions also allow the execution of simultaneous synchronous learning in Isaac Gym, Isaac Orbit, and Omniverse Isaac Gym.

Agents can specify a scope consisting of set of sub-environments among all available environments. During each timestep, the trainer collects the actions taken by each agent within their respective scopes and creates a single action vector that is fed into the environment stepping pipeline. After stepping, observations, rewards, and any other information are split and delivered to each agent based on their scope for learning and optimization.

This approach allows simultaneous learning and comparison of agents, hyperparameters, and other components in a single run. In addition, the modular design of the library allows resources such as memory (buffer replay or rollout buffer) to be shared between agents.

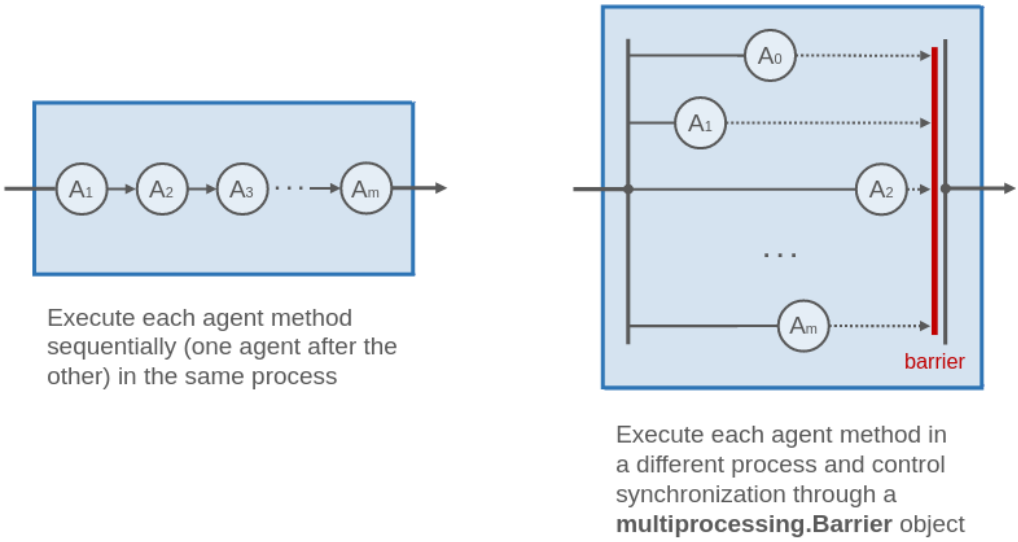

FIGURE 4.10: Sequential (left) vs parallel (right) execution.

4.2.1.9 Utils

In addition, as mentioned above, a set of utilities are offered to perform, among others, the following operations:

- Loading and post-processing of TensorBoard files: Utility to load and iterate over Tensorboard log files generated during training or evaluation. For example, this utility allows to generate comparison and average plots of the training results to understand the behavior of the system.
- Loading and post-processing of exported memory files: Utility for loading and iterating over previously generated and exported memory files containing the transitions resulting from the agent's interaction with the environment.
- Instantiate models quickly: Utilities for quickly creating instances of different types of models by minimally specifying the parameters of their architectures. This utility allows the definition of models in a similar way to other RL libraries, but this practice typically hides model details from the user and makes customization hard.
- Computation of inverse kinematics for robotic manipulators in Isaac Gym and Omniverse Isaac Gym: Unify under a single and reusable function all the procedures used to compute the inverse kinematics, using the damped least squares method show in Equation 4.1, of a robotic manipulator for Isaac Gym and Omniverse Isaac Gym environments.

$$\Delta\theta = J^T(JJ^T + \lambda^2 I)^{-1}\,\vec{e} \tag{4.1}$$

  Where $\Delta\theta$ is the change in joint angles, J is the robotic manipulator Jacobian, λ is a non-zero damping constant, and $\vec{e}$ is the Cartesian pose error (position and orientation).
- Hugging Face integration: The Hugging Face Hub is a platform for building, training and deploying ML models, as well as accessing a variety of datasets and metrics for further analysis and validation.

This utility allows to download and use several trained models (agent checkpoints) for different environments/tasks of Gym/Gymnasium, Isaac Gym, Omniverse Isaac Gym, etc. that are available in the Hugging Face Hub. These models can be used as comparative benchmarks, to collect environment transitions in memory (e.g. for offline RL), or to pre-initialize agents to perform similar tasks, etc.

4.2.2 Documentation

The documentation is written using reStructuredText[3] and hosted online by Read the Docs under the URL `https://skrl.readthedocs.io` as shown in Figure 4.11.

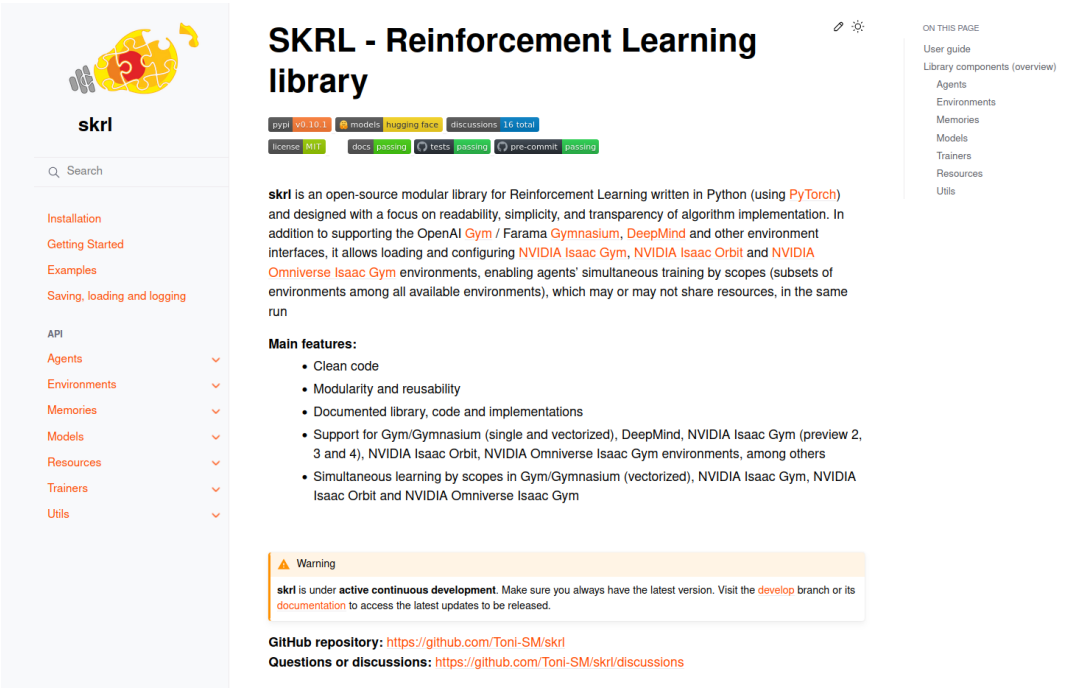

FIGURE 4.11: Screenshot of the *skrl* documentation home page

Apart from the library installation steps and API details (classes, functions, parameters, return values, etc.), snippets are also included to guide the development of new components or algorithms. In addition, a detailed mathematical description of the implementation of the RL agents is provided.

Examples of use cases are included with their respective scripts and description of functionalities such as tracking and visualizing metrics. In addition, examples are provided showing basic use cases of the Franka Emika Panda and KUKA LBR iiwa manipulator robots in the real world and *sim2real* applications in ROS and Robot Operating System - version 2 (ROS2) to guide and support advanced RL implementations.

[3] A plaintext markup language: `https://docutils.sourceforge.io/rst.html`

4.3 Training time and resource consumption reduction via simultaneous learning in parallel environments

As described above, *skrl* offers the capability to perform simultaneous training and evaluation of several agents by scopes, a unique feature of this library as far as it is known. This feature, combined with the capability of off-policy algorithms to learn a policy using data generated by the current policy, other policies, or external sources, can reduce the overall training time and further increase the obtained reward using the same computing resources.

For this, the 10 learning runs for DDPG, TD3 and SAC agents, with different trainers and memory modalities, using the same and hyperparameters and the number of parallel environments for Omniverse Isaac Gym Ant is carried out. Standalone and simultaneous sequential and parallel (without shared memory) training was done on 64 environments per agents. Simultaneous sequential training (with shared memory) was also done on 64 environments for all agents (where the scope was divided equally among the available agents).

The architecture of the Policy and Q networks used by all the agents is the same. The networks receive the observation space as input for the Policy, and the concatenated components of the observation space and action space for Q networks as input. They are followed by two dense hidden layers of 512 and 256 neurons respectively, each with ReLU activation functions. The Policy network has eight output neurons that use the hyperbolic tangent function to fit the action space to the expected interval $[-1, 1]$, while the Q networks use a linear activation function for their output neuron to estimate an unbounded Q-value.

In the Table 6.2 are listed the hyperparameter values configured and allowed by the *skrl* library.

Hyperparameters	**Value**
Optimizer	Adam
Policy/Value learning rate	$5 \cdot 10^{-4}$
Actor/Critic learning rate	$5 \cdot 10^{-4}$
Entropy learning rate (SAC)	$5 \cdot 10^{-3}$
Initial entropy value	1.0
Experience replay buffer size	15625
Batch size	4096
Discount factor	0.99
Ornstein-Uhlenbeck noise (DDPG)	θ: 0.15, σ: 0.2, base scale: 0.5
Gaussian noise (TD3)	mean: 0.0, std: 0.1
Exploration decay (type)	linear
Exploration decay (initial scale)	1.0
Exploration decay (final scale)	10^{-3}
Smooth regularization noise (TD3)	mean: 0.0, std: 0.2
Smooth regularization clip (TD3)	[-0.5, 0.5]
Soft update hyperparameter	0.005
Random timesteps	80
Learning starts	80
Gradient norm clip	0.0
State preprocessor	running-standard-scaler

TABLE 4.1: Hyperparameter configured and allowed by *skrl* library.

Figure 4.12 shows the mean reward and standard deviation of both standalone and simultaneous training (with shared memories) for each agent.

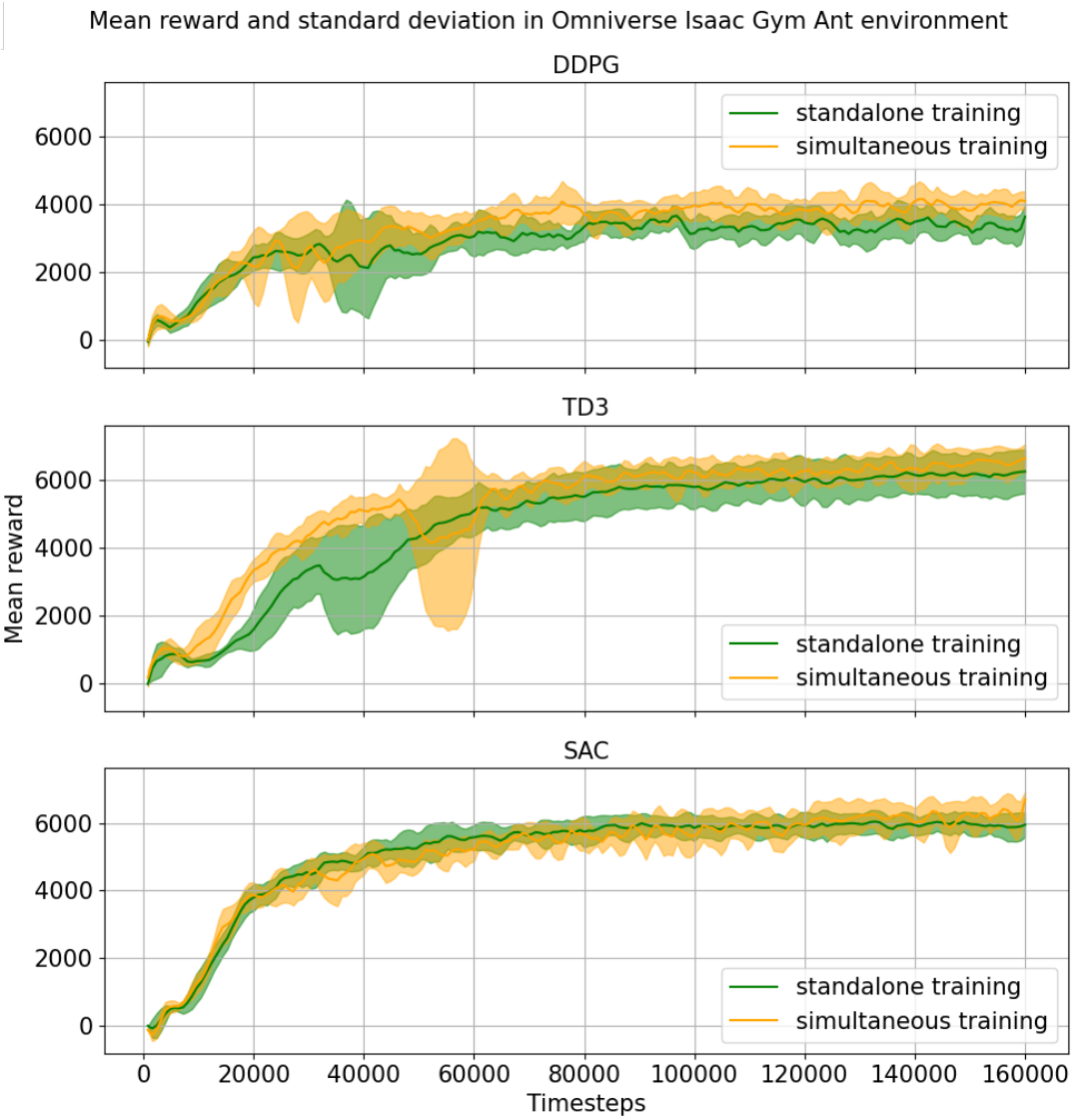

FIGURE 4.12: Mean reward and standard deviation of both standalone and simultaneous training (with shared memories) of the DDPG, TD3 and SAC actor-critic algorithms in the Omniverse Isaac Gym Ant environment.

Simultaneous training with shared memory yields slightly higher reward values than standalone training. This result can be attributed to training on data collected from different sources. In this case, the data collection comes from three different algorithms, each one with a different exploration strategy, making it possible to cover the observation-action space in a broader extent.

Regarding the training time, the use of sequential and parallel simultaneous trainers (with or without memory sharing) reduces the total training time, in about

one-fourth, compared to the overall training time of each algorithm alone, as shown in the Figure 4.13.

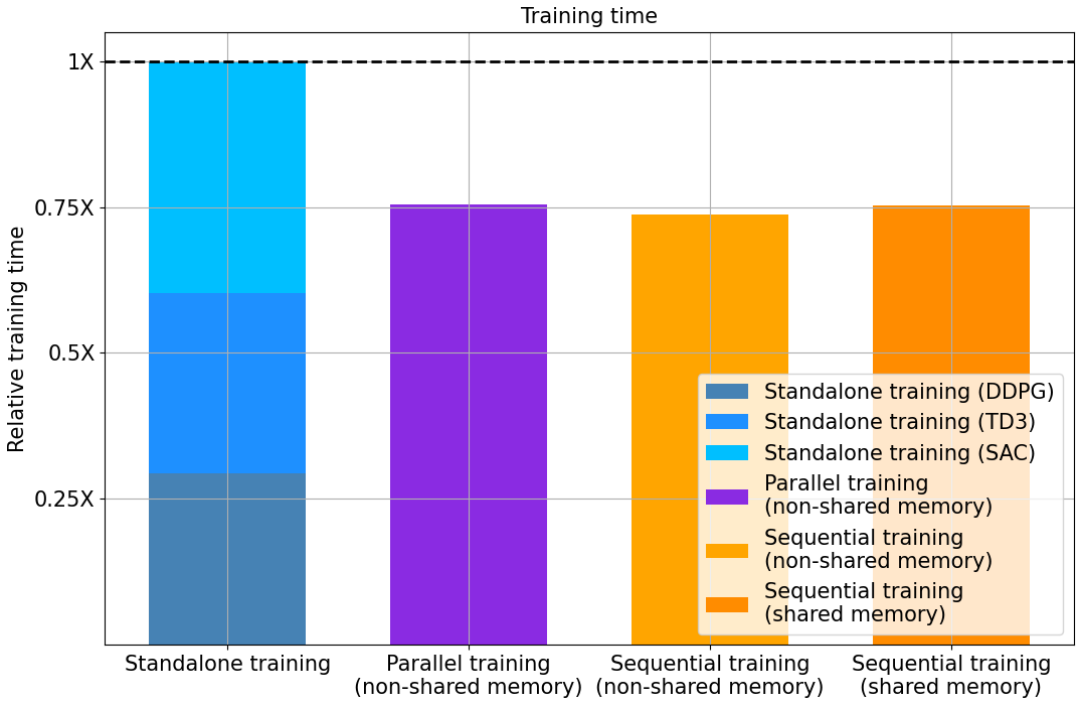

FIGURE 4.13: Relative times for standalone and simultaneous training using different trainers and sharing memory modes.

A significant portion of the training time reduction is due to the fact that in both sequential and simultaneous parallel training, the interaction with the environments (which are created one-time) is performed only once for all involved agents.

Figure 4.14 shows the relative GPU consumption for standalone and simultaneous training using different trainers and sharing memory modalities.

According to the plot, the GPU usage is nearly the same for both standalone and simultaneous sequential training with shared memory. Since shared memory training is performed on the same number of parallel environments as standalone training, its replay buffer (and thus GPU) usage is the same. The small difference is due to the presence of the models (the approximation functions) required by each agent.

In the case of simultaneous sequential and parallel training without shared memory, the GPU consumption of the library components, excluding the consumption of the simulator and PyTorch CUDA kernels[4], is at least 3 times higher than that of standalone training. For parallel training, the PyTorch CUDA kernels are loaded in each process, which significantly increases the total GPU consumption.

[4] PyTorch compiled and optimized code designed specifically to run on NVIDIA GPUs using CUDA C language.

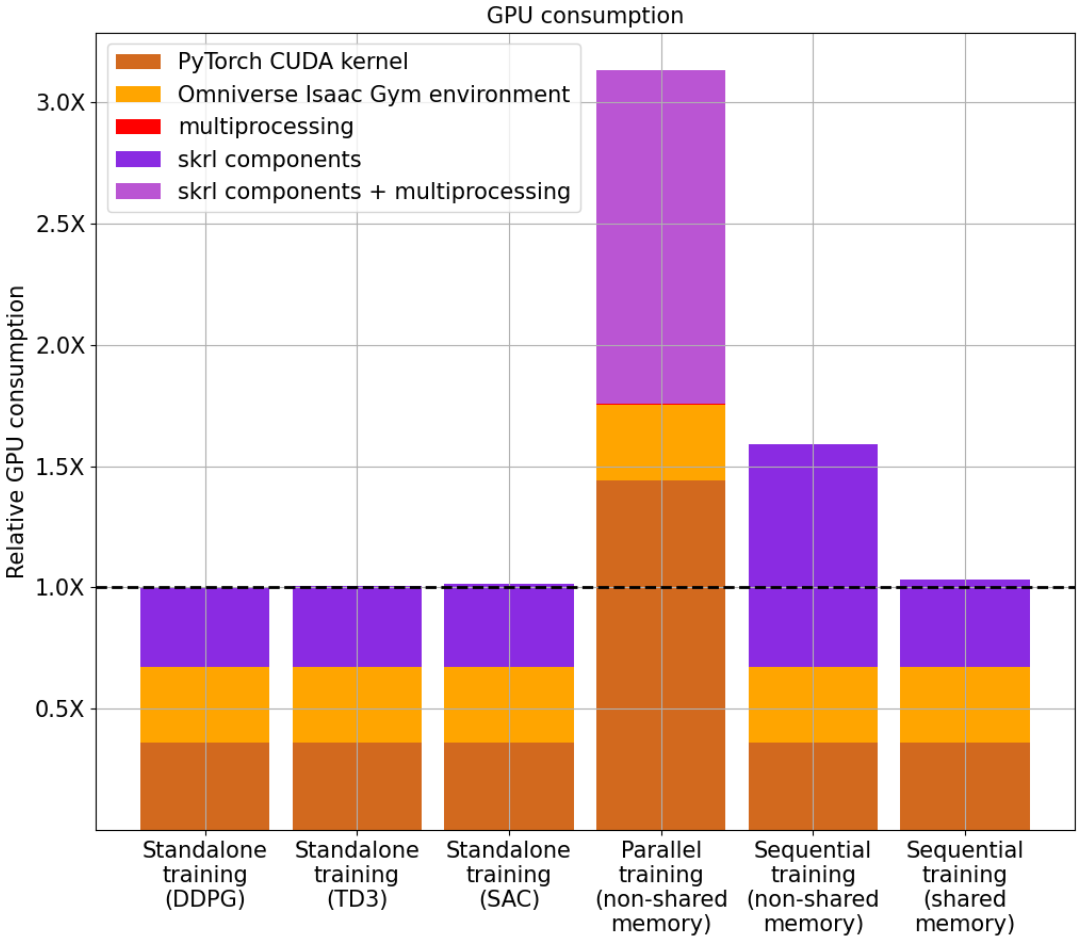

FIGURE 4.14: Relative GPU consumption for standalone and simultaneous training with different sharing memory modalities.

4.4 Chapter conclusions

skrl is a RL library that allows researchers to to compose their experiments using a modular API. Its development has focused on the readability, simplicity, and transparency of the algorithm implementations. In addition to supporting the traditional environment interfaces such as OpenAI Gym / Farama Gymnasium and DeepMind, it also allows loading and training on NVIDIA Isaac Gym, Isaac Orbit and Omniverse Isaac Gym environments. In the latter environments, it enables simultaneous training of agents by scopes (subsets of environments among all available environments) that may or may not share resources in the same run.

In particular, this last feature allows off-policy RL algorithms that share replay memory to reduce their training time and computational resource consumption (in terms of GPU memory) while increasing the mean perceived reward.

This feature answers research question 2 (a): By simultaneously training off-policy RL algorithms it is possible to reduce the training time of all agents by about 25%. In addition, the use of shared memory can reduce GPU consumption for simultaneous training by up to 35% and 70%, respectively, compared to sequential and parallel training without shared memory.

Chapter 5

Reducing Contact Forces and Jamming States using Force Overlay in Disassembly Tasks

The content of this chapter addresses research question 2 b): How can the capabilities of a collaborative robot for disassembly tasks be leveraged for reducing contact forces and jamming states?

The work presented in this chapter is partially published in the paper: Antonio Serrano-Muñoz, Íñigo Elguea-Aguinaco, Dimitris Chrysostomou, Simon Bøgh, Nestor Arana-Arexolaleiba. "A Scalable and Unified Multi-Control Framework for KUKA LBR iiwa Collaborative Robots". In: 2023 IEEE / SICE International Symposium on System Integration (SII). *IEEE. 2023, pp. 1–5.* URL: https://ieeexplore.ieee.org/abstract/document/10039308.

KUKA LBR iiwa robots are among the most popular robotic manipulators and are widely recognized for their capabilities and reliability in collaborative robotics and safety applications thanks to their advanced control and sensor technology.

One of the distinctive capabilities of such manipulators, which has been used in contact-rich manipulation and even in disassembly research, is the generation of oscillating motions (via force overlays) to help redistribute the forces resulting from contact between the manipulated parts and avoid jamming. However, it is unknown to what extent the different overlays supported by the KUKA LBR iiwa, and with different parameterizations (such as oscillation amplitude and frequency), can help to reduce the forces resulting from the contact of the manipulated parts, especially in disassembly.

This chapter describes the empirical research conducted to determine the extent to which the use of force overlay to generate oscillatory motion under different parameterizations helps to reduce contact forces and the occurrence of jamming during the execution of a disassembly task. It also describes a new control framework for KUKA LBR iiwa robots that allows accessing and parameterizing the force overlap mode of such manipulators among other desirable capabilities for RL and disassembly tasks.

5.1 Contact force redistribution via oscillating motions

Oscillating motions can help to redistribute the forces resulting from the friction caused by the contact of the manipulated parts as well as from the occurrence of jamming.

Some authors have applied oscillating motions in their works. For example, the Lissajous curve overlay has been used in contact-rich manipulation tasks such as cleaning on non-flat surfaces [146], filleting geometric profiles [147] or searching the nut hole for screwing tasks [148]. Also, the spiral overlay has been used to detect a hollow for insertion [149] as well as for the assembly of wooden components [150].

In disassembly, the sinusoidal overlay was used for evaluating strategies to separate a pin along a door chain [74], as discussed in Chapter 3. For this, the authors used superposition only in one of the motion axes and with a single parametrization.

Although oscillating movements have been used in the published literature, it is not known to what extent the available overlays (with different parameterizations) can help to reduce the forces resulting from friction caused by the contact of the manipulated parts, particularly in disassembly.

This empirical research investigates the use of the KUKA LBR iiwa Cartesian impedance controller with overlaid force oscillation to reduce contact forces and the occurrence of jamming during the execution of a disassembly task. For this purpose, the different types of force overlays (sinusoidal, Lissajous, and spiral) under different parameters are evaluated in a disassembly task.

To control and access the capabilities of KUKA LBR iiwa, there are some implementations using different workflows (such as Python, MATLAB, ROS and ROS2; they in an standalone implementation). However, only few of them allow the use of some of the available force overlay types (and under limited parameterization) required to conduct the study about contact force reduction. Therefore, it is necessary to have an implementation able to access, to a greater extent, the capabilities of these cobots.

The following section describes the implemented framework. In the subsequent section, and with an understanding of the accessible capabilities, the experiments and results are presented.

5.2 Scalable and unified multi-control framework for KUKA LBR iiwa cobots

This section describes *libiiwa*, a control framework for the KUKA LBR iiwa collaborative robots. The design of this framework responds to the needs of our lab, since this robot is used in our work. As an extra, the framework unifies and enables control and communication through ROS and ROS2 [151], but also provides direct control for those applications where minimal control frequency is required through a scalable, simple, and well-documented API.

5.2.1 Framework architecture

The Figure 5.1 shows the architecture of the implemented framework. It is divided into three main blocks (where each block represents a physical device) as described below, from the bottom to the top block.

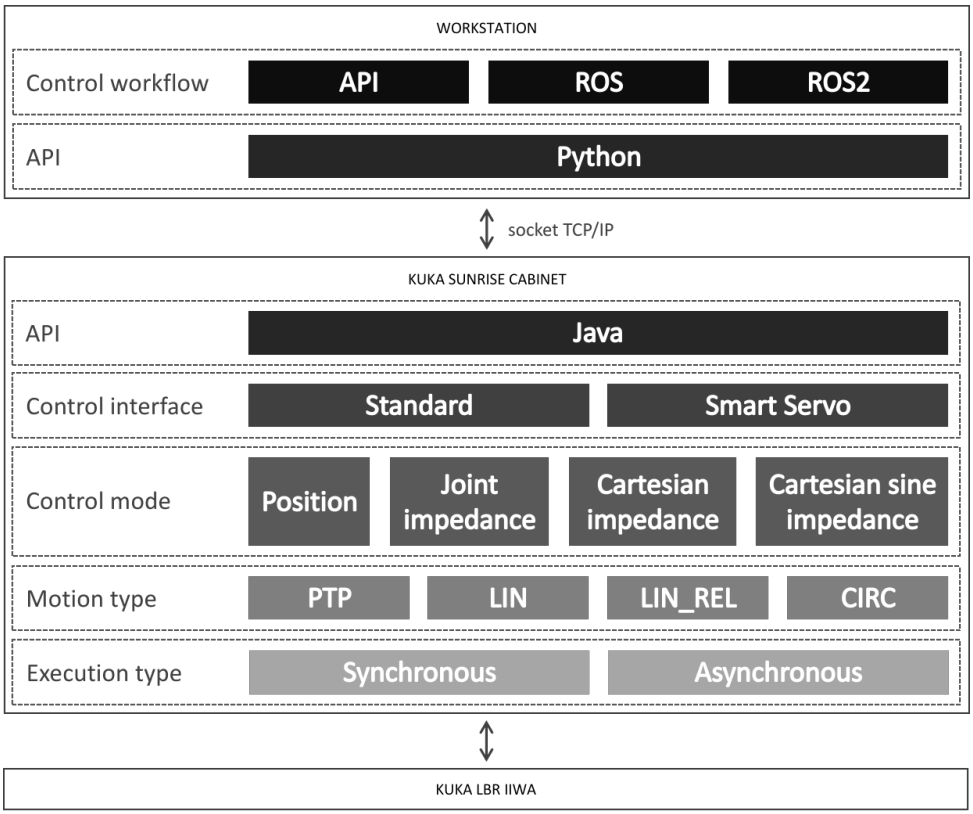

FIGURE 5.1: Framework architecture divided into nested blocks according to the devices involved and the robot features and capabilities, as well as the APIs and control workflows provided.

5.2.1.1 KUKA LBR iiwa

The KUKA LBR iiwa[1] is a seven-degree-of-freedom serial manipulator with force and torque sensors along each axis, as illustrated in Figure 5.2. This sensor distribution allows both position and impedance control, providing compliant behavior in force-sensitive tasks.

5.2.1.2 KUKA Sunrise Cabinet

KUKA Sunrise Cabinet is the controller workstation of the robotic manipulator, which has specific software, hardware, and interfaces to control it. KUKA Sunrise.OS is the cabinet's operating system, which provides toolboxes and libraries programmed in JAVA to read and modify the robot's state. The robot's control system or the KUKA Smartpad is the only source of access to any sensory data or information relevant to the current task.

[1]LBR iiwa: www.kuka.com/en-de/products/robot-systems/industrial-robots/lbr-iiwa

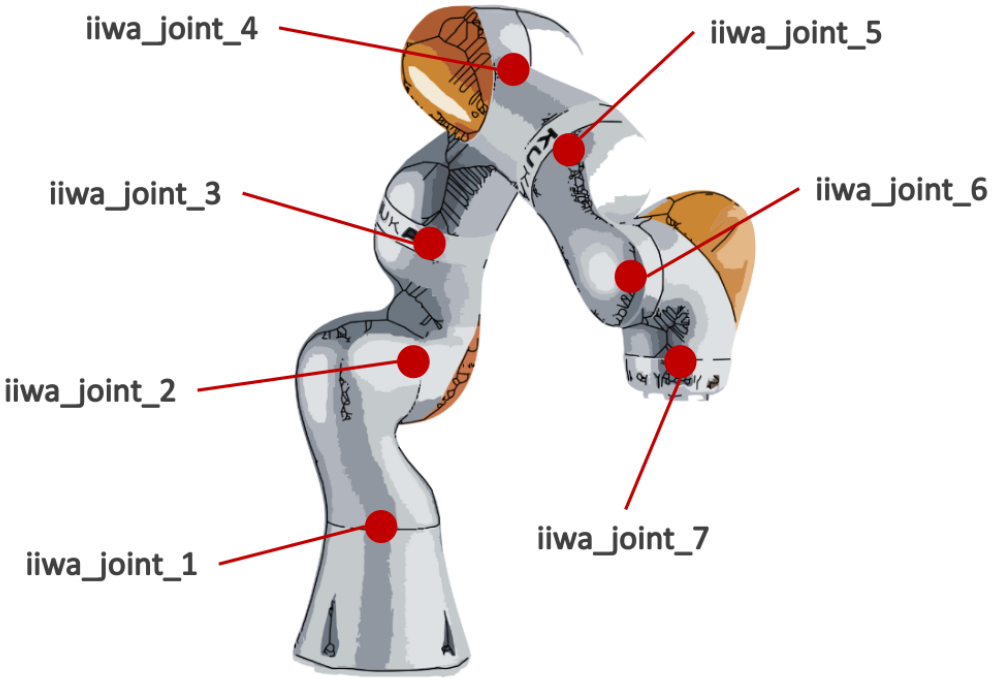

FIGURE 5.2: KUKA LBR iiwa robot and representation of its joints.

The robot can be programmed through both software and hardware interfaces. Software interfaces include the Standard Interface and the Servo Interface, while hardware interfaces include the Fast Response Interface (FRI), as described below:

- Standard Interface: A software interface that allows viewing the current state of the robot, setting parameters, and programming discontinuous blocking or non-blocking motion sequences.
- Servo (or serving) Interface: A real-time software interface that allows the implementation of non-deterministic applications and continuous and smooth motion composed of multiple points. This interface requires the presence of some KUKA-specific software libraries in the cabinet.
- Fast Response Interface (FRI): A hardware interface that allows direct, low-level, real-time access to the KUKA robot controller from external computers at high rates of up to 1 kHz. This interface requires KUKA libraries written in C++ and Java. This interface requires the presence of some KUKA-specific hardware in the cabinet.

Since our lab setup does not have the FRI hardware interface, the framework is developed on the Standard and Servo software interfaces.

To access most of the specific features and capabilities of the LBR iiwa, a subcomponent has been programmed in JAVA language that defines an interface using the KUKA Sunrise.OS libraries: *libiiwa - JAVA API*. The API allows communication and control from external stations via TCP/IP protocol. It can also be used to quickly develop applications that can be executed from the cabinet itself.

The following robot features and capabilities can be configured and used:

Execution type

Execution type defines how the movements of the robot will be executed by the internal controller. The types supported by the implemented API are:

- Synchronous: The synchronous type executes a motion command after the current motion has been completed, blocking the program flow.
- Asynchronous: The asynchronous type executes a motion command without interrupting the program.

Motion type

Motion type defines the types of movements to be performed by the robot to go from one pose to another. The types supported by the implemented API are:

- Point-to-point (PTP): Executes a point-to-point motion to the endpoint.
- Linear (LIN): Executes a straight-line motion to the endpoint.
- Linear relative to its current position (LIN_REL): Executes a linear motion relative to the previous position.
- Circular (CIRC): Executes a circular motion defined by an auxiliary position and an end position.

Control mode

Control mode defines which controller can be used to operate the robot. The API allows operating in the following modes:

- Position: Execute the specified motion with the highest possible positional accuracy and without path deviation.
- Joint impedance: Virtual spring damper system with configurable stiffness and damping values for each joint.
- Cartesian impedance: Virtual spring damper system with configurable stiffness and damping values for Cartesian space. This allows the robot to respond to external forces in a compliant manner.
- Cartesian sine impedance: Special form of the Cartesian impedance controller that allows overlaying force values on the motion. Available force overlays are:
  - Constant force: overlay a constant force in one Cartesian direction, that does not change over time.
  - Sine: overlay a simple force oscillation in one Cartesian direction.
  - Lissajous: overlay a 2-dimensional oscillation in one plane that generates a Lissajous curve.
  - Spiral: overlay a spiral-shaped force oscillation in one plane.

Control interface

Control interface defines which interface will be used to operate the robot controllers. The API allows selecting between the two software interfaces:

- Standard: A software interface that allows viewing the current state of the robot, setting parameters, and programming discontinuous blocking or non-blocking motion sequences.
- Servo: A real-time software interface that allows the implementation of non-deterministic applications and continuous and smooth motion composed of multiple points.

Conditions

Conditions can be used to monitor the robot's control and trigger specific reactions (such as stopping a current motion) when definable limits are exceeded or not reached. The following conditions can be defined:

- Cartesian force: For each Cartesian axis, define the force condition at which the robot must stop its motion.
- Joint torque: Define the axis/joint torque condition (lower and upper limits) for the specified axis at which the robot must stop its motion.

The *libiiwa - JAVA API* also defines a communication protocol (described in Subsection 5.2.2.1) over TCP/IP to allow control and reading of the robot status from external workstations. Other APIs and functionalities are implemented on top of this protocol.

5.2.1.3 Workstation

An external computer that acts as an access interface to the operator, to other external control programs such as learned or designed control policies, or to ROS/ROS2 environments, for example. External access or control workflows are based on a library component programmed in Python, (*libiiwa - Python API*), which communicates via TCP/IP with the mentioned pair (*libiiwa - JAVA API*).

5.2.2 Framework API

5.2.2.1 Communication protocol

The communication protocol designed and implemented allows to control the robot from external workstations. Its structure is designed as a number sequence of floating (32 bits, default configuration) or double (64 bits) precision, which can be configured in both, the Java and Python API. This structure simplifies the segmentation and use of the transmitted information, extending the control to other programming languages such as C/C++, MATLAB, JavaScript, and others. The communication

protocol byte order, size, and alignment uses the network byte order (which is always big-endian as defined in IETF RFC 1700[2])

The communication has to be initiated from the external control workstation which sends a command request and receives the robot state as response as shown in Figure 5.3.

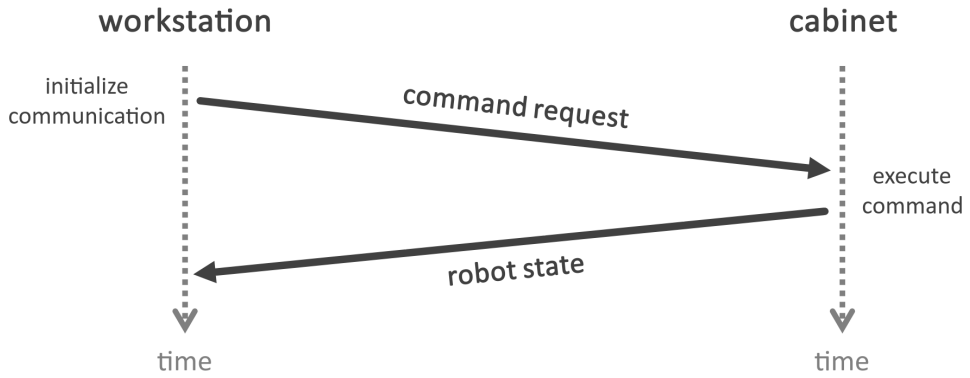

FIGURE 5.3: Communication protocol flow between the external workstation (from which the request must be initiated) and the cabinet (which returns the robot state as a response).

Command (request)

The command allows to control the robot movement, either in Cartesian or joint action space, to configure the operating modes, set limits for velocities, accelerations, and jerks and define stop conditions. Its structure, shown in Figure 5.4, is a sequence of 8 numerical values, where the first value is the code of the command to be executed (interpreted as an integer value), while the other values are the data that the command may or may not require. The maximum number of data values corresponds to the number of joints of the robot: 7 in the case of the LBR iiwa.

Refer to the communication protocol description in the framework documentation (`https://libiiwa.readthedocs.io/en/latest/intro/protocol.html`) for more details about the command codes and their associated data.

Robot state (response)

The state of the robot received as a response is a sequence of 38 numeric values. The distribution of these values corresponds to the following information: command status (1), joint positions (7), joint velocities (7), joint torques (7), Cartesian position (3), Cartesian orientation (3), Cartesian forces (3), Cartesian torques (3), last error (1), has fired condition (1), is ready to move (1), and has active motion (1).

Refer to the communication protocol description in the framework documentation (`https://libiiwa.readthedocs.io/en/latest/intro/protocol.html`) for more details about the robot state and the data type of its components.

[2]IETF RFC 1700: `https://www.rfc-editor.org/rfc/rfc1700`

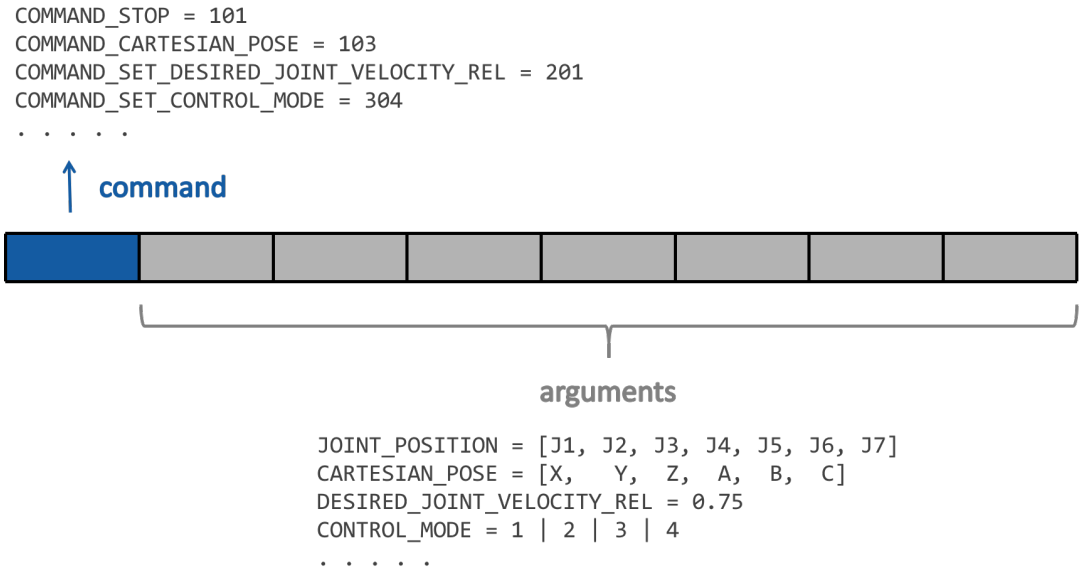

FIGURE 5.4: Structure of the communication protocol request command. The request is composed of the command code to be executed and its respective parameters.

5.2.2.2 Control workflow

This section describes the various control workflows that have been implemented in this framework up to the time of this writing.

Python API

The Python API allows direct control of the robot from a simple interface, by instantiating a class. This API allows rapid prototyping and solution creation, given the nature of the programming language, especially in the field of ML, where Python is the most popular language.

Since the Python module does not require any special dependencies except for NumPy, it is possible to use it from any operating system, not just Linux.

ROS & ROS2 API

On top of the Python API, a node has been developed for both ROS and ROS2 that allows the framework (and thus the manipulator) to be integrated into a ROS environment. Therefore, this implementation has the advantage of being easy to debug and extend.

The ROS nodes for both versions implement the entire Python API as follows: subscription topics for manipulating the robot in Cartesian and joint control space, and publication topics for exporting sensor information such as joint position, velocity, and torque, the Cartesian pose of the end-effector, and the force and torque measured in that frame. The messages defined in the standard, geometry, and sensor packages are used for topics.

It also implements ROS services to configure and use the features and capabilities described above. For the services, specific messages were defined, since the message definition of the standard support is limited.

MoveIt integration

MoveIt is the most widely used and popular motion planning and manipulation software in the robotics community [152]. By default, it uses the ROS Control *FollowJointTrajectory*[3] action service as a low-level controller for controlling the manipulator [153].

The ROS nodes of the framework implement this action service, which makes it possible to use MoveIt to control the LBR iiwa as showcased in Figure 5.5. For both versions, the topic names, execution frequency, and other parameters can be modified from the *roslaunch* files provided.

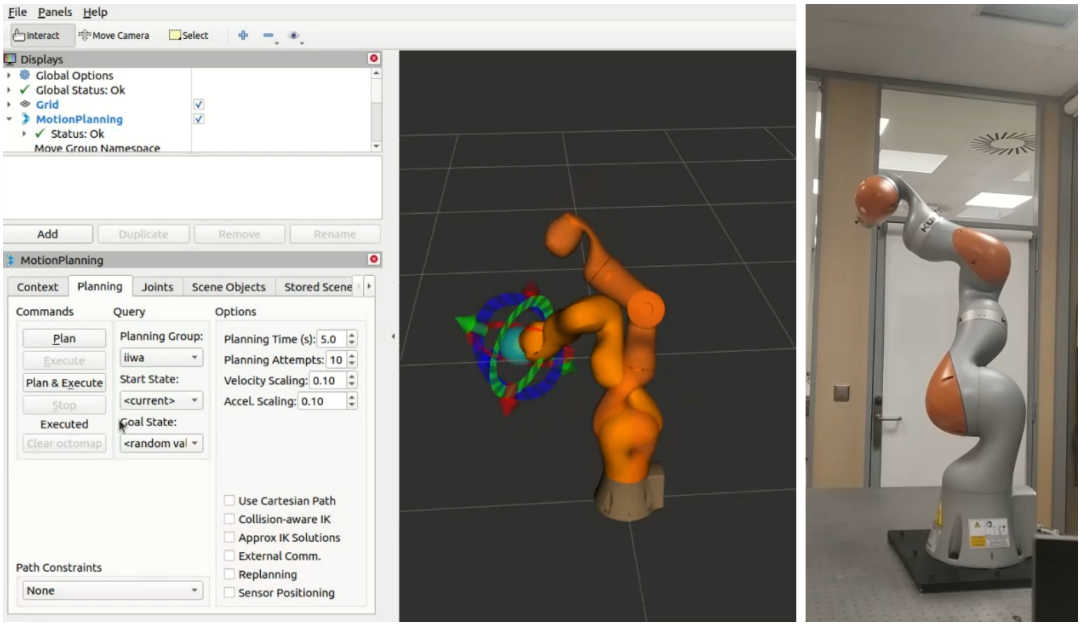

FIGURE 5.5: MoveIt integration with a real KUKA LBR iiwa robot.

FollowJointTrajectory defines a sequence of joint configurations (trajectory) to be followed by the controller. The implementation allows specifying the behavior of the trajectory execution in the following 2 manners:

- Follow the whole trajectory:

  The execution of the next joint configuration in the sequence of trajectories parameterized in the *FollowJointTrajectory* action service is updated according to Equation 5.1:

$$\sum_{j}^{J=7} ||q_j - q_{j_{trajectory}}|| <= \epsilon \tag{5.1}$$

[3]FollowJointTrajectory: https://docs.ros.org/en/noetic/api/control_msgs/html/action/FollowJointTrajectory.html

Where q is the robot joint positions and ϵ is the trajectory update threshold.

The execution is sensitive to different values of speed, acceleration, and jerk. The ϵ parameter must be adjusted in cases where the trajectory runs intermittently or there are missing joint configurations.

- Go to last trajectory joints configuration: The controller ignores the middle joint configuration and acts to position the robot at the last joint configuration parameterized by the *FollowJointTrajectory* action service.

5.2.2.3 Documentation

The documentation is written using reStructuredText[4] and hosted online by Read the Docs under the URL `https://libiiwa.readthedocs.io` as shown in Figure 5.6.

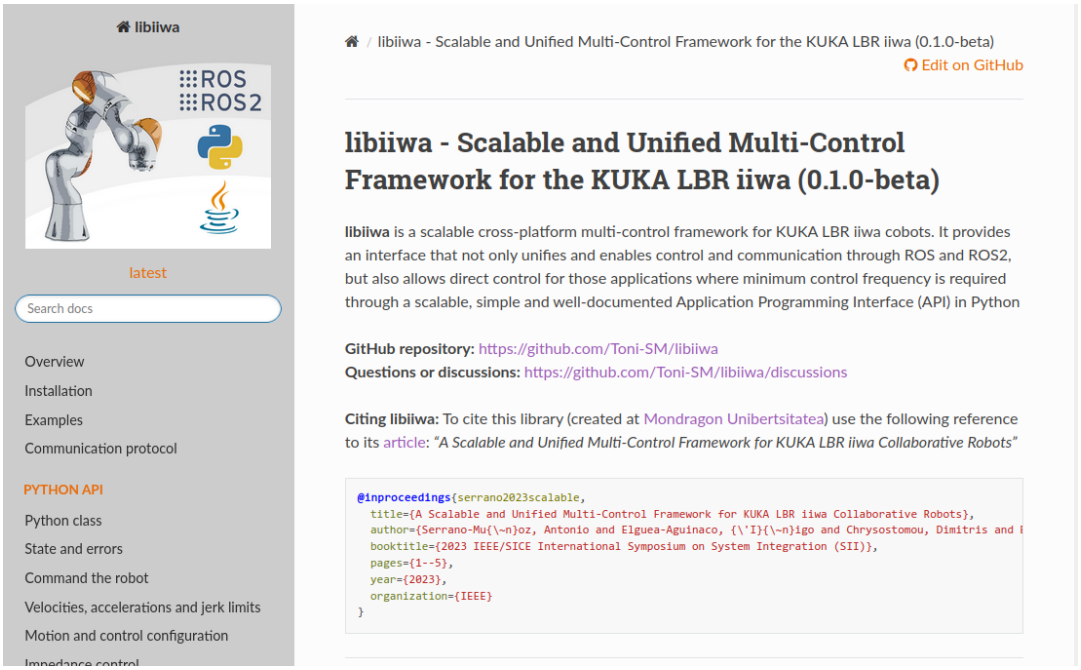

FIGURE 5.6: Screenshot of the libiiwa documentation home page

Apart from the library installation steps and API details (classes, functions, parameters, return values, etc.), snippets are also included for the Python, ROS and ROS2 APIs. For the last two, the code snippets are extended to call topics and services from the terminal and from Python code using the *rospy* and *rclpy* libraries for both versions.

Examples of use cases are included with their respective scripts or ROS packages.

5.2.3 Comparison with existing implementations

Table 5.1 presents a comparison between the proposed framework and other existing implementations in terms of the capabilities and features of KUKA LBR iiwa robots, as well as the control workflows described previously.

[4] reStructuredText is a plaintext markup language: `https://docutils.sourceforge.io/rst.html`

The table shows that the proposed framework covers most of the features and capabilities of the robot, providing the potential for the development of a wide range of robotic applications at different levels of complexity. This opens up new possibilities for the exploitation of the robot's capabilities in various domains and tasks.

Feature	**Specification**	**Frameworks/libraries**							
		libiiwa	**iiwa_stack [154]**	**KUKA-IIWA-API [155]**	**iiwa_ros [156]**	**iiwa_ros2 [157]**	**lbr_fri_ros2_stack [158]**	**KUKA Toolbox (Matlab) [159]**	**iiwaPy/iiwaPy3 [160]**
Execution type	Synchronous	X						X	X
	Asynchronous	X	X	X				X	X
Motion type	PTP	X	X	X				X	X
	LIN	X	X	X				X	X
	LIN_REL	X		X				X	X
	CIRC	X	X	X				X	X
	SPLINE		X						
Control mode	Position	X	X	X				X	X
	Joint impedance	X	X						
	Cartesian impedance	X	X	X				X	X
	Cartesian sine impedance	X	/						
Control interface	Standard	X		X				X	X
	Servo	X	X					X	X
	FRI				X	X	X		
Condition	Force	X							
	Torque	X							
Control workflow	Direct	X						X	X
	ROS	X	X	X	X				
	ROS2	X				X	X		

TABLE 5.1: Comparison of libiiwa with other related frameworks/libraries. Shaded cells do not apply to the marked fields.

5.2.3.1 Relevant features for the execution of disassembly tasks using reinforcement learning

From the list of capabilities and features of KUKA LBR iiwa robots, some are particular relevant or desirable for carrying out disassembly tasks using RL. Among these features are:

- Cartesian sine impedance control mode: Oscillating motion can redistribute the forces resulting from friction between the contact points of the parts to be

disassembled due to variations in pressure and sliding direction. This redistribution helps to prevent localized areas from experiencing excessive friction and to reduce the likelihood of jamming [74][161].

While *iiwa_stack* allows configuring only the overlay of a single-axis sine oscillatory motion, *libiiwa* offers access to all types of motion overlays and other configurations particular of this control mode.

- Force and torque conditions: Disassembly tasks often involve interactions with objects that can cause excessive forces and torques that could lead to damage or injury [162]. Defining conditions that internally monitor the robot control is relevant to ensure the safety of the operator, the environment and the robotic manipulator itself.

  Compared with other libraries, *libiiwa* enables the configuration of internal monitoring operations of the robot control and the triggering of specific reactions for force and torque limits. In addition, it allows setting maximum permissible values for impedance control such as control force, velocity and Cartesian deviation.

- Control workflows: Python is the widest used programming language for research and application development in RL, thanks to the existence of libraries and frameworks that provide high-level functionality for both implementing RL interfaces and codifying ML algorithms [163][164].

  Besides providing the most complete API in Python, *libiiwa* also allows control through both ROS and ROS2, making it suitable for developing a wide variety of robotic applications, including RL.

5.3 Reducing contact forces and jamming states using Cartesian impedance control with overlaid force oscillation

This section experimentally investigates the use of the KUKA LBR iiwa Cartesian impedance controller with overlaid force oscillation to reduce contact forces and the occurrence of jamming (measured as the change in the time required to complete the motion) during the execution of a disassembly task. For this purpose, the different types of force overlays (sinusoidal, Lissajous, and spiral), under different amplitudes and frequencies of oscillation, are evaluated in a disassembly task.

5.3.1 Experimental setup

The experimental setup consists of two elements: a fixed slotted base and an object (that fits inside the slot) attached to the manipulator end-effector as shown in Figure 5.7. The fixed slotted base is placed in the direction of the Y-axis of the manipulator. The gap between the base and the object is 0.25 millimeters.

For each experiment, the robot end-effector is controlled at a frequency of 15Hz to go from the initial position to the final position (for a total displacement of 30 centimeters). Although the action (a) move the end-effector in the Y-axis of the manipulator, a random uniform noise (U) is added in the X and Z-axis to intensify jamming and friction states according to Equation 5.2.

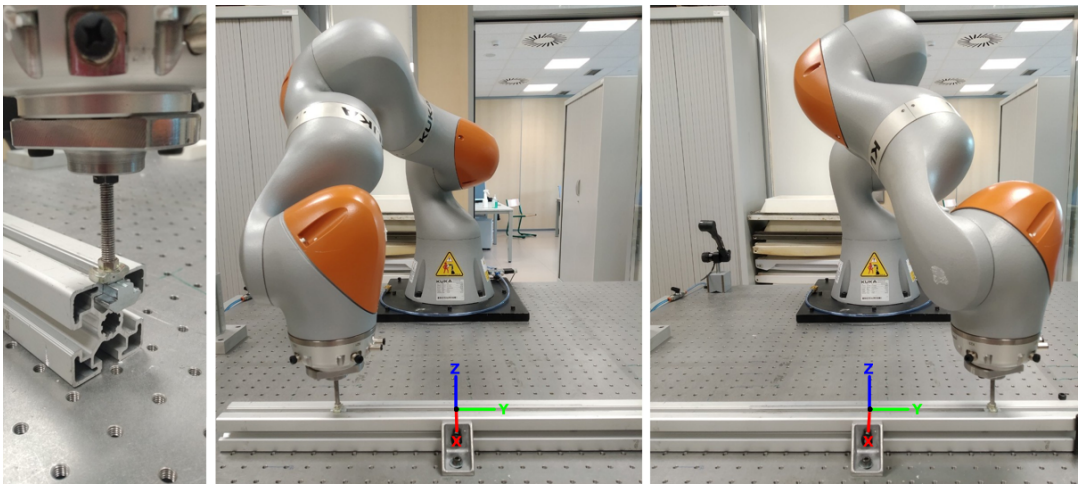

FIGURE 5.7: Experimental setup: Fixed slotted base and object attached to the manipulator end-effector (left). Initial position (center) and final position of the experiment (right).

$$a = [\ \Delta x,\ \Delta y,\ \Delta z\ ] = [\ U(-1,1)\,cm,\ 1\,cm,\ U(-1,1)\,cm\ ] \tag{5.2}$$

For each force overlay type, parameterized by frequency (2.5, 5.0, 7.5, 10.0, 12.5 and 15.0 Hz), amplitude (1.0, 2.0, 3.0, 4.0 and 5.0 Newtons), axis of application (X, Y and Z for sine type), and plane of application (XY, XZ, YZ for Lissajous and spiral types), 5 experiments were performed. In addition and as a baseline, 5 experiments were performed without any force overlay. Five different seeds are set for each of the 5 experiments and common for the different parameterizations. A threshold of 30 Newtons was established to analyze the contact forces involved. Cartesian stiffness and damping were set to 3500 N and 250 Nm respectively.

5.3.2 Results

Figure 5.8 shows the mean and standard deviation of the end-effector's Cartesian force vector magnitude exceeding 30 Newtons (measured every 15 Hz) for each type of force overlay (parameterized by frequency, amplitude and the axis or plane of actuation). For this, and the next plots, the values for the baseline execution (no-overlay) is added for comparison.

According to the plot, practically all tested overlay types (in their different parameterizations) average below non-overlay measurements. Only for punctual frequency and amplitude parameterization pairs (e.g.: (2.5 Hz, 1.0 N), (15 Hz, 2 N) or (15 Hz, 3 N)), and for the YZ-plane, the force measurements generated by overlaying Lissajous curves and spiral shapes exceed, up to 1.3 Newtons, the non-overlaid measurements.

The overlay types that on average have lower contact forces, up to 10% compared to no-overlay, are Lissajous curves and spiral shapes in the XZ-plane and sinusoidal oscillation in the X-axis, in that order. It is relevant to note that both the X-axis and the XZ-plane are perpendicular to the direction of extraction.

Similarly, Figure 5.9 plots the number of such occurrences during the execution of the experiments. In terms of the amount of contact force exceeding the specified

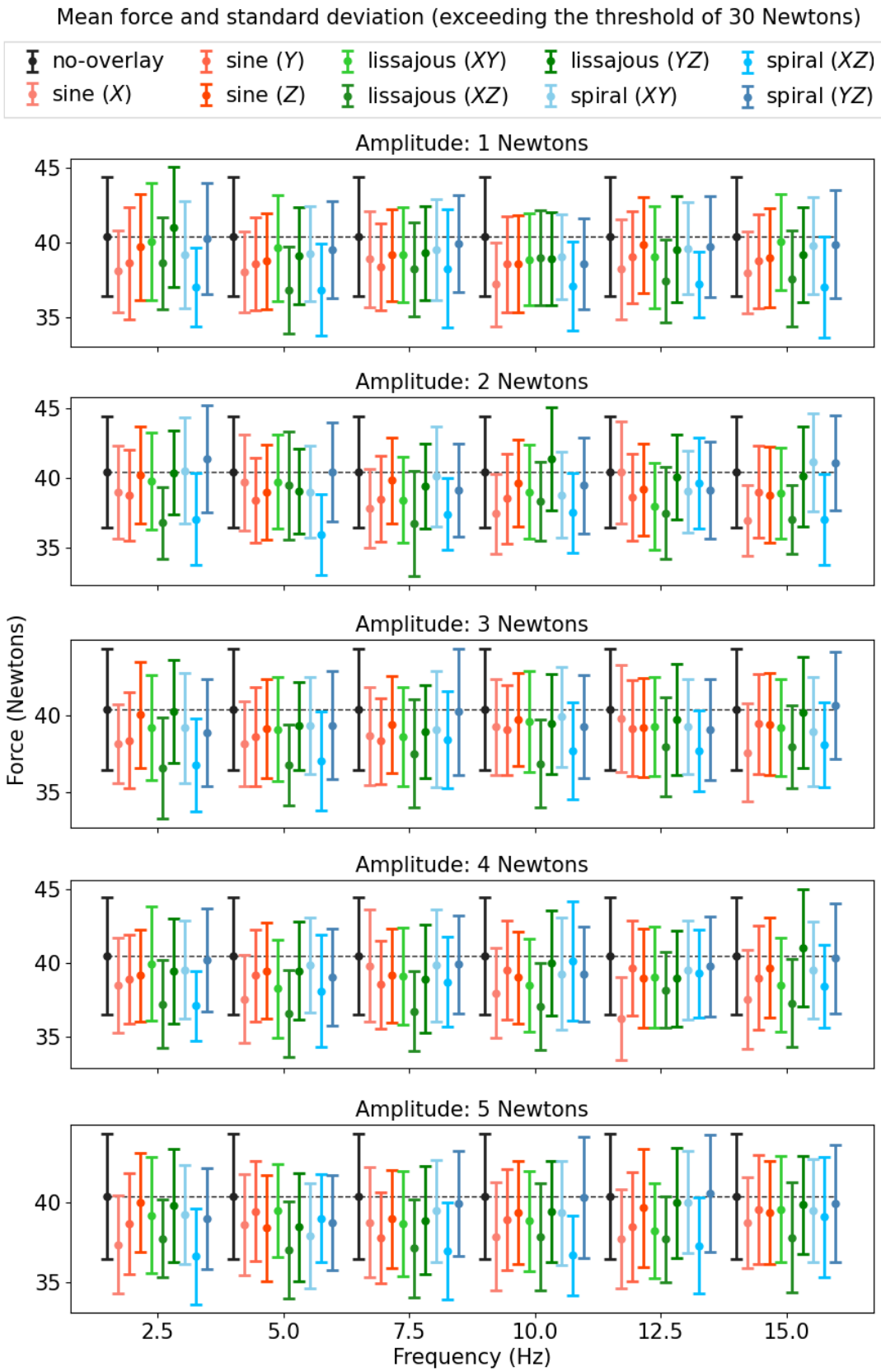

FIGURE 5.8: Mean and standard deviation of the end-effector's Cartesian force vector magnitude exceeding 30 Newtons for each force overlay type under different parameters.

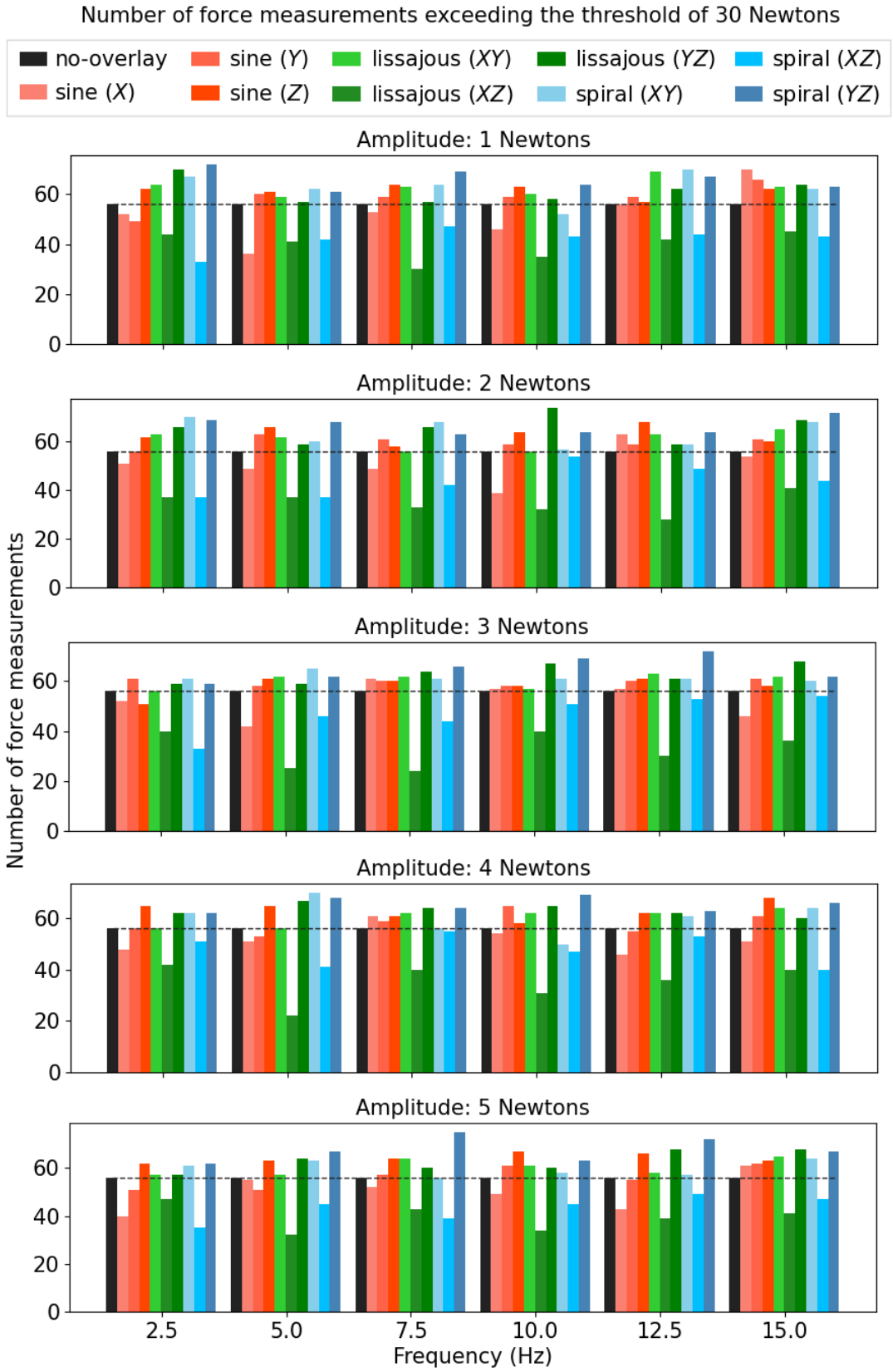

FIGURE 5.9: Number of end-effector's Cartesian force vector magnitude exceeding 30 Newtons for each force overlay type under different parameters.

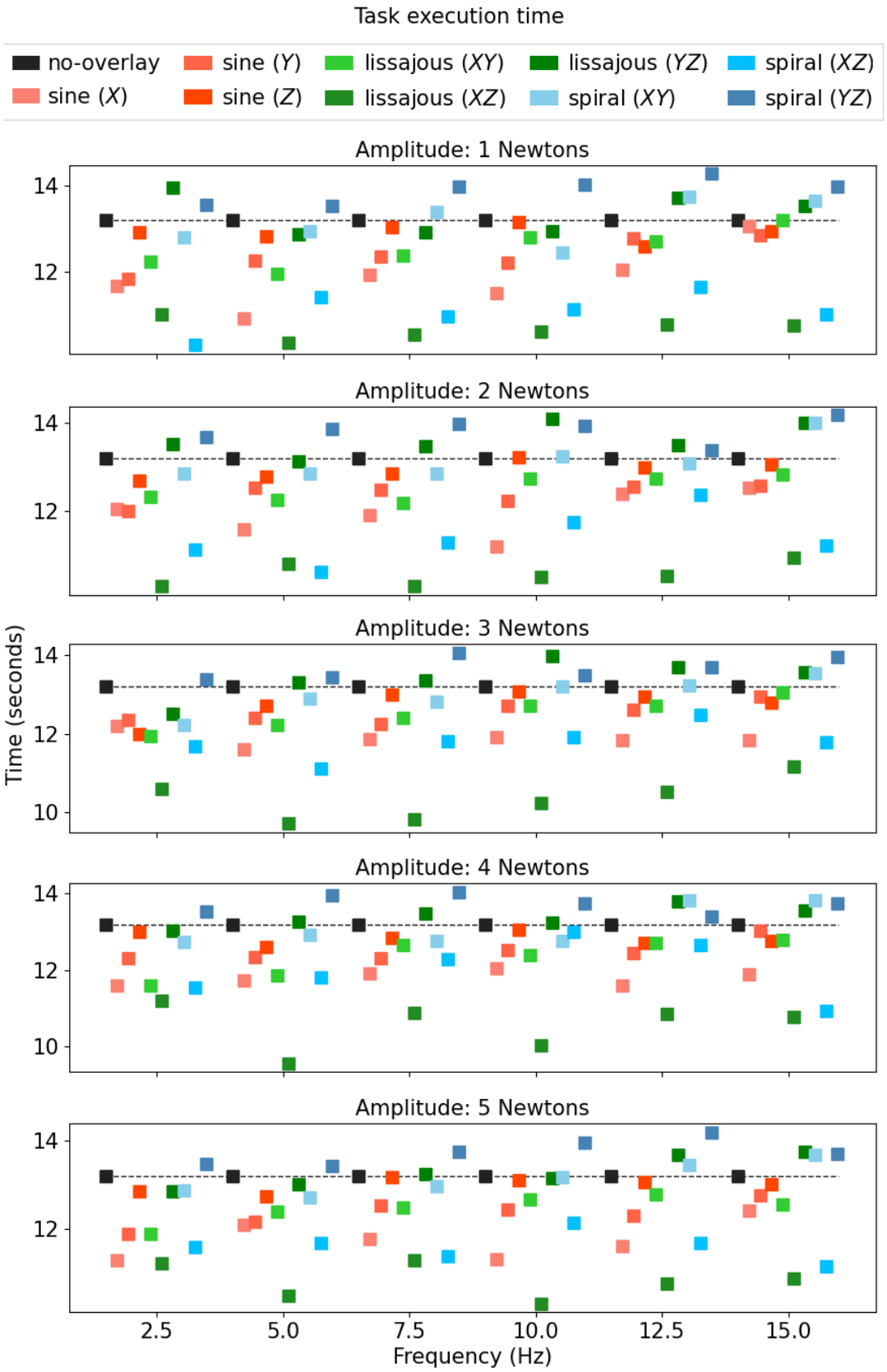

FIGURE 5.10: Task execution time for each force overlay type under different parameters.

threshold, Lissajous curves and spiral shapes in the XZ-plane and sinusoidal oscillation in the X-axis offer the most significant reductions. These reductions are up to 55% for the Lissajous curves, up to 45% for the spiral shapes, and up to 32% for the sinusoidal oscillations compared to non-overlay measurements.

Although there are some cases in which the occurrences exceed the baseline as much as 27.5% (e.g. (12.5 Hz, 3 N)), their magnitudes are still less than the latter.

Figure 5.10 depicts the mean time taken to perform the disassembly task for each set of experiments. The plotted data is used indirectly to analyse the probability of jamming states (the greater the number of jams, the greater the time required to complete the extraction).

Regarding the duration of task execution, Lissajous curves and spiral shapes in the XZ-plane and sinusoidal oscillations in the X-axis again exhibit the most significant reductions. For Lissajous curves, spiral shapes, and sinusoidal oscillations in the XZ-plane and on the X-axis, execution times are reduced by up to 28%, 22%, and 18%, respectively, compared to non-overlay execution. Such time reductions can be associated with a decrease in the probability of reaching jamming states.

5.4 Chapter conclusions

The research carried out answers research question 2 (b): Using the force overlay types under different parameters reduces the contact forces caused by friction and the probability of jamming states (measured as the change in the time required to complete the motion) in the execution of disassembly tasks. In particular, Lissajous curves and spiral shapes, applied in the plane perpendicular to the extraction direction, yield the most effective results. The results show reductions of up to 10% and 55% in the mean force magnitude and occurrence of contact force exceeding the established threshold, respectively, and a 28% reduction in task completion time (and thus the mitigation of jamming states) compared to non-overlay execution.

For KUKA LBR iiwa manipulator robots, *libiiwa*, a control framework is presented with different workflows, including both, ROS and ROS2 packages for software interoperability in robotics, and direct control for RL applications. The proposed interface outperforms current approaches in terms of functionality, accessibility to robot capabilities, and integration. This framework can be useful to the robotics community for developing research and applications using such manipulators, thus taking advantage of the functionalities offered by this robot.

Chapter 6

Reinforcement Learning for Generalization of Collaborative Disassembly Tasks with High Variability

The content of this chapter addresses, in part, research question 1: To what extent can RL algorithms generalize the execution of disassembly tasks using collaborative robots?

The work presented in this chapter is partially derived from the work carried out for the paper: Íñigo Elguea-Aguinaco, Antonio Serrano-Muñoz, Dimitrios Chrysostomou, Ibai Inziarte-Hidalgo, Simon Bøgh, and Nestor Arana-Arexolaleiba. "Goal-Conditioned Reinforcement Learning within a Human-Robot Disassembly Environment". In: Applied Sciences 12.22 (2022), p. 11610. URL: https://www.mdpi.com/2076-3417/12/22/11610.

The research conducted in Chapter 3 revealed how some RL algorithms can learn and generalize object extraction tasks to various initial conditions, such as positions and rotations.

To perform disassembly tasks, agents must also be able to handle the variability of EOL product states such as different geometries and changes in the physical properties of objects, such as friction, for example. However, it is not known to what extent RL algorithms can generalize disassembly tasks with high degree of variability.

This chapter updates the state-of-the-art of related publications in the domain of RL and disassembly up to the time of writing. Also, it proposes a RL method to generalize disassembly tasks in Cartesian space with high degree of variability in terms of geometric length and spacing between parts, as well as physical friction between them.

6.1 An updated review on disassembly tasks

The following section provides a recent and updated overview of the state-of-the-art of research and application in disassembly tasks up to the time of writing.

6.1.1 Disassembly tasks

Since the previous review discussed in Chapter 3, few works have been published as well in the execution of disassembly tasks. A snapshot of these works is shown in Figure 6.1.

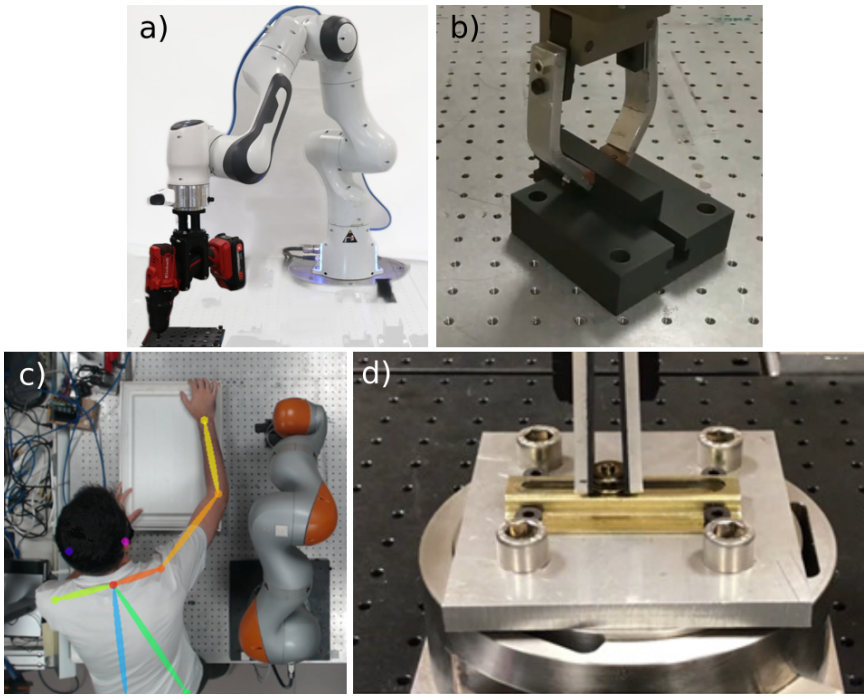

FIGURE 6.1: Snapshot of the publications of a) Hjorth et al. [165], b) Serrano-Muñoz et al. [64], c) Elguea-Aguinaco et al. [68], and d) Qu, Wang, and Pham [166].

Hjorth et al. [165] propose an algorithm to perform energy-aware impedance control using a global power-limited energy reservoir. Such an algorithm is evaluated on an unscrewing task and compared with a standard impedance controller and a hybrid force-impedance controller. Unlike the latter, the proposed implementation guarantees system passivity, safe task execution, and avoids contact loss. A drawback of the presented control algorithm is the need to manually design the parameterization of the energy flow limit, which could limit its generalization to tasks requiring different energy levels.

Serrano-Muñoz et al. [64] conducted (extended in Chapter 3), to their knowledge, the first study on the generalization of object removal skills when applied to contact-rich disassembly tasks. For this purpose, they implemented a disassembly task involving one translational DOF on which two off-policy RL algorithms were trained in simulation. Subsequently, the generalization capabilities were evaluated in both simulation and the real world. Although the results of this initial study show generalization capabilities for different initial positions and rotations, its scope is limited to robot end-effector motions in only 2 DOFs for extraction in the XY-plane, as well as for the geometry described for the test case.

Elguea-Aguinaco et al. [68] extended previous work to present a novel strategy that combines the execution of the same disassembly task with real-time collision avoidance through machine learning for safe human-robot interaction. To achieve this, a reward function was designed that, at each timestep, guides the learning,

and therefore the extraction task, in a direction subject to the location of a human collaborator that avoids colliding with the person. In addition, this work scales the task implementation from a single environment to hundreds of environments using NVIDIA Isaac Gym preview and *skrl*. Although the authors succeed in completing the task conditioned by the presence of the human being, from the point of view of disassembly, they have the same limitations as their previous work. Their scope is limited to extraction movements in the XY-plane and for the same geometry.

Qu, Wang, and Pham [166] recently extended the Herold et al. [74] work (described in Chapter 3) to the field of RL. They conducted a research study using DDPG, and also validated with other off-policy algorithms such as SAC and TD3, to learn how to separate a bolt along a door chain. The disassembly task involves one translational DOF. This work performs an empirical study of the effects of changes in robot precision on the performance of controllers learned by adding uncorrelated Gaussian noise to the robot's actions. In contrast to previous RL works, training is performed in the real world. The work has as drawbacks the reward function, which guides the policy in the same fixed direction corresponding to the manipulator end-effector X-axis and is designed for that particular geometry. Moreover, it uses absolute information that prevents the generalization of the task to different initial configurations.

6.1.1.1 Analysis

An overview of the disassembly tasks presented in the papers of the literature review is shown in Table 6.1. The table summarizes the use or not of RL and the different disassembly tasks.

Ref.	**RL**	**Disassembly task**
Hjorth et al. [165]	No	Unscrewing
Serrano-Muñoz et al. [64]	Yes	Contact-rich extraction
Elguea-Aguinaco et al. [68]	Yes	Contact-rich extraction
Qu, Wang, and Pham [166]	Yes	Contact-rich extraction

TABLE 6.1: A summary of recent articles addressing disassembly tasks up to the time of this writing.

Research on RL for disassembly tasks has been conducted in previously unexplored DOFs. One translational DOF: separating a bolt along a door chain in the x-axis by Qu, Wang, and Pham [166]. Two translational DOFs: extracting a block from a slotted base in the XY plane by Serrano-Muñoz et al. [64] and by Elguea-Aguinaco et al. [68]. However, works with a larger number of DOF has not been reported.

Observation spaces still incorporate absolute information. Furthermore, the tasks start from the same spatial position. Qu, Wang, and Pham [166] use the absolute spatial Cartesian pose of the manipulator end-effector while Elguea-Aguinaco et al. [68] use the same information, but in the XY plane of both the manipulator's end effect and the fixed base. As discussed in Chapter 3, the absolute information prevents generalization of the learning to new spatial positions or to different configurations of the robot manipulator.

The reward functions guide the policy in a specific direction, making it physically impossible to complete tasks in other directions. In Qu, Wang, and Pham [166]'s

work, this function only takes into account one-way advances on the X-axis. In Serrano-Muñoz et al. [64] and Elguea-Aguinaco et al. [68] work, while the reward functions allow movement in different directions, they do not prevent the policy from falling into intermittent movement by exploiting design flaws.

Observations, reward functions, and episode completion conditions are strongly anchored to the specific geometry used during training. Serrano-Muñoz et al. [64] and Elguea-Aguinaco et al. [68] setups only support geometries with a length of 0.1 meters while in Qu, Wang, and Pham [166] it is 0.06 meters. Exploration outside these ranges is not performed during training. The use of RL methods strongly anchored to the geometries used in related publications makes generalization to new geometries almost unfeasible.

The high degree of variability in the state of the EOL products leads to uncertainties in the geometries, physical properties and the way of carrying out the extraction of the objects to be disassembled, for example. In this sense, existing works in the field of RL are limited to specific geometries and have a reward function that does not guarantee generalization in the view of such variability. Furthermore, they are limited to manipulations on one axis or Cartesian planes and the observations still incorporate absolute information.

Consequently, this chapter aims to provide an RL method capable of generalizing the performance of disassembly tasks under high variability.

6.2 Reinforcement learning for disassembly tasks with high variability

This section describes the research carried out to handle the high degree of variability of the states of the objects to be disassembled in terms of geometric length and spacing between the objects in contact, as well as physical friction between them, for the generalization of extraction tasks. In addition, the RL method proposed in this research is extended to all translational Cartesian DOFs to operate in the space rather than on a single axis or a plane.

For this purpose, the extraction of a rigid object that is inside and in contact with a rigid slotted part is proposed as a use case. Both objects have variable geometric dimensions in terms of length and air space between them, as well as physical frictions.

6.2.1 Reinforcement learning experimental setup

The experimental setup consists of a robotic manipulator and an object composed of two rigid bodies, with variable geometric dimensions and physical frictions, in contact with each other. The objective is, with RL algorithms, to learn how to control a robotic manipulator to remove the detached object from the fixed slotted base that can be oriented towards any direction in Cartesian space.

6.2.1.1 Simulated setup

An environment consisting of a manipulator robot (KUKA LBR iiwa) and a fixed base was designed. The robot has a 2x2 cm cube attached to it by a fixed joint and

placed 20 cm from its end-effector in the Z-axis, which fits into the slotted base as shown in Figure 6.2.

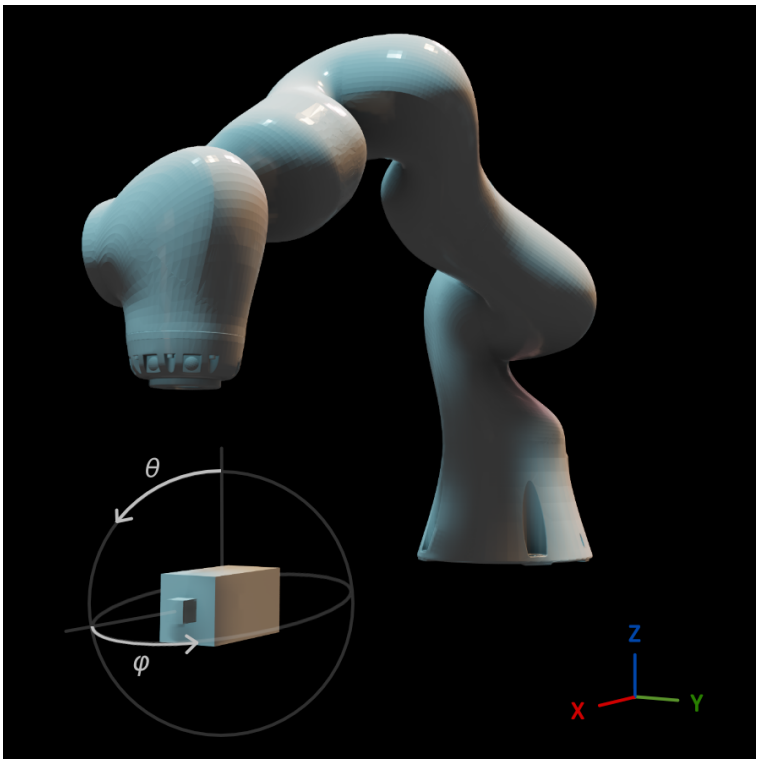

FIGURE 6.2: Experimental setup in the simulation and its reference system. Spherical coordinate notation described by polar (θ) and azimuthal (ϕ) angles is used to define the base orientation.

The combined object (fixed base and cube) is placed 0.65 meters in front of the robot, at the same level as the robot's base. The length of the geometry of the fixed base and the width and height of its slot can be modified. The static and dynamic friction coefficients of both the base and the cube attached to the robot can also be modified to generate high degree of variability.

6.2.2 Reinforcement learning formulation

A MDP with a finite-horizon discounted return was used to frame the problem. The agent will learn how to move the robotic manipulator's TCP on the Cartesian space X, Y and Z-axes to perform the extraction of two rigid objects in contact (contact-rich extraction skill).

During each timestep of interaction with the environment, the agent is presented with an observation o of the state $s \in S$ of the environment, which is not fully observable. The agent then selects an action $a \in A$ from the action space using a parameterized policy π_θ. The environment, which changes according to the agent's action, provides a reward signal $r_t = R(s_t, a_t, s_{t+1})$ to the agent, indicating how good or bad the new state is. The agent's objective is to optimize the cumulative reward, discounted by a factor $\gamma \in (0, 1]$, by adjusting the policy's behavior through some optimization algorithm.

The following subsections describe the RL formulation:

6.2.2.1 Reinforcement learning elements

Observation space

The observation space is defined as a 9-dimensional vector. The components of the observation space are position, velocity and force, all in the Cartesian space as shown in Equation 6.1.

$$o_t = [\ p_{tcp_t} - p_{tcp_0},\ p_{tcp_t} - p_{tcp_{t-1}},\ \vec{F}\ ] \tag{6.1}$$

The position component is calculated as the relative position of the TCP (p_{tcp_t}) with respect to its initial pose (p_{tcp_0}) in the last 15 timesteps. As discussed in Chapter 3, the use of relative information is relevant to the generalization of the task for different initial poses as long as the manipulation is performed within the operating range of the robot.

The velocity is calculated as the difference between the current TCP position (p_{tcp_t}) and its previous position ($p_{tcp_{t-1}}$). Since the captures are taken at the same frequency, the time fraction between the two measurements becomes a constant assumed by the neural network (approximation function).

Force ($\vec{F}$) information provides feedback on the mechanical interaction between the manipulated parts

Action space

The action space is a 3-dimensional vector. Each component maps to the respective translation (in centimeters) of the robot's TCP in the $[-1, 1]$ continuous interval in the Cartesian space as shown in Equation 6.2.

$$a_t = [\ \Delta x,\ \Delta y,\ \Delta z\ ] \tag{6.2}$$

The action space definition allows movements of the manipulator control frame in its 3 translational DOFs, making it possible to reach any position in Cartesian space within the robot's range.

Reward

To enforce the execution of the extraction task, a dense reward function is proposed in which the agent receives an instantaneous value, at each interaction with the environment, as a measure of how good or bad its decision-making was.

The reward value for the extraction task is determined by calculating the cumulative spatial displacement progress of the robot's TCP at each iteration, as formulated in Equation 6.3.

$$r_{task_t} = \left\| \sum_{i=1}^{t} (p_{tcp_i} - p_{tcp_{i-1}}) \right\| \tag{6.3}$$

The reward formulation overcomes the limitations of the definitions of state-of-the-art work in disassembly and has the following benefits:

- There is no restriction on the direction of extraction to any particular combination of Cartesian axes.
- For the same extraction direction, it does not limit its sense. Once a motion sense is defined, the maximum reward values are obtained by performing actions in the same sense.

  Moreover, it prevents the policy from falling into intermittent movement by taking advantage of design flaws. Figure 6.3 shows an example of such behavior. In this figure, random or intermittent decision making produces a lower reward value than following the same direction and sense.
- It encourages a fast execution of the extraction task in any of the identified directions ans senses. Higher reward values are obtained for larger displacements (in the same direction and sense).

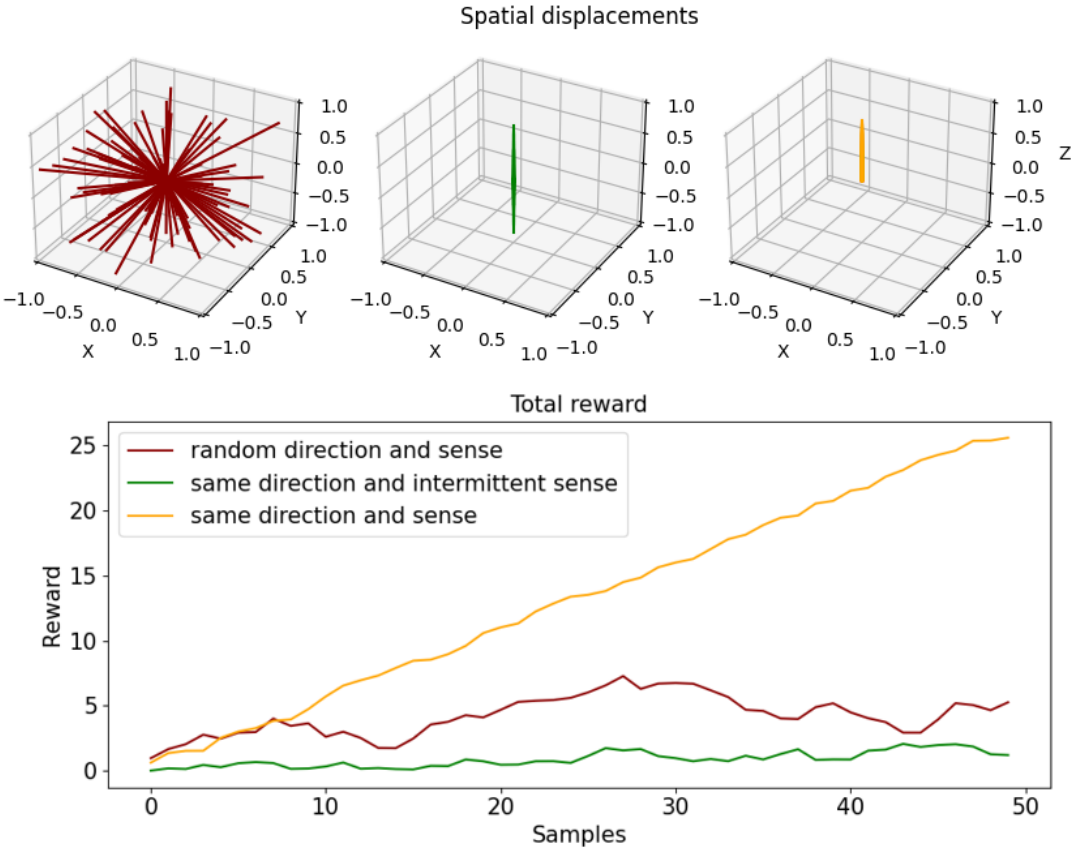

FIGURE 6.3: Examples of the reward function formulated in Equation 6.3 for 50 spatial displacement samples for both random direction and sense (red), and for the same direction with intermittent sense (green) and with the same sense (orange). Random or intermittent decision making produces a lower reward value than following the same direction and sense.

Episode termination

To generalize the extraction learning to geometries with different dimensions, short episodic training is performed. Each episode lasts 1 second of physics simulation, where the interaction with the environment occurs at a frequency of 15 Hz.

This configuration tries to learn to learn to extract fragments instead of doing the whole task, as is done in the existing state-of-the-art. Learning to extract fragments has the following advantages:

- Task execution is not limited to one or several objects with specific known dimensions.
- Extraction execution for fragments is adjustable to the variations in the geometry of the parts involved and their physical properties.

Note that during the evaluation, it is necessary to use a window (with size 15 timesteps) to keep the value of the TCP initial position (p_{tcp_0}) updated in order to calculate its relative position for building the observation space.

6.2.2.2 Reinforcement learning algorithms

Agents

For training and evaluation, the following actor-critic algorithms were selected:

- Deep Deterministic Policy Gradient (DDPG)
- Twin Delayed Deep Deterministic policy gradient (TD3)
- Soft Actor-Critic (SAC)
- Proximal Policy Optimization (PPO)

Either DDPG, TD3 and SAC are off-policy algorithms while PPO is an on-policy algorithm. The PPO selection is performed to take advantage of massive environment parallelization.

Model architectures

The architecture of the Policy and Q networks used by the DDPG, TD3 and SAC agents as well as the Value network used by PPO are the same. The networks receive the observation space as input for both, Policy and Value networks, and the concatenated components of the observation space and action space for Q networks as input.

The model's inputs are followed by two hidden dense layers of 64 neurons each with Exponential Linear Unit (ELU) activation functions. Compared to the previous study, this time the ELU activation function was chosen. ELU is a smoother function than ReLU and, unlike the latter, allows negative information to be explicitly captured, potentially improving the network's performance and leading to faster convergence in some cases [167][168].

The Policy network has 3 output neurons that use the hyperbolic tangent function to fit the action space to the expected interval $[-1, 1]$. Both, the Q and Value networks use a linear activation function for their output neuron to estimate an unbounded Q-value and state-value respectively.

6.2.3 Experiments and results

6.2.3.1 Experiment implementation

The RL environment was defined using the NVIDIA Omniverse Isaac Gym interface in Python to allow running massive parallel simulations. To take advantage of this simulation platform, the environment is replicated until 1024 instances of it are obtained as shown in Figure 6.4.

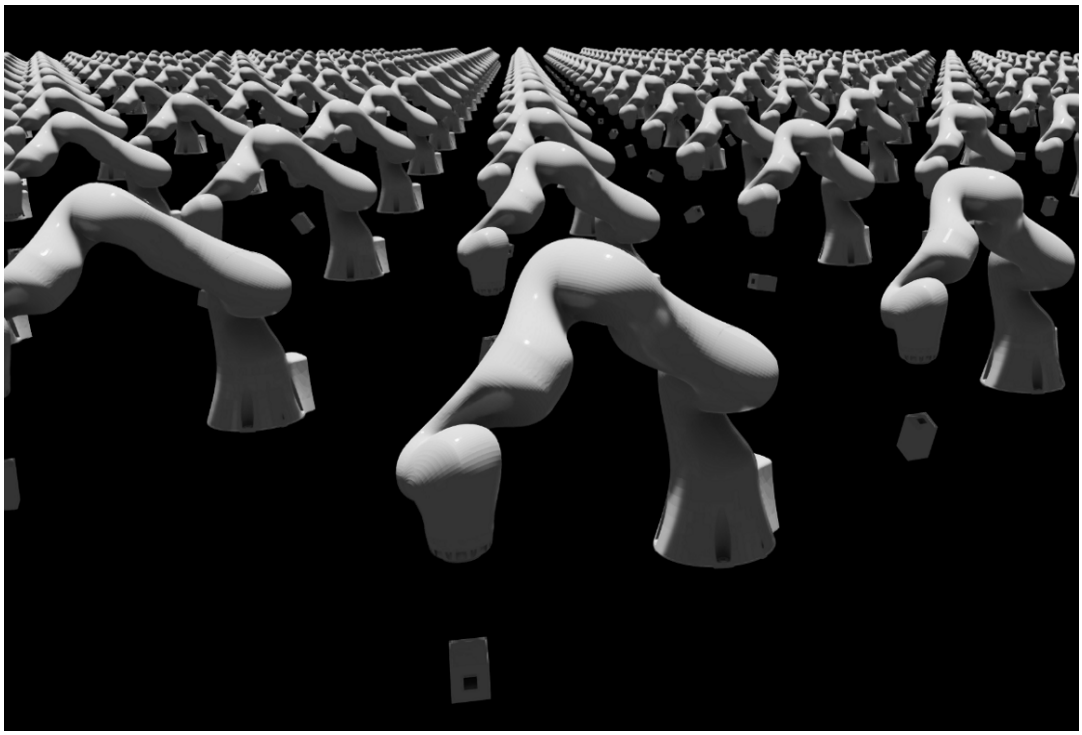

FIGURE 6.4: Snapshot of a subset of the 1024 parallel environments in simulation.

For both, learning and evaluation the *skrl* library described in Chapter 4 was used. Using *skrl*, PPO training was done standalone, while DDPG, TD3 and SAC were trained simultaneously with shared memory to reduce training times, GPU consumption and improve their learning. In the Table 6.2 are listed the hyperparameter values configured and allowed by the *skrl* library.

Hyperparameters	**Value**
Optimizer	Adam
Policy/Value learning rate	$5 \cdot 10^{-4}$
Actor/Critic learning rate	$5 \cdot 10^{-4}$
Entropy learning rate (SAC)	$5 \cdot 10^{-3}$
Rollout buffer size	16
Experience replay buffer size	10^5

Rollouts	16
Learning epoch	8
Number of mini batches	8
Batch size	PPO: 256, DDPG, TD3 and SAC: 4096
Discount factor	0.99
Ratio/value clip	0.2
Ornstein-Uhlenbeck noise (DDPG)	θ: 0.15, σ: 0.2, base scale: 0.1
Gaussian noise (TD3)	mean: 0.0, std: 0.1
Exploration decay (type)	linear
Exploration decay (initial scale)	1.0
Exploration decay (final scale)	10^{-3}
Smooth regularization noise (TD3)	mean: 0.0, std: 0.2
Smooth regularization clip (TD3)	[-0.5, 0.5]
Soft update hyperparameter	0.005
Maximum timesteps per episode	15

TABLE 6.2: Hyperparameter configured and allowed by *skrl* library.

To generate high degree of variability, both the air gap between the fixed base and the cube attached to the robot and the physical friction between the two objects were randomized. A set of air gaps between the fixed base and the object (0.0, 0.10, 0.25, 0.50, 1.00 millimeters), as well as the dynamic and static friction coefficients (0.00, 0.05, 0.10, 0.15, 0.20, 0.25, 0.50, 0.75, 1.00), were uniformly distributed across all parallel environments.

Each time an environment is reseted, the orientation of the fixed base (and the object attached to the robot) is randomized by uniform sampling (in spherical coordinates), in the range 0 to 180 degrees in polar and -90 to 90 degrees in azimuthal angles. Since the fixed base is symmetrical, the angular samples in these intervals cover all possible orientations in the space.

The training were done using a local workstation with an Intel(R) Xeon(R) W-2295 CPU @ 3.00 GHz, 125 GiB of RAM, and a GPU Quadro RTX 6000. The algorithms were trained 6 times, with different seeds for the random number generator, to identify repetitive behaviors. PPO was trained for 20000 timesteps while DDPG, TD3 and SAC were trained for 75000 timesteps. The policies with the highest mean reward during training were selected for evaluation.

6.2.3.2 Training

Figure 6.5 shows the mean reward and its standard deviation for PPO, trained individually, and DDPG, TD3 and SAC, trained simultaneously with shared memory. Timpesteps axis have been normalized to plot all curves on the same time scale, since the number of timesteps executed is different for PPO and for DDPG, TD3 and SAC.

According to the results, PPO and SAC reach very similar mean reward values. Although SAC quickly reaches maximum values, its performance declines very slightly in the second half of the training. In the case of DDPG and TD3, their statistics reach a stable behavior, without growth or decay, from very early in the training process. Both algorithms are ranked under PPO and SAC, with DDPG being the one with the worst performance during training.

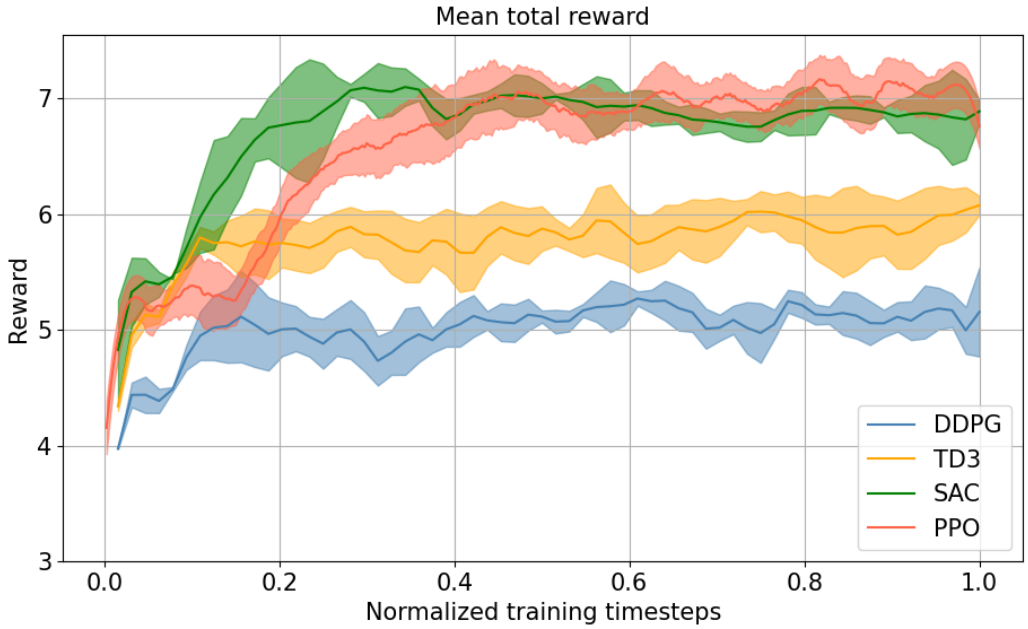

FIGURE 6.5: Simulation training results: Mean reward and standard deviation during training for PPO, DDPG, TD3 and SAC agents.

6.2.3.3 Evaluation of the generalization capabilities under variability in simulation

To evaluate the RL algorithms' generalization capabilities in simulation, a set of orientations resulting from the combination of polar angles (in the range 0 to 180 degrees with a 10 degrees of step) and azimuthal angles (in the range -90 to 90 degrees with a 5 degrees of step) were defined, derived in a total of 648 environments with different orientations each.

For this phase, a maximum number of 100 timesteps was set as the termination condition (along with the completion of the extraction task) for the current episode. 500 timesteps, for at least 5 episodes, were run for the combination of friction coefficients (0.00, 0.05, 0.10, 0.15, 0.20, 0.25, 0.50, 0.75, 1.00) and air gaps (0.0, 0.10, 0.25, 0.50, 1.00 millimeters) defined during training.

Figure 6.6 displays, in a global view, the task completion ratio of all experiments for the involved algorithms. The task completion ratio is the outcome of the discrimination of the execution status (whether the task has been completed or not) in the specified amount of timesteps for the evaluation. In addition, it includes the task completion ratio from the execution of a baseline defined by the execution of actions that follow the fixed base orientation with the maximum possible displacement.

Since the baseline corresponds to the optimal performance of the extraction task (maximum displacement in the same orientation of the fixed base), it is expected that the results obtained with the trained policies will be close to the baseline values, but never exceed them.

For friction coefficients close to zero, where the probability of jamming is practically null (for actions close to the orientation direction of the fixed base) the successful execution of the task with the learned policies is completed with a high ratio. However, although the successful execution of the tasks for higher values of the friction coefficients increase their ratio, as the air gap between the fixed base and the

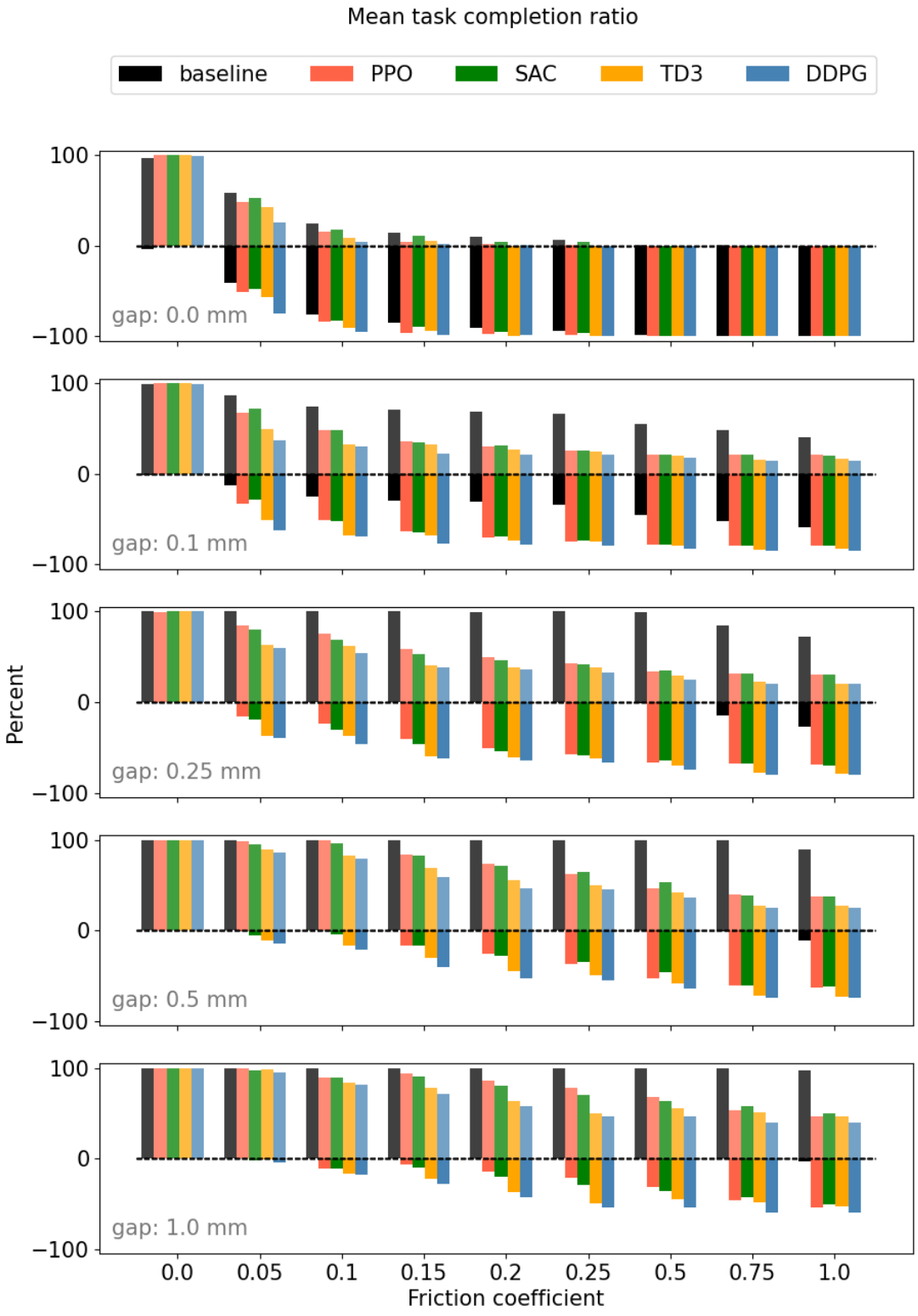

FIGURE 6.6: Task completion ratio resulting from the discrimination of the execution state (completed or not) for the combination of friction coefficients and air gaps defined during training. The baseline was generated with actions that follow the fixed base orientation with the maximum possible displacement. Negative values indicate that the execution could not be completed within the specified timesteps.

object to be extracted grows, they remain below 50% of the baseline ratio.

Figure 6.7 represents graphically the mean duration (in timesteps) of the last 5 episodes performed by the PPO (the optimal algorithm according to training results) for 2 pairs of coefficient of friction and air gap: (0.0, 0.0 mm) and (1.0, 1.0 mm), among all available pairs.

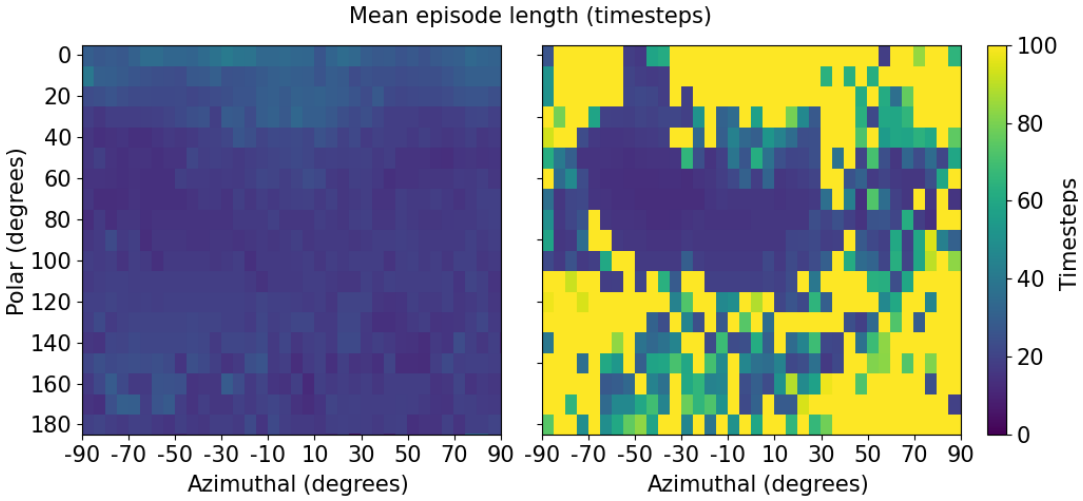

FIGURE 6.7: Mean duration (in timesteps) of the last 5 episodes for the PPO algorithm for the pairs of coefficients of friction and air gap (0.0, 0.0 mm), on the left, and (1.0, 1.0 mm), on the right, respectively.

Such figure allows to visualize in detail (with respect to the different spatial orientations of the parts) how the trained policy behaves for different pairs of polar and azimuthal angles.

6.3 Chapter conclusions

The results presented in this chapter respond, to a large extent, to research question 1: The proposed RL method is able to generalize, in different extends, extraction disassembly tasks with high variability in the geometric length, air gap as well as the physical friction of the objects involved. For low friction coefficients, as the gap between the manipulated parts increases, the task execution is completed at high percentage values (above 95%). Not so, for high friction coefficients, where the percentage values drop below 50%.

To generalize the task to different geometric lengths and manipulations in Cartesian three-dimensional space, the use of relative information to form the observation space, the reward function and the training of the agents for short periods of time are essential. The definition of the reward function does not restrict the extraction direction to any particular Cartesian axis, but promotes the execution of the task for any of the identified directions and senses.

However, since the reward function does not explicitly define a direction to take, the decisions made by the trained policies are spread over finding actions that yield some extraction, leaving little room for task execution once a direction is found (if found), thereby reducing the probability of task completion. Furthermore, such a definition does not guarantee the reliability of the trained system with respect to a specific extraction sense.

Chapter 7

Disambiguating Disassembly Task Sense and Narrowing Action Space Through Human Operator Experience.

The content of this chapter addresses, in part, research question 1: To what extent can RL algorithms generalize the execution of disassembly tasks using collaborative robots?

The work presented in this chapter is partially derived from the work carried out for the paper: Íñigo Elguea-Aguinaco, Antonio Serrano-Muñoz, Dimitrios Chrysostomou, Ibai Inziarte-Hidalgo, Simon Bøgh, and Nestor Arana-Arexolaleiba. "Goal-Conditioned Reinforcement Learning within a Human-Robot Disassembly Environment". In: Applied Sciences 12.22 (2022), p. 11610. URL: *https://www.mdpi.com/2076-3417/12/22/11610*.

The research conducted in Chapter 6 showed how some RL algorithms can learn and generalize object extraction tasks with a high degree of variability in terms of the geometric dimensions and physical friction of the manipulated objects. However, disassembly is not always feasible, which makes a human-robot collaboration approach essential to address this challenge.

This chapter proposes an adaptation of the RL method described in the previous chapter (Chapter 6) that incorporates the human operator's experience as a source of information to narrow the action space around the extraction direction, as well as to disambiguate the extraction sense of the disassembly task.

7.1 Reinforcement learning for disassembly tasks with execution disambiguation

Performing the extraction task in a preferred sense, when the geometry is symmetrical, may be necessary or desirable in certain cases. For example, when it is necessary to perform the movements in a specific sense for safety reasons or due to workspace constraints; or when there is an irregularity/obstruction due to the conservation/transport/handling state of an EOL product that blocks extraction in one of the senses.

The use of conventional cameras may help to identify workspace boundaries and/or safety requirements [68][169]. However, in case of irregularities/obstructions in the manipulated parts, they may suffer from occlusions [170] since mechanical interactions occur inside the slotted components.

A review by Goli, Wang, and Saadat [171] investigates the perspective of self-learning robotics for disassembly automation. The authors highlight that a human-robot disassembly system can be a more efficient solution than manual disassembly systems due to the high level of uncertainty and unplanned operations in the field. They also emphasize the relevance of RL in robotic manipulation tasks.

In this particular scenario, the human operator can make use of his expert knowledge and help by guiding with the extraction sense. The objective is not for the human operator to provide the exact and specific solution in terms of the magnitude of the displacement, the direction and the sense of the extraction. In this case, it is desired that the human provides a hint as to where (direction and sense) the extraction should be performed, with the autonomous agent making the final decision.

7.1.1 Gathering human hints for extraction sense disambiguation.

To obtain a human operator hint, the force measurement system of the robotic manipulator itself is used as the front-end interface. By exerting an external force on the manipulator end-effector, the human operator can provide a hint of which direction and sense (encoded as a Cartesian space vector) to follow, as shown in Figure 7.1.

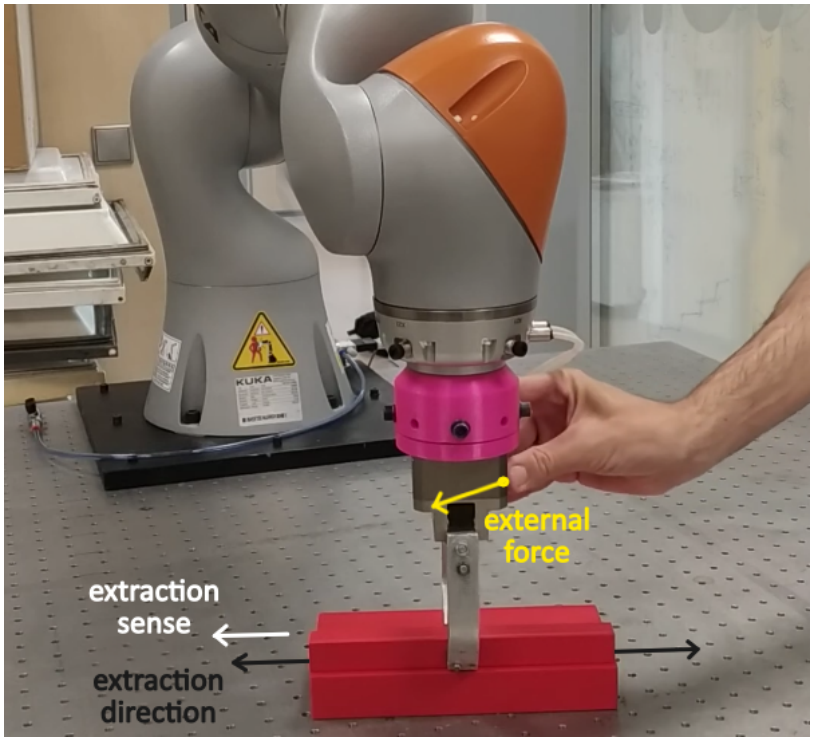

FIGURE 7.1: Using human hint for extraction sense disambiguation.

The Cartesian space vector for the hint ($\vec{hint}$) is computed as the normalized vector of the external force ($\vec{F}_{ext}$), plus a small number to avoid division by zero, sensed in the end-effector frame and rotated by the orientation (q_{ee}) of that frame, as specified by Equation 7.1.

$$\vec{hint} = q_{ee} \otimes \frac{\vec{F}_{ext}}{||\vec{F}_{ext}|| + 10^{-8}} \tag{7.1}$$

In order to evaluate the feasibility of using the manipulator's force sensors as the interface, human hints were sampled with respect to 24 reference points (ground truths). Figure 7.2 shows the spatial distribution of the ground truths and of the captured samples.

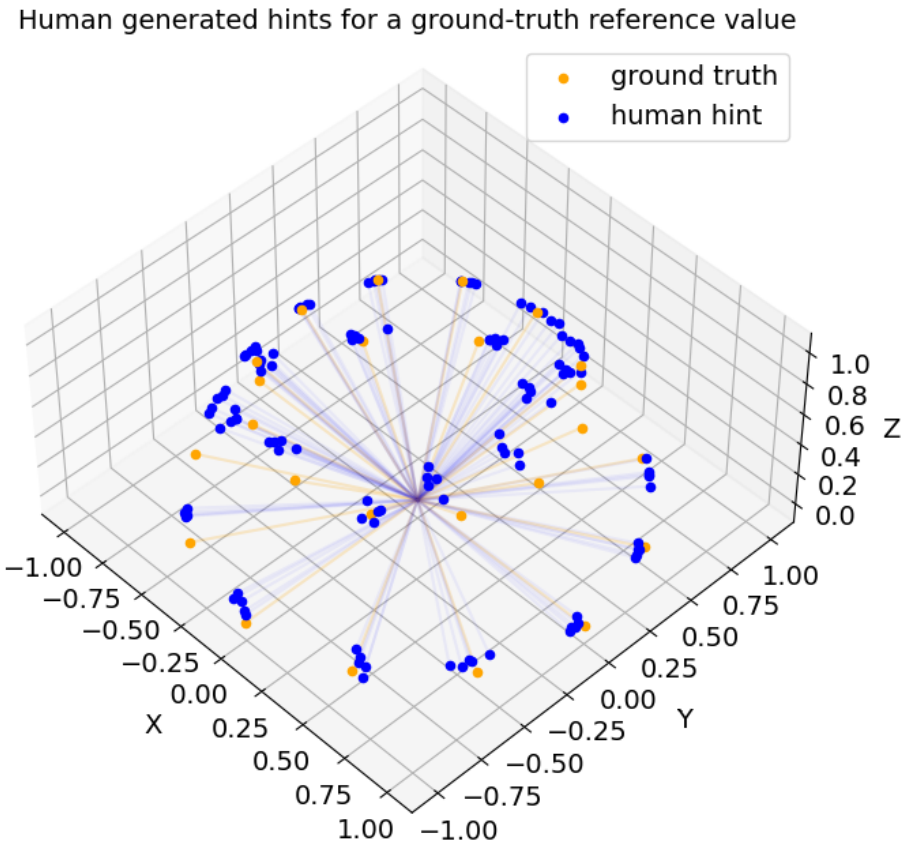

FIGURE 7.2: Spatial distribution of the ground truth values for the fixed polar angles of 45 and 90 degrees and the azimuthal angles in the interval -180 to 180 degrees spaced every 30 degrees (in orange) and the acquired samples of the human hints (in blue) for these ground truth values.

For each reference point (resulting from the combination of the fixed polar angles 45 and 90 degrees and the azimuthal angles in the interval -180 to 180 degrees spaced every 30 degrees) 5 samples of the external force exerted by the human (in the direction and sense of the references) were captured.

Even when the hints computed from the external force exerted by the human are close to the ground truth values, there is an observable spatial difference even between the samples made for a same reference. Figure 7.3 shows the differences, measured as the angle between the hint vectors and their respective ground truth values for the data set.

The angular differences remain mostly below 7.5 degrees for references parallel to the horizontal plane (90 degrees polar angle). For the 45 degree polar angle, the difference is greater. For both example cases, the differences can be up to 15 degrees.

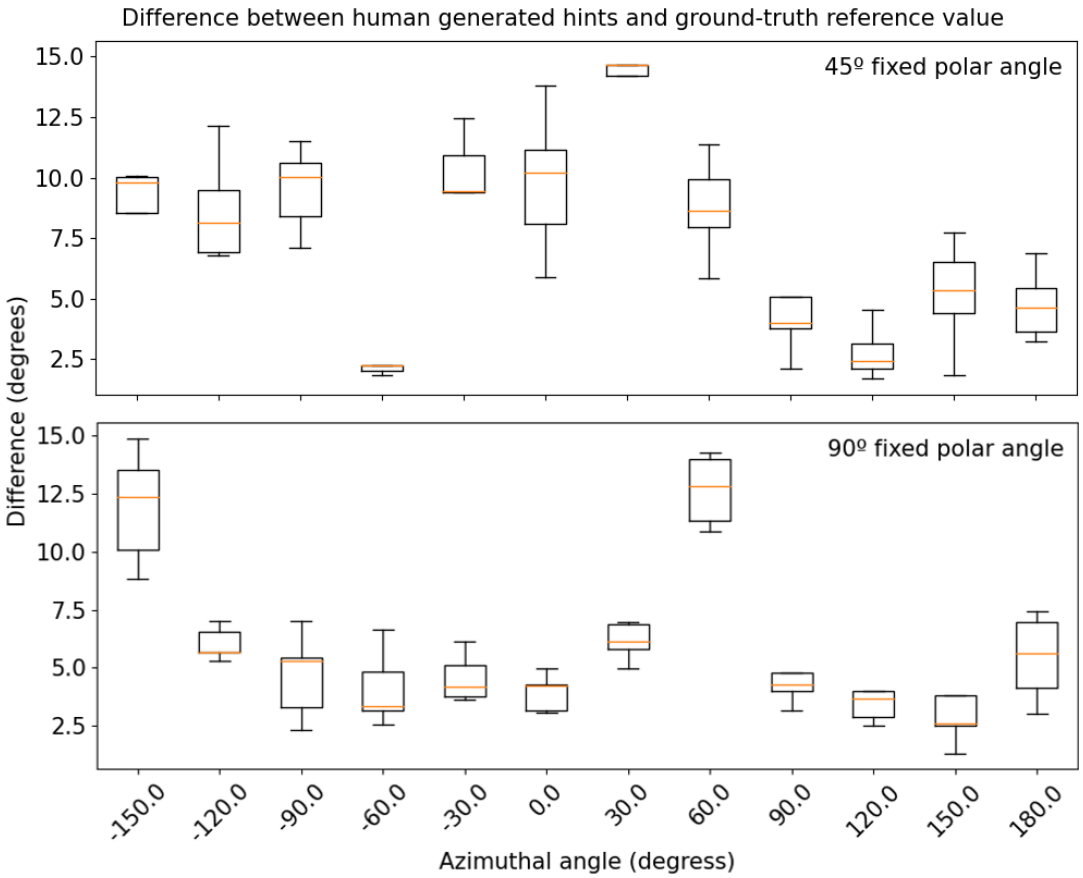

FIGURE 7.3: Angular difference between predefined ground truth vectors and hint vectors resulting from the external force exerted by the human to match reference data.

Despite the differences between the ground truth values and the vectors calculated from the external force exerted by the human, the latter can be used to provide hints to disambiguate the sense of the extraction task since the difference is small enough to distinguish from the opposite side (detectable starting at 90 degrees).

7.1.2 Reinforcement learning experimental setup

The experimental setup is the same as the one described in Chapter 6 but this time, incorporating the hint information.

7.1.2.1 Simulated setup

The simulation setup includes the possibility of generating hints whose directions and senses can be parameterized. The parametrization allows to define a sense (to one side or the other) for the same direction. Also, it enables the adjust of the hint direction (by a pair of polar and azimuthal angles) to match or not the exact direction of the extraction. This configuration makes it possible to simulate the directional deviations of the human-generated hint according to the analysis performed as well as a specific sense.

7.1.2.2 Real-world setup

For real-world evaluation five different environments were used as shown in Figure 7.4. The selection of these environments is based on the creation of variability with the resources available in our laboratory.

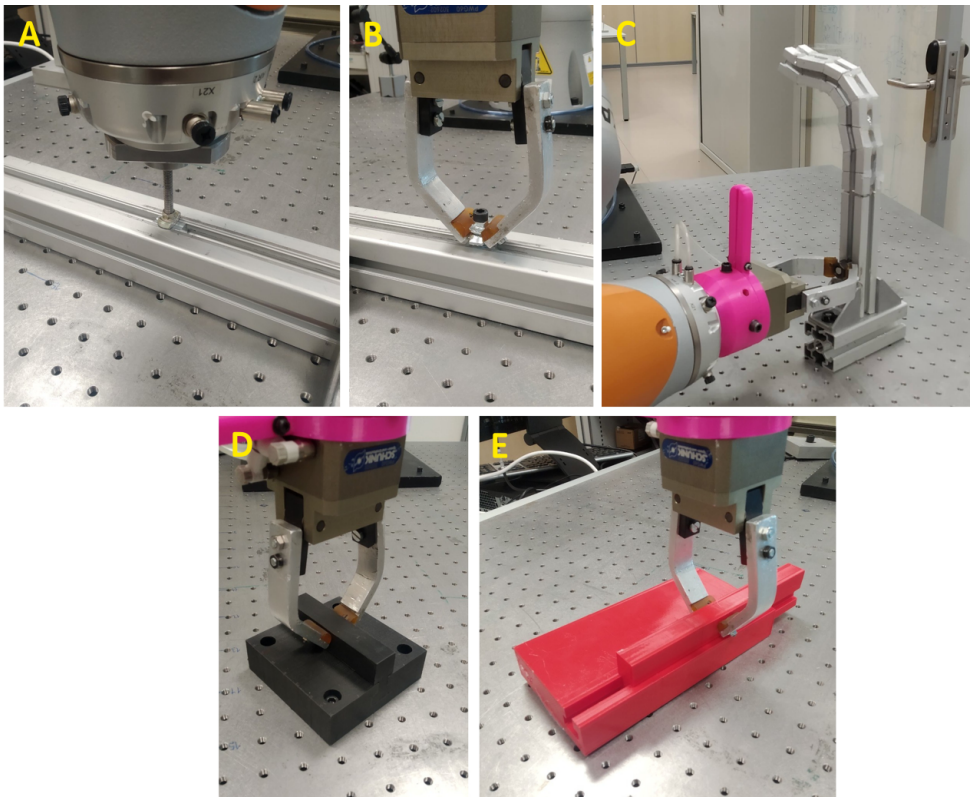

FIGURE 7.4: Different real-world environments for the evaluation of the learned policies.

The real-world environments are composed of plastic and metal parts, with different coefficients of friction, air gaps between the contacting parts and geometries in terms of length and dimensions of the fixed base slot as well as the shape (with sharp or rounded edges) of the object to be extracted, Table 7.1 list each environment specifications.

Specification	**Environments**				
	A	**B**	**C**	**D**	**E**
Base length (cm)	45	45	40	10	10
Object length (cm)	2.0	2.0	2.5	20	20
Air gap	0.15	0.15	0.30	0.05	0.55
Base material	Aluminum	Aluminum	Aluminum	Plastic	Plastic
Object material	Aluminum	Aluminum	Steel	Plastic	Plastic
Friction coefficient	0.85 - 1.06	0.85 - 1.06	0.4 - 0.8	0.1 - 0.4	0.1 - 0.4

TABLE 7.1: Real-world environment specifications.

Robot control in the real world was done using the *libiiwa* Python API as described in Chapter 5. In order to guarantee the safety of the operator, the environment and the robotic manipulator itself, force conditions were defined, with maximum admissible values up to 60 Newtons for each Cartesian axis, which activate

stop reactions when the limits are reached.

In addition, to reduce the contact forces caused by friction and the probability of jamming states during the execution of the disassembly tasks, force overlay in the form of Lissajous curves was used. The Lissajous curve, parameterized with amplitude and frequency values of 3 Newtons and 7.5 Hz respectively, was applied in the plane perpendicular to the direction of action taken by the policy. This configuration is based on the empirical study carried out in Chapter 5.

The generation of the human hint is performed by the operator using the same procedure described in Subsection 7.1.1.

7.1.3 Reinforcement learning formulation

The formulation of the RL setup remains the same except for the observation and reward components. These two components are modified to use the human-generated hint to disambiguate the sense of the extraction task.

7.1.3.1 Reinforcement learning elements

Observation space

The vector of human-generated hint is added to the previously defined observation space, which is now a 12-dimensional space as shown in Equation 7.2.

$$o_t = [\ p_{tcp_t} - p_{tcp_0},\ p_{tcp_t} - p_{tcp_{t-1}},\ \vec{F},\ \vec{hint}\ ] \tag{7.2}$$

Reward

The reward part dedicated to the integration of the hint provided by the human tries to reduce the difference between this information (*hint*) and the action taken by the policy (a_t).

For this purpose, in the first instance, an auxiliary value (ρ_t) involving both normalized values is calculated according to the Equation 7.3. The closer, in direction and sense, is the action taken from the provided hint, the closer to zero is its value.

$$\rho_t = 0.5\ \|\widehat{hint} - \widehat{a}_t\| \tag{7.3}$$

Then, the reward function related to the hint is defined by Equation 7.4.

$$r_{hint_t} = \texttt{clip}(\|\widehat{a}_t\|,\ 0,\ 1)\ e^{-5\rho_t} \tag{7.4}$$

This function has two components. One is a negative exponential of the auxiliary value, with the aim of penalizing the actions taken by the policy that do not follow the specified hint. The other, it penalizes actions whose norm (and therefore the expected displacement resulting from its application) is small.

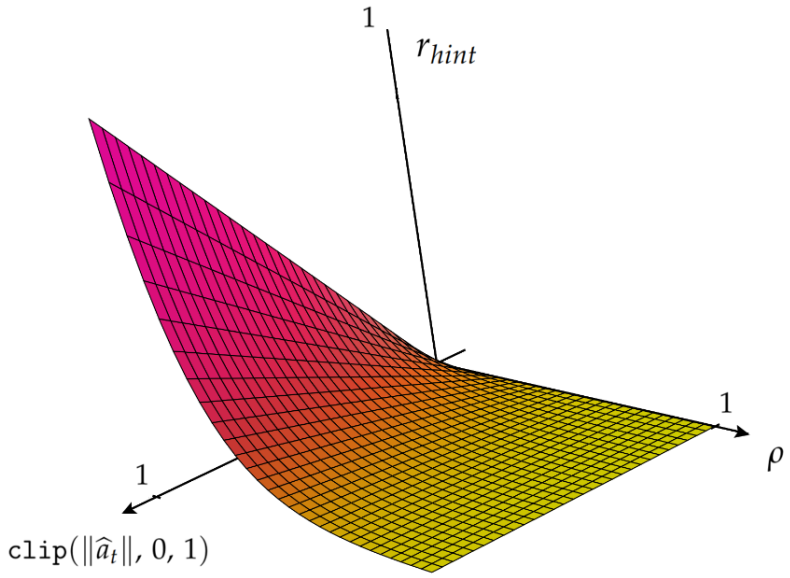

FIGURE 7.5: Graphical representation of the hint-reward function.

A representation of this reward function can be found in Figure 7.5.

The final reward function is now a weighted sum of the reward functions of the task execution and the following of the human hint, as formulated in Equation 7.5. The weights (w_{task} and w_{hint}) allow to adjust the relevance of both members or turn them off.

$$r_t = w_{task}\, r_{task_t} + w_{hint}\, r_{hint_t} \tag{7.5}$$

7.1.4 Experiments and results

7.1.4.1 Experiment implementation

The implementation of the experiments follows the same configuration described in Chapter 6 for simulation. To this configuration is added the randomization of the hints, parameterized by polar and azimuthal angles, to generate angular deviations of up to 15 degrees per spherical coordinate angle with respect to the fixed base orientation direction (ground truth). The target sense of extraction is also randomized.

The algorithms (PPO, DDPG, TD3 and SAC) were trained 6 times, with different seeds for the random number generator, to identify repetitive behaviors. The same hyperparameters from the previous chapter were used for training.

7.1.4.2 Training

Figure 7.6 shows the mean reward and its standard deviation for PPO, trained individually, and DDPG, TD3 and SAC, trained simultaneously with shared memory. As in the previous experiments, timpesteps axis have been normalized to plot all curves on the same time scale, since the number of timesteps executed is different for PPO and for DDPG, TD3 and SAC.

According to the results, the agents learn in a similar way as those trained in the previous chapter, but this time they reach a higher mean reward. PPO and SAC

FIGURE 7.6: Simulation training results: Mean reward and standard deviation during training for PPO, DDPG, TD3 and SAC agents.

reach very similar mean reward values and maintain a slight increasing trend practically during the whole training run. In the case of DDPG and TD3, both algorithms are positioned below PPO and SAC, being DDPG the one with the worst performance during training.

7.1.4.3 Evaluation of the generalization capabilities under variability in simulation and real-world

Evaluation in simulation

To evaluate the RL algorithms' capabilities, the same set of orientations resulting from the combination of polar angles (in the range 0 to 180 degrees with a 10 degrees of step) and azimuthal angles (in the range -90 to 90 degrees with a 5 degrees of step) were used for a total of 648 environments with different orientations each. To this configuration is added the adjustment of the angular deviation (0.0, 3.0, 6.0, 9.0, 12.0, 15.0 degrees) of the generated hint with respect to the actual orientation of the fixed base. The extraction sense was uniformly distributed in a random manner.

For this phase, a maximum number of 100 timesteps was set as the termination condition (along with the completion of the extraction task) for the current episode. 500 timesteps, to guarantee the execution of at least 5 episodes, were run for the combination of friction coefficients and air gaps defined during training. A window of size 15 timesteps, to keep the value of the initial position of the TCP (p_{TCP_0}) updated, was used for building the observation space.

Figure 7.7, 7.8, 7.9, 7.10 and 7.11 show a view of the overall task completion ratio for all the different values of friction coefficients (0.00, 0.05, 0.10, 0.15, 0.20, 0.25, 0.50, 0.75, 1.00), angular deviations (0.0, 3.0, 6.0, 9.0, 12.0, 15.0 degrees) and air gaps (0.0, 0.10, 0.25, 0.50, 1.00 millimeters), grouped by the air gap values.

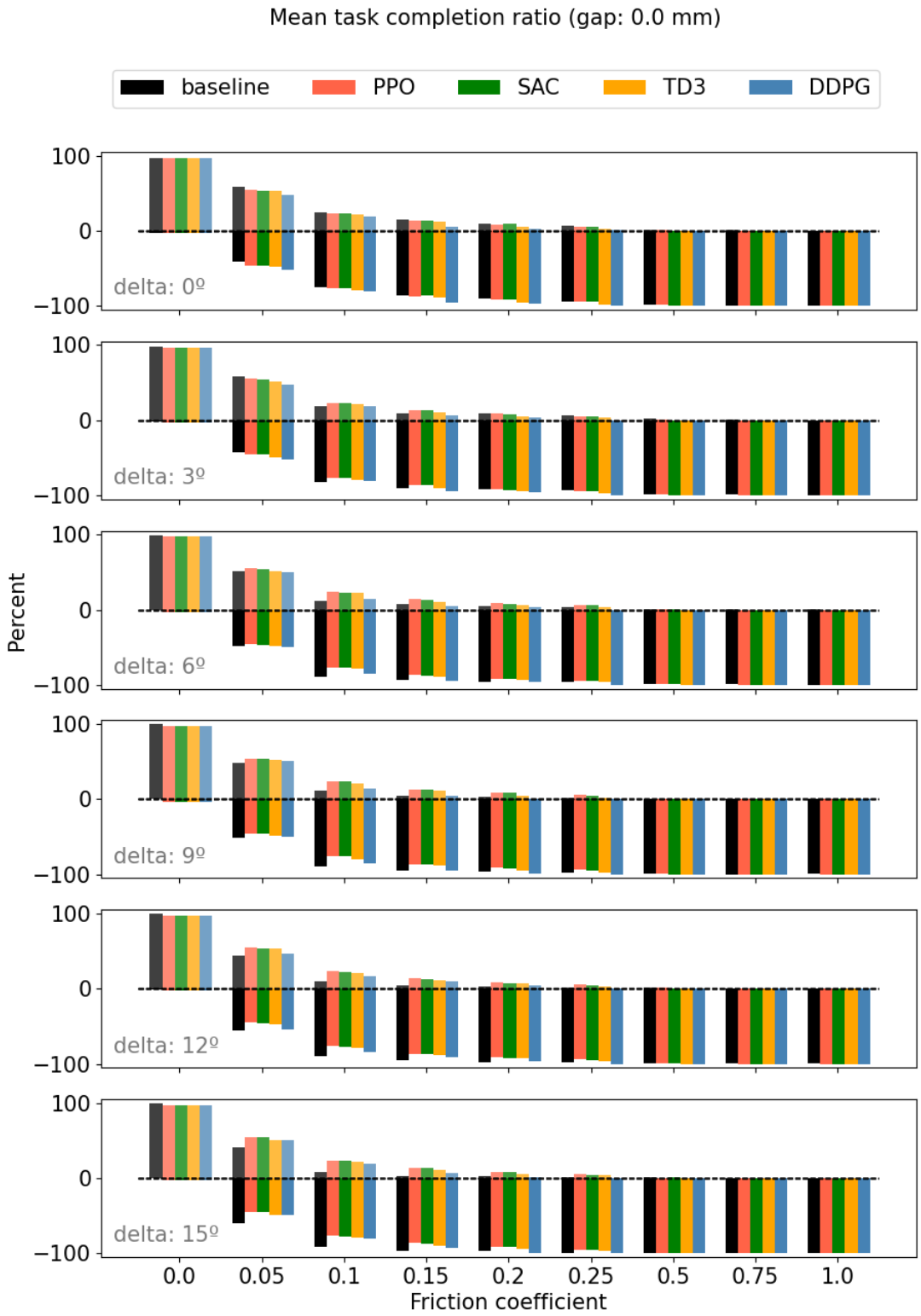

FIGURE 7.7: Task completion ratio resulting from discrimination of execution status (completed or not) for an air gap of 0.0 mm. The baseline was generated with actions that follow the generated hints with the maximum possible displacement. Negative values indicate that the execution could not be completed within the specified number of timesteps.

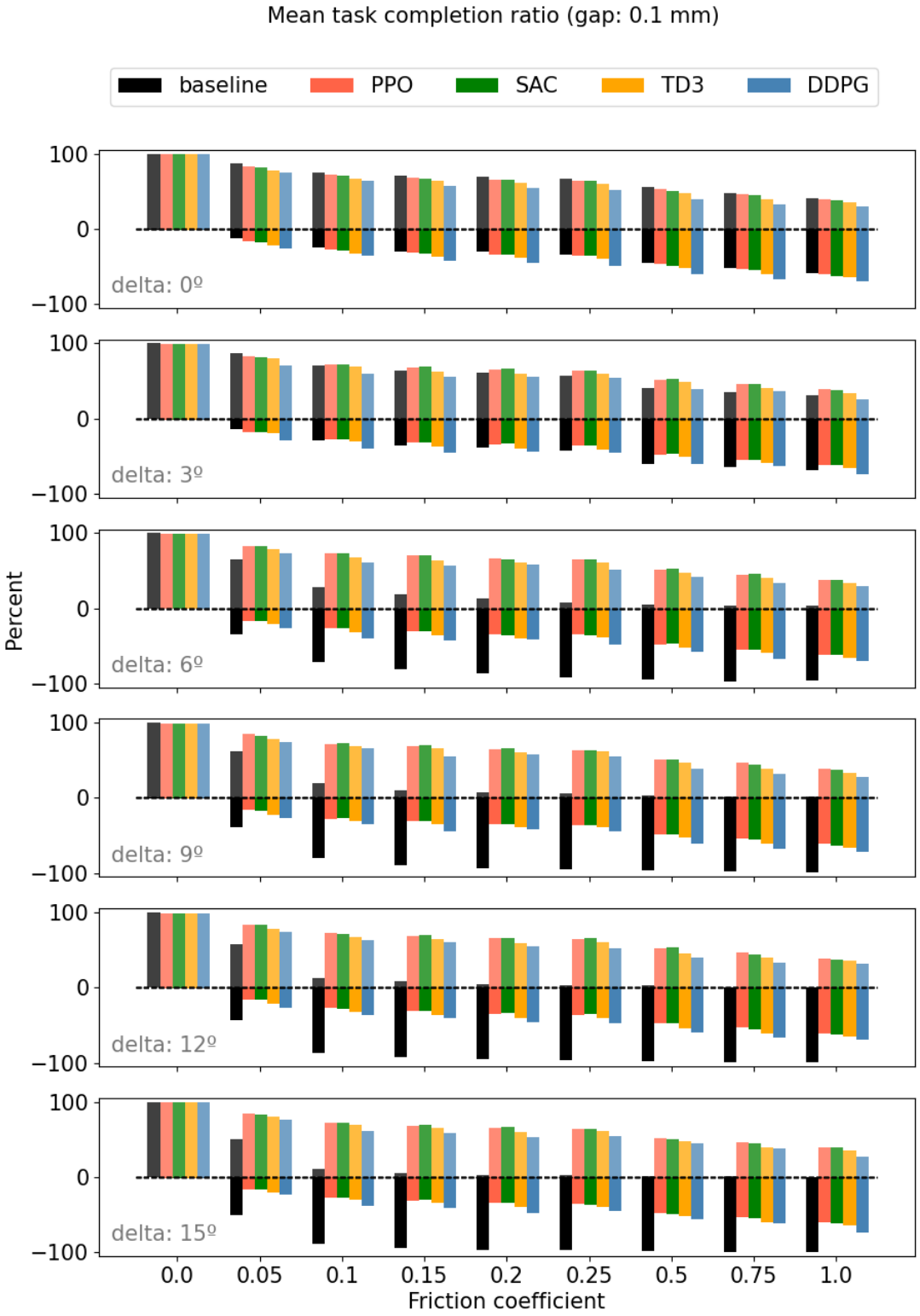

FIGURE 7.8: Task completion ratio resulting from discrimination of execution status (completed or not) for an air gap of 0.1 mm. The baseline was generated with actions that follow the generated hints with the maximum possible displacement. Negative values indicate that the execution could not be completed within the specified number of timesteps.

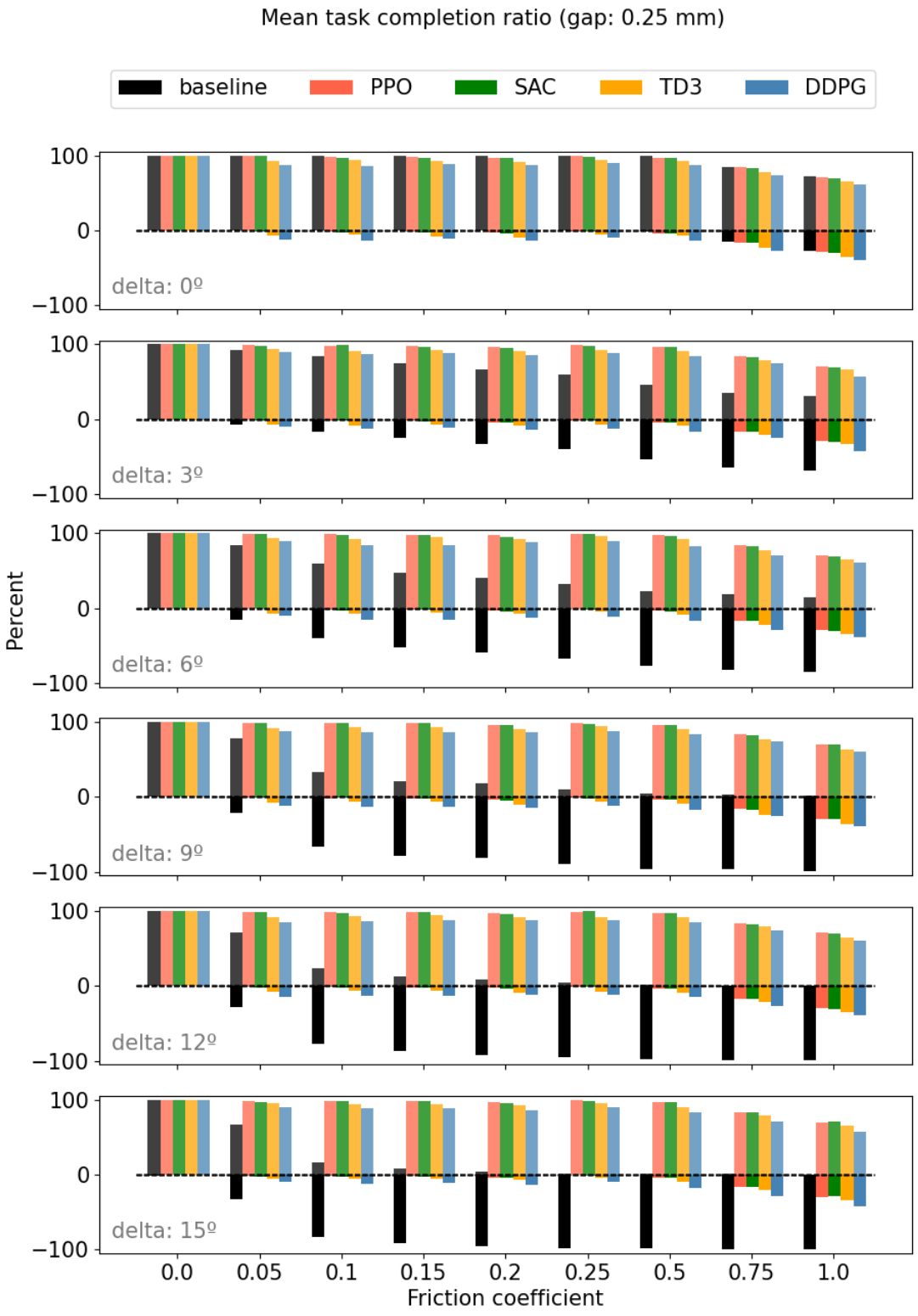

FIGURE 7.9: Task completion ratio resulting from discrimination of execution status (completed or not) for an air gap of 0.25 mm. The baseline was generated with actions that follow the generated hints with the maximum possible displacement. Negative values indicate that the execution could not be completed within the specified number of timesteps.

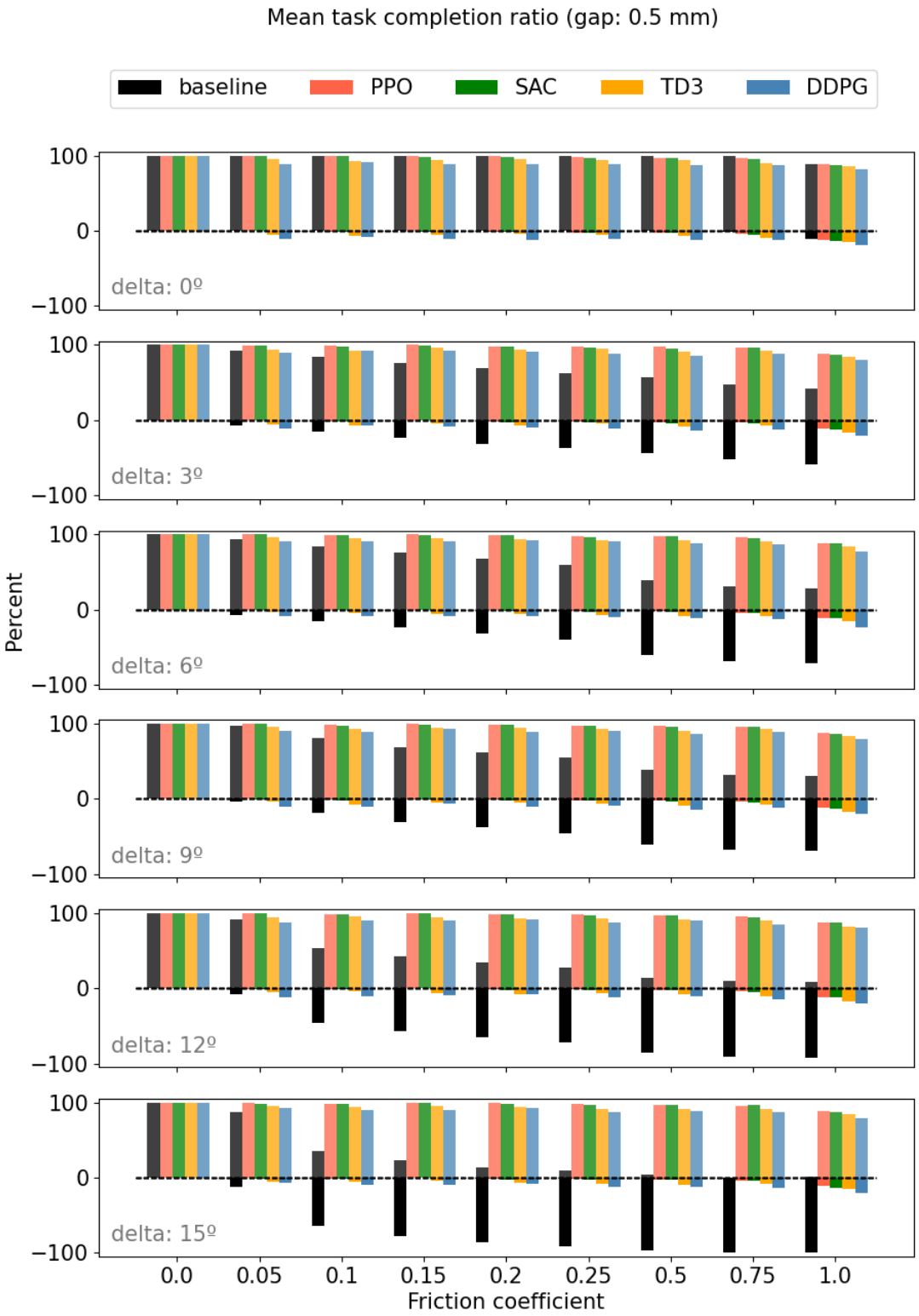

FIGURE 7.10: Task completion ratio resulting from discrimination of execution status (completed or not) for an air gap of 0.5 mm. The baseline was generated with actions that follow the generated hints with the maximum possible displacement. Negative values indicate that the execution could not be completed within the specified number of timesteps.

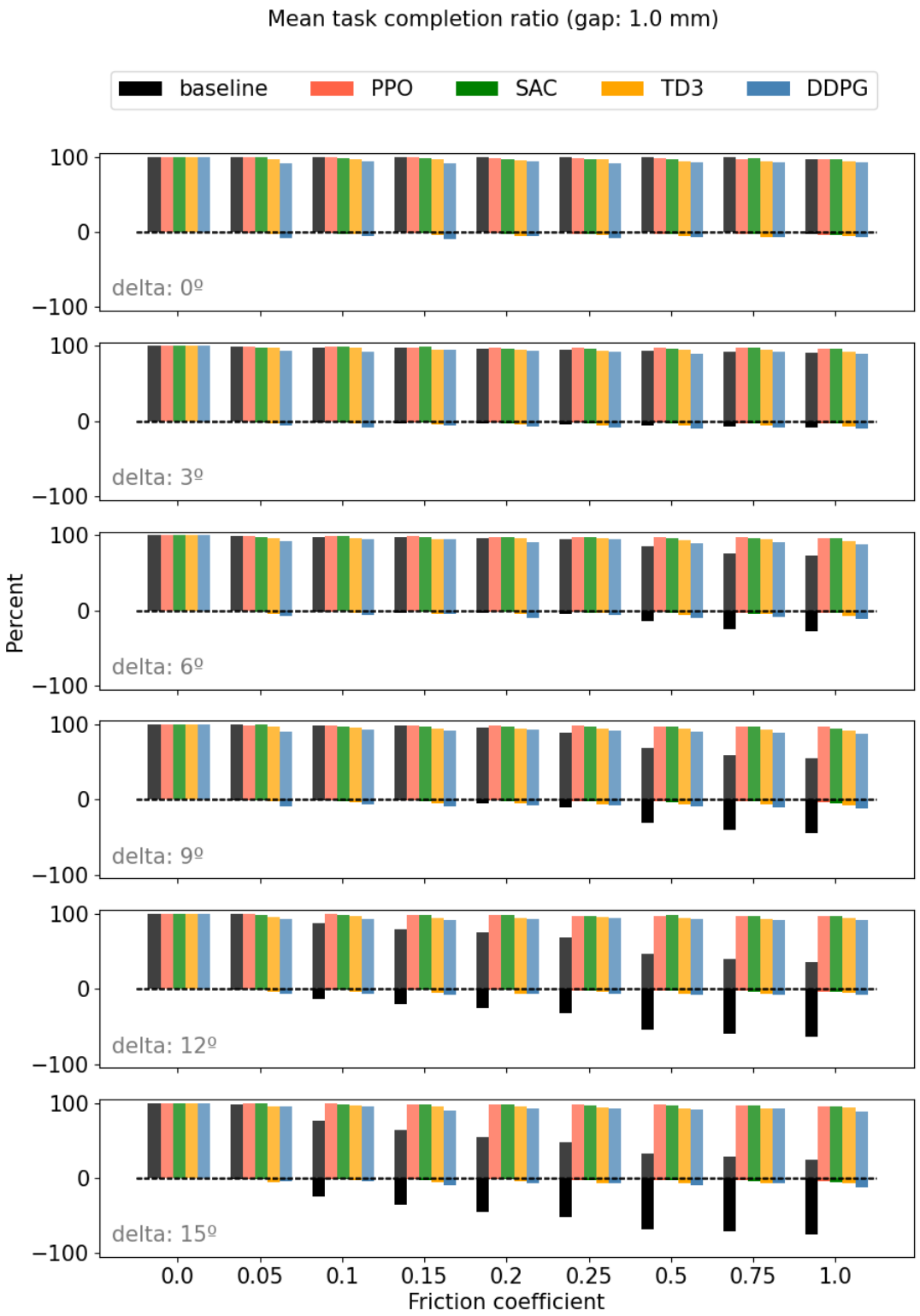

FIGURE 7.11: Task completion ratio resulting from discrimination of execution status (completed or not) for an air gap of 1.0 mm. The baseline was generated with actions that follow the generated hints with the maximum possible displacement. Negative values indicate that the execution could not be completed within the specified number of timesteps.

For friction coefficients near zero, where the probability of jamming is almost zero for actions close to the orientation direction of the fixed base, the successful execution of the task with the learned policies is completed with a high ratio. This phenomenon is also reproduced for the baseline, which is limited to following the generated hint, even for angular deviation values close to 15 degrees.

When the air gap is very small or non-existent, as the friction coefficient values increase, the success ratio of task execution for the number of defined timesteps decreases for both, the baseline and the learned policies for all angular deviations of the generated hints.

For air gaps that allow for more mobility, as the angular deviation of the provided hints increases, the performance of the learned policies is maintained even at high friction coefficient values. However, since the baseline performs actions that deviate it from the correct extraction direction, the successful completion ratio of the task decreases.

Action space distribution

Considering the distribution of the actions taken by the policy during the evaluation shown in Figure 7.12, the hints provided are also useful to reduce, considerably, the search for movements that allow the object to be extracted from the fixed base.

For the RL formulation without human hint (described in Chapter 6), the data show that although the actions are distributed around the exact extraction vector (ground truth), they are not concentrated but scattered over a range of up to 50%, 65%, and 25% of the action space amplitude for the X, Y, and Z axes, respectively. Because the reward function (used during training) does not explicitly define a direction and sense to take, part of execution is spent in finding actions that produce some motion, leaving little room for extraction once a direction is found (if it is ever found).

For the RL formulation with human hint, although there are small deviations of up to 30% of the amplitude of the action space for each axis, the decisions taken are mainly concentrated around the exact extraction vector (ground truth).

These results shows an improvement in the generalization and extraction task execution capabilities when policies are provided with a hint (albeit not an exact one) as to where to focus their actions compared to when nothing is provided, as discussed earlier in this chapter.

Extraction sense

Since the slot of the fixed base goes through it from one side to the other, it is possible to perform the extraction in two opposite senses for the same direction, which may be relevant or desirable in certain circumstances.

To carry out an analysis, in simulation and in the real world, an extraction sense is defined and the difference between this definition and the extraction sense taken by the policy during the execution of the task is calculated. If the policy performed the task in the same defined sense, the execution is marked as “right”, otherwise it is marked as “wrong”. Figure 7.13 shows the reliability, in terms of the extraction sense, of the extraction task.

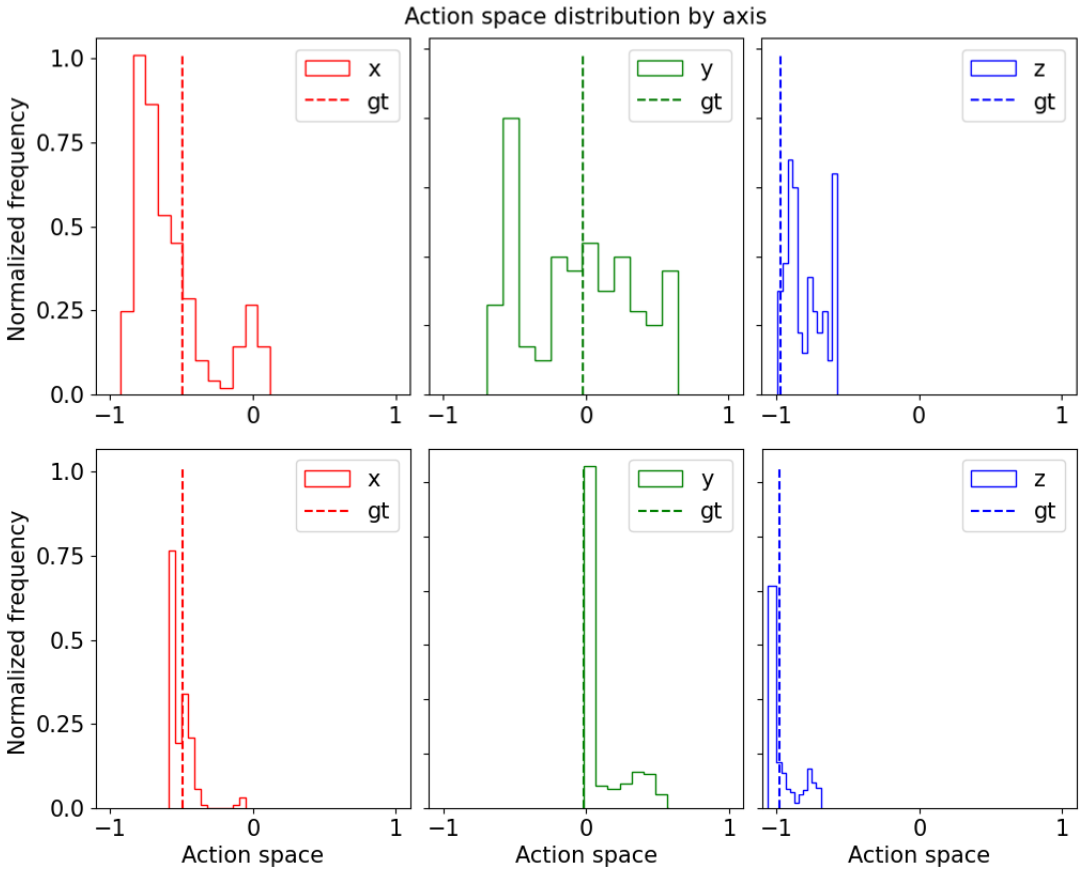

FIGURE 7.12: Action space distribution of the last 5 episodes, executed by the best policy trained without (top chart) and with human hint (bottom chart) with the PPO algorithm, for the pair of coefficients of friction and air gap (1.0, 1.0 mm) and the polar and azimuthal angles -50 and 15 degrees respectively. For reference, the components of the exact vector (ground truth: gt) of the extraction direction are plotted. The human hint has 15 degrees of angular deviation.

The results show that the trained RL system, according to the RL formulation (described in Chapter 6) does not take into account the sense of extraction during the task execution. For the evaluated algorithms, the distribution of "right" and "wrong" senses is more or less similar. Although there are slight tendencies towards execution in the same sense for all cases, these are not significant enough to guarantee the reliability of the trained system with respect to the extraction sense.

The RL formulation with human hint information, to disambiguate the execution, is able to follow such information and thus guarantee to a greater extent (compared to policies trained without the hint information) the reliability of the trained system with respect to the task extraction sense.

Transference to the real world

In addition, the study evaluated the generalization of the learned skill in the real world. For this purpose, the tests were performed for different real-world environments. The spatial position and orientation of the target objects were sampled . The best agent performed five episodes for each particular case.

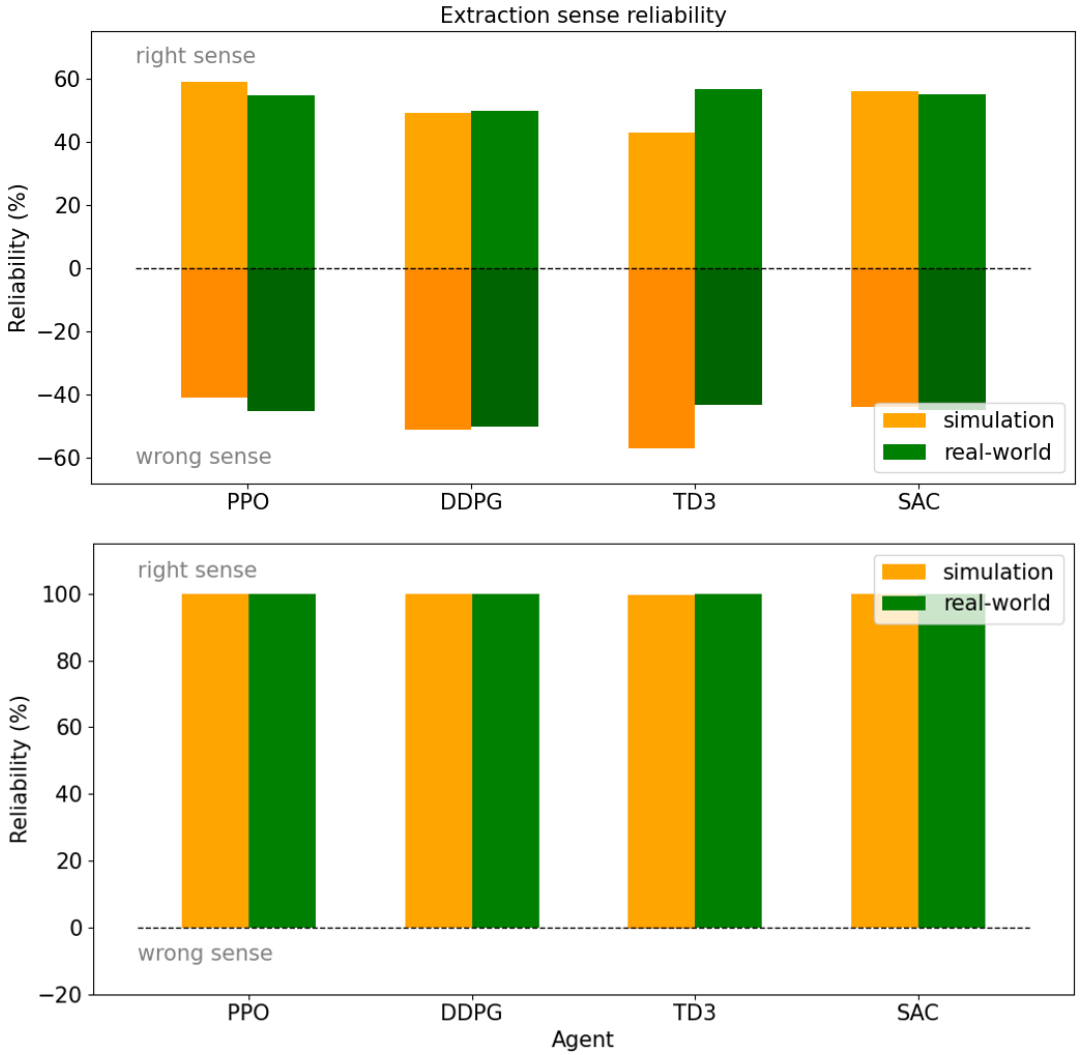

FIGURE 7.13: Reliability of the learned extraction task, without (top chart) and with human hint (bottom chart), in terms of extraction sense for both simulation and real-world evaluation.

Figure 7.14, 7.15, 7.16, 7.17 and 7.18 show a sequence of extraction for environments A, B, C, D, E respectively.

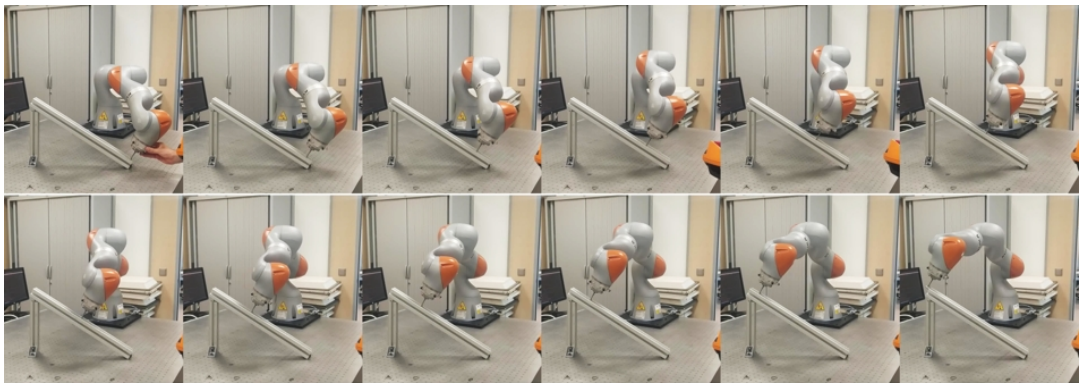

FIGURE 7.14: Extraction sequence for environment A using the best trained policy in simulation. Frames were taken each 42 timesteps.

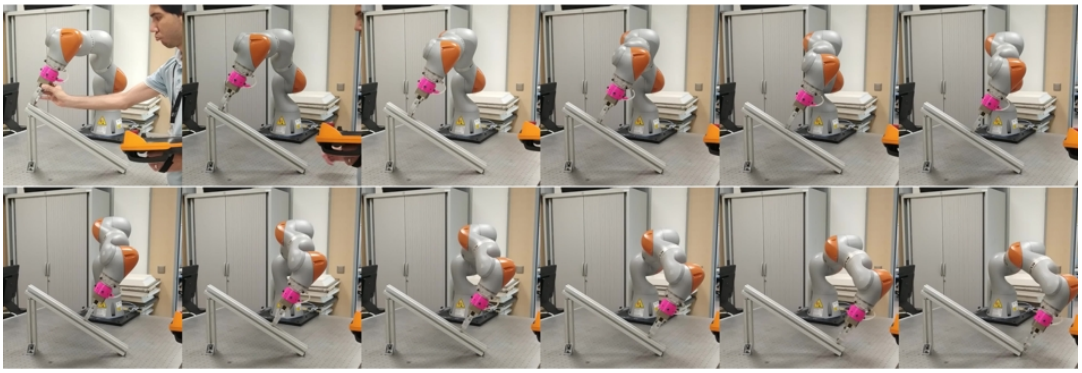

FIGURE 7.15: Extraction sequence for environment B using the best trained policy in simulation. Frames were taken each 36 timesteps.

FIGURE 7.16: Extraction sequence for environment C using the best trained policy in simulation. Frames were taken each 42 timesteps.

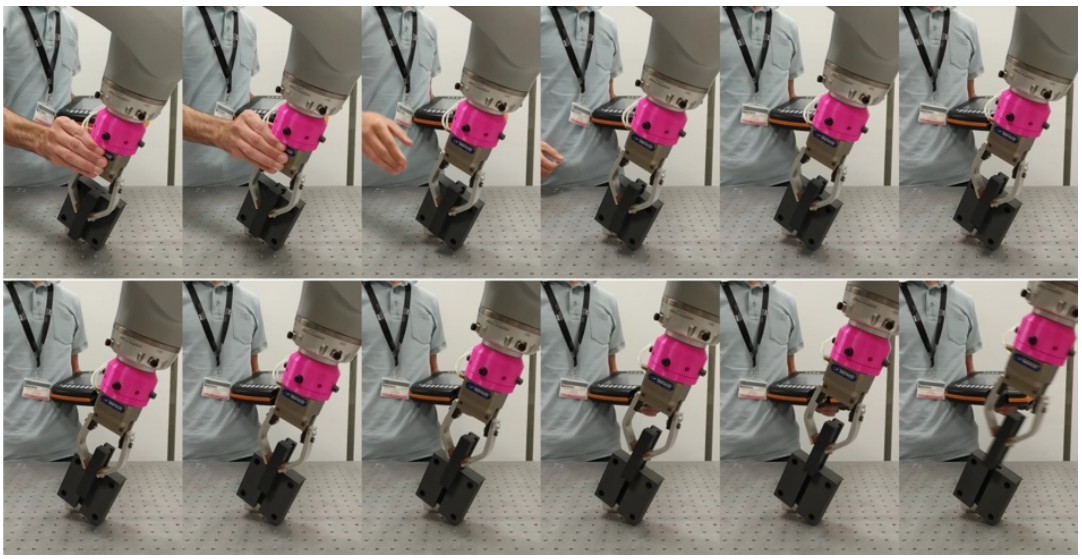

FIGURE 7.17: Extraction sequence for environment D using the best trained policy in simulation. Frames were taken each 14 timesteps.

Table 7.2 shows a record of the statistics of the evaluations performed in the real world using the baseline and the best trained policy.

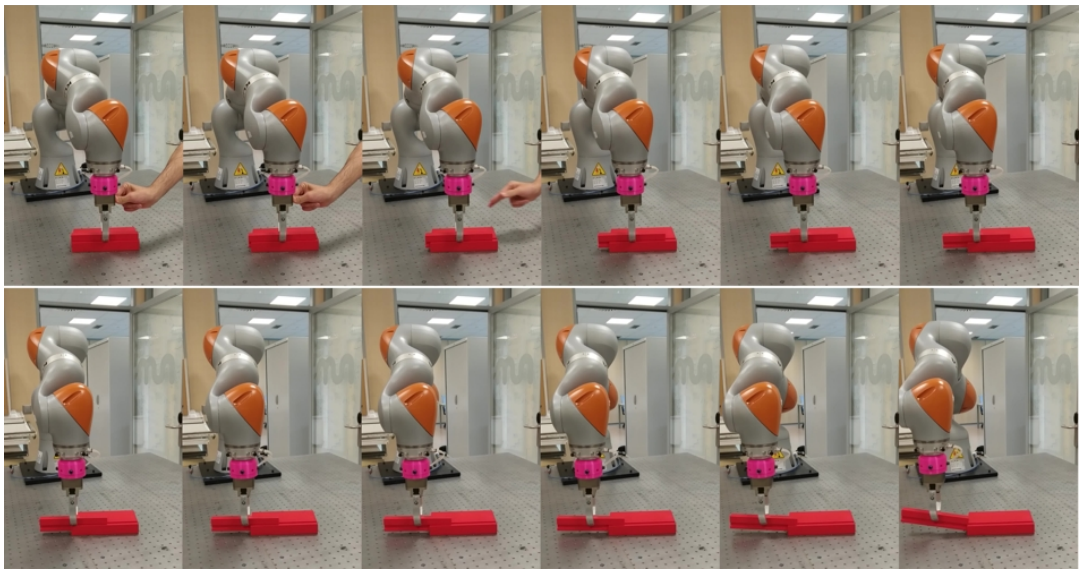

FIGURE 7.18: Extraction sequence for environment E using the best trained policy in simulation. Frames were taken each 16 timesteps.

Metric	Actor	Environments				
		A	B	C	D	E
Success ratio	baseline	1 / 5	3 / 5	0 / 5	2 / 5	4 / 5
	trained policy	5 / 5	5 / 5	5 / 5	5 / 5	5 / 5
Mean number of timesteps	baseline	282	242	-	97	115
	trained policy	254	223	257	84	94

TABLE 7.2: Real-world evaluations statistics of the baselines and trained policy. The mean number of timesteps was computed for successful episodes only.

According to the results listed in the table, the policy trained using RL executes the extraction task satisfactorily for all environments and evaluations while the baseline fails in the majority. In particular, the baseline cannot perform the extraction task for the environment *C*. This environment has pronounced curvatures that make the sole use of the information provided by the human operator insufficient to perform the disassembly.

Furthermore, the execution of the extraction task is performed, between 8% and 15%, faster using the learned policy, which is capable of adapting to variations in the environment, than the baseline.

7.2 Chapter conclusions

The research conducted responds, for the most part, to research question 1: The extended RL method for extraction disassembly tasks is able to generalize such tasks to different geometric properties and physical friction of the objects involved. Under this method, the execution of the task is completed with percentage values above 95% for all the evaluated scenarios.

The introduction of the operator's experience to generate hints that indicate (even if not exactly) where the extraction should be performed and the redefinition of the reward function are key not only to generalize the task, but also to significantly reduce the search for movements to extract the objects and to guarantee to a greater extent the reliability of the trained system with respect to a given extraction sense.

Chapter 8

Towards the Implementation of Disassembly (and Assembly) Tasks Using Ready-to-Use Packages for Rapid Robotic Prototyping

The integration of robotic automation into industrial processes around the world has become essential to improve production speed and accuracy while reducing operating costs. However, as processes demand greater flexibility and agility, new obstacles and challenges have arisen in the use of robots, particularly with regard to the significant engineering costs of work cells.

These costs, in terms of resources and time, can include the design, manufacture, and installation of equipment and accessories such as conveyors, sensors, and third-party software, for example.

This chapter proposes the creation of a collection of ROS and non-ROS packages that can be used to prototype such projects quickly. The packages are designed to reduce the development and deployment time of robotic systems. In addition, they are designed to provide users with a high degree of customization and flexibility to accelerate the creation and testing of collaborative robotic disassembly and assembly systems.

8.1 Solutions for creating robotic systems

Some solutions exist to accelerate the creation of robotic systems, which can help reduce development time and costs while improving efficiency and flexibility.

Simulation

Robotic simulation has become a fundamental part of the development and testing of new robotic systems, as well as robotics research [172].

Computer simulation allows engineers and personnel to model their designs in a virtual environment. This practice helps to dramatically reduce the time and cost associated with the design and development of robotic solutions, enabling companies to bring new robotic systems to market more quickly and efficiently [173][174]. It also makes it possible to identify problems or limitations of a design before it is

physically built, allowing adjustments and improvements to be made early in the development process.

Simulations can be used to test a wide range of scenarios, environments, and working conditions (which may be difficult or impossible to replicate in the real world), allowing the performance and capabilities of a robotic system to be fully evaluated prior to deployment and reducing the risk of damage during the operational phase [175][176].

In summary, robotic simulation has proven to be a powerful tool for accelerating the development of robotic systems, allowing designers and engineers to rapidly iterate and improve their designs, ultimately resulting in more capable, reliable and cost-effective implementations. Although robotic simulation is an important tool for developing such systems, it also has some limitations. One of the most important is that it cannot fully reproduce the complexity and unpredictability of the real world [177][178].

Ready-to-use packages

Ready-to-use packages are pre-written, pre-built if applicable, and tested software components that can be easily integrated into robotic systems. These packages can include libraries, algorithms, controllers and drivers, and other software components that are commonly used in robotics development.

By using ready-to-use packages, developers can significantly reduce the time and effort required to develop complex robotic systems, as they do not have to build everything from scratch and can focus on the unique aspects of their systems. In addition, they can be especially useful for developers who do not have in-depth knowledge of all the software components required to create a robotic system and leverage the expertise of other members of the community.

The standardized framework for building and integrating robotic systems provided by ROS, an open source platform, is a critical aspect of the development process. The platform comes with a large collection of ready-to-use packages developed and maintained by a broad community of experts. ROS also provides tools and infrastructure that support the development and sharing of new packages [56]. This allows developers to contribute their own software components to the robotics community, fostering collaboration and innovation in the field.

Among the most relevant ROS packages (or frameworks) is *MoveIt*, which focuses on motion planning, manipulation, kinematics, and control [152][179]. In addition, *Nav2* provides perception, planning, control, localization, visualization, and other capabilities for building reliable mobile autonomous systems [180].

Examples of notable ROS package collections are *ros_control* and *NVIDIA Isaac ROS*. The first one is a set of packages, originally developed for the PR2 robot, that includes a controller interface, controller manager, gears, and other tools to make controllers generic to all robots [153]. The last one is a collection of individual hardware-accelerated packages for image processing and computer vision and perception using AI [181].

While the large collection of ready-to-use ROS packages can significantly speed the creation of robotic systems, it can also create challenges related to integration, consistency, and scalability.

With so many packages available, it can be difficult to determine which ones are best suited for a particular project and how to integrate them into a cohesive system. It is important to ensure that each package is properly integrated into the overall system and that there are no conflicts or compatibility issues. This can be especially difficult when working with multiple packages developed by different people and organizations [59]. In these scenarios, it is critical to pay special attention to architecture and design to ensure that robotic systems are reliable, efficient, and easy to maintain, even as they grow in complexity.

Skill-based system

Skill-based systems provide an easy-to-use interface for programming robots, allowing users to define tasks and goals using natural language or other intuitive modes without the need for specialized programming skills. These systems typically operate at three abstraction levels. The primitive level represents the lowest level of abstraction, defining actions that are directly executed by the robotic system. The skill level combines different primitives to create more complex behaviors. Finally, at the task level, higher-level goals and objectives are defined. Using this approach, users can program the system without having to worry about the specific details of how these behaviors will be executed.

Some solutions have been developed on top of ROS. *SkiROS* [182] is a framework for knowledge integration and autonomous mission execution. For the cognitive part, symbolic representations designed by the personnel are formalized through an ontology. A Graphical User Interface (GUI) helps to organize the robot knowledge and its behavior. In the same vein, *SkillMaN* [183] proposes a planning and execution framework for mobile robotic manipulators. Such a system also has the ability to store experience-relevant information for later use, and a recovery module that provides knowledge for interpreting failures and suggests recovery strategies. However, manual processing is required to generate semantic descriptions for the task at hand.

In [184] a way for combining the skill based programming with an architecture based on finite state machines is proposed for controlling dual-arm robots. In this work, the ROS parameter server is used to enable the communication between skills. The implementation allows for sequential execution of skills, but lacks task modeling and scheduling.

Another solution is the *Skill Based System* (SBS) [185][186][187], a task-level programming software tool intended for use on both stationary and mobile collaborative robots. This development comprises the three abstraction layers (task, skill and device primitives) and services (use of multiple devices to provide advanced functionality to the skill layer), and provides a GUI to program, teach, monitor and control the tasks. ROS is employed to enable communication between most of the software nodes. The system has been used in several research projects [187] and for for performing a screwing tasks [186].

REpac, an extendable framework for creating skill-based industrial robot applications is presented in [188]. This framework allows robot skills to be provided and shared via a central marketplace and to be freely combined in the application. The skills have to be programmed to be compatible with three extension stages: loading and registration, knowledge management, and automated task planning. Although

the system enables easy configuration of the skills, the development cost is high compared to the subsequent benefits in operation.

Although skill-based systems are designed to be used by operators with no experience in robot programming, they require expert personnel to develop and maintain, whose knowledge extends beyond their base, such as ROS, as well as a significant amount of time. Such systems may not be as flexible or customizable and may not provide the level of control and precision needed for more complex or specialized tasks.

8.2 Framework description

This document presents an organized rich collection of ready-to-use, modular, and highly configurable ROS and non-ROS packages to accelerate the creation of robotic systems for collaborative manufacturing and remanufacturing projects.

This implementation provides a set of packages specifically designed to address the integration, consistency, and scalability challenges that often arise when working with multiple packages developed by different people and organizations. By being generic and highly configurable, the packages can be easily adapted to the specific needs of different projects, ensuring a cohesive and reliable overall system.

In addition, the packages in this collection can be integrated into the solutions described above. For example, they can form part of a simulation system, be combined with other ready-to-use packages, or provide additional functionality to existing skills-based systems.

Three elements are key to this: First, the definition of a common structure for the packages. Second, the semantic naming of packages. And third, the use of control version utilities for storage, sharing and composition of projects. These elements are discussed in the following subsections.

8.2.1 Package structure

Having a common structure for packages is critical when developing modular applications. This standardization enables seamless integration and compatibility between different packages, resulting in a more efficient and reliable system. In addition, a common framework streamlines the programming process and promotes a better understanding of package operations, making them more reusable across projects. Consistency in package design also improves documentation and support by allowing developers to leverage prior knowledge and experience to create more robust and trustworthy packages.

The packages structure is based on the common structure that ROS packages[1] must follow. Some folders and files are added to this structure to ensure integration with repositories and version control systems, compiled files and installation and configuration scripts as show in Table 8.1 and Table 8.2.

Group	Name	Description	Type
	bin	Binaries files	folder

[1]ROS packages: http://wiki.ros.org/Packages

C/C++ code	include	Headers exposed for public consumption	folder
	src	Source code	folder
Python code	scripts	Script	folder
Test	test	Unit tests	folder
ROS messages	msg	ROS topic definition (custom messages)	folder
	srv	ROS service definition (custom messages)	folder
	action	ROS actionlib definition (custom specification)	folder
ROS launch	launch	ROS launch files	folder
Git project	.git	Git repository	folder
	.gitignore	List of files and folder to be ignored	file
	.gitmodules	Git submodule definitions	file
	README.md	Information about the project	file
	LICENCE	License file	file
ROS package	package.xml	Catkin package manifest	file
	CMakeList.txt	CMake's input for building software packages	file
Setup	install.bash	Shell script to install and setup the package	file

TABLE 8.1: Directory and file structure of ROS packages.

Group	**Name**	**Description**	**Type**
C/C++ code	bin	Binaries files	folder
	include	Headers exposed for public consumption	folder
	src	Source code	folder
Python code	scripts	Script	folder
Test	test	Unit tests	folder
Git project	.git	Git repository	folder
	.gitignore	List of files and folder to be ignored	file
	.gitmodules	Git submodule definitions	file
	README.md	Information about the project	file
	LICENCE	License file	file
Setup	install.bash	Shell script to install and setup the package	file

TABLE 8.2: Directory and file structure of non-ROS packages.

8.2.2 Semantic package naming

The use of meaningful and descriptive names provides a structured approach to organizing packages, which facilitates the management and maintenance of software systems. Such semantic naming helps convey the purpose and functionality

of each package, allowing developers to easily identify and locate implementations that meet their needs.

The names are assigned under the format `skros_CATEGORY_SUBCATEGORY_NAME`.

Category

In *skros*, categories are named using a semantic taxonomy inspired by the abstraction layers of skill-based systems. In this sense, the following categories are defined:

- Device: this refers to a physical component or tool that is integrated into the overall system to perform specific functions. Devices can include robots, sensors, actuators, among others that are used to observe and interact with the environment.
- Service: this refer to software implementations that process or return some type of data during execution and are not tied to a specific physical device. Note that although they have the same name, they are not related to ROS services (a communication protocol for sending and receiving requests and responses).
- Extension/Extra: this refer to software implementations that provide some specific functionality that does not fall into the above classifications. For example, they may be simulation files or scripts to perform pre- or post-processing of some data in an offline fashion.
- Task: this is a high level of organization that groups, under the same directory (ROS workspace), one or more of the above elements as needed, as well as external packages or other files, to support the execution/control of some simple or complex process. Typically, it contains a file, a main program, or a state machine that controls the operation of the system.

Subategory

Subcategories are labels that help organize similar components under the same semantic name. This allows for better organization, search, and initial understanding of the packages. Examples of such subcategories for the "device" category could be the terms actuators, cameras, robots, etc. to help organize the different types of devices.

Name

The name, as the term implies, provides a unique identifier for each type of component within the appropriate categories and subcategories.

8.2.3 Storage, exchange and project composition

Each package is conceived as a main directory that includes other directories and files. Such packages are built as Git[2] repositories (a distributed version control system) and are stored and shared through GitLab (cloud repository hosting site).

Building the packages as repositories allows for issue tracking, code review, continuous integration and deployment, as well as integrated wiki documentation.

To compose tasks (and projects), Git submodule[3] is used. This is a feature of Git that allows to include a repository as a subdirectory of another repository, keeping them separate. This composition makes it possible to design tasks that follow the architecture shown in Figure 8.1.

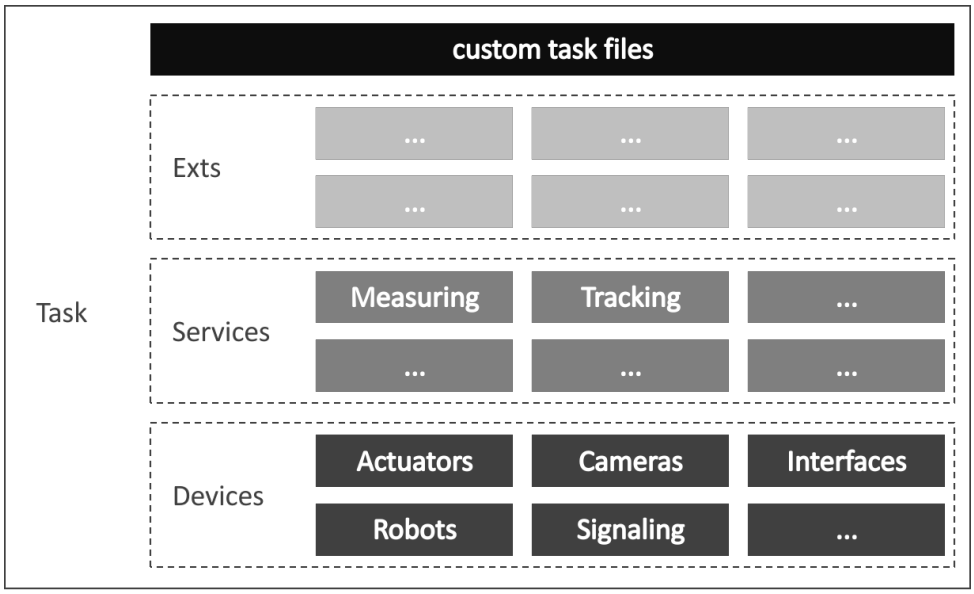

FIGURE 8.1: Skros's task architecture.

Using Git submodules for task composition provides certain flexibility and advantages when managing complex compositions. Among them are:

- Modularity: Using submodules allows breaking down a complex project into smaller, independent modules or components.
- Dependency management: Submodules provide a convenient way to manage dependencies between different components of a project. This reduces the risk of conflicts and compatibility issues.
- Version control: Each submodule has its own repository, allowing version control on an individual basis. This makes it easier to manage and update specific components without affecting the entire project.
- Simplified project configuration: Project configuration allows reusable loading of dependencies, ensuring that the correct versions of submodules are extracted and configured correctly.
- Collaboration: Submodules enable collaborative development but maintain a centralized project structure that facilitates integration of changes.

[2]Git: https://git-scm.com
[3]Git submodule: https://git-scm.com/book/en/v2/Git-Tools-Submodules

8.3 Main framework components

A description of the primary components, grouped by category and subcategory, that have been developed to the time of this writing follows.

8.3.1 Devices

This section lists and describes the main devices implemented in the following subcategories.

8.3.1.1 Actuators

Actuators are components of an automation system that convert electrical, hydraulic, or pneumatic signals, for example, into physical motion. Actuators are used to control various mechanical devices in automated systems, such as valves, motors, and others.

Gripper

The *GripperCommand* is a ROS Control action message to send commands to a gripper. By setting a maximum force value that can be applied by the gripper, this action interface makes it easy to control those actuators [179].

The ROS package *skros_device_actuator_gripper* provides an implementation for the *GripperCommand* action service. Figure 8.2 shows the graphical representation of the main node of the ROS package (device:gripper) as well as possible nodes involved and their topics.

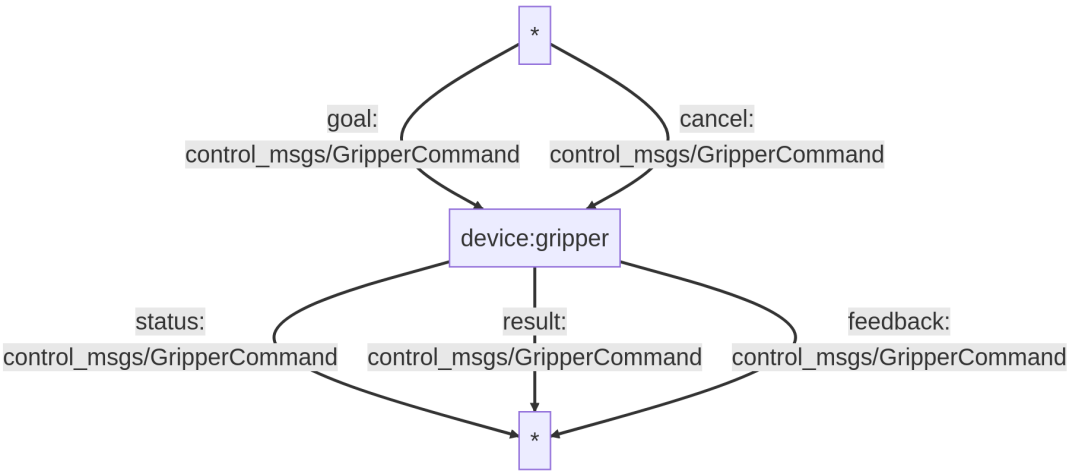

FIGURE 8.2: Graphical representation of the nodes and topics involved in the ROS *skros_device_actuator_gripper* package.

The node provides only a communication interface to the ROS environment. It is the user's responsibility to implement the interface between the node and the specific control hardware for the gripper or similar device. To assist in the creation of the hardware access interface, a code for using the general-purpose input/output (I/O) pins of the NVIDIA Jetson Nano is provided.

For example, Figure 8.3 shows the physical deployment of the controller using such a package on a high gripping force light industrial pneumatic gripper.

FIGURE 8.3: A pneumatic gripper with generative designed fingers printed using additive manufacturing for disassembly and its control system.

8.3.1.2 Cameras

Cameras (digital cameras) are devices used in automation to capture images or video of the environment, such as products, parts, or components. Cameras can be installed at various points along the production line or on robots to provide visual information to the system.

USB Camera

USB cameras (or webcams) are digital cameras that use USB technology, typically USB 2.0 or USB 3.0, to transfer image data to a computer or other device. They may have high-resolution image sensors, the ability to adjust focus and exposure, and support for various image and video formats.

They are often used as webcams for videoconferencing, but can also be used for other applications, such as video recording, security and surveillance, scientific research, and industrial automation. They are versatile imaging devices compatible with a wide range of operating systems such as Windows, MacOS and Linux, and with a wide range of different manufacturers and models, from simple and affordable cameras for personal use to very advanced cameras for professional and industrial use.

The ROS package *skros_device_camera_usbCamera* provides a wrapper for the ROS *cv_camera* package. The *cv_camera* package uses the open-source computer vision library OpenCV [189] capture object to capture the image from various sources such as USB cameras, remote cameras in steaming, video files, among others. This package also allows to configure various parameters and image formats. Figure 8.4 shows the graphical representation of the main node of the ROS package (device:usbCamera) as well as possible nodes involved and their topics.

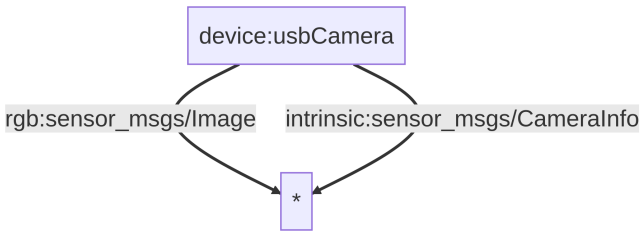

FIGURE 8.4: Graphical representation of the nodes and topics involved in the ROS *skros_device_camera_usbCamera* package.

The wrapper has two functionalities. The first functionality allows the installation of the base ROS package by executing all the necessary steps to obtain and install dependencies from a single file. The second one, allows to unify under a common framework, components whose name and topics have an understandable semantics for the usability and scalability of the system.

RealSense

RealSense cameras are a family of high-resolution depth-sensing cameras that include infrared sensors in addition to RGB sensors as shown in Figure 8.6, developed by Intel Corporation [190]. These cameras use various technologies, such as laser or stereo vision, to capture depth information and enable 3D imaging and sensing.

They are compatible with various operating systems, such as Windows, Linux, and Android, and have various SDKs and APIs for developers to create custom applications, as well as ROS packages for publishing related topics. They are primarily designed for use in robotics, but can also be found in gaming, virtual and augmented reality, and 3D scanning applications.

The ROS package *skros_device_camera_realSense* provides a wrapper for the ROS *realsense2_camera* package for Intel RealSense devices connected via USB 3.0 interface. Figure 8.5 shows the graphical representation of the main node of the ROS package (device:realSense) as well as possible nodes involved and their topics.

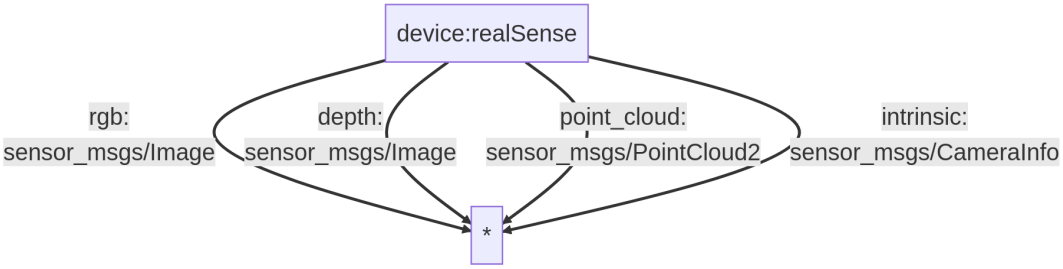

FIGURE 8.5: Graphical representation of the nodes and topics involved in the ROS *skros_device_camera_realSense* package.

The wrapper has two functionalities. The first functionality allows the installation of the base ROS package by performing all necessary steps to obtain and install dependencies from a single file. The second one, allows to unify under a common framework, the component whose name and topics have an understandable semantics for ease of use and scalability of the system.

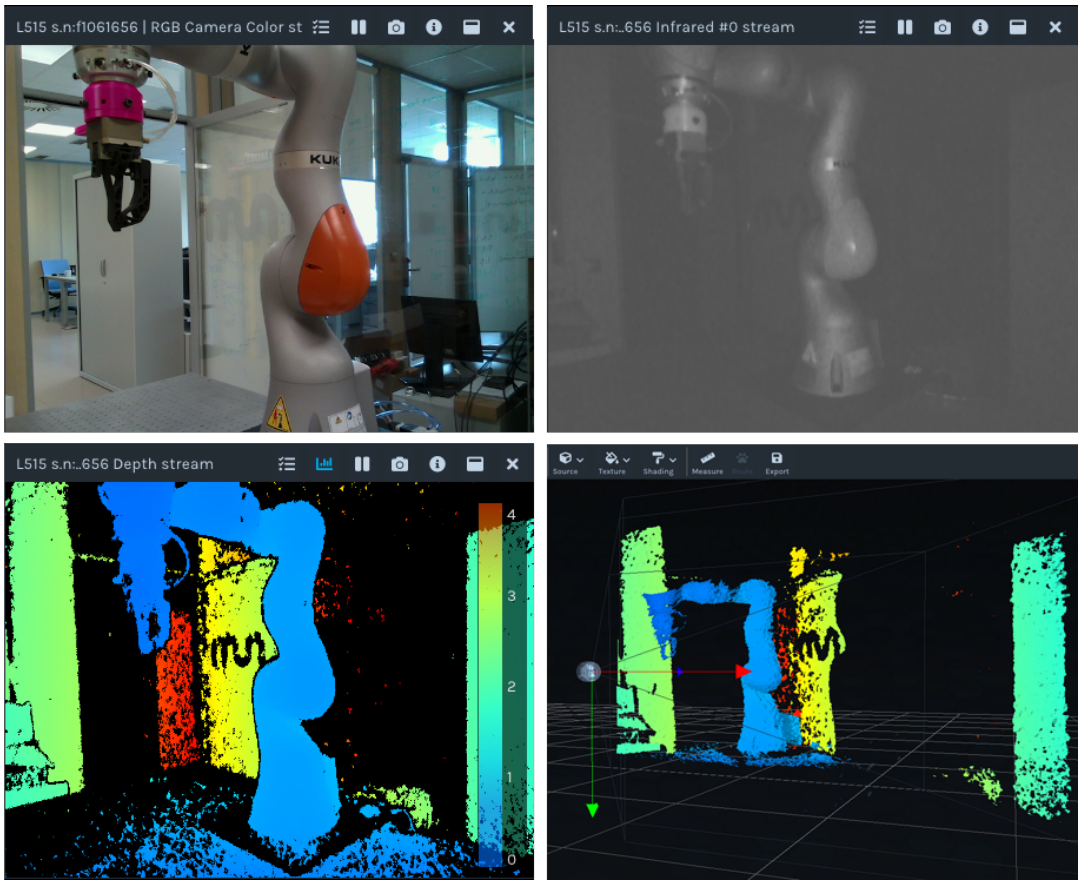

FIGURE 8.6: RealSense capture (from left to right and from top to bottom): RGB, infrared, depth (colored) and point cloud.

8.3.1.3 Interfaces

Interfaces refer to the inputs through which a human interacts with a given electronic device. Their purpose is to allow the user to operate the device intuitively and effectively, with a minimum of training or technical knowledge. They can be graphical, such as a touch screen, or voice-based, using speech synthesis and speech recognition technologies, for example.

TedCube

The TedCube is a touchless natural user interface developed by TedCas, a healthcare technology provider. This device allows healthcare professionals to operate any medical system using voice commands and hand gestures. The system uses Microsoft Kinect and the Myo Armband gesture control system for gesture recognition and includes an advanced user interface for managing medical instruments and accessing medical data in a hospital environment [191][192].

Although it is primarily designed for healthcare environments in smart operating rooms, its use can be extended to other scenarios such as robotics.

The ROS package *skros_device_interface_tedCube* provides a ROS interface for using the TedCube to control any device as part of a ROS environment. Figure 8.7

shows the graphical representation of the main node of the package (device:tedCube) as well as possible nodes involved and their topics.

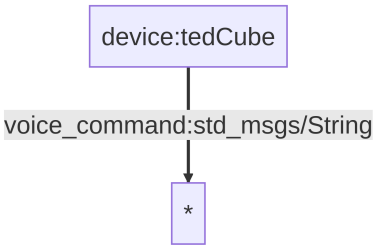

FIGURE 8.7: Graphical representation of the nodes and topics involved in the ROS *skros_device_interface_tedCube* package.

This package implements the communication with the TedCube and an initial processing step of the messages coming only from the voice control interface through a Representational State Transfer (REST) API. Although not yet implemented, gesture control could be easily integrated. The data are published to the ROS environment encoded in JavaScript Object Notation (JSON) format. The Figure 8.8 show a sample of a custom voice control configuration.

In addition, to facilitate testing or system integration, a fake interface based on an easily configurable Graphical User Interface (GUI) is provided through an input file that maps buttons to voice commands. This functionality allows you to use the TedCube without being physically present.

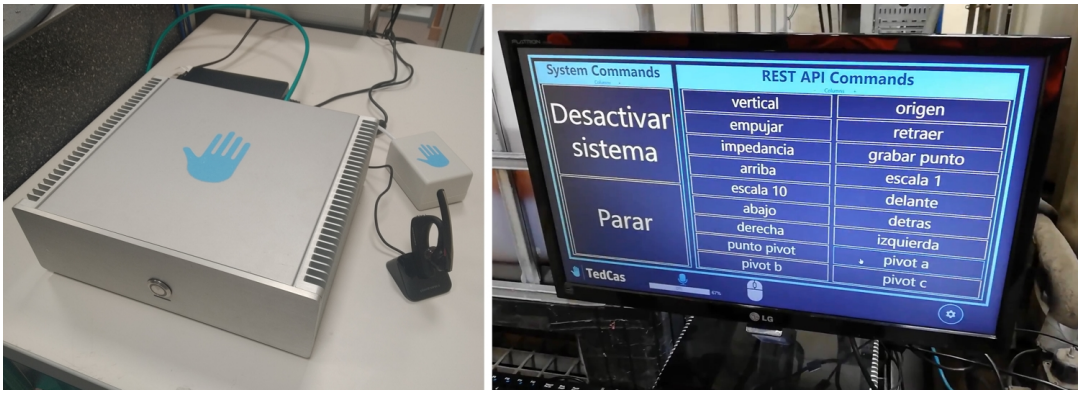

FIGURE 8.8: A TedCube device (on the left) and an example of a voice interface configuration (on the right).

8.3.1.4 Robots

A robot is a sophisticated machine that can perform complex tasks either under remote control or autonomously. Typically, a robot consists of a set of actuators, sensors, controllers, and a communication system that work together to achieve the desired functionality.

KUKA LBR iiwa

The KUKA LBR iiwa is a seven-degree-of-freedom serial robotic manipulator with force/torque sensors along each axis. This sensor distribution allows both position and impedance control, providing compliant behavior in force-sensitive tasks as described in Chapter 5.

The ROS package *skros_device_robot_libiiwa* provides a wrapper for the ROS *libiiwa* package. Figure 8.9 shows the graphical representation of the main node of the ROS package (device:libiiwa) as well as possible nodes involved and their topics.

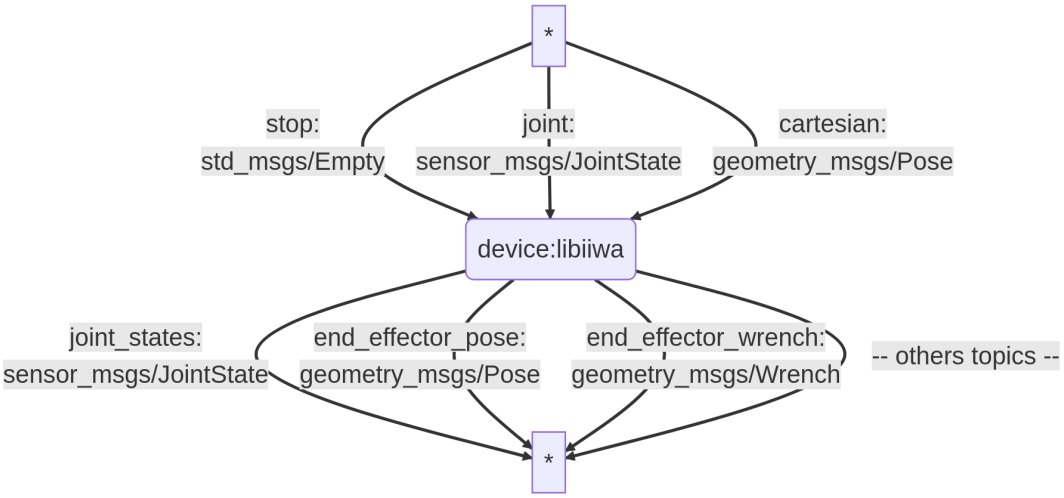

FIGURE 8.9: Graphical representation of the nodes and topics involved in the ROS *skros_device_robot_libiiwa* package.

The wrapper has two functionalities. The first functionality allows the installation of the base ROS package, and not of the other *libiiwa* components, performing all the necessary steps to obtain and install the dependencies from a single file. The second one, allows to unify under a common framework, the component whose name and topics have an understandable semantics for ease of use and scalability of the system.

8.3.1.5 Signaling

Signaling is the exchange or provision of information to the operators or machines of an automated system. It can be essential for coordinating the behavior of different parts of a system, as well as for providing information about a particular work status or safety condition.

Industrial signal/light tower

Industrial light towers are a typical and useful way to implement light signaling in automation. They typically consist of a vertical mast with multiple colored light sources attached. In addition, they may include a buzzer sound emitting device. Such towers can be mounted on a machine or at a specific location in a plant to provide visual signaling to operators or machines.

They can provide a variety of signaling information, including status, safety and communication information. Table 8.3 lists the colors of the indicator lights and their meaning with respect to the machine status as described in IEC 60204-1 [193]. The

light towers must have the applicable colors in the same order, from top to bottom, as described in that table.

Color	Meaning	Explanation	Action by operator
Red	Emergency	Hazardous condition	Immediate action to deal with hazardous condition (for example switching off the machine supply, being alert to the hazardous condition and staying clear of the machine)
Yellow	Abnormal	Abnormal or impending critical condition	Monitoring and/or intervention (for example by re-establishing the intended function)
Blue	Mandatory	Indication of a condition that requires action by the operator	Mandatory action
Green	Normal	Normal condition	Optional
White	Neutral	Other conditions. May be used whenever doubt exists about the application of the other colors	Monitoring

TABLE 8.3: Colors for indicator lights and their meanings with respect to the condition of the machine. Content adapted from the IEC 60204-1 standard.

The ROS package *skros_device_signaling_lightTower* provides a ROS implementation to handle up to five-color generic light towers and an alarm buzzer. Figure 8.10 shows the graphical representation of the main node of the ROS package (device:lightTower) as well as possible nodes involved and their topics.

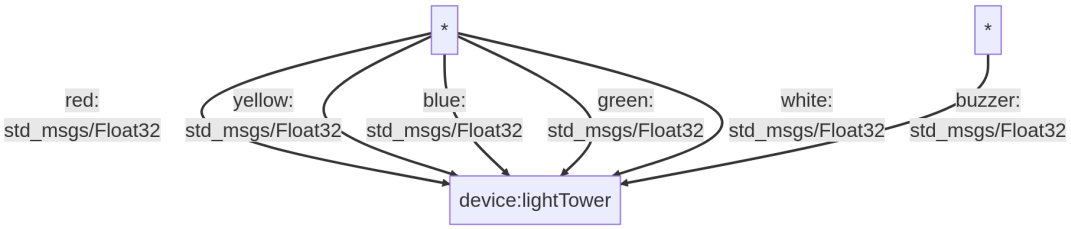

FIGURE 8.10: Graphical representation of the nodes and topics involved in the ROS *skros_device_signaling_lightTower* package.

The implementation allows, by setting a floating precision value, to disable (0) or enable each light independently, permanently (`inf`) or intermittently (value greater than 0, in seconds).

The node provides only a communication interface to the ROS environment. It is the user's responsibility to implement the interface between the node and the specific control hardware for the light tower or similar device. To assist in the creation of the hardware access interface, a code for using the general-purpose input/output (I/O) pins of the NVIDIA Jetson Nano is provided.

Light projector

Projection of light or images onto a work area is a technique that consists of projecting images, symbols, or messages onto a workpiece or work area. The goal is to provide workers with visual guidance or information to help them perform their tasks more efficiently and accurately [194], or to provide safety messages or warnings to prevent accidents and ensure a safe work environment [195][196].

The ROS package *skros_device_signaling_lightProjector* provides an implementation for projecting colors or patterns from a ROS environment. Figure 8.11 shows the graphical representation of the main node of the ROS package (device:lightProjector) as well as possible nodes involved and their topics.

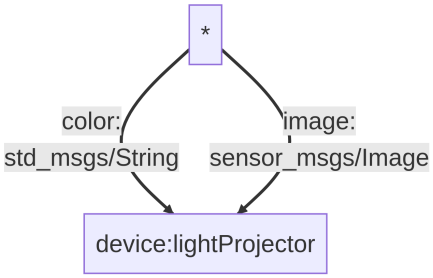

FIGURE 8.11: Graphical representation of the nodes and topics involved in the ROS *skros_device_signaling_lightProjector* package.

The node generates a full-screen display of the input information on the device on which it is located. This visualization can be displayed on the work environment using a conventional or laser projector as shown in Figure 8.12.

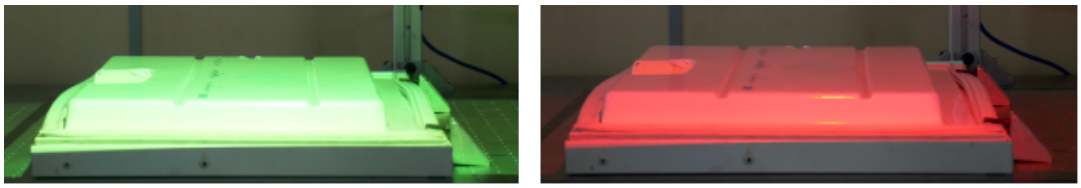

FIGURE 8.12: Example of illumination of the work area using a conventional light projector to indicate different conditions.

The input information can be a string with the color name or its hexadecimal value of the X11 standard colors [197], or an RGB image. Through the continuous publication of images (frames) to this node, it is possible to display videos.

8.3.2 Services

This section lists and describes the main services implemented in the following subcategories.

8.3.2.1 Measuring

Measurement in the context of services is the process of determining or estimating numerical values associated with the size, quantity, or degree of something, for example, using a standardized system of units.

Transformation distance

ROS transform is a powerful feature designed to provide a standardized way to represent and manage coordinate frames in a robotic system and to perform transformations between them in an easy and efficient way [198].

The ROS package *skros_service_measuring_tfDistance* implements a node that calculates and publishes the distance between two specified links. This can be useful in applications where monitoring the distance between two elements is required, for example, to make a safety decision. Figure 8.13 shows the graphical representation of the main node of the ROS package (device:tfDistance) as well as possible nodes involved and their topics.

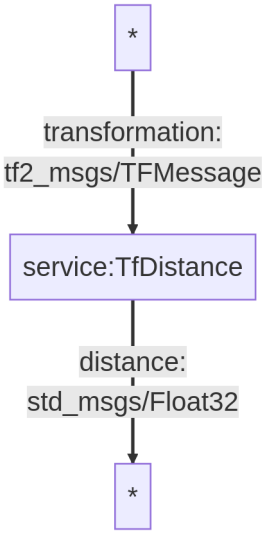

FIGURE 8.13: Graphical representation of the nodes and topics involved in the ROS *skros_service_measuring_tfDistance* package.

The implementation internally subscribes to the `/tf` through the standard ROS package *tf2_ros*, which provides an API to create and use a transform listener to perform query operations on the ROS transform.

The selection of links between which the distance is to be calculated can be configured through the node's own launch file. This allows multiple instances with different configurations to be launched to calculate the distance between different links.

8.3.2.2 Tracking

Tracking refers to the ability of a system to follow or track a moving object or its trajectory. This can be accomplished by using a variety of sensors and algorithms that provide data on the object's position, velocity, and acceleration, for example.

Skeleton tracking

Human skeleton (or skeletal) tracking refers to a method employed in the field of computer vision and image processing that enables real-time tracking of human movements by detecting and localizing the joints of the human body [199][200].

The *skros_service_tracking_skeletonTracking* package allows to detect and localize the joints of the human body on an image in RGB format and to publish, to the ROS transformation, the spatial pose of the identified skeletons. Figure 8.14 shows the graphical representation of the main node of the package (service:skeletonTracking) as well as possible nodes involved and their topics.

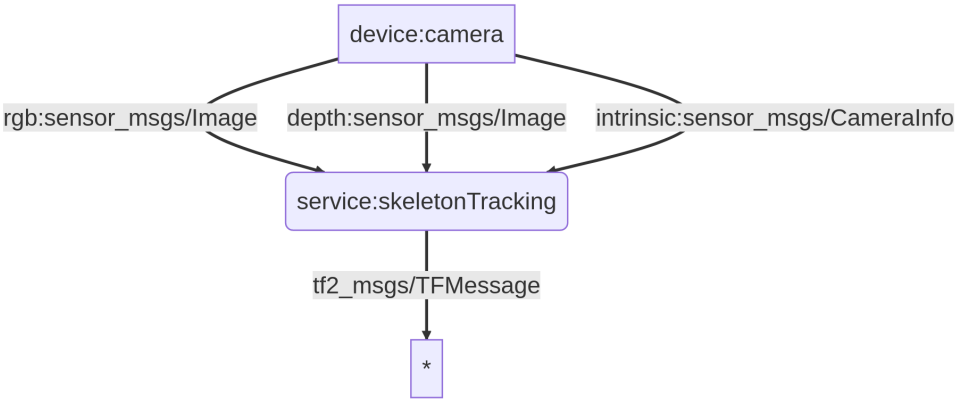

FIGURE 8.14: Graphical representation of the nodes and topics involved in the ROS *skros_service_tracking_skeletonTracking* package.

The logic behind the functionality of such a package consists of two components: 1) the detection of the human skeletons in a color image and 2) the spatial deprojection of the identified joints using a depth image.

For the first phase, the implementation supports the switching between two backends to estimate the location of the human skeletons present in a frame as shown in Figure 8.15:

- YOLOv7: version 7 of the YOLO (You Only Look Once) family of one-step real-time object detection algorithms introduced in July 2022 [201]. Skeleton detection was added as an extra feature in that release.
- MediaPipe: an open-source and cross-platform framework designed to create machine learning pipelines for live and streaming media applications [202]. Although it has a lower precision compared to YOLOv7, unlike the latter, it is possible to perform real-time identification on the CPU. This is especially useful for testing on systems that do not have dedicated hardware for algorithm acceleration, such as GPUs.

To allow compatibility between both backends, only the following 17 common joints are used in the subsequent pipeline: *nose*, *left_eye*, *right_eye*, *left_ear*, *right_ear*, *left_shoulder*, *right_shoulder*, *left_elbow*, *right_elbow*, *left_wrist*, *right_wrist*, *left_hip*, *right_hip*, *left_knee*, *right_knee*, *left_ankle*, *right_ankle*.

For the deprojection and joint spatial pose estimation phase, *pyrealsense2*, a Python wrapper of the Intel RealSense SDK 2.0 [190] for accessing these cameras, is used. This library allows to compute the corresponding point in 3D space relative to the

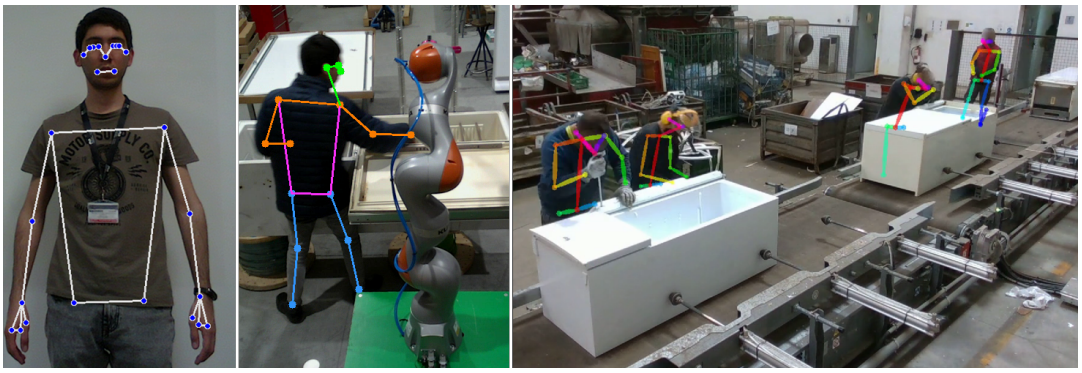

FIGURE 8.15: Estimation of human skeleton positions using the MediaPipe (left) and YOLOv7 (middle and right) backends.

camera, given the pixel coordinates, the depth in the image and the distortion coefficients. After that, the spatial points are grouped and published to the ROS transformation (`/tf`).

Since no estimation of the human pose is made based on the modeling of the skeletal structure of the human body, only the position of the joints can be determined, not their rotation. All the rotations are set to the default value $wxyz$ $[1, 0, 0, 0]$ as quaternion.

8.4 Industrial/research tasks applications

skros has been developed, tested and used in the following industrial/research projects.

8.4.1 Projects

8.4.1.1 chARmER

The Assistive Robotic Disassembly System for Recycling (chARmER) project envisions the creation of a sustainable manufacturing system that enables flexible, hybrid human-robot automation for disassembly processes. This approach can not only generate a business from EOL products, but also reduce their environmental impact while improving working conditions for people with physical and intellectual disabilities.

The consortium of this project comprises five, Spanish and Danish, companies who have been awarded the SMART S0218 seal (from European Union's SMART EUREKA programme). These companies have collaborated to pool their expertise in various areas such as robotics, human-robot interaction, AI, disassembly, natural language, manufacturing, and occupational health, among others, to develop the proposed solution.

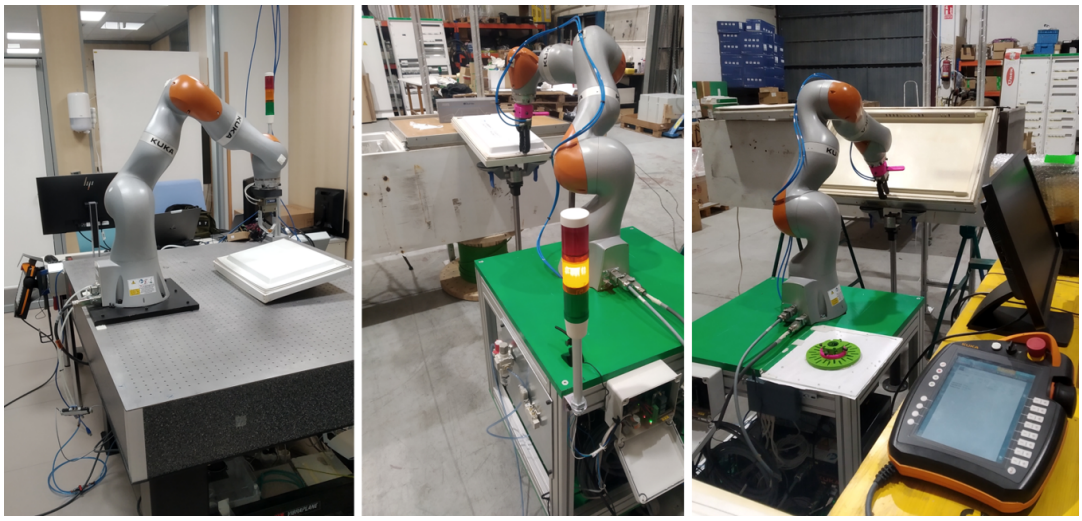

FIGURE 8.16: Demos presented during the chARmER project: laboratory prototype (left), factory prototype (center and right).

8.4.1.2 VALU3S

The objective of the European Union's Horizon 2020 ECSEL Joint Undertaking project Verification and Validation of Automated Systems' Safety and Security (VALU3S)[4] is to evaluate the latest verification and validation (V&V) methods and tools and to create a multi-domain framework to provide a clear structure for the various components and elements required to perform the V&V process. The primary benefit of this framework is to reduce the time and cost required to verify and validate automated systems for safety, cybersecurity, and privacy requirements.

The VALU3S project consortium includes a combination of 25 industrial partners, 6 leading research institutes and 10 universities from 10 different countries, working together to achieve the project goal.

Within this project, *skros* has been used in the "Use Case 7: Human-robot collaboration in a disassembly process with workers with disabilities" (based on chARmER's use case) as shown in Figure 8.17.

This use case uses simulation to implement a collaborative robotic system in a disassembly process with disabled workers. The solution enables intelligent human-robot interaction and learning, as well as the use of augmented reality technologies to validate the robotic solution in different virtual scenarios before implementing it in a real environment. It uses the KUKA LBR Iiwa manipulator and ISO/TS 15066:2016 monitoring to regulate the robot's speed depending on the safety zone in which the operator is located.

8.4.2 Lessons learned

The development and utilization of the collection of ROS and non-ROS packages described above, for the development of disassembly and assembly tasks in particular and any other robotic task in general, has produced relevant feedback from which lessons have been learned. In some occasions, these feedbacks have even led

[4]VALU3S: https://valu3s.eu/

FIGURE 8.17: Real (top) and simulated (bottom) case study evaluation scenario in VALU3S.

to the redesign of certain packages to increase and improve their functionality and integration.

Regarding the creation of new packages, the development process of the current packages and their use has demonstrated the need for programming and ROS knowledge for their implementation. In addition, a careful design is required to define input/output parameters that guarantee the reconfigurability of the implemented packages for any application launched from the ROS launch files. Otherwise, it would be necessary to modify the source code at the slightest need of the target applications, limiting its generality and hindering its scalability and interoperability between different tasks or projects. Furthermore, there is the need to know Git, both for version control during the development of new packages, and for the installation and use of existing packages via Git submodules.

Regarding the use of the existing packages in the proposed collection, their reconfigurable designs and their modular and semantic conception have proven to accelerate the creation of prototypes and robotic applications for both assembly and disassembly tasks, as well as non-robotic applications that use the devices and services system for other purposes.

In general, the collection of existing packages as well as their conception and design has been well accepted both by those persons involved in their development and by those who have used or heard about them, not only within the context of the

previously mentioned industrial/research tasks applications, but also by staff and exchange students from other research centers and universities.

8.5 Chapter conclusions

skros is a rich and organized collection of ready-to-use ROS and non-ROS packages that allows to rapidly create and deploy robotic applications for the execution of robotic assembly and disassembly tasks, as well as general robotic and non-robotic applications. Its modular, parameterizable and semantic conception and design provide a high level of flexibility, reconfigurability and customization for both simulation and real-world applications.

Chapter 9

Conclusions and Future Work

9.1 Thesis conclusions

This dissertation explored RL for learning and generalization of disassembly tasks, in particular contact-rich extraction tasks, with uncertainties due to geometric variability and physical friction of the objects to be disassembled using the KUKA LBR iiwa collaborative robot.

The research conducted shows that some RL algorithms can learn object extraction skills by interacting with the environment, and can generalize these skills to different initial conditions, such as positions and rotations, as well as to different physical frictions and geometric variations, such as the air gap between parts and different lengths. The following elements are essential to this:

- The use of relative information to generalize the task execution for different initial positions and orientations, as long as the manipulation is within the robot's operating range.
- The definition of a reward function for task execution that does not restrict the policy to making decisions in a specific direction and sense, making it physically impossible to execute the tasks in other directions and senses.
- Training for short periods of time, not to learn to complete the whole task, but to learn to extract fragments and thus achieve generalization to different geometries with unknown dimensions.

Moreover, the research showed how the incorporation of the human operator's experience, as a source of information for the disambiguation of the extraction task, are key to both significantly reduce the search for movements to extract the objects as well as further guarantee the reliability of the trained system with respect to extraction in a given removal sense. In this case, the human operator provides a hint, by exerting an external force on the manipulator end-effector, as to where (in terms of direction and sense) the extraction task should be performed, with the autonomous agent making the final decision.

Under this method, the execution of the disassembly task is completed with percentage values above 95% for all the evaluated scenarios.

Also, as part of this investigation and to overcome existing barriers for RL and robotics in the domain of applied engineering, three major developments were made:

- Given that training RL algorithms can be time-consuming and require considerable computational resources to handle complex environments and high-dimensional state and action spaces, *skrl* a modular RL library was created. This library focuses on readability, simplicity, and transparency of algorithm implementations and their customization.

  In addition to supporting the interfaces of traditional environments such as OpenAI Gym / Farama Gymnasium and DeepMind, it also allows loading and training in NVIDIA Isaac Gym, Isaac Orbit, and Omniverse Isaac Gym environments. In the latter environments, it enables simultaneous training of agents by domains that may or may not share resources, reducing training time while consuming the same amount of resources and increasing perceived reward.

- KUKA LBR iiwa collaborative robot is one of the most famous collaborative robots in industry and research. *libiiwa* is presented as a control framework for such manipulators, with different workflows including ROS and ROS2 as well as direct control for reinforcement learning applications. The proposed interface outperforms current approaches in terms of functionality, access to robot capabilities (including parameterization and use of all types of force overlays), and integration.

  Using such a framework, an experimental study was conducted on how the use of different types of oscillating force overlays reduces the contact forces caused by friction and the probability of jamming states during the execution of disassembly tasks. In particular, Lissajous curves and spiral shapes applied in the plane perpendicular to the disassembly direction provide the most effective results.

- To rapidly create and deploy robotic applications for carrying out robotic assembly and disassembly tasks, *skros*, a rich and organized collection of ready-to-use ROS and non-ROS packages, is proposed. The modular concept and design of such packages provides a high degree of flexibility, reconfigurability, and customization for both simulation and real-world applications (See Chapter 8).

9.2 Future work

Although this dissertation addresses the execution of disassembly tasks using RL algorithms, there are several remaining avenues for further exploration of the field. The following points outline potential areas of future work that could advance the presented research and extend its domain.

- Although current research focuses on uncertainties due to geometric variability (in terms of length and air gap between manipulated parts) and physical friction, there are other sources of uncertainty in disassembly scenarios in the real world that can be investigated. For example, the geometric shapes of the parts or the environmental factors and use/storage conditions. Future work could investigate how RL algorithms can deal with these uncertainties and adapt their policies accordingly. This could involve incorporating additional sensory inputs or using uncertainty quantification techniques to make the system more robust and adaptable to different disassembly scenarios.

- Current research is mainly concerned with disassembly tasks in a limited subset of Cartesian space (translational DOFs). Extending the domain of disassembly tasks to all available DOFs in Cartesian space, including rotational motions, poses significant challenges. Addressing the increasing complexity of multi-DOF disassembly tasks would require sophisticated RL algorithms, efficient representations of state and action spaces, and careful consideration of safety constraints.

- The current research has been conducted on rigid objects, but in the real world it is common to disassemble deformable objects, such as soft materials or flexible components. Learning to disassemble deformable solid objects poses additional challenges due to their dynamic nature and variable stiffness. Future work could explore how RL algorithms can be adapted to the complexities of interacting with deformable objects, for example by incorporating tactile or gripping force information to improve manipulation strategies and adapt to the changing properties of the object during the disassembly process.

- Although the present research explored the inclusion of human operators as a source of information for disambiguation, there is potential for further research on how to improve human-robot collaboration in disassembly tasks. Investigating shared control strategies, where the human operator and the RL agent work together to accomplish the task, could lead to more efficient disassembly processes as long as safety is guaranteed.

- Moving RL-based disassembly algorithms from research prototypes to real-world industrial applications requires addressing several challenges. Future work could focus on validating the proposed methods in practical manufacturing environments, taking into account factors such as real-time constraints, safety requirements, and hardware limitations. In addition, investigating how to seamlessly integrate RL-based disassembly systems into existing robotic automation setups and manufacturing processes is critical for wider adoption and commercial deployment.

Bibliography

[1] Leonid Hryhorovych Melnyk, Oleksandr Vasylovych Kubatko, Iryna Borysivna Dehtyarova, Iryna Borysivna Dehtiarova, Oleksandr Mykhailovych Matsenko, and Oleksandr Dmytrovych Rozhko. "The effect of industrial revolutions on the transformation of social and economic systems". In: (2019).

[2] Peter N Stearns. *The industrial revolution in world history*. Routledge, 2020.

[3] Peter Terwiesch and Christopher Ganz. "Trends in automation". In: *Springer handbook of automation* (2009), pp. 127–143.

[4] Dietrich Imkamp, Jürgen Berthold, Michael Heizmann, Karin Kniel, Eberhard Manske, Martin Peterek, Robert Schmitt, Jochen Seidler, and Klaus-Dieter Sommer. "Challenges and trends in manufacturing measurement technology–the "Industrie 4.0" concept". In: *Journal of sensors and sensor systems* 5.2 (2016), pp. 325–335.

[5] Luis M Camarinha-Matos. "Collaborative networked organizations: Status and trends in manufacturing". In: *Annual Reviews in Control* 33.2 (2009), pp. 199–208.

[6] Heiner Lasi, Peter Fettke, Hans-Georg Kemper, Thomas Feld, and Michael Hoffmann. "Industry 4.0". In: *Business & information systems engineering* 6 (2014), pp. 239–242.

[7] Chunguang Bai, Patrick Dallasega, Guido Orzes, and Joseph Sarkis. "Industry 4.0 technologies assessment: A sustainability perspective". In: *International journal of production economics* 229 (2020), p. 107776.

[8] Peter P Groumpos. "A critical historical and scientific overview of all industrial revolutions". In: *IFAC-PapersOnLine* 54.13 (2021), pp. 464–471.

[9] Vijay P Singh, Shalini Yadav, Krishna Kumar Yadav, and Ram Narayan Yadava. "An Overview on Environmental Degradation and Mitigation". In: *Environmental Degradation: Challenges and Strategies for Mitigation* (2022), pp. 3–15.

[10] Sveinung Jørgensen, Lars Jacob Tynes Pedersen, Sveinung Jørgensen, and Lars Jacob Tynes Pedersen. "The circular rather than the linear economy". In: *RESTART sustainable business model innovation* (2018), pp. 103–120.

[11] Peter Birch Sørensen. "From the linear economy to the circular economy: A basic model". In: *Finanz-Archiv: Zeitschrift für das Gesamte Finanzwesen* 74.1 (2018), pp. 71–87.

[12] Furkan Sariatli. "Linear economy versus circular economy: a comparative and analyzer study for optimization of economy for sustainability". In: *Visegrad Journal on Bioeconomy and Sustainable Development* 6.1 (2017), pp. 31–34.

[13] Mitsutaka Matsumoto and Winifred Ijomah. "Remanufacturing". In: *Handbook of sustainable engineering*. Springer Netherlands, 2013, pp. 389–408.

[14] Idiano D'Adamo and Paolo Rosa. "Remanufacturing in industry: advices from the field". In: *The International Journal of Advanced Manufacturing Technology* 86.9 (2016), pp. 2575–2584.

[15] F Javier Ramírez, Juan A Aledo, Jose A Gamez, and Duc T Pham. "Economic modelling of robotic disassembly in end-of-life product recovery for remanufacturing". In: *Computers & Industrial Engineering* 142 (2020), p. 106339.

[16] Xuhui Xia, Huixian Zhu, Zelin Zhang, Xiang Liu, Lei Wang, and Jianhua Cao. "3D-based multi-objective cooperative disassembly sequence planning method for remanufacturing". In: *The International Journal of Advanced Manufacturing Technology* 106.9 (2020), pp. 4611–4622.

[17] Deepak Singhal, Sushanta Tripathy, and Sarat Kumar Jena. "Remanufacturing for the circular economy: Study and evaluation of critical factors". In: *Resources, Conservation and Recycling* 156 (2020), p. 104681.

[18] Benjamin T Hazen, Diane A Mollenkopf, and Yacan Wang. "Remanufacturing for the circular economy: An examination of consumer switching behavior". In: *Business Strategy and the Environment* 26.4 (2017), pp. 451–464.

[19] Erik Sundin and Hui Mien Lee. "In what way is remanufacturing good for the environment?" In: *Design for innovative value towards a sustainable society: Proceedings of EcoDesign 2011: 7th International Symposium on Environmentally Conscious Design and Inverse Manufacturing*. Springer. 2012, pp. 552–557.

[20] Johan Östlin, Erik Sundin, and Mats Björkman. "Product life-cycle implications for remanufacturing strategies". In: *Journal of cleaner production* 17.11 (2009), pp. 999–1009.

[21] Anjar Priyono, Winifred Ijomah, and Umit S Bititci. "Disassembly for remanufacturing: A systematic literature review, new model development and future research needs". In: *Journal of Industrial Engineering and Management (JIEM)* 9.4 (2016), pp. 899–932.

[22] T Harjula, Brian Rapoza, Winston A Knight, and Geoffrey Boothroyd. "Design for disassembly and the environment". In: *CIRP annals* 45.1 (1996), pp. 109–114.

[23] Askiner Gungor and Surendra M Gupta. "Disassembly line balancing". In: (1999).

[24] Seamus M McGovern and Surendra M Gupta. *Disassembly Line: Balancing and Modeling*. McGraw-Hill Education, 2011.

[25] Eren Özceylan, Can B Kalayci, Aşkıner Güngör, and Surendra M Gupta. "Disassembly line balancing problem: a review of the state of the art and future directions". In: *International Journal of Production Research* 57.15-16 (2019), pp. 4805–4827.

[26] Zude Zhou, Jiayi Liu, Duc Truong Pham, Wenjun Xu, F Javier Ramirez, Chunqian Ji, and Quan Liu. "Disassembly sequence planning: recent developments and future trends". In: *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture* 233.5 (2019), pp. 1450–1471.

[27] Xiwang Guo, MengChu Zhou, Abdullah Abusorrah, Fahad Alsokhiry, and Khaled Sedraoui. "Disassembly sequence planning: a survey". In: *IEEE/CAA Journal of Automatica Sinica* 8.7 (2020), pp. 1308–1324.

[28] Hendrik Poschmann, Holger Brueggemann, and Daniel Goldmann. "Disassembly 4.0: A review on using robotics in disassembly tasks as a way of automation". In: *Chemie Ingenieur Technik* 92.4 (2020), pp. 341–359.

[29] Geoffrey Boothroyd. *Assembly automation and product design*. crc press, 2005.

[30] Jörgen Frohm, Veronica Lindström, Mats Winroth, and Johan Stahre. "The industry's view on automation in manufacturing". In: *IFAC Proceedings Volumes* 39.4 (2006), pp. 453–458.

[31] Karina Cecilia Arredondo-Soto, Arturo Realyvasquez-Vargas, Aidé Aracely Maldonado-Macías, and Jorge García-Alcaraz. "Impact of human resources on remanufacturing process, internal complexity, perceived quality of core, numerosity, and key process indicators". In: *Robotics and computer-integrated manufacturing* 59 (2019), pp. 168–176.

[32] SL Soh, SK Ong, and AYC Nee. "Design for disassembly for remanufacturing: methodology and technology". In: *Procedia CIRP* 15 (2014), pp. 407–412.

[33] Federico Vicentini. "Collaborative robotics: a survey". In: *Journal of Mechanical Design* 143.4 (2021).

[34] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[35] P Read Montague. "Reinforcement learning: an introduction, by Sutton, RS and Barto, AG". In: *Trends in cognitive sciences* 3.9 (1999), p. 360.

[36] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. "Playing Atari with Deep Reinforcement Learning". In: *ArXiv* abs/1312.5602 (2013).

[37] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, L. Sifre, Dharshan Kumaran, Thore Graepel, Timothy P. Lillicrap, Karen Simonyan, and Demis Hassabis. "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play". In: *Science* 362 (2018), pp. 1140 –1144.

[38] M. Mehdi Afsar, Trafford Crump, and Behrouz Homayoun Far. "Reinforcement Learning based Recommender Systems: A Survey". In: *ACM Computing Surveys* 55 (2021), pp. 1 –38.

[39] Liviu Marina and Andreea Sandu. "Deep reinforcement learning for autonomous vehicles-state of the art". In: *Bulletin of the Transilvania University of Brasov. Engineering Sciences. Series I* 10.2 (2017), pp. 195–202.

[40] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. "Hierarchical text-conditional image generation with clip latents". In: *arXiv preprint arXiv:2204.06125* (2022).

[41] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. "Language models are few-shot learners". In: *Advances in neural information processing systems* 33 (2020), pp. 1877–1901.

[42] OpenAI. "GPT-4 Technical Report". In: *ArXiv* abs/2303.08774 (2023).

[43] Smruti Amarjyoti. "Deep reinforcement learning for robotic manipulation-the state of the art". In: *arXiv preprint arXiv:1701.08878* (2017).

[44] Akash Nagaraj, Mukund Sood, and Bhagya M. Patil. "A Concise Introduction to Reinforcement Learning in Robotics". In: *ArXiv* abs/2210.07397 (2022).

[45] Robert Platt. "Grasp Learning: Models, Methods, and Performance". In: *Annual Review of Control, Robotics, and Autonomous Systems* 6 (2022).

[46] Dong Han, Beni Mulyana, Vladimir Stankovic, and Samuel Cheng. "A Survey on Deep Reinforcement Learning Algorithms for Robotic Manipulation". In: *Sensors* 23.7 (2023), p. 3762.

[47] Zhiyao Wen, Wentao Zhang, and Minfeng Qian. "A comprehensive review of deep reinforcement learning for object detection". In: *2021 international symposium on artificial intelligence and its application on media (ISAIAM)*. IEEE. 2021, pp. 146–150.

[48] Jens Kober, J Andrew Bagnell, and Jan Peters. "Reinforcement learning in robotics: A survey". In: *The International Journal of Robotics Research* 32.11 (2013), pp. 1238–1274.

[49] Thanh Thi Nguyen, Ngoc Duy Nguyen, and Saeid Nahavandi. "Deep reinforcement learning for multiagent systems: A review of challenges, solutions, and applications". In: *IEEE transactions on cybernetics* 50.9 (2020), pp. 3826–3839.

[50] Yuxi Li. "Deep reinforcement learning: An overview". In: *arXiv preprint arXiv: 1701.07274* (2017).

[51] Yuxi Li. "Reinforcement learning in practice: Opportunities and challenges". In: *arXiv preprint arXiv:2202.11296* (2022).

[52] Pavel Osinenko, Dmitrii Dobriborsci, and Wolfgang Aumer. "Reinforcement learning with guarantees: a review". In: *IFAC-PapersOnLine* 55.15 (2022), pp. 123–128.

[53] Lukas Brunke, Melissa Greeff, Adam W Hall, Zhaocong Yuan, Siqi Zhou, Jacopo Panerati, and Angela P Schoellig. "Safe learning in robotics: From learning-based control to safe reinforcement learning". In: *Annual Review of Control, Robotics, and Autonomous Systems* 5 (2022), pp. 411–444.

[54] Byoung-Wook Choi. "A Review and Outlook of Robotic Software Frameworks". In: *The Journal of Korea Robotics Society* 5.2 (2010), pp. 169–176.

[55] Abdelfetah Hentout, Abderraouf Maoudj, and Brahim Bouzouia. "A survey of development frameworks for robotics". In: *2016 8th International Conference on Modelling, Identification and Control (ICMIC)*. IEEE. 2016, pp. 67–72.

[56] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, Andrew Y Ng, et al. "ROS: an open-source Robot Operating System". In: *ICRA workshop on open source software*. Vol. 3. 3.2. Kobe, Japan. 2009, p. 5.

[57] SJ Shin, CL Cho, HS Jeon, SH Yoon, and TY Kim. "A Survey on Deep Reinforcement Learning Libraries". In: *Electronics and Telecommunications Trends* 34.6 (2019), pp. 87–99.

[58] Klaas-Jan Stol and Muhammad Ali Babar. "Challenges in using open source software in product development: a review of the literature". In: *Proceedings of the 3rd international workshop on emerging trends in free/libre/open source software research and development*. 2010, pp. 17–22.

[59] Pablo Estefo, Jocelyn Simmonds, Romain Robbes, and Johan Fabry. "The robot operating system: Package reuse and community dynamics". In: *Journal of Systems and Software* 151 (2019), pp. 226–242.

[60] Sophia Kolak, Afsoon Afzal, Claire Le Goues, Michael Hilton, and Christopher Steven Timperley. "It takes a village to build a robot: An empirical study of the ROS ecosystem". In: *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE. 2020, pp. 430–440.

[61] Xin Tan and Minghui Zhou. "Scaling open source software communities: Challenges and practices of decentralization". In: *IEEE Software* 39.1 (2020), pp. 70–75.

[62] Tineke M Egyedi and Ruben van Wendel de Joode. "Standardization and other coordination mechanisms in open source software". In: *International Journal of IT Standards and Standardization Research (IJITSR)* 2.2 (2004), pp. 1–17.

[63] Giri Panamoottil Krishnan and Nikolaos Tsantalis. "Unification and refactoring of clones". In: *2014 Software Evolution Week-IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE)*. IEEE. 2014, pp. 104–113.

[64] Antonio Serrano-Muñoz, Nestor Arana-Arexolaleiba, Dimitrios Chrysostomou, and Simon Bøgh. "Learning and generalising object extraction skill for contact-rich disassembly tasks: an introductory study". In: *The International Journal of Advanced Manufacturing Technology* (2021), pp. 1–13. URL: https://link.springer.com/article/10.1007/s00170-021-08086-z.

[65] Íñigo Elguea-Aguinaco, Antonio Serrano-Muñoz, Dimitrios Chrysostomou, Ibai Inziarte-Hidalgo, Simon Bøgh, and Nestor Arana-Arexolaleiba. "A review on reinforcement learning for contact-rich robotic manipulation tasks". In: *Robotics and Computer-Integrated Manufacturing* 81 (2023), p. 102517. URL: https://www.sciencedirect.com/science/article/pii/S0736584522001995.

[66] Antonio Serrano-Munoz, Dimitrios Chrysostomou, Simon Bøgh, and Nestor Arana-Arexolaleiba. "skrl: Modular and flexible library for reinforcement learning". In: *Journal of Machine Learning Research* 24.254 (2023), pp. 1–9. URL: https://www.jmlr.org/papers/v24/23-0112.html.

[67] Antonio Serrano-Muñoz, Íñigo Elguea-Aguinaco, Dimitris Chrysostomou, Simon Bøgh, and Nestor Arana-Arexolaleiba. "A Scalable and Unified Multi-Control Framework for KUKA LBR iiwa Collaborative Robots". In: *2023 IEEE / SICE International Symposium on System Integration (SII)*. IEEE. 2023, pp. 1–5. URL: https://ieeexplore.ieee.org/abstract/document/10039308.

[68] Íñigo Elguea-Aguinaco, Antonio Serrano-Muñoz, Dimitrios Chrysostomou, Ibai Inziarte-Hidalgo, Simon Bøgh, and Nestor Arana-Arexolaleiba. "Goal-Conditioned Reinforcement Learning within a Human-Robot Disassembly Environment". In: *Applied Sciences* 12.22 (2022), p. 11610. URL: https://www.mdpi.com/2076-3417/12/22/11610.

[69] Idoia Altuna-Galfarsoro, Antonio Serrano-Muñoz, Iker Castro Alfaro, Jon Aurrekoetxea, and Nestor Arana-Arexolaleiba. "Generative design of 3D printed grippers for robot/human collaborative environments". In: *2021 IEEE International Workshop of Electronics, Control, Measurement, Signals and their application to Mechatronics (ECMSM)*. IEEE. 2021, pp. 1–5. URL: https://ieeexplore.ieee.org/abstract/document/9468855.

[70] Ainhoa Apraiz Iriarte, Ganix Lasa Erle, Antonio Serrano-Muñoz, Íñigo Elguea-Aguinaco, and Nestor Arana-Arexolaleiba. "Evaluation of the User Experience of an industrial robotic environment in Virtual Reality". In: *International Congress on Project Management and Engineering*. Vol. 7. 3. AEIPRO, 2022. URL: http://dspace.aeipro.com/xmlui/handle/123456789/3288.

[71] Ainhoa Apraiz Iriarte, Ganix Lasa, Maitane Mazmela, Nestor Arana-Arexolaleiba, Antonio Serrano-Muñoz, Íñigo Elguea-Aguinaco, and Amaia Etxabe. "Evaluating the Effect of Speed and Acceleration on Human Factors During an Assembly Task in Human-Robot Collaboration". In: *Available at SSRN 4371145* ().

[72] Thomas M Moerland, Joost Broekens, Aske Plaat, Catholijn M Jonker, et al. "Model-based reinforcement learning: A survey". In: *Foundations and Trends in Machine Learning* 16.1 (2023), pp. 1–118.

[73] Yongquan Zhang, Hong Lu, Duc Truong Pham, Yongjing Wang, Mo Qu, Joey Lim, and Shizhong Su. "Peg–hole disassembly using active compliance". In: *Royal Society open science* 6.8 (2019), p. 190476.

[74] R Herold, YJ Wang, DT Pham, J Huang, C Ji, and S Su. "Using active adjustment and compliance in robotic disassembly". In: *Industry 4.0–Shaping The Future of The Digital World*. CRC Press, 2020, pp. 101–105.

[75] Jun Huang, Duc Truong Pham, Yongjing Wang, Mo Qu, Chunqian Ji, Shizhong Su, Wenjun Xu, Quan Liu, and Zude Zhou. "A case study in human–robot collaboration in the disassembly of press-fitted components". In: *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture* 234.3 (2020), pp. 654–664.

[76] Christoffer B Kristensen, Frederik A Sørensen, Hjalte B Nielsen, Martin S Andersen, Søren P Bendtsen, and Simon Bøgh. "Towards a Robot Simulation Framework for E-waste Disassembly Using Reinforcement Learning". In: *Procedia Manufacturing* 38 (2019), pp. 225–232.

[77] Mihael Simonič, Leon Žlajpah, Aleš Ude, and Bojan Nemec. "Autonomous Learning of Assembly Tasks from the Corresponding Disassembly Tasks". In: *2019 IEEE-RAS 19th International Conference on Humanoid Robots (Humanoids)*. IEEE. 2019, pp. 230–236.

[78] Youhao Li, Qujiang Lei, ChaoPeng Cheng, Gong Zhang, Weijun Wang, and Zheng Xu. "A review: Machine learning on robotic grasping". In: *Eleventh International Conference on Machine Vision (ICMV 2018)*. Vol. 11041. SPIE. 2019, pp. 775–783.

[79] Manuel Graña, Marcos Alonso, and Alberto Izaguirre. "A panoramic survey on grasping research trends and topics". In: *Cybernetics and Systems* 50.1 (2019), pp. 40–57.

[80] Kilian Kleeberger, Richard Bormann, Werner Kraus, and Marco F Huber. "A survey on learning-based robotic grasping". In: *Current Robotics Reports* 1 (2020), pp. 239–249.

[81] Junfeng Ding, Chen Wang, and Cewu Lu. "Transferable force-torque dynamics model for peg-in-hole task". In: *arXiv preprint arXiv:1912.00260* (2019).

[82] Zhimin Hou, Zhihu Li, Chenwei Hsu, Kuangen Zhang, and Jing Xu. "Fuzzy logic-driven variable time-scale prediction-based reinforcement learning for robotic multiple peg-in-hole assembly". In: *IEEE Transactions on Automation Science and Engineering* 19.1 (2020), pp. 218–229.

[83] Cristian C Beltran-Hernandez, Damien Petit, Ixchel G Ramirez-Alpizar, and Kensuke Harada. "Variable compliance control for robotic peg-in-hole assembly: A deep-reinforcement-learning approach". In: *Applied Sciences* 10.19 (2020), p. 6923.

[84] Gerrit Schoettler, Ashvin Nair, Jianlan Luo, Shikhar Bahl, Juan Aparicio Ojea, Eugen Solowjow, and Sergey Levine. "Deep reinforcement learning for industrial insertion tasks with visual inputs and natural rewards". In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2020, pp. 5548–5555.

[85] Masahide Oikawa, Tsukasa Kusakabe, Kyo Kutsuzawa, Sho Sakaino, and Toshiaki Tsuji. "Reinforcement learning for robotic assembly using non-diagonal stiffness matrix". In: *IEEE Robotics and Automation Letters* 6.2 (2021), pp. 2737–2744.

[86] Justin Fu, Sergey Levine, and Pieter Abbeel. "One-shot learning of manipulation skills with online dynamics adaptation and neural network priors". In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2016, pp. 4019–4026.

[87] Jose Sanchez, Juan-Antonio Corrales, Belhassen-Chedli Bouzgarrou, and Youcef Mezouar. "Robotic manipulation and sensing of deformable objects in domestic and industrial applications: a survey". In: *The International Journal of Robotics Research* 37.7 (2018), pp. 688–716.

[88] Michał Bednarek and Krzysztof Walas. "Comparative assessment of reinforcement learning algorithms in the taskof robotic manipulation of deformable linear objects". In: *2019 4th International Conference on Robotics and Automation Engineering (ICRAE)*. IEEE. 2019, pp. 173–177.

[89] Jan Matas, Stephen James, and Andrew J Davison. "Sim-to-real reinforcement learning for deformable object manipulation". In: *Conference on Robot Learning*. PMLR. 2018, pp. 734–743.

[90] Vladimír Petrík and Ville Kyrki. "Feedback-based fabric strip folding". In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2019, pp. 773–778.

[91] Sanjay Krishnan, Animesh Garg, Richard Liaw, Brijen Thananjeyan, Lauren Miller, Florian T Pokorny, and Ken Goldberg. "SWIRL: A sequential windowed inverse reinforcement learning algorithm for robot tasks with delayed rewards". In: *The international journal of robotics research* 38.2-3 (2019), pp. 126–145.

[92] Thanh Nguyen, Ngoc Duy Nguyen, Fernando Bello, and Saeid Nahavandi. "A new tensioning method using deep reinforcement learning for surgical pattern cutting". In: *2019 IEEE international conference on industrial technology (ICIT)*. IEEE. 2019, pp. 1339–1344.

[93] Adria Colomé, Antoni Planells, and Carme Torras. "A friction-model-based framework for reinforcement learning of robotic tasks in non-rigid environments". In: *2015 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2015, pp. 5649–5654.

[94] Aleksandra Anna Apolinarska, Matteo Pacher, Hui Li, Nicholas Cote, Rafael Pastrana, Fabio Gramazio, and Matthias Kohler. "Robotic assembly of timber joints using reinforcement learning". In: *Automation in Construction* 125 (2021), p. 103569.

[95] Yunlei Shi, Zhaopeng Chen, Hongxu Liu, Sebastian Riedel, Chunhui Gao, Qian Feng, Jun Deng, and Jianwei Zhang. "Proactive action visual residual reinforcement learning for contact-rich tasks using a torque-controlled robot". In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2021, pp. 765–771.

[96] Jihong Zhu, Andrea Cherubini, Claire Dune, David Navarro-Alarcon, Farshid Alambeigi, Dmitry Berenson, Fanny Ficuciello, Kensuke Harada, Jens Kober, Xiang Li, et al. "Challenges and outlook in robotic manipulation of deformable objects". In: *IEEE Robotics & Automation Magazine* 29.3 (2022), pp. 67–77.

[97] Oliver Kroemer, Scott Niekum, and George Konidaris. "A review of robot learning for manipulation: Challenges, representations, and algorithms". In: *The Journal of Machine Learning Research* 22.1 (2021), pp. 1395–1476.

[98] Haifeng Han, Gavin Paul, and Takamitsu Matsubara. "Model-based reinforcement learning approach for deformable linear object manipulation". In: *2017 13th IEEE Conference on Automation Science and Engineering (CASE)*. IEEE. 2017, pp. 750–755.

[99] Ngoc Duy Nguyen, Thanh Nguyen, Saeid Nahavandi, Asim Bhatti, and Glenn Guest. "Manipulating soft tissues by deep reinforcement learning for autonomous robotic surgery". In: *2019 IEEE International Systems Conference (SysCon)*. IEEE. 2019, pp. 1–7.

[100] Mythra V Balakuntala, Upinder Kaur, Xin Ma, Juan Wachs, and Richard M Voyles. "Learning multimodal contact-rich skills from demonstrations without reward engineering". In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2021, pp. 4679–4685.

[101] Cristian Camilo Beltran-Hernandez, Damien Petit, Ixchel Georgina Ramirez-Alpizar, Takayuki Nishi, Shinichi Kikuchi, Takamitsu Matsubara, and Kensuke Harada. "Learning force control for contact-rich manipulation tasks with rigid position-controlled robots". In: *IEEE Robotics and Automation Letters* 5.4 (2020), pp. 5709–5716.

[102] Marco Braun and Sebastian Wrede. "Incorporation of expert knowledge for learning robotic assembly tasks". In: *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. Vol. 1. IEEE. 2020, pp. 1594–1601.

[103] Jianlan Luo, Eugen Solowjow, Chengtao Wen, Juan Aparicio Ojea, Alice M Agogino, Aviv Tamar, and Pieter Abbeel. "Reinforcement learning on variable impedance controller for high-precision robotic assembly". In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE. 2019, pp. 3080–3087.

[104] Xin Zhao, Huan Zhao, Pengfei Chen, and Han Ding. "Model accelerated reinforcement learning for high precision robotic assembly". In: *International Journal of Intelligent Robotics and Applications* 4 (2020), pp. 202–216.

[105] Florian Wirnshofer, Philipp Sebastian Schmitt, Georg von Wichert, and Wolfram Burgard. "Controlling Contact-Rich Manipulation Under Partial Observability." In: *Robotics: Science and Systems*. 2020.

[106] Fares J Abu-Dakka and Matteo Saveriano. "Variable impedance control and learning—a review". In: *Frontiers in Robotics and AI* 7 (2020), p. 590681.

[107] Pawel Ladosz, Lilian Weng, Minwoo Kim, and Hyondong Oh. "Exploration in deep reinforcement learning: A survey". In: *Information Fusion* (2022).

[108] Zheng Wu, Wenzhao Lian, Vaibhav Unhelkar, Masayoshi Tomizuka, and Stefan Schaal. "Learning dense rewards for contact-rich manipulation tasks". In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2021, pp. 6214–6221.

[109] Nikola Vulin, Sammy Christen, Stefan Stevšić, and Otmar Hilliges. "Improved learning of robot manipulation tasks via tactile intrinsic motivation". In: *IEEE Robotics and Automation Letters* 6.2 (2021), pp. 2194–2201.

[110] Serkan Cabi, Sergio Gómez Colmenarejo, Alexander Novikov, Scott Reed, Ksenia Konyushkova, Rae Jeong, Konrad Zolna, Yusuf Aytar, David Budden, Mel Vecerik, et al. "Scaling data-driven robotics with reward sketching and batch reinforcement learning". In: *arXiv preprint arXiv:1909.12200* (2019).

[111] S Levine, N Wagener, and P Abbeel. "Learning contact-rich manipulation skills with guided policy search (2015)". In: *arXiv preprint arXiv:1501.05611* (2015).

[112] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. "End-to-end training of deep visuomotor policies". In: *The Journal of Machine Learning Research* 17.1 (2016), pp. 1334–1373.

[113] Rae Jeong, Jackie Kay, Francesco Romano, Thomas Lampe, Tom Rothorl, Abbas Abdolmaleki, Tom Erez, Yuval Tassa, and Francesco Nori. "Modelling generalized forces with reinforcement learning for sim-to-real transfer". In: *arXiv preprint arXiv:1910.09471* (2019).

[114] Masashi Hamaya, Robert Lee, Kazutoshi Tanaka, Felix von Drigalski, Chisato Nakashima, Yoshiya Shibata, and Yoshihisa Ijiri. "Learning robotic assembly tasks with lower dimensional systems by leveraging physical softness and environmental constraints". In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2020, pp. 7747–7753.

[115] Shahbaz Abdul Khader, Hang Yin, Pietro Falco, and Danica Kragic. "Stability-guaranteed reinforcement learning for contact-rich manipulation". In: *IEEE Robotics and Automation Letters* 6.1 (2020), pp. 1–8.

[116] Michelle A Lee, Yuke Zhu, Peter Zachares, Matthew Tan, Krishnan Srinivasan, Silvio Savarese, Li Fei-Fei, Animesh Garg, and Jeannette Bohg. "Making sense of vision and touch: Learning multimodal representations for contact-rich tasks". In: *IEEE Transactions on Robotics* 36.3 (2020), pp. 582–596.

[117] Oren Spector and Miriam Zacksenhouse. "Deep reinforcement learning for contact-rich skills using compliant movement primitives". In: *arXiv preprint arXiv:2008.13223* (2020).

[118] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. "Deep reinforcement learning: A brief survey". In: *IEEE Signal Processing Magazine* 34.6 (2017), pp. 26–38.

[119] Wenshuai Zhao, Jorge Peña Queralta, and Tomi Westerlund. "Sim-to-real transfer in deep reinforcement learning for robotics: a survey". In: *2020 IEEE symposium series on computational intelligence (SSCI)*. IEEE. 2020, pp. 737–744.

[120] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. "Domain randomization for transferring deep neural networks from simulation to the real world". In: *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE. 2017, pp. 23–30.

[121] Lin Shao, Toki Migimatsu, and Jeannette Bohg. "Learning to scaffold the development of robotic manipulation skills". In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2020, pp. 5671–5677.

[122] Christoph Hennersperger, Bernhard Fuerst, Salvatore Virga, Oliver Zettinig, Benjamin Frisch, Thomas Neff, and Nassir Navab. "Towards MRI-based autonomous robotic US acquisitions: a first feasibility study". In: *IEEE transactions on medical imaging* 36.2 (2017), pp. 538–548.

[123] Vijay Konda and John Tsitsiklis. "Actor-critic algorithms". In: *Advances in neural information processing systems* 12 (1999).

[124] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. "Continuous control with deep reinforcement learning". In: *arXiv preprint arXiv:1509.02971* (2015).

[125] Scott Fujimoto, Herke Hoof, and David Meger. "Addressing function approximation error in actor-critic methods". In: *International conference on machine learning*. PMLR. 2018, pp. 1587–1596.

[126] Viktor Makoviychuk, Lukasz Wawrzyniak, Yunrong Guo, Michelle Lu, Kier Storey, Miles Macklin, David Hoeller, Nikita Rudin, Arthur Allshire, Ankur Handa, et al. "Isaac gym: High performance gpu-based physics simulation for robot learning". In: *arXiv preprint arXiv:2108.10470* (2021).

[127] Jacky Liang, Viktor Makoviychuk, Ankur Handa, Nuttapong Chentanez, Miles Macklin, and Dieter Fox. "Gpu-accelerated robotic simulation for distributed reinforcement learning". In: *Conference on Robot Learning*. PMLR. 2018, pp. 270–282.

[128] Mayank Mittal, Calvin Yu, Qinxi Yu, Jingzhou Liu, Nikita Rudin, David Hoeller, Jia Lin Yuan, Pooria Poorsarvi Tehrani, Ritvik Singh, Yunrong Guo, Hammad Mazhar, Ajay Mandlekar, Buck Babich, Gavriel State, Marco Hutter, and Animesh Garg. *ORBIT: A Unified Simulation Framework for Interactive Robot Learning Environments*. 2023. eprint: arXiv:2301.04195.

[129] Nikita Rudin, David Hoeller, Philipp Reist, and Marco Hutter. "Learning to walk in minutes using massively parallel deep reinforcement learning". In: *Conference on Robot Learning*. PMLR. 2022, pp. 91–100.

[130] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. "Openai gym". In: *arXiv preprint arXiv:1606.01540* (2016).

[131] Farama-Foundation. *Gymnasium*. https://github.com/Farama-Foundation/Gymnasium. 2022.

[132] Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, et al. "Deepmind control suite". In: *arXiv preprint arXiv:1801.00690* (2018).

[133] Saran Tunyasuvunakool, Alistair Muldal, Yotam Doron, Siqi Liu, Steven Bohez, Josh Merel, Tom Erez, Timothy Lillicrap, Nicolas Heess, and Yuval Tassa. "dm_control: Software and tasks for continuous control". In: *Software Impacts* 6 (2020), p. 100022.

[134] Yuke Zhu, Josiah Wong, Ajay Mandlekar, and Roberto Martín-Martín. "robosuite: A modular simulation framework and benchmark for robot learning". In: *arXiv preprint arXiv:2009.12293* (2020).

[135] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. "Asynchronous methods for deep reinforcement learning". In: *International conference on machine learning*. PMLR. 2016, pp. 1928–1937.

[136] Xue Bin Peng, Ze Ma, Pieter Abbeel, Sergey Levine, and Angjoo Kanazawa. "Amp: Adversarial motion priors for stylized physics-based character control". In: *ACM Transactions on Graphics (TOG)* 40.4 (2021), pp. 1–20.

[137] István Szita and András Lörincz. "Learning Tetris using the noisy cross-entropy method". In: *Neural computation* 18.12 (2006), pp. 2936–2941.

[138] Hado Van Hasselt, Arthur Guez, and David Silver. "Deep reinforcement learning with double q-learning". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 30. 1. 2016.

[139] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. "Playing atari with deep reinforcement learning". In: *arXiv preprint arXiv:1312.5602* (2013).

[140] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. "Proximal policy optimization algorithms". In: *arXiv preprint arXiv:1707.06347* (2017).

[141] Christopher John Cornish Hellaby Watkins. "Learning from delayed rewards". In: (1989).

[142] Md Masudur Rahman and Yexiang Xue. "Robust Policy Optimization in Deep Reinforcement Learning". In: *arXiv preprint arXiv:2212.07536* (2022).

[143] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor". In: *International conference on machine learning*. PMLR. 2018, pp. 1861–1870.

[144] Gavin A Rummery and Mahesan Niranjan. *On-line Q-learning using connectionist systems*. Vol. 37. University of Cambridge, Department of Engineering Cambridge, UK, 1994.

[145] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. "Trust region policy optimization". In: *International conference on machine learning*. PMLR. 2015, pp. 1889–1897.

[146] Ariyan M Kabir, Krishnanand N Kaipa, Jeremy Marvel, and Satyandra K Gupta. "Automated planning for robotic cleaning using multiple setups and oscillatory tool motions". In: *IEEE Transactions on Automation Science and Engineering* 14.3 (2017), pp. 1364–1377.

[147] Srinivasan Lakshminarayanan, Omey Mohan Manyar, and Domenico Campolo. "Toolpath generation for robot filleting". In: *Advanced Surface Enhancement: Proceedings of the 1st International Conference on Advanced Surface Enhancement (INCASE 2019)—Research Towards Industrialisation*. Springer. 2020, pp. 273–280.

[148] Ulrich Berger, Duc Tho Le, Wenchao Zou, and Mayur Andulkar. "Towards a Mobile Robotic Assembly System Using a Compliant Robot". In: *Tagungsband des 2. Kongresses Montage Handhabung Industrieroboter*. Springer. 2017, pp. 145–154.

[149] Vinu Vijayakumaran Nair, Daniel Kuhn, and Vera Hummel. "Development of an easy teaching and simulation solution for an autonomous mobile robot system". In: *Procedia manufacturing* 31 (2019), pp. 270–276.

[150] Sven Stumm, Johannes Braumann, Martin von Hilchen, and Sigrid Brell-Cokcan. "On-site robotic construction assistance for assembly using a-priori knowledge and human-robot collaboration". In: *Advances in Robot Design and Intelligent Control: Proceedings of the 25th Conference on Robotics in Alpe-Adria-Danube Region (RAAD16)*. Springer. 2017, pp. 583–592.

[151] Steven Macenski, Tully Foote, Brian Gerkey, Chris Lalancette, and William Woodall. "Robot Operating System 2: Design, architecture, and uses in the wild". In: *Science Robotics* 7.66 (2022), eabm6074.

[152] David Coleman, Ioan Sucan, Sachin Chitta, and Nikolaus Correll. "Reducing the barrier to entry of complex robotic software: a moveit! case study". In: *arXiv preprint arXiv:1404.3785* (2014).

[153] Sachin Chitta, Eitan Marder-Eppstein, Wim Meeussen, Vijay Pradeep, Adolfo Rodríguez Tsouroukdissian, Jonathan Bohren, David Coleman, Bence Magyar, Gennaro Raiola, Mathias Lüdtke, et al. "ros_control: A generic and simple control framework for ROS". In: *The Journal of Open Source Software* 2.20 (2017), pp. 456–456.

[154] Salvatore Virga, Oliver Zettinig, Marco Esposito, Karin Pfister, Benjamin Frisch, Thomas Neff, Nassir Navab, and Christoph Hennersperger. "Automatic force-compliant robotic ultrasound screening of abdominal aortic aneurysms". In: *2016 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE. 2016, pp. 508–513.

[155] Robin Edwards and Robert Bush. *A Python interface to the KUKA LBR IIWA R800/R820*. 2020. URL: https://github.com/rivelinrobotics/iiwapy.

[156] Konstantinos Chatzilygeroudis, Bernardo Fichera, and Aude Billard. *iiwa_ros: A ROS Stack for KUKA's IIWA robots using the Fast Research Interface*. 2019. URL: http://github.com/epfl-lasa/iiwa_ros.

[157] ICube-Robotics. *IIWA_ROS2*. 2022. URL: https://github.com/ICube-Robotics/iiwa_ros2.

[158] Martin Huber, Esteve Fernandez, and Christopher Mower. *lbr_fri_ros2_stack: ROS 2 packages for the KUKA LBR*. 2020. URL: https://github.com/kCL-BMEIS/lbr_fri_ros2_stack.

[159] M. Safeea and P. Neto. "KUKA Sunrise Toolbox: Interfacing Collaborative Robots With MATLAB". In: *IEEE Robotics Automation Magazine* 26.1 (2019), pp. 91–96. ISSN: 1070-9932. DOI: 10.1109/MRA.2018.2877776.

[160] Mohammad Safeea. *iiwaPy2/iiwaPy3: Python library used to control KUKA iiwa robots, the 7R800 and the 14R820, from an external computer*. 2020. URL: https://github.com/Modi1987/iiwaPy3.

[161] Arun Dayal Udai, Subir Kumar Saha, and Ashutosh Dayal. "Overlaid orthogonal force oscillations for robot assisted localization and assembly". In: *ISME J. Mech. Des.* 2.1 (2019), pp. 09–25.

[162] Sebastian Hjorth and Dimitrios Chrysostomou. "Human–robot collaboration in industrial environments: A literature review on non-destructive disassembly". In: *Robotics and Computer-Integrated Manufacturing* 73 (2022), p. 102208.

[163] Andrea Lonza. *Reinforcement Learning Algorithms with Python: Learn, understand, and develop smart algorithms for addressing AI challenges*. Packt Publishing Ltd, 2019.

[164] O Gym and Nimish Sanghi. *Deep Reinforcement Learning with Python*. Springer, 2021.

[165] Sebastian Hjorth, Edoardo Lamon, Dimitrios Chrysostomou, and Arash Ajoudani. "Design of an Energy-Aware Cartesian Impedance Controller for Collaborative Disassembly". In: *arXiv preprint arXiv:2302.03587* (2023).

[166] Mo Qu, Yongjing Wang, and Duc Truong Pham. "Robotic Disassembly Task Training and Skill Transfer Using Reinforcement Learning". In: *IEEE Transactions on Industrial Informatics* (2023).

[167] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. "Fast and accurate deep network learning by exponential linear units (elus)". In: *arXiv preprint arXiv:1511.07289* (2015).

[168] Shiv Ram Dubey, Satish Kumar Singh, and Bidyut Baran Chaudhuri. "Activation functions in deep learning: A comprehensive survey and benchmark". In: *Neurocomputing* (2022).

[169] Petr Oščádal, Tomáš Kot, Tomáš Spurnỳ, Jiří Suder, Michal Vocetka, Libor Dobeš, and Zdenko Bobovskỳ. "Camera Arrangement Optimization for Workspace Monitoring in Human–Robot Collaboration". In: *Sensors* 23.1 (2022), p. 295.

[170] Himanshu Chandel and Sonia Vatta. "Occlusion detection and handling: a review". In: *International Journal of Computer Applications* 120.10 (2015).

[171] Farzaneh Goli, Yongjing Wang, and Mozafar Saadat. "Perspective of self-learning robotics for disassembly automation". In: *2022 27th International Conference on Automation and Computing (ICAC)*. IEEE. 2022, pp. 1–6.

[172] Jack Collins, Shelvin Chand, Anthony Vanderkop, and David Howard. "A review of physics simulators for robotic applications". In: *IEEE Access* 9 (2021), pp. 51416–51431.

[173] C Karen Liu and Dan Negrut. "The role of physics-based simulators in robotics". In: *Annual Review of Control, Robotics, and Autonomous Systems* 4 (2021), pp. 35–58.

[174] HeeSun Choi, Cindy Crump, Christian Duriez, Asher Elmquist, Gregory Hager, David Han, Frank Hearl, Jessica Hodgins, Abhinandan Jain, Frederick Leve, et al. "On the use of simulation in robotics: Opportunities, challenges, and suggestions for moving forward". In: *Proceedings of the National Academy of Sciences* 118.1 (2021), e1907856118.

[175] Aaron Staranowicz and Gian Luca Mariottini. "A survey and comparison of commercial and open-source robotic simulator software". In: *Proceedings of the 4th International Conference on PErvasive Technologies Related to Assistive Environments*. 2011, pp. 1–8.

[176] Afsoon Afzal, Deborah S Katz, Claire Le Goues, and Christopher S Timperley. "A study on the challenges of using robotics simulators for testing". In: *arXiv preprint arXiv:2004.07368* (2020).

[177] Mirella Santos Pessoa De Melo, José Gomes da Silva Neto, Pedro Jorge Lima Da Silva, João Marcelo Xavier Natario Teixeira, and Veronica Teichrieb. "Analysis and comparison of robotics 3d simulators". In: *2019 21st Symposium on Virtual and Augmented Reality (SVR)*. IEEE. 2019, pp. 242–251.

[178] Sebastian Höfer, Kostas Bekris, Ankur Handa, Juan Camilo Gamboa, Melissa Mozifian, Florian Golemo, Chris Atkeson, Dieter Fox, Ken Goldberg, John Leonard, et al. "Sim2Real in robotics and automation: Applications and challenges". In: *IEEE transactions on automation science and engineering* 18.2 (2021), pp. 398–400.

[179] Sachin Chitta. "MoveIt!: an introduction". In: *Robot Operating System (ROS) The Complete Reference (Volume 1)* (2016), pp. 3–27.

[180] Steve Macenski, Francisco Martín, Ruffin White, and Jonatan Ginés Clavero. "The marathon 2: A navigation system". In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2020, pp. 2718–2725.

[181] NVIDIA. *NVIDIA Isaac ROS. High-performance computing for robotics built on ROS 2*. 2021. URL: https://github.com/NVIDIA-ISAAC-ROS.

[182] Francesco Rovida, Matthew Crosby, Dirk Holz, Athanasios S Polydoros, Bjarne Großmann, Ronald PA Petrick, and Volker Krüger. "SkiROS—a skill-based robot control platform on top of ROS". In: *Robot Operating System (ROS) The Complete Reference (Volume 2)* (2017), pp. 121–160.

[183] Mohammed Diab, Mihai Pomarlan, Daniel Beßler, Aliakbar Akbari, Jan Rosell, John Bateman, and Michael Beetz. "SkillMaN—A skill-based robotic manipulation framework based on perception and reasoning". In: *Robotics and Autonomous Systems* 134 (2020), p. 103653.

[184] Héctor Herrero, Amine Abou Moughlbay, Jose Luis Outón, Damien Sallé, and Karmele López de Ipiña. "Skill based robot programming: Assembly, vision and Workspace Monitoring skill interaction". In: *Neurocomputing* 255 (2017), pp. 61–70.

[185] Casper Schou, Jens S Damgaard, Simon Bøgh, and Ole Madsen. "Human-robot interface for instructing industrial tasks using kinesthetic teaching". In: *IEEE ISR 2013*. IEEE. 2013, pp. 1–6.

[186] Paul J Koch, Marike K van Amstel, Patrycja Dębska, Moritz A Thormann, Adrian J Tetzlaff, Simon Bøgh, and Dimitrios Chrysostomou. "A skill-based robot co-worker for industrial maintenance tasks". In: *Procedia Manufacturing* 11 (2017), pp. 83–90.

[187] Casper Schou, Rasmus Skovgaard Andersen, Dimitrios Chrysostomou, Simon Bøgh, and Ole Madsen. "Skill-based instruction of collaborative robots in industrial settings". In: *Robotics and Computer-Integrated Manufacturing* 53 (2018), pp. 72–80.

[188] Lisa Heuss, Clemens Gonnermann, and Gunther Reinhart. "An extendable framework for intelligent and easily configurable skills-based industrial robot applications". In: *The International Journal of Advanced Manufacturing Technology* 120.9-10 (2022), pp. 6269–6285.

[189] Gary Bradski. "The openCV library." In: *Dr. Dobb's Journal: Software Tools for the Professional Programmer* 25.11 (2000), pp. 120–123.

[190] Leonid Keselman, John Iselin Woodfill, Anders Grunnet-Jepsen, and Achintya Bhowmik. "Intel realsense stereoscopic depth cameras". In: *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*. 2017, pp. 1–10.

[191] Francisco M Sánchez-Margallo, Juan A Sánchez-Margallo, José L Moyano-Cuevas, Eva María Pérez, and Juan Maestre. "Use of natural user interfaces for image navigation during laparoscopic surgery: initial experience". In: *Minimally Invasive Therapy & Allied Technologies* 26.5 (2017), pp. 253–261.

[192] Juan A Sánchez-Margallo, José Castillo Rabazo, Carlos Plaza de Miguel, Peter Gloor, David Durán Rey, Manuel Ramón González-Portillo, Isabel López Agudelo, and Francisco M Sánchez-Margallo. "Wearable Technology for Assessment and Surgical Assistance in Minimally Invasive Surgery". In: *Advances in Minimally Invasive Surgery*. IntechOpen, 2021.

[193] IEC IEC. "60204-5.1: Safety of machinery-Electrical equipment of machines-Part 1: General requirements". In: *Geneva, Switzerland: International Electrotechnical Commission* (2009).

[194] Rasmus S Andersen, Ole Madsen, Thomas B Moeslund, and Heni Ben Amor. "Projecting robot intentions into human environments". In: *2016 25th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*. IEEE. 2016, pp. 294–301.

[195] Christian Vogel, Maik Poggendorf, Christoph Walter, and Norbert Elkmann. "Towards safe physical human-robot collaboration: A projection-based safety system". In: *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2011, pp. 3355–3360.

[196] Christian Vogel, Christoph Walter, and Norbert Elkmann. "Safeguarding and supporting future human-robot cooperative manufacturing processes by a projection-and camera-based technology". In: *Procedia Manufacturing* 11 (2017), pp. 39–46.

[197] *Named-color. Cascading style sheets: MDN*. URL: https://developer.mozilla.org/en-US/docs/Web/CSS/named-color.

[198] Tully Foote. "tf: The transform library". In: *2013 IEEE Conference on Technologies for Practical Robot Applications (TePRA)*. IEEE. 2013, pp. 1–6.

[199] Bin Ren, Mengyuan Liu, Runwei Ding, and Hong Liu. "A survey on 3d skeleton-based action recognition using learning method". In: *arXiv preprint arXiv:2002.05907* (2020).

[200] Liangchen Song, Gang Yu, Junsong Yuan, and Zicheng Liu. "Human pose estimation and its application to action recognition: A survey". In: *Journal of Visual Communication and Image Representation* 76 (2021), p. 103055.

[201] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. "YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors". In: *arXiv preprint arXiv:2207.02696* (2022).

[202] Camillo Lugaresi, Jiuqiang Tang, Hadon Nash, Chris McClanahan, Esha Uboweja, Michael Hays, Fan Zhang, Chuo-Ling Chang, Ming Guang Yong, Juhyun Lee, et al. "Mediapipe: A framework for building perception pipelines". In: *arXiv preprint arXiv:1906.08172* (2019).