Politecnico di Torino

Politecnico di Torino

Master's Degree in Computer Engineering LM32

A.y. 2025/2026

Graduation Session March 2026

From Simulation to Real-World Assembly

Deep Reinforcement Learning for Robotic Assembly on the Flexiv Rizon 4s

Supervisors:

Prof. Giuseppe Bruno Averta

Ing. Claudio Chieppa

Candidate:

Emanuele Giuseppe Siani

Abstract

Robotic assembly relies on the precise joining, fastening, and positioning of components. While classical motion-planning techniques excel in tightly structured workspaces requiring millimetric precision, they lack the flexibility to adapt to unstructured failures, often necessitating manual intervention or rigid programmed reactions. Deep Reinforcement Learning (DRL) offers a paradigm shift, allowing agents to learn optimal decisions through trial-and-error without explicit programming. Although DRL has demonstrated success in simpler or looser assembly benchmarks, learned policies frequently fall short of the positional accuracy required for strict industrial tolerances, often exhibiting stochastic behavior that is incompatible with high-precision manufacturing.

The research specifically targets the Gear Meshing task, a contact-rich scenario selected because it demands exceptional precision, far exceeding the requirements of standard pick-and-place baselines. This task involves inserting a gear onto a peg with a 0.5 mm clearance while aligning teeth with adjacent gears, a process requiring complex physical interaction. Using the Proximal Policy Optimization (PPO) algorithm, the agent was trained in simulation using state-of-the-art frameworks. To bridge the simulation to reality (Sim-to-Real) gap, training was conducted within a customized version of the NVIDIA FORGE environment. While originally designed for the Franka Panda manipulator, this environment was leveraged to implement extensive Domain Randomization, varying both the manipulated assets and the robot's dynamics and customized for the Flexiv Rizon 4s manipulator. The policy relies exclusively on proprioceptive data: the current pose and the forces measured at the Tool Center Point (TCP), thereby eliminating the need for external sensors and minimizing hardware complexity.

This work documents the first known deployment of this specific task on the Flexiv Rizon 4s and introduces a novel workflow for exporting policies from the rl-games library to a physical industrial controller. To ensure operational safety, the policy incorporates a force-thresholding mechanism leveraging the robot's integrated torque sensors, bypassing the need for extensive gain tuning during deployment. Experimental results validate the feasibility of an end-to-end, zero-shot RL pipeline for high-precision assembly. While physical deployment highlighted the persistent challenges of the reality gap, successful task completion was achieved. This research demonstrates that Reinforcement Learning can effectively handle low-clearance industrial tasks, opening new avenues for autonomous precision manufacturing. Future work will explore Imitation Learning, and multimodal visual feedback to further enhance robustness and precision.

Acknowledgements

Un ringraziamento al Professor Giuseppe Bruno Averta per la guida fornita durante la stesura della tesi.

Ringrazio Claudio Chieppa per l'opportunità di tesi propostami.

Ringrazio i colleghi Simone Voto, Federico Pretini, Anna Fumagalli ed Elisa Cenedese per il supporto tecnico fornitomi durante l'attività di tirocinio.

Table of Contents

List of Tables

List of Figures

List of Abbrevavtions

A2C

Asynchronous Actor-Critic

A3C

Asynchronous Asynchronous Actor-Critic

ADR

Automatic Domain Randomization

AI

Artificial Intelligence

CPS

Cyber Physical Systems

CPU

Central Processing Unit

CUDA

Compute Unified Device Architecture

DDS

Data Distribution Service

DOF

Degree of Freedom

DR

Domain Randomization

DRL

Deep Reinforcement Learning

ELU

Exponential Linear Unit

EMA

Exponential Moving Average

EoAT

End-of-Arm Tool

FORGE

FOrce-Guided Exploration for Robust Contact-Rich Manipulation under Uncertainty

GPU

Graphics Processing Unit

FP

False Positive

IK

Inverse Kinematics

IL

Imitation Learning

IMU

Inertial Measurement Unit

IPC

Inter-Process Communication

KL

Kullback-Leibler

LSTM

Long Short-Term Memory

MDP

Markov Decision Process

MLP

Multi-Layer Perceptron

ONNX

Open Neural Network Exchange

OSIC

Operational Space Impedance Controller

PID

Proportional-Integral-Derivative

POMDP

Partially Observable Markov Decision Process

PPO

Proximal Policy Optimization

PTP

Point-to-Point

QoS

Quality of Service

RDK

Robotic Development Kit

RL

Reinforcement Learning

RNN

Recurrent Neural Networks

ROS

Robot Operating System

ROS2

Robot Operating System 2

RTOS

Real-Time Operating System

TCP

Tool Center Point

TRPO

Trust Region Policy Optimization

UDR

Uniform Domain Randomization

USD

Universal Scene Description

Chapter 1

Introduction

Robotic systems constitute the backbone of modern industrial production, facilitating end-to-end assembly through state-of-the-art manipulators. Traditionally, these tasks are executed within highly structured and deterministic workspaces that demand millimetric precision. In such contexts, conventional control strategies, relying on inverse kinematics, trajectory planning, and sample-based motion control, represent the industry standard [1]. These methodologies achieve high repeatability and accuracy through explicitly programmed control pipelines and rigorously defined environmental parameters.

Despite their widespread adoption, these classical techniques exhibit a fundamental lack of adaptability when faced with unstructured failures or minor environmental perturbations. Such systems typically require manual intervention or the implementation of rigid, pre-programmed recovery routines to address unforeseen deviations. In this landscape, Deep Reinforcement Learning (DRL) offers a significant paradigm shift. By enabling agents to learn optimal control policies autonomously through interaction, mapping current states to actions based on a scalar reward signal, DRL provides a framework for dynamic adaptation to failures and non-deterministic conditions.

Although Reinforcement Learning has seen extensive application in fields such as humanoid locomotion, its deployment in high-precision industrial assembly remains relatively limited. The stochasticity of learned policies often represent a barrier to achieve the stringent tolerances required for industrial-grade tasks. This thesis addresses this gap by presenting a comprehensive end-to-end pipeline for training, evaluating, and deploying a DRL agent on a physical industrial manipulator.

In order to investigate the difficulties of high-precision assembly, the *Gear Meshing* task was chosen. Representing a contact-rich scenario selected because it demands exceptional precision, far exceeding the requirements of standard pick-and-place baselines. This task involves inserting a gear onto a peg with a 0.5 mm clearance while aligning teeth with adjacent gears, a process requiring complex

physical interaction.

The learning architecture utilizes the Proximal Policy Optimization (PPO) algorithm [2], leveraged through the `rl-games` library. Training was conducted within the `NVIDIA Isaac Lab` framework, a GPU-accelerated platform built upon `NVIDIA Isaac Sim` and the `Omniverse` ecosystem [3, 4]. This environment enables high-fidelity, physically-based simulations, allowing for the massive parallelization required for efficient DRL training.

The robotic arm employed for this research is the **Flexiv Rizon 4s** [5], a 7-degree-of-freedom (7-DOF) redundant manipulator specifically designed for adaptive tasks. The Rizon 4s is distinguished by its industrial-grade force control and high-performance joint-torque sensors, providing a sensing resolution of up to 0.03 N. These features make it an ideal candidate for tasks requiring "human-like" dexterity and the ability to respond to complex contact forces during the assembly process.

In order to bridge the significant gap between simulation amd the physical world, training was conducted within a customized version of the `FORGE` environment [6]. Originally developed for the Franka Emika Panda, `FORGE` represents the state-of-the-art for contact-rich Reinforcement Learning (RL) tasks. Built upon the `IndustReal` framwork [7], it extends capabilities by incorporating force observations, a force-thresholding mechanism to condition the agent's behavior as well as success prediction for early termination. The agent exploits only proprioceptive information: the current pose, forces measured at the Tool Center Point (TCP), and the previous action, thereby eliminating the need for external sensors and minimizing hardware complexity. To facilitate zero-shot transfer, the environment utilizes extensive Domain Randomization (DR) [8], stochastically varying asset geometries, robot dynamics, and control parameters. This approach exposes the agent to a diverse distribution of environmental conditions, effectively modeling the measurement noise and physical discrepancies inherent in real-world deployment as well as making the robot able to operate in a variety of starting conditions.

The robustness of the trained policy was validated through a rigorous tri-level hierarchy:

1. **In-Environment Training performance:** The initial validation layer involving the training of the agent to confirm convergence and basic task proficiency.

2. **Sim-to-Sim Replica:** A secondary layer utilizing a high-fidelity replica of the training scene in `NVIDIA Isaac Sim`. This stage integrates Robot Operating System 2 (ROS2) [9] nodes to stream robot state and receive control commands, while policy inference is handled by a standalone Python script using `PyTorch`.

3. **Physical Deployment:** The final validation involves zero-shot transfer to the physical **Flexiv Rizon 4s**. The deployment maintains the same ROS2-based control logic, with the agent operating on a real-time Linux kernel

machine. To replicate the simulated conditions, all manipulated components were 3D-printed.

The primary contributions of this thesis are as follows:

- **First - Deployment on the Flexiv Rizon 4s:** This work documents the first known application of this high-precision gear meshing task on the Rizon 4s, establishing a performance benchmark for this adaptive manipulator in RL-driven assembly.

- **Second - Exportation of `rl-games` Policies:** It introduces a novel workflow for the exportation and real-world deployment of policies trained using the `rl-games` library, bridging the software gap between specialized RL frameworks and industrial controllers.

- **Third - End-to-End Reinforcement Learning Pipeline:** The thesis provides a comprehensive overview of a complete pipeline for contact-rich, high-precision tasks, detailing the training dynamics, the custom Sim-to-Sim validation architecture, and the challenges encountered during physical deployment.

1.1 Thesis Outline

The remainder of this thesis is organized into the following chapters:

- **Chapter 2:** provides a comprehensive review of the current landscape in robotic assembly. It examines the inherent challenges of high-precision manipulation, the fundamental principles of Reinforcement Learning, and the methodologies employed to mitigate the Sim-to-Real gap, including the domain randomization techniques.

- **Chapter 3** delineates the technical framework of the research. This includes a detailed description of the simulation framework, reinforcement learning libraries, and deployment pipeline. Additionally, it specifies the hardware used for training and deployment, as well as the characteristics of the Flexiv Rizon 4s robotic arm.

- **Chapter 4:** offers an in-depth analysis of the agent's architecture and the specific configurations of the `FORGE` environment. It details the state and action spaces, the reward function design, and the contact-physics parameters relevant to the gear meshing task.

- **Chapter 5:** describes the multi-stage validation pipeline. It outlines the development of the high-fidelity Sim-to-Sim IsaacSim-based replica, and the procedural steps for physical robot deployment.
- **Chapter 6:** presents the data collected across all testing phases. It analyzes training performance, evaluates metrics in simulated and physical environments, and discusses the findings of conducted studies.
- **Chapter 7**: summarizes the primary contributions of this thesis. Outlines the lessons learned regarding high-precision Sim-to-Real transfer and proposes potential avenues for future research, such as the integration of multimodal perception, the realization of a full assembly automated sequence as well as more advanced learning paradigms.

Chapter 2

Background and Literature Review

Before detailing the methodologies, experimental procedures and results of this research, it is of fundamental relevance to establish the theoretical foundation upon which this work of research is built. For the purpose, this chapter provides a comprehensive overview of the core concepts in robotic manipulation and Reinforcement Learning (RL). Furthermore, it highlights the specific problem addressed by this thesis, discussing the current state-of-the-art within the field of robotic assembly and identify the limitations of traditional control strategies this research aims to overcome.

2.1 Robotic Assembly and Manipulation Tasks

Modern manufacturing is founded on the concept of the "industrial robot", formally defined by the ISO 8373:2021 standard as a *reprogrammable, multifunctional manipulator designed to move material, parts, tools, or specialized devices through variable programmed motions for the performance of a variety of tasks* [10]. This definition encapsulates the versatility that has lead to a widespread adoption of robotic manipulators in production lines such as automotive assembly (see Figure 2.1).

When programmed with classical control techniques, such as such as Inverse Kinematic (IK) solvers, Proportional-Integral-Derivative (PID) loops, or pre-defined trajectory planners, the robots excel in completing their assigned tasks with exceptional accuracy and repeatability. These systems are designed for structured environments where the task parameters, such as the exact pose of the manipulated objects or the friction coefficients of a surface, are determined a-priori. However, the performances of these "scripted" behaviors is fundamentally determined by the

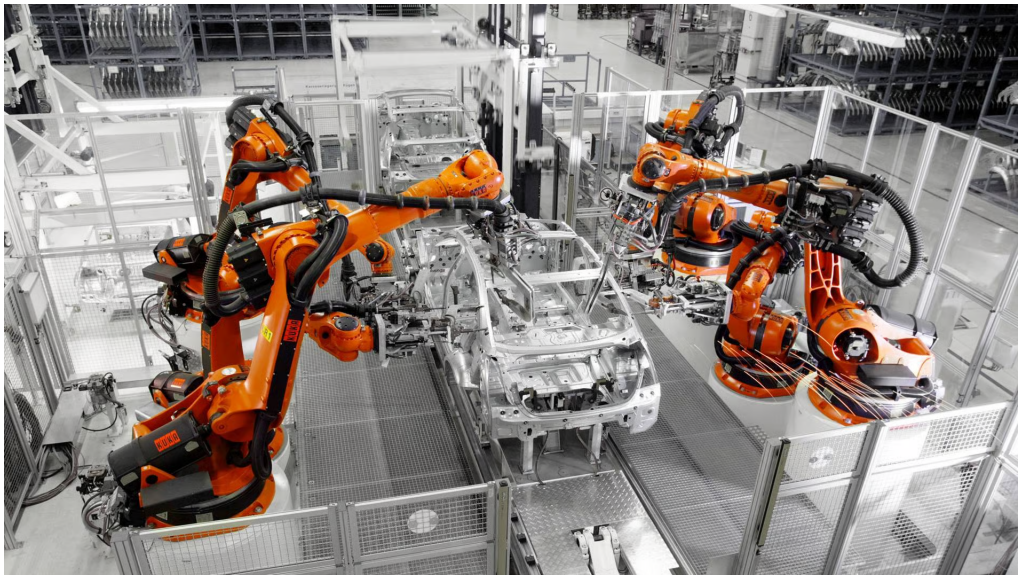


Figure 2.1: Industrial automotive assembly utilizing robotic manipulators [11].

environment remaining consistent with the initial model throughout the entire task life-cycle. This represents the primary limitation of the aforementioned control techniques; if any unforeseen event disturbs the environment (i.e. like microscopic misalignment) the robotic manipulator fails. Purely deterministic controllers, in such cases, lack the adaptability to recover from unmodeled events, causing the robot to typically incur in mechanical jamming [12].

The primary inspiration of this thesis is taken from this exact vulnerability. By investigating the effectiveness of Reinforcement Learning in assembly tasks as an adaptive control strategy, this research explores a paradigm shift from *programmed determinism* to *learned adaptability*. The ultimate objective is to establish a baseline for robotic manipulation where the trained agent must rely on real-time sensor data, rather than pre-scripted rules, autonomously learning an adaptive strategy that leads to task completion.

2.2 Reinforcement Learning Overview

Within the broad domain of Artificial Intelligence (AI), Reinforcement Learning represents the learning paradigm we, as individuals, are the most familiar with. Its core principle is that an autonomous agent, during training, must experiment within an environment to discover which actions yield the highest cumulative benefit. To do so, it constantly receives a delayed scalar signal, the reward, that evaluates its current state. The ultimate objective is to maximize the cumulative reward over

time [13]. During training, agents experience a wide variety of scenarios, including edge cases that may be difficult for engineers to account for through traditional scripting. This continuous exposure allows the agent to learn a behavior that is extremely robust to unforeseen dynamics.

To further detail the Reinforcement Learning framework, it is crucial to underline how the behavior of the agent differs from the training phase to the post-training deployment stage. During training, the agent's actions are primarily driven by trial and error. Here, the agent may take sub-optimal decisions to discover the environment's response and identify high-reward regions of the state space. More formally, this stage is called the **exploration phase** and represents the basis of Reinforcement Learning.

Conversely, once the agent is fully trained, it leverages its learned knowledge to select the optimal (or near-optimal) actions with the aim of maximizing the task performance. This stage is called **exploitation**.

A fundamental characteristic of Reinforcement Learning is that the agent aims to maximize the return, defined as the summation of rewards collected over an entire task execution (episode). This allows the agent to learn long-term strategies, potentially accepting a low immediate reward (or even a penalty) if such an action is a prerequisite for reaching a high-value goal state later in the episode.

2.2.1 Comparison between Reinforcement Learning and Other Paradigms

To fully understand the paradigm shift introduced by Reinforcement Learning, it is helpful to compare it with other classical learning techniques such as **Supervised Learning** and **Unsupervised Learning**. In the first category, the learning procedure is characterized by a set of labeled data (e.g., predicting a house's market price based on its features). The feedback is instantaneous and provided by a "teacher" in the form of the correct label.

In the second learning strategy, Unsupervised Learning, instead the model aims to discover relevant common patterns or clusters relying on unlabeled data, hence without exploiting any external judge or explicit feedback signal [14].

Reinforcement Learning completely differs from the two aforementioned techniques. Its foundations are rooted in the exploration that the agent, autonomously, conducts . There are no labels or pre-scripted rules. The only guiding elements are the state of the environment, the actions that the agent can take, and the reward he gets. This fundamental "unstructureness" is what enables Reinforcement Learning agents to excel in adaptation.

2.2.2 Formalization of Reinforcement Learning Elements

Having discussed the key concepts of Reinforcement Learning in order to highlight its strengths and differences from more common learning techniques, a rigorous explanation of the key elements characterizing this framework is useful for the following sections. Its key components are as follows:

- **Agent:** The autonomous entity that observes the environment and executes actions.
- **Environment:** The external system with which the agent interacts, encompassing all dynamics and physical constraints.
- **Observations (Ω):** The partial or complete information the agent receives about the environment's state. In robotics, this typically includes proprioceptive data (joint positions, velocities) and external sensor data (force-torque measurements).
- **Action (a):** The set of possible interactions available to the agent (e.g., applying a Cartesian delta or a joint torque).
- **Policy (π):** The mapping from observations to actions. A policy can be deterministic (outputting a single action) or stochastic (outputting a probability distribution over actions).
- **Reward (R):** The scalar feedback signal received immediately following an action.
- **Value Functions (V, Q):** While the reward represents immediate gain, value functions estimate the expected future return. The State-Value function $V(s)$ represents the value of being in a state, while the Action-Value function $Q(s, a)$ represents the value of taking a specific action in a specific state.
- **Model:** A representation of the environment's transition dynamics, $P(s'|s, a)$.

Figure 2.2 depicts the learning flow of Reinforcement Learning. The trained agent, at each training step t, observes the environment and computes an action a_t to take according to its policy π. As a consequence, the environment emits a reward signal r_t that the agent stores. This iterative process forms the basis of the learning signal used to update the policy.

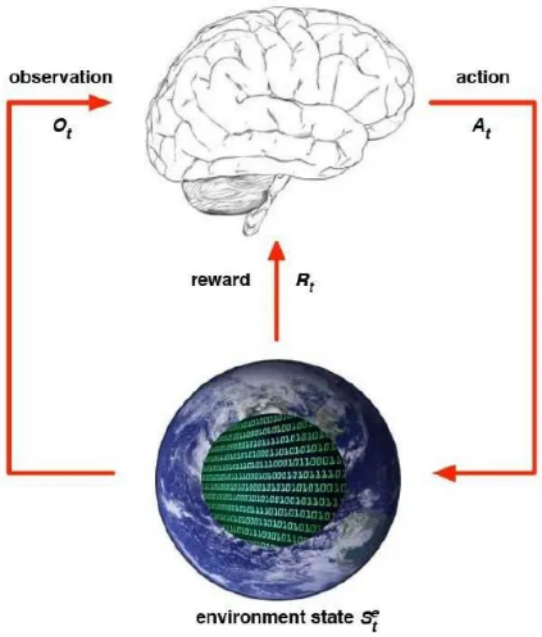


Figure 2.2: Example schema of Reinforcement Learning.

2.2.3 Reinforcement Learning Mathematical Modeling

To provide a rigorous overview of the mathematical concepts constituting the foundation of Reinforcement Learning, it is essential to first define the framework of Markov Decision Processes (MDP). An MDP is formally defined as a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ where:

- $\mathcal{S}$ is a finite set of states
- $\mathcal{A}$ is a finite set of actions a
- $\mathcal{P}$ is a state transition matrix, s.t.

$$P^a_{ss'} = P(S_{t+1} = s' \mid S_t = s, A_t = a)$$

- $\mathcal{R}$ is a reward function, s.t.

$$R^a_s = \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a]$$

- γ is a discount factor, $\gamma \in [0,1]$

The key premise of such processes is the Markov Property, which posits that *"the future is independent of the past given the present"*. This implies that the current

state S_t of the process captures all relevant information from the history, therefore the agent is capable of choosing the most profitable action to execute by using only current information.

In the context of MDPs the ultimate objective of the agent is to maximize the *expected return*, defined as the discounted sum of future rewards:

$$G_t = R_{t+1} + \gamma R_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{2.1}$$

From Equation 2.1 it is evident that the discount factor γ is a weight given to the future rewards, therefore a value near to 0 leads to "myopic" evaluation while 1 leads to "far-sighted" evaluation.

Comparison between MDPs and Reinforcement Learning MDPs constitute the structural backbone of Reinforcement Learning, however a fundamental distinction lies between the two. Within the context of RL, typically, the environment's dynamics (i.e. the transition matrix $\mathcal{P}$) are unknown, therefore the agent must approximate these underlying distributions through iterative interaction and trial-and-error.

Value Functions and the Bellman Equation The aim of the training procedure is to learn a policy $\pi(a|s)$ that maximizes the expected return. To enable the agent to evaluate the long-term desirability of specific states or actions, the **state-value function** $V^\pi(s)$ and the **action-value function** $Q^\pi(s,a)$ are introduced:

$$V^\pi(s) = \mathbb{E}_\pi \left[G_t \mid s_t = s \right] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \,\middle|\, s_t = s \right] \tag{2.2}$$

$$Q^\pi(s,a) = \mathbb{E}_\pi \left[G_t \mid s_t = s, a_t = a \right] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \,\middle|\, s_t = s, a_t = a \right] \tag{2.3}$$

These equations are fundamental for the RL agent since they allow to quantify the potential benefit of the state or the state-action pair. In fact, intuitively, $V^\pi(s)$ measures the profitability of starting from a specified state s and then following the policy π. On the other hand, $Q^\pi(s,a)$ quantifies the benefit of starting from state s, taking a specified action a and then continuing to follow the policy π. Thanks to these measures, the agents are able to evaluate the goodness of their policy and update it to increase the expected cumulative return.

These functions satisfy recursive identities known as the **Bellman Equations** [15], defined as:

$$V^\pi(s) = \mathbb{E}_{a\sim\pi(\cdot|s)} \mathbb{E}_{s'\sim P(\cdot|s,a)} \left[R(s,a) + \gamma V^\pi(s') \right] \tag{2.4}$$

$$Q^{\pi}(s,a) = \mathbb{E}_{s' \sim P(\cdot|s,a)} \mathbb{E}_{a' \sim \pi(\cdot|s')} \left[R(s,a) + \gamma Q^{\pi}(s',a') \right] \tag{2.5}$$

Both the equations 2.4 and 2.5 intuitively indicate how the value of a state (or state-action pair) is equal to the immediate reward R_t plus the discounted value of successor states (or state-action pairs). This property is derived from the recursive nature of the return G_t (see Equation 2.1).

In conclusion, the optimal value functions, indicated respectively as $V^{\star}$ and $Q^{\star}$, satisfy the following Bellman optimality equations:

$$V^{\star}(s) = \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s' \mid s,a) \left[R(s,a) + \gamma V^{\star}(s') \right] \tag{2.6}$$

$$Q^{\star}(s,a) = \sum_{s' \in \mathcal{S}} P(s' \mid s,a) \left[R(s,a) + \gamma \max_{a' \in \mathcal{A}} Q^{\star}(s',a') \right] \tag{2.7}$$

For model-based Reinforcement Learning, these equations can be solved directly via dynamic programming. If the dynamics of the environment are known, then the value functions can be computed, and the policy can be iteratively updated to maximize them. In contrast, this research focuses on model-free methods, where the agent must estimate these optimal values directly from sampled trajectories without explicit knowledge of the transition probabilities $P^{a}_{ss'}$.

2.2.4 Taxonomy of Reinforcement Learning Algorithms

To contextualize the selection of the algorithm used in this research, it is essential to categorize the landscape of Reinforcement Learning methods based on their architecture.

Figure 2.3 illustrates a hierarchy of the Reinforcement Learning algorithms currently representing the state-of-the-art in the field, grouped according to their main learning paradigms. It is evident that the primary distinction relies on whether or not a model of the environment is available.

Model-Based vs. Model-Free RL The fundamental difference between these types of algorithms lies in the availability, or tentative acquisition, of the state transition matrix $P(s'|s,a)$.

- **Model-Based RL:** For these kind of algorithms, the agent is either given a model of the environment (e.g. AlfaGo) or attempts to learn one from experience (e.g. World Models). While these techniques are highly sample-efficient, they are often computationally intensive. Moreover, if applied to an environment governed by complex non-linear dynamics (i.e. robotic assembly), notoriously difficult to be modeled analytically, they often lead to performance degradation due to the model significantly deviating from reality.

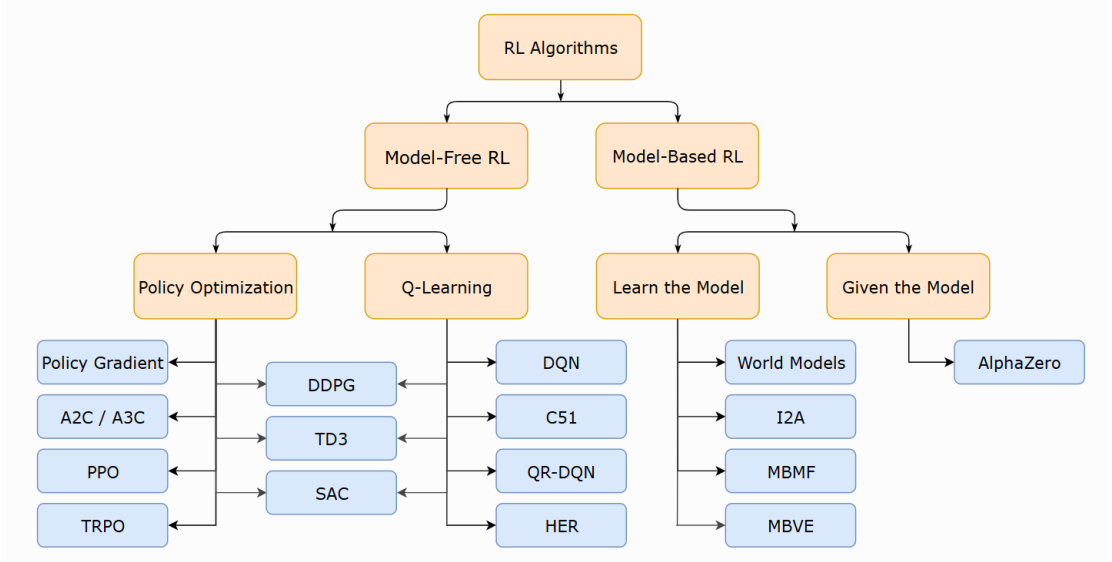


Figure 2.3: Taxonomy of Reinforcement Learning algorithms [16].

- **Model-Free RL:** Conversely, model-free RL algorithms, which constitutes the core of this research, operate without an explicit internal representation of the environment. They learn directly through trial and error. This makes them the most suitable choice for high complexity environments like contact-rich industrial tasks where the underlying physics are too intricate to be captured by a simplified transition function.

Value-Based, Policy-Based, and Actor-Critic Methods Within the broad variety of Model-Free RL algorithms, another fundamental distinction is represented by their optimization target:

- **Value-Based (Q-Learning):** Algorithms such as DQN or QR-DQN aim to learn an optimal action-value function $Q^*(s, a)$. The policy is implicitly derived by selecting actions greedily: $a = \arg\max_a Q(s, a)$. While effective for discrete action spaces, these methods often struggle with the continuous control required in robotics.
- **Policy Optimization:** These algorithms, in contrast, totally turn the paradigm upside down. They directly optimize the parameters θ of a stochastic policy $\pi_\theta(a|s)$ using gradient ascent on the expected return. The learned policies, therefore, result in superior efficiency in high-dimensional, continuous action spaces where it's difficult to build a faithful representation of the world.

These two strategies converge in the **Actor-Critic** framework, of which the algorithms are characterized by two sets of parameters. The first, the **Critic**,

estimates the action-value function; hence it is responsible for policy evaluation. The **Actor**, conversely, learns the policy and updates its parameters in the direction suggested by the Critic. Modern algorithms, such as the Proximal Policy Optimization (PPO) utilized in this work, often leverage this dual-structure to minimize gradient variance and ensure stable policy updates.

2.2.5 Proximal Policy Optimization (PPO)

Proximal Policy Optimization (PPO) is one of the most widely adopted algorithms in the modern landscape of robotic Reinforcement Learning, primarily due to its balance between performance and training stability. As established in the previous taxonomy, PPO is a model-free, on-policy actor-critic algorithm that belongs to the Policy Optimization family. Its derivation is fundamentally rooted in the need to resolve the instabilities inherent in standard Policy Gradient methods while avoiding the prohibitive computational costs of second-order optimization techniques [2].

Since PPO is selected as the only learning algorithm for this research, it is fundamental to detail the path leading to its formulation as well its main characteristics, thus enabling to understand the rationale behind its choice.

Stabilization in Policy Gradient Methods As previously established, Policy Gradient methods optimize a parameterized policy by ascending the gradient of the expected return. The fundamental **Policy Gradient Theorem** [17] formalizes this gradient $\nabla_\theta J(\theta)$ as:

$$\nabla_\theta J(\theta) = \mathbb{E}\pi \left[\nabla\theta \log \pi_\theta(a_t \mid s_t) G_t\right] \tag{2.8}$$

where G_t represents the cumulative return (Equation 2.1). Despite their theoretical advantages, policy-based method generally suffer high-variance leading to destructive policy updates. Excessive magnitudes in the parameters θ variation during a single gradient step, can cause the policy to collapse, losing previously learned beneficial behaviors due to outliers or localized stochastic noise. To mitigate this variance, different strategies are adopted within the broad family of Policy-based methods.

One of the most effective concepts in literature is the transition from the raw cumulative reward, to the **Advantage Function** $\mathcal{A}^\pi(s, a)$, defined as:

$$A^\pi(s_t, a_t) = Q^\pi(s_t, a_t) - V^\pi(s_t), \tag{2.9}$$

Intuitively, $A^\pi(s, a)$ quantifies the relative benefit of taking a specific action a compared to the average behavior dictated by the current policy at state s. Substituting the return G_t with $A^\pi(s, a)$ yields a more stable gradient:

$$\nabla_\theta J(\theta) \approx \mathbb{E}_\pi \left[\nabla_\theta \log \pi_\theta(a_t|s_t) A^\pi(s_t, a_t) \right], \tag{2.10}$$

This strategy allows the agent to focus on actions that perform better than expected, thus stabilizing the gradient updates. Despite reducing variance, the methods based on Advantage Function, however, do not prevent excessively large policy updates.

In order to address this limitation, Trust Region Policy Optimization (TRPO) methods had been developed [18]. The key idea is to constraint the policy update within a "Trust Region" using the **Kullback–Leibler (KL) divergence**:

$$D_{KL}(\pi_\theta|\pi_{\theta_{old}}) \approx \frac{1}{2}(\theta - \theta_{old})^T H (\theta - \theta_{old}) < \epsilon \tag{2.11}$$

Therefore D_{KL} is used as a constraint to ensure the policy doesn't change too drastically in one update, while H is the Hessian matrix of the KL divergence, used in the second-order approximation.

While TRPO-based algorithms proved to excel in policy stabilization, the computational burden introduced by the Hessian matrix creates a practical difficulty in its actual real-world implementation. To solve this specific issue, PPO is derived as an approximation of TRPO.

PPO Mathematical Formulation The key principle at the basis of PPO is to relax the mathematical computation of TRPO by employing a first-order **Clipped Surrogate Objective**. This ensures the policy updates do not exceed unwanted magnitudes without requiring complex matrix inversions.

Defining $\rho_t(\theta)$ as the importance sampling ratio between the current and previous policy:

$$\rho_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \tag{2.12}$$

PPO optimizes the following objective function J^{CLIP}:

$$J^{CLIP} = \min \left[\rho_t(\theta) A^{\pi_\theta}(s,a), \text{clip}(\rho_t(\theta), 1-\epsilon, 1+\epsilon) A^{\pi_\theta}(s,a) \right] \tag{2.13}$$

where ϵ is a hyperparameter (typically 0.1 or 0.2) that limits the maximum acceptable change in the policy ratio. By taking the minimum of the unclipped and clipped objectives, PPO effectively ignores updates that would push the policy too far away from the current distribution, particularly when the advantage is positive (to prevent over-optimism) or negative (to prevent drastic avoidance).

In conclusion, PPO offers a robust and computationally efficient approximation of Trust Region methods. By utilizing the clipped objective alongside an actor-critic architecture, PPO provides the stability necessary for high-dimensional continuous

control tasks, such as gear meshing, where small errors in the early stages of training could otherwise lead to divergent behavior.

2.3 Simulation to Reality gap

Robots operating in real-world environments face numerous challenges due to the unpredictability and variability of their surroundings. Directly training a robot in the physical world can be prohibitively expensive, time-consuming, and risky, especially when dealing with complex tasks. Reinforcement learning has emerged as a promising approach for teaching robots autonomous control, but its reliance on large-scale data collection poses significant practical constraints. Moreover, in real-world applications, the specification of key concepts of RL such as the reward is complex.

Simulated environments, paved the way for the adoption of Reinforcement Learning by overcoming these limitations. The foundation of RL is built upon the concept of exploration, which intrinsically requires the agent to make wrong decisions and potentially cause damage. In this context, a simulation framework provides a safe environment where agents can experience failure and collapse without material consequences. Additionally, simulation provides "perfect" ground-truth data for sensor inputs, facilitating the construction of complex observation vectors that would otherwise require costly external metrology in the real world.

However, the transition from simulated environments to the real world, referred to as Sim-to-Real transfer, presents a critical challenge [19]. The gap, formally, refers to the differences in dynamics, sensory noise, and environmental interactions between the simulation and real-world systems, often resulting in poor performance when policies trained in simulation are deployed on real robots.

For high-precision assembly tasks like Gear Meshing, characterized by a 0.5 mm diametrical clearance, the gap is primarily driven by:

- **Physics Modeling Errors:** Inaccurate approximations of non-linear friction, contact mechanics (stiffness/damping), and mechanical backlashes.

- **Observation Noise:** Discrepancies between the idealized, noise-free simulated sensors and the stochastic, latency-affected measurements of real-world torque and proprioceptive sensors.

- **System Latency:** Asymmetries in communication timing between the control unit and the actuators, which can lead to divergent behavior in high-frequency feedback loops.

2.3.1 Domain Randomization (DR)

In order to bridge the Sim-to-Real gap, the most widely adopted technique is Domain Randomization (DR). Pioneered by researches at OpenAI [8], DR introduction represented a major breakthrough in Reinforcement Learning adoption, significantly improving the performance of the policies deployed on real robots.

The core concept of this methodology is to expose the agent, during the training phase, to a wide variety of the main environmental parameters. If the agent is trained on a sufficiently broad distribution, the real world will appear to the policy as just another randomized variation.

Intuitively, the utility of DR is two-fold:

1. **Real-World Parameter Coverage:** Under significant uncertainty about the specific value of physical parameters, randomization within a rational interval could statistically enable the agent to encounter the specific "true" values it will face during physical deployment. In this context, accurate modeling of the interval boundaries and types of the distribution from which to extract the parameters is fundamental.

2. **Invariance Learning:** By exposing the agent to constant variations in non-essential parameters, the policy becomes robust to these variations, thus learning to prioritize the invariant features and ignore noise.

Within the Domain Randomization techniques, the most widely adopted are Uniform Domain Randomization (UDR) and Automatic Domain Randomization (ADR). The first involves extracting the randomized parameters from a uniform distribution, where the boundaries are defined a-priori and kept constant throughout the entire training procedure. Conversely, ADR aims to progressively learn the appropriate parameter randomization distribution by specifically adjusting its boundaries: these are expanded when the agent performs well on samples drawn near the current limits, and contracted otherwise.

In conclusion, Domain Randomization transforms the Sim-to-Real problem from one of system identification to one of robustness. By treating the real world as a noisy outlier of the training distribution, DR enables the zero-shot transfer of complex manipulation policies, providing the necessary resilience for high-precision tasks like Gear Meshing where the margin for error is minimal.

2.4 Challenges of Applying Reinforcement Learning to High-Precision Robotic Tasks

Reinforcement Learning has undoubtedly achieved remarkable success in various robotic domains, especially in locomotion and basic pick-and-place operations.

However, its application to high-precision industrial assembly represents a barrier of significant difficulty. Unlike the mentioned domains, these task involve sub-millimetric clearances, thus introducing a set of challenges that standard RL frameworks often struggle to handle.

2.4.1 The Precision-Stochasticity Paradox

As extensively discussed, the core strengths of Reinforcement Learning, such as adaptability and robustness to unforeseen disturbances, should enable agents to excel in assembly tasks, theoretically representing a superior alternative to traditional scripted controllers in less structured environments. However, the main limitations in this field are directly derived from the very nature of RL: stochasticity. For algorithms such as PPO, the stochasticity of the learned policy is a structurally desired characteristic that constitutes the main strength of the exploration-exploitation trade-off [2]. Conversely, this intrinsically random nature represents a barrier in industrial assembly, where deterministic precision is fundamental. A task characterized by a 0.5 mm clearance requires high-frequency deterministic corrections; even marginal variance in the control signal can result in a mechanical jam or "dead-lock" state.

Standard RL algorithms inherently struggle to handle the strict dimensional tolerances of industrial manufacturing. To test these limits, the *ManiSkill2* framework introduced the `PlugCharger` peg-in-hole assembly task, deliberately modeled with realistic dimensions to enforce a strict 0.5 mm clearance [20]. When standard visual RL policies were trained on this sub-millimetric task from scratch for 25 million time steps, they reached a state of complete failure, achieving a success rate of just 0.01% [20]. The benchmark authors concluded that existing RL algorithms are currently insufficient to perform such highly precise controls, emphasizing the necessity for novel compliance strategies.

To further isolate why RL policies fail in these environments, ablation studies explicitly controlled the difficulty of the assembly clearances. When the clearance threshold was increased by a factor of 10 to make the task artificially easier, the RL agents achieved significantly higher performance. This empirical evidence directly supports what can be termed the *Precision-Stochasticity Paradox*: standard RL frameworks possess the adaptability for general manipulation, but the stochastic nature of their exploration and execution makes them highly fragile when the task strictly demands sub-millimetric deterministic accuracy.

This paradox defines the core technical challenge of this research: the development of a policy that translates high-fidelity proprioceptive force-torque signatures into precise, deterministic corrective movements. By learning a high-precision compliance strategy through force-feedback, this work aims to bridge the gap between the stochastic flexibility of RL and the deterministic requirements of industrial

gear meshing.

Chapter 3

Methodologies

This chapter provides a comprehensive overview of the methodologies utilized throughout this research. In particular, it details the training and Simulation framework within the environment was modeled and the agent was trained, as well as the hardware architecture supporting this stage. Furthermore, the deployment pipeline and associated software tools are described, with specific emphasis on the communication protocol and the robotic manipulator employed.

3.1 Training and Simulation Frameworks

The training process has been performed using the state of the art frameworks for reinforcement learning training and high-fidelity physics-based robotics simulation. To support these computationally intensive tools, a powerful workstation was utilized, ensuring the necessary processing power to execute the complete training and evaluation pipeline efficiently.

3.1.1 IsaacSim

NVIDIA IsaacSim is an advanced physical simulator built to serve as a robust development platform for AI-enabled robotics systems, often referred to as AI-enabled Cyber Phyisical Systems (CPS) [3]. Its primary utility lies in providing a physically accurate virtual framework that allows for the iterative testing of complex robotic tasks without the operational risks or hardware costs associated with physical experimentation.

In the context of Reinforcement Learning (RL) this safety represents a key enabler. The inherent exploration-driven nature of RL requires agents to execute suboptimal or "wrong" actions to discover the underlying reward structure. In high-precision assembly tasks, such exploratory behavior in the real world could

result in mechanical failure of the manipulator or damage to the manipulated components. IsaacSim mitigates these risks by providing a risk-free environment for stochastic exploration.

The core capabilities of IsaacSim are to be found in its seamless integration with NVIDIA hardware and software ecosystem, offering support for high-speed, GPU-accelerated simulation and AI training.

Its key architectural features, as delineated by the authors of [3], include:

- **Universal Scene Description (USD)**: IsaacSim utilizes the USD framework, originally developed by Pixar Studios. In the context of NVIDIA IsaacSim, this open-source 3D framework allows for the hierarchical assembly of complex virtual environments, comprising the robotic manipulator, the manipulated gears, and secondary scene elements such as worktables and lighting, all within a unified file format.

- **Sensor Simulation and Privileged Information:**: A fundamental advantage of IsaacSim is its ability to simulate diverse sensor modalities, including IMUs, cameras, and force-torque sensors. IsaacSim provides built-in libraries for sensors that replicate the real hardware.

- **Massive Parallelization**: Unlike CPU-based simulators, IsaacSim supports the execution of hundredths of environments in parallel. This massive parallelization allows for rapid accumulation of information and the collection of samples that are strongly correlated in time. By exposing the agent to a vast variety of initial conditions and domain-randomized parameters in parallel, the framework significantly enhances both the speed and the quality of the policy's convergence.

In this thesis, the application of IsaacSim is twofold. **First**, it serves as the foundation for `NVIDIA IsaacLab`, the extension used for agent training. **Second**, a high-fidelity replica of the real-deployment scene has been produced within IsaacSim for the Sim-to-Sim validation process. This replica, other than managing the physics simulation, utilizes the `ros2_bridge` extension. Such software library enables a seamless communication interface to stream inbound telemetry and outbound control commands, thereby mirroring the real-world ROS2 deployment architecture.

3.1.2 IsaacLab

Building upon the capabilities of NVIDIA IsaacSim, this research utilizes `IsaacLab` for training. Representing the successor to the GPU-native Isaac Gym, designed specifically for large-scale multi-modal robot learning. IsaacLab, within its main

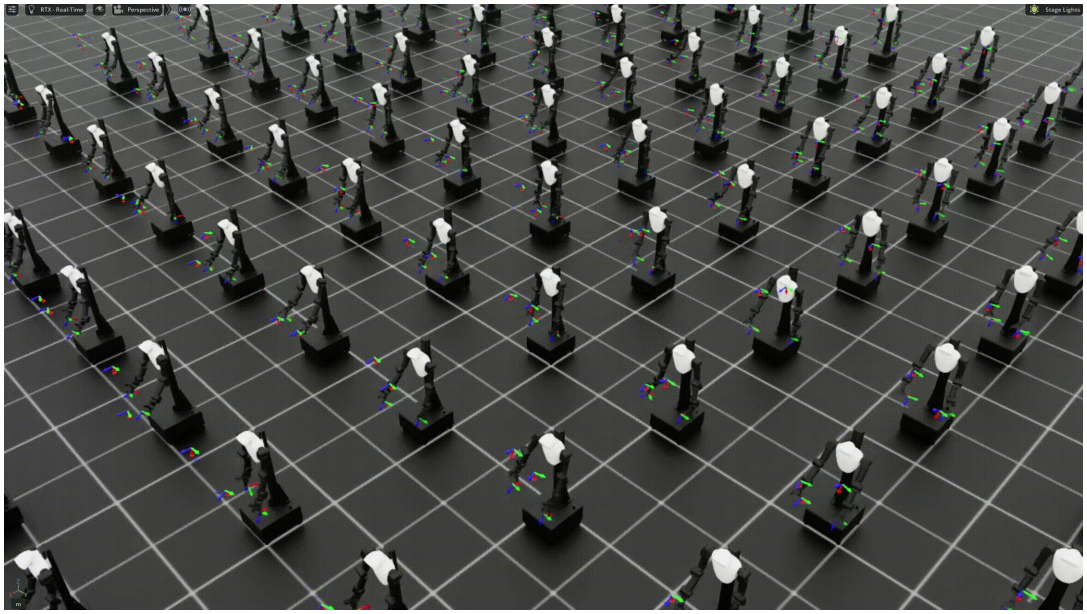


Figure 3.1: Example of parallelization within NVIDIA IsaacSim

characteristics, serves as a modular, task-oriented wrapper optimized for Reinforcement Learning and Imitation Learning (IL) [21].

NVIDIA IsaacLab provides a comprehensive framework for robot learning, covering the enntirety of the training-testing pipeline from environment setup to policy training.

As delineated by the authors of [4], IsaacLab offers two distinct paradigms for environment design

1. **Manager-Based Workflow:** A high-level, modular approach that encapsulates Markov Decision Process (MDP) components, such as rewards, observations, and terminations, into reusable managers, promoting code extensibility.

2. **Direct Workflow:** A performance-centric interface that allows for low-level interaction with simulation buffers, prioritized for computationally sensitive training pipelines.

Within the diverse suite of environments offered by IsaacLab, this research employs the `FORGE` environment. Supporting, at the time of writing, only Franka Emika Panda, it represents a method for Sim-to-Real transfer of force-aware manipulation policies in the presence of significant pose uncertainty. From the implementation point of view, it falls in the category of the Direct Worflows, thus enabling for low-level interaction within the physics environment [6].

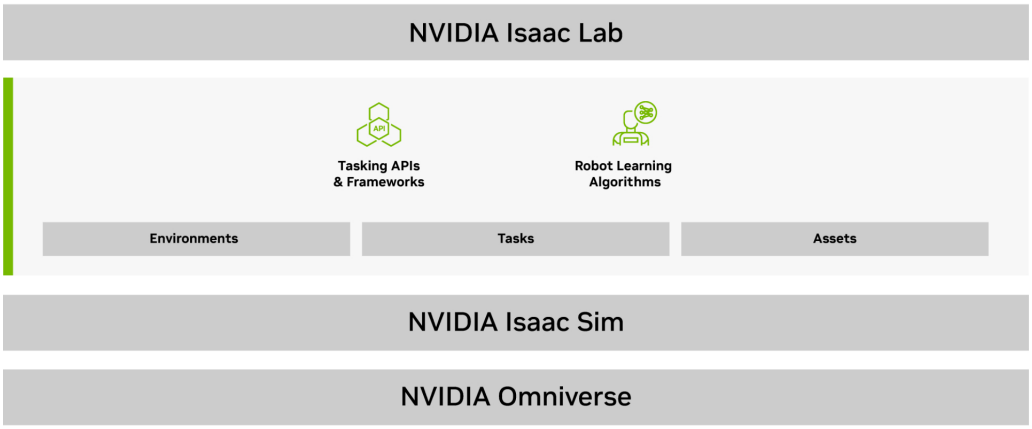


Figure 3.2: NVIDIA IsaacLab architecture [21]

As a wrapper of IsaacSim, IsaacLab enhances the robot assets by exposing an extended version of the `Articulation` class. A pivotal feature for this research is the `get_link_incoming_joint_force` method, which enables the agent to measure the multidimensional spatial forces experienced by the gripper. Its intensive usage within training allows the agent to develop the sensitivity required for contact-rich assembly, as detailed in the subsequent sections.

3.1.3 Workstation

The entirety of the training and Sim-to-Sim validation workflows were executed on a high-performance dedicated workstation. To ensure research reproducibility and isolate the complex dependencies required by the simulation stack, a containerized approach was adopted using `Docker`. A specialized container was architected to host the `NVIDIA IsaacLab` and `IsaacSim` installations, ensuring a consistent execution environment across different stages of the pipeline. The details of the containerized environment are reported Appendix A

The technical specifications of the hardware and software environment of the workstation are summarized below:

- **CPU:** AMD Ryzen 9 9950X. This 16-core, 32-thread processor (operating between 600 MHz and 5.75 GHz) provides the high single-core clock speeds necessary to mitigate Python-side bottlenecks in the environment-step logic, while the high core count supports background ROS2 processing.

- **RAM:** 64 GB DDR5.
- **GPU:** NVIDIA RTX A6000 with 48 GB (49,140 MiB) of GDDR6 VRAM. This substantial memory buffer is a critical enabler for vectorized Reinforcement Learning, allowing for the storage of large observation buffers and the concurrent execution of up to 512 parallel environments without VRAM saturation.
- **Driver and CUDA Stack:** NVIDIA Driver version 570.172.08 with CUDA Runtime 12.8.
- **Operating System:** Ubuntu 24.04.3 LTS (Noble Numbat) utilizing Linux Kernel 6.14.0-29-generic.
- **Containerization Architecture:** Docker Engine 28.4.0 integrated with the NVIDIA Container Toolkit 1.18.0-rc.3, enabling low-latency GPU passthrough to the virtualized environment.

The utilization of this enterprise-grade hardware was fundamental to the success of the training phase. By enabling the parallelization of 512 simultaneous environments, the workstation allowed for the rapid collection of millions of interaction steps, significantly reducing the wall-clock time required for the PPO agent to converge on a stable policy for the contact-rich gear meshing task.

3.2 Deployment Pipeline and Tools

This section details the deployment tools and procedures, together with the software and hardware employed. The architecture is designed to maintain parity between the simulation-based validation (Sim-to-Sim) and the physical robot deployment. Both pipelines share the control logic, with the primary distinction being the execution environment: for Sim-to-Sim validation, the policy interacts with the virtual robot and assets simulated within `IsaacSim` on the workstation detailed in Section 3.1.3, whereas real-world deployment utilizes the physical manipulator and assets, with the deployment script running on a real-time Linux machine, as described in Section 3.2.3.

3.2.1 Robot Operating System (ROS) 2

In order to implement the communication for both inbound telemetry and outbound control of the robot, the Robot Operating System (ROS) 2 is employed. In this research, the deployment is achieved through an independent python script that runs the policy inference. In order to firstly construct the observation dictionary

and second controlling the robot, communication with the robot is needed. In this context, ROS2 serves as the primary middleware for orchestrating the robot's modular software components. ROS2 is an open-source, peer-to-peer robotics framework that facilitates communication between heterogeneous processes, referred to as *nodes*, through a structured messaging protocol.

The main characteristics of ROS2 rely on the *Data Distribution Service* (DDS) as its underlying communication layer. Unlike the centralized "Master" node architecture of its predecessor (ROS1), ROS2 utilizes a fully decentralized, discovery-based system. This transition to DDS provides several industrial-grade advantages:

- **Quality of Service (QoS):** Allows for fine-grained control over data transmission, such as setting reliability policies (Reliable vs. Best-Effort), durability, and history depth, which is critical for high-frequency sensor streams.
- **Real-Time Capabilities:** The architecture supports real-time execution requirements by minimizing latency and jitter in inter-process communication (IPC).
- **Security:** Integration of DDS-Security standards (SROS2) enables authentication and encryption for sensitive robotic data.

In the context of this research, ROS2 serves as the bridge between the deployment script and the broader robotic stack. This integration is managed by the adoption of the `rlcpy` python library in the deployment script. For the robot, the implementation varies from real-robot deloyment and sim-to-sim evaluation.

Sim-to-Sim In simulation, ROS2 connectivity is achieved through the *ROS2 Bridge* extension within `NVIDIA Omniverse`, which translates internal simulator tensors into standard ROS2 messages. Moreover, an additional layer of abstraction was needed. Flexiv robots utilize a custom messaging, namely (`flexiv_msgs`) . Its details are to be found in Appendix B. In order to adapt IsaacSim ROS2 messages to Flexiv standard, an additional python layer has been developed re-encapsulating the standard IsaacSim messages into the specific formats required by the Flexiv software stack. This ensures that the inference logic remains identical to the real-world deployment.

Real deployment For physical deployment, communication is managed directly by the `flexiv_ros2` drivers [22]. This bridge serves as a ROS2 wrapper for the Flexiv Robotic Development Kit (RDK), implementing the hardware interface for `ros2_control` and `MoveIt!2`. The drivers are executed on a dedicated Linux workstation connected to the robot via a private Ethernet link. This configuration enables low-latency communication within the local subnet, allowing for real-time joint torque and position control of the Rizon 4s.

3.2.2 Flexiv Rizon4s Manipulator

The robotic manipulator utilized in this research is the **Flexiv Rizon4s**, a 7-degree-of-freedom (7-DOF) adaptive manipulator engineered to replicate the dexterity and flexibility of a human arm. The Rizon4s features a redundant kinematic architecture with a total weight of 21 kg and a maximum payload capacity of 4 kg. Its workspace is characterized by a maximum reach of 919 mm when fully extended.

Although currently less prevalent in academic literature compared to competitors such as the Franka Emika Panda or Universal Robots manipulators, the choice of this manipulator was crucial. Rizon4s, indeed, was selected specifically for its industry-leading, force-aware manipulation capabilities. It integrates high-performance joint-torque sensors and advanced whole-body force control algorithms, facilitating human-like sensitivity in contact-rich tasks. A critical feature for this research is the manipulator's force-sensing accuracy of 0.03 N and a maximum Tool Center Point (TCP) force capacity of 150 N. This level of precision is fundamental to the gear meshing task, where the agent must interpret subtle resistance forces to achieve successful alignment. The multidimensional force-torque data experienced by the TCP is streamed in real-time via the ROS2 interface, as detailed in Appendix B.

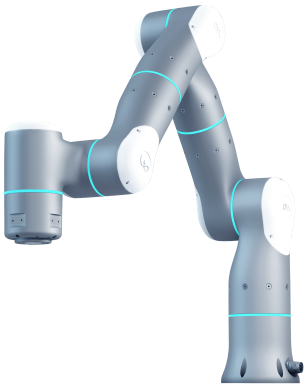


Figure 3.3: The Flexiv Rizon 4s 7-DOF adaptive manipulator.

As every standard industrial manipulator, the robot is interfaced via a dedicated Teach Pendant, an Ubuntu-based tablet running *Flexiv Elements*. This proprietary operating system provides an intuitive, flowchart-based programming environment that utilizes "primitives" to simplify complex adaptive tasks. For the purposes of this thesis, while manual operation and trajectory recording are possible through *Flexiv Elements*, the deployment of the reinforcement learning policy is managed

through the custom pipeline detailed in Section 3.3.

HESPER robot controller

The robotic arm is powered by the **HESPER** controller, a high-performance external computing unit dedicated to realtime execution of force control algorithms. Featuring a 12 kg weight and a power rating of 500 W, the HESPER unit is engineered to manage the computational density of whole-body force control. It serves as the primary gateway for the robot's industrial fieldbus communications, supporting protocols such as TCP/IP and Profinet.

In the architecture of this research, the HESPER controller resides at the base of the manipulator, facilitating low-latency TCP/IP communication with the real-time linux machine to ensure the responsive execution of the deployment script.

Figure 3.4: The Flexiv HESPER robot controller, responsible for 1 kHz deterministic control loops.

GRAV Gripper

In order to enable the manipulator to interact with the physical world, the robotic arm is equipped with the **GRAV** gripper, a 2-finger force-controlled end-of-arm tool (EoAT). The GRAV gripper is specifically designed for seamless integration with the Rizon series and provides a programmable gripping force range from 1 N to 100 N with a 1 N force control accuracy, allowing for the secure yet sensitive handling of the 3D-printed gear assets. Its communication with the robot is integrated into the manipulator itself.

In the context of this research, the gripper ensured for tight holding of the held asset.

Figure 3.5: The Flexiv GRAV force-controlled gripper.

3.2.3 Real-Time Linux Machine

Initial experimental trials indicated that standard Linux distributions are unsuitable for the direct control of high-precision assembly tasks via ROS2. When utilizing a conventional kernel, the communication between the deployment script and the HESPER controller exhibited significant temporal *jitter*. This non-deterministic behavior resulted in irregular command intervals, manifesting as "choppy" or oscillating manipulator movements that compromised the agent's ability to perform delicate meshing.

To achieve the required determinism, the deployment machine was configured with a real-time Linux kernel utilizing the `PREEMPT_RT` patch. This modification transforms the standard kernel into a hard real-time operating system (RTOS) by making nearly all kernel code paths preemptible. By assigning the ROS2 inference nodes high scheduling priority and minimizing interrupt latency, the `PREEMPT_RT` kernel ensured a stable 1 kHz control loop. This transition to a deterministic execution environment was critical for bridging the Sim-to-Real gap, as it ensured that the policy's temporal dynamics in the physical world remained congruent with the discretized time steps (Δt) of the training simulation.

3.3 Deployment Setup

This section delineates the architectural structure of the deployment pipeline for both Sim-to-Sim validation and Sim-to-Real implementation. Particular emphasis is placed on the communication flow between the distinct software and hardware agents. Figure 3.6 illustrates the bifurcated workflow utilized in this research.

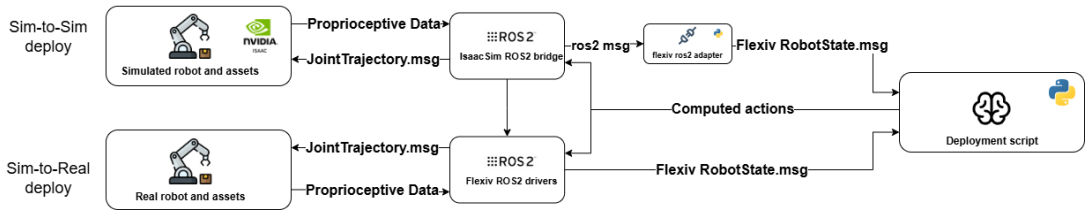


Figure 3.6: Schematic representation of the Sim-to-Sim and Sim-to-Real deployment pipelines.

Sim-to-Sim Pipeline In the simulation-based validation setup, the robot is modeled within a high-fidelity replica of the training environment in `NVIDIA IsaacSim`. During task execution, the virtual manipulator continuously streams its proprioceptive state via the ROS2 bridge. This state vector includes the Flange pose (comprising Cartesian coordinates $p = [x, y, z]^\top$ and orientation in quaternions $q = [q_w, q_x, q_y, q_z]$) and the external wrench measured at the Flange, expressed in the world coordinate frame.

To maintain strict parity with the physical hardware, the deployment script performs a spatial transformation of these measurements. Specifically, the wrench data is re-projected from the Flange reference frame to the Tool Center Point (TCP) frame used during training phase. The raw data stream from the `IsaacSim` ROS2 nodes is then processed and re-serialized into the `flexiv_msgs/RobotStates` format. This intermediate abstraction layer ensures information consistency, regardless of whether the source is a simulator or physical hardware.

Sim-to-Real Pipeline The physical deployment phase transitions from synthetic modeling to the real-world manipulator. Utilizing the Flexiv HESPER control unit, proprioceptive telemetry is acquired directly through the `flexiv_ros2` drivers, which provide access to the robot's hardware abstraction layer. Unlike the simulation pipeline, this stage does not require an additional message-translation layer, as the data is natively published in the proprietary Flexiv format.

To ensure safe and smooth interaction during the contact-rich gear meshing process, the robot is operated under a **Joint Trajectory Impedance Controller** provided by the Flexiv RDK. This controller implements a mass-spring-damper virtual dynamic at the joint level, allowing the robot to exhibit compliant behavior when encountering the mechanical resistance of the gear teeth.

Despite the discrepancies in the underlying physics engines between IsaacSim and the real manipulator, both pipelines utilize an identical control logic implemented within the python deployment script. At each discretization step (dt), the deployment script computes the optimal Joint Trajectory. These commands are published to the robot, which closes the loop by tracking the desired TCP position

while maintaining the specified impedance parameters. This unified approach minimizes the divergence between the simulated agent's behavior and its real-world performance.

Chapter 4

Training Procedure

This chapter provides a comprehensive overview of the training methodology utilized to develop the Reinforcement Learning (RL) policy. To ensure the reproducibility of the research, each phase of this critical stage is documented in detail. The analysis encompasses the core architectural features of the training environment, the formulation of the task as a Partially Observable Markov Decision Process (POMDP), and the domain randomization strategies employed to mitigate the Sim-to-Real gap.

Furthermore, the actions computed by the policy are detailed. During training, they are manually transformed to ensure smooth robot control. Such behavior is replicated in the deployment script that is detailed in the next sections. Lastly the adaptations needed to switch from the natively supported Franka Emika Panda to Flexiv Rizon4s are described, together with the agent configuration for all the trainings conducted.

4.1 FORGE Environment description

The environment utilized in this research is `FORGE` (**FOrce-Guided Exploration for Robust Contact-Rich Manipulation under Uncertainty**), as introduced by [6]. As the nomenclature evidences, the environment is specifically designed to leverage force-torque feedback to overcome the complexities of contact-rich manipulation under significant spatial uncertainty.

Building on top of the IndustReal framework (see [7]), FORGE introduces a critical expansion: the integration of force observations into the agent's state representation. Such enhancement enables for smooth agent's behavior which is essential in low-clearance assembly tasks. To fully exploit these features, a manipulator capable of high-fidelity force sensing is required; thus, the Flexiv Rizon 4s, with its industry-leading adaptive control and sub-Newton sensitivity, serves as

an exceptional candidate for this research.

The FORGE environment implements three primary assembly tasks:

- **Peg Insertion**: Involves the insertion of a round peg (8 mm diameter) into a socket with **0.5mm** diametrical clearance.
- **Gear Meshing**: Requires the agent to mount a gear onto a fixed peg with a 0.5 **mm** clearance while simultaneously aligning its teeth with adjacent, pre-placed gears. This task necessitates complex physical reasoning and multi-point contact management.
- **Nut threading**: Requires a nut to be threaded onto a peg until mechanical interlock is achieved, preventing vertical extraction.

Figure 4.1 represents the three FORGE tasks with their original implementation within the Franka Emika Panda manipulator.

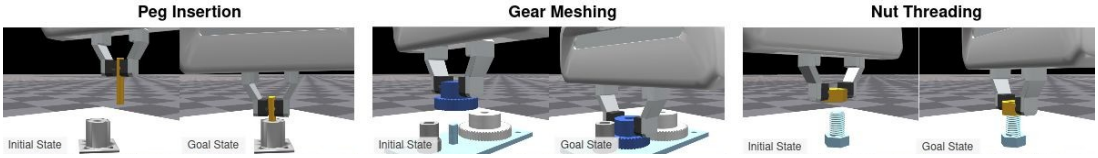


Figure 4.1: Visual representation of the initial and goal states for the three FORGE benchmarks, originally implemented with the Franka Emika Panda [6].

For the scope of this research, the *Gear Meshing* task was selected as the primary focus. This task represents the highest degree of difficulty within the FORGE suite, as it requires not only high-precision insertion but also the dynamic alignment of three distinct sets of gear teeth. This interaction presents a non-trivial challenge for RL agents, requiring a sophisticated balance between reaching the goal pose and responding to the reactive forces generated by tooth-on-tooth collisions.

4.1.1 Innovations Relative to Previous Works

The FORGE environment represents a major step ahead over its predecessor, the IndustReal framework. While sharing the same three implemented tasks, FORGE extends IndustReal with three architectural contributions designed to improve the robustness and safety of learned policies:

1. **Force-Thresholding Mechanism:** During policy execution a force-threshold is set. If the forces experienced at the TCP exceed this predefined threshold, the agent receives a penalizing negative reward. This constraint incentivizes the discovery of "gentle" assembly strategies, reducing mechanical stress on

the components. Furthermore, this mechanism effectively decouples policy performance from rigid controller gains, mitigating the need for exhaustive gain tuning during physical deployment.

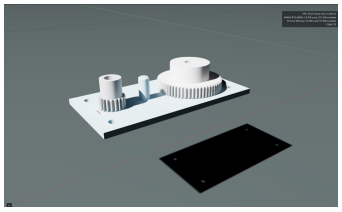
2. **Comprehensive Dynamics Randomization scheme**: To facilitate zero-shot Sim-to-Real transfer, FORGE implements an extensive randomization scheme. This includes stochastic perturbations of the robot's link dynamics, joint friction, and control latencies, as well as the physical properties (e.g., mass, friction, and localized geometry) of the manipulated assets. This approach ensures the policy is resilient to the discrepancies between the simulated physics engine and real-world dynamics.

3. **Success prediction**: Alongside the control policy, a success predictor is trained to output a scalar value ranging from -1 to $+1$, representing the estimated proximity to task completion. This feature is at the foundation of the deployment architecture, providing an objective metric for the deployment script to trigger early termination upon successful assembly.

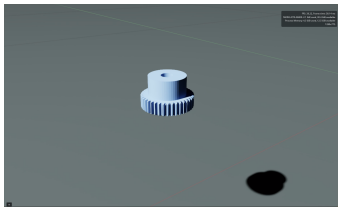
4.1.2 The Gear Meshing Task: System Architecture and Spatial Configuration

The Gear Meshing task is the primary focus of this research due to its high dimensionality and contact-rich complexity relative to standard assembly benchmarks. The completion of the task requires the coordination of three main actors, respectively: the robotic manipulator, the held gear (held asset), and the fixed assembly base (fixed asset).

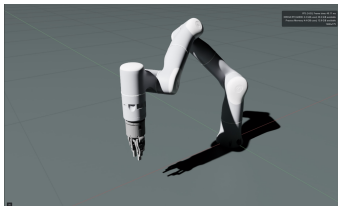
Figure 4.2 illustrates the physical components of the training environment.



(a) Fixed asset: Central peg and flanking gears.



(b) The Held asset: Medium-sized gear.



(c) Rizon 4s robotic manipulator.

Figure 4.2: Core actors of the training environment.

The fixed asset is constituted of a central mounting peg and flanked by two secondary gears of different dimension, which impose strict rotational alignment constraints on the teeth. The held asset, represented by a medium size gear, is held tightly in the manipulator gripper.

The initial configuration of such components is randomized at the start of each training episode (see Section 4.4 for the randomization scheme in the details) . Regardless of the fixed asset's base position, the TCP is always located between 25 mm and 45 mm above the insertion peg. To ensure the policy generalizes to slight positioning errors, a spatial tolerance is applied to this starting pose: ± 2 **cm** along the X and Y axes.

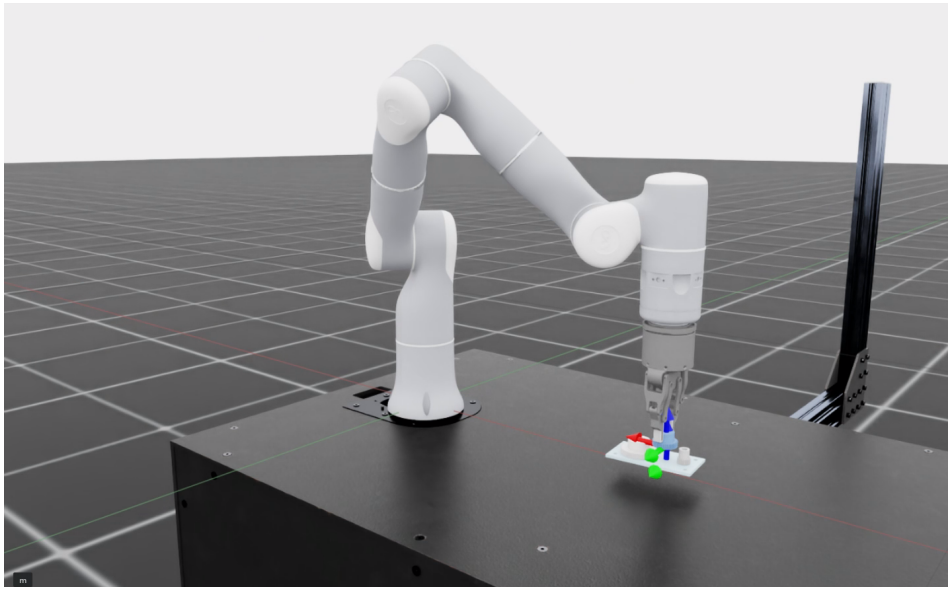


Figure 4.3: Example of a training starting condition

As highlighted in Figure 4.3, the task starts with the manipulator in this approach state. The Figure also depicts the two reference frames associated to the TCP pose (p^{tcp}) and the fixed base (p^{fixed}) which represents the insertion target. The global world reference frame is coincident with the robot's base frame, providing a consistent origin for the transformation matrices

4.2 Partially Observable Markov Decision Process formulation

The problem undertaken by the FORGE environment is formulated as a Partially Observable Markov Decision Process (POMDP).

4.2.1 State Space ($\mathcal{S}$)

The state $s_t \in \mathcal{S}$ represents the ground-truth configuration of the system. Following the `state_order` specification, the state vector is structured as:

- **Ground Truth Poses**: The absolute poses of the Tool Center Point (TCP) (p^{tcp}), the fixed base (p^{fixed}), and the held gear (p^{held}), all in $SE(3)$.
- **Relative Transform**: The relative position of the held gear to the fixed target $p_{rel}^{held} \in \mathcal{R}^{\ni}$.
- **Velocities**: Ground-truth linear and angular velocities for the TCP and held object ($v^{tcp}, v^{held} \in \mathbb{R}^6$).
- **Joint Configuration**: The current robot joint positions.
- **Contact Forces**: The true contact force ($F^{tcp} \in \mathbb{R}^3$).
- **Dynamics and Controller Properties (Ψ)**: Time-invariant information about the robot, controller, and parts (`task_prop_gains`, `ema_factor`, `pos_threshold`, `rot_threshold`).

4.2.2 Observation Space (Ω)

The observation vector $o_t \in \Omega$ consists of noisy estimates and task-relevant parameters available to the policy. Following, it includes:

- **Noisy TCP Pose**: The relative fingertip position and orientation ($\hat{p}^{tcp} \in SE(3)$).
- **Noisy TCP Velocity**: Estimated linear and angular velocities ($\hat{v}^{tcp} \in \mathbb{R}^6$).
- **Smoothed Estimated Contact Force**: The force-torque sensor data ($\hat{F}_{smooth}^{tcp} \in \mathbb{R}^3$).
- **Force threshold**: The scalar force threshold τ_f conditioning agent behavior.
- **Previous action**: The previous action vector a_{t-1}, comprising the delta targets and the success prediction.

Please note that the observations are affected by manually inserted noise. Noise configuration is detailed in Section 4.5

4.2.3 Reward Function Design

The reward function aims to lead the agent to perform insertion task. It is structured as a combination of continuous distance-based positive rewards, action regularization and discrete bonuses in case of success.

Keypoint Alignment To capture both translational and rotational errors, the environment utilizes a set of N keypoints defined in the local frame of the manipulated assets. The keypoints are equally spaced by an offset of 0.15. Let $\mathbf{p}_i^{held}$ and $\mathbf{p}_i^{fixed} \in SE(3)$ be the world-frame poses of the i-th keypoint for the held and fixed objects, respectively. The mean keypoint distance d_{kp} is defined as:

$$d_{kp} = \frac{1}{N} \sum_{i=1}^{N} \|\mathbf{p}_i^{held} - \mathbf{p}_i^{fixed}\|_2 \tag{4.1}$$

This distance is processed through a multi-stage *squashing function* to provide different levels of precision. The squashing function is defined as:

$$S(d, a, b) = \frac{1}{1 + a \cdot d^b} \tag{4.2}$$

where a and b are coefficients determining the sensitivity of the reward to the distance. The total keypoint reward is the sum of three such terms:

$$r_{kp} = w_{base} S(d_{kp}, a_0, b_0) + w_{coarse} S(d_{kp}, a_1, b_1) + w_{fine} S(d_{kp}, a_2, b_2) \tag{4.3}$$

This multi-resolution approach provides a broad gradient for initial reaching (r_{base}), a medium gradient for alignment (r_{coarse}), and a highly sensitive gradient for the final insertion phase (r_{fine}).

Regularization and Success prediction To ensure kinematic smoothness and penalize aggressive behaviors, two penalty terms are applied to the actions $\mathbf{a}_t$:

- **Action Magnitude Penalty**: $r_{mag} = -w_{mag}\|\mathbf{a}_t\|_2$, penalizing high-effort control signals.
- **Action Gradient Penalty**: $r_{grad} = -w_{grad}\|\mathbf{a}_t - \mathbf{a}_{t-1}\|_2$, encouraging temporal smoothness and reducing oscillations.

To ensure kinematic smoothness and facilitate deployment, the reward includes terms for action magnitude, temporal gradients, and success forecasting:

- **Asset-Relative Penalty**: Penalizes large displacements relative to the asset frame, normalized by thresholds τ_{pos} and τ_{rot}:

$$r_{act,rel} = -w_r \left(\frac{\|\Delta p\|_2}{\tau_{pos}} + \frac{|\Delta \psi|}{\tau_{rot}} \right) \tag{4.4}$$

- **Success Prediction Error**: As described in Sec 4.1.1, the policy outputs an auxiliary prediction $\hat{s}_t \in [-1, 1]$. The agent is penalized for the absolute error relative to the ground-truth success $\mathbb{I}_s$:

$$r_{pred} = -\omega_p \cdot |\mathbb{I}_s - \hat{s}_t| \tag{4.5}$$

  This penalty is only activated once the agent achieves a stable success ratio of 0.25, ensuring the auxiliary task does not interfere with early-stage exploration.

Force-Thresholding and Safety A central innovation of the FORGE environment is the penalization of excessive contact forces. This encourages the policy to perform "gentle" exploration and prevents hardware damage during real-robot deployment. The penalty is calculated using a Rectified Linear Unit (ReLU):

$$r_{force} = -\omega_f \cdot \max(0, \|\mathbf{F}^{tcp}_{smooth}\|_2 - \tau_f) \tag{4.6}$$

where $\|\mathbf{F}_{smooth}\|_2$ is the magnitude of the smoothed force-torque data and τ_f is the safety threshold. The safety threshold is randomized each episode for each agent, with a minimum value of 5 and a maximum of 10.

Unified Reward Formulation The final reward $\mathcal{R}_t$ at each timestep is the weighted sum of these components, plus a discrete bonus for engagement and final success:

$$\mathcal{R}_t = r_{kp} + r_{mag} + r_{grad} + r_{force} + r_{act,rel} + r_{reg} + \omega_e \mathbb{I}_{engaged} + \omega_s \mathbb{I}_{success} \tag{4.7}$$

4.3 Control Logic

At training-time, the manipulator's motion is controlled by an Operational Space Impedance Controller (OSIC), a control strategy that is further replicated in real-deployment through Flexiv RDK. This control strategy translates the high-level Cartesian targets generated by the policy into low-level joint torques, effectively modeling the robot's end-effector as a virtual spring-damper system.

While the learned policy updates the target at a frequency of 15 Hz, the underlying impedance controller and physics simulation execute at $120Hz$. As a result, each policy action is kept constant for a total of 8 consecutive simulation steps, ensuring smooth trajectory tracking.

The controller computes a virtual wrench, comprising force $\boldsymbol{F}_{cmd}$ and torque $\boldsymbol{\tau}_{cmd}$ at the TCP to minimize Cartesian error relative to the target pose while damping the resultant motion. The control law is defined as follows:

$$\boldsymbol{F}_{cmd} = \boldsymbol{K}_p^{pos} \boldsymbol{e}_p - \boldsymbol{K}_d^{pos} \dot{\boldsymbol{x}} \tag{4.8}$$

$$\boldsymbol{\tau}_{task} = \boldsymbol{K}_p^{rot} \boldsymbol{e}_o - \boldsymbol{K}_d^{rot} \boldsymbol{\omega} \tag{4.9}$$

where:

- $\boldsymbol{K}_p^{pos}, \boldsymbol{K}_p^{rot} \in \mathbb{R}^{3\times3}$ are diagonal stiffness matrices that define the "rigidity" of the virtual spring.
- $\boldsymbol{K}_d^{pos}, \boldsymbol{K}_d^{rot} \in \mathbb{R}^{3\times3}$ are damping matrices. To ensure a non-oscillatory response, these are configured for critical damping, where $\boldsymbol{K}_d = 2\sqrt{\boldsymbol{K}_p}$.

- $\boldsymbol{e}_p$ and $\boldsymbol{e}_o$ represent the translational and rotational error vectors, respectively.
- $\dot{\boldsymbol{x}}$ is the linear velocity and $\boldsymbol{\omega}$ is the angular velocity of the end-effector.

The Cartesian wrench is mapped to joint torques $\boldsymbol{\tau}_{imp}$ by means of the transpose of the geometric Jacobian matrix $\boldsymbol{J}(\boldsymbol{q}_{joint})$:

$$\boldsymbol{\tau}_{imp} = \boldsymbol{J}(\boldsymbol{q}_{joint})^T \begin{bmatrix} \boldsymbol{F}_{cmd} \\ \boldsymbol{\tau}_{task} \end{bmatrix} \tag{4.10}$$

Furthermore, the final torque sent to the robot motors includes terms for gravity compensation $\boldsymbol{g}(\boldsymbol{q}_{joint})$ and Coriolis/centrifugal forces $\boldsymbol{c}(\boldsymbol{q}_{joint}, \dot{\boldsymbol{q}}_{joint})$, thus linearizing the dynamics:

$$\boldsymbol{\tau}_{total} = \boldsymbol{\tau}_{imp} + \boldsymbol{g}(\boldsymbol{q}_{joint}) + \boldsymbol{c}(\boldsymbol{q}_{joint}, \dot{\boldsymbol{q}}_{joint}) \tag{4.11}$$

This control strategy ensures that the agent interacts with a simplified, quasi-linear system where actions directly correspond to task-space displacements.

4.3.1 Policy Actions and Post-Processing

The RL policy generates a 7-dimensional action vector $\mathbf{a}_t \in \mathbb{R}^7$ at each inference step. While the final component represents success prediction $\hat{s}_t$ (detailed in Section 4.1.1), the first coefficients define the desired displacement $(\Delta p, \Delta\theta)$ in the fixed asset's reference frame. To ensure smooth manipulator's motion and compatibility with the world-frame Operational Space Controller, the policy's actions require a multi-stage post-processing that is also replicated in the deployment script.

Transformation to World Frame Since the control logic expects to receive the target pose of the TCP expressed in the world-frame, a transformation is needed to map the actions from the fixed asset frame to the world frame. Also, since the actions represent deltas from the previous step, the TCP position p^{tcp} is summed, thus obtaining the target TCP position to reach.

Position Action Scaling and Clipping In order to stabilize training and prevent aggressive trajectories, the raw position deltas are modulated by a scaling factor `action_scale` (σ_{pos}) of 0.05. Furthermore, a clipping is applied to restrict the maximum displacement per step:

- **Scaling:** The raw outputs are multiplied by $\sigma_{pos} = 0.05$, thus avoiding actions that are excessive in magnitude.
- **Clipping:** Moreover, each translation component is constrained to $\lambda_{pos} = \pm 0.02\,\text{m}$. This 2 cm per-step limit ensures that even if the policy outputs a saturated signal, the robot maintains a safe velocity profile suitable for high-precision meshing.

Rotational Constraints Unlike the position actions, manual scaling and clipping are omitted to allow the agent full authority over the alignment phase. However, to focus the learning to the relevant workspace, the orientation is limited to the $(+x, -y)$ quadrant of the world frame. In addition, the rotation are limited to the only Z-axis by zeroing the x, y components. This effectively reduces the orientation search to the yaw angle required to align the gears' teeth.

Furthermore, rotation deltas are clipped in order to allow for maximum rotation of $\pm 5.56°$

Actions smoothing As a last preprocessing step, actions are smoothed to further prevent jittery agent's behavior. Equation 4.12 describes the Exponential Mean Average smoothing the actions are subject to.

$$a_t^{smooth} = \alpha \cdot a_t + (1 - \alpha) \cdot a_{t-1}^{smooth} \tag{4.12}$$

4.4 Randomization Scheme

Key enabler for robust simulation to reality transfer is the randomization of the environment. By exposing the agent to a variety of starting conditions, it learns to adapt to a broad set of environmental parameters.

Within the FORGE environment, the randomizations impact multiple parameters, specifically: geometrical/positional randomization, physics randomization and control randomization.

Geometrical/positional randomization The starting poses of the manipulated assets are randomized. Specifically, the fixed asset position is offset by a random value sampled from a uniform distribution, scaled by a 0.05 coefficient. Hence, the fixed base initial position is randomized by a $\pm 5cm$ in all the axis directions. Within geometrical randomization, also its initial orientation is randomized. The Z-axis yaw angle orientation is, indeed, randomized of a total of ± 15.0 degrees.

As described in Section 4.1.2, the robotic manipulator starting configuration is selected with respect of the fixed asset randomized position. The robot, hence, is set above the insertion peg at a on offset of $35mm$ by applying its Inverse Kinematics (IK). Furthermore, randomization is applied in order to bring the robot's hand within a range of $\pm 0.02m$ along the X and Y axis, while $\pm 0.01m$ with respect to the Z-axis. Also the hand orientation value is randomized, with the only Z-axis orientation randomly selected within $\pm 22{,}35°$.

In order to simulate an imperfect grasp, the held asset initial position (within the gripper) is randomized of $\pm 0.003m$ along x and z axis.

Physical properties randomization In order to take into account for diverse physical parameters, the mass and friction coefficients of some the actors are randomized. Specifically, mass randomization only takes place for the held asset, extracted from a uniform distribution within the range of $[0.000, 0.010]kg$.

Friction randomization is, conversely, applied only to the fixed asset, randomly selected within a range of $[0.25, 1{,}25]$.

Control dynamics randomization Regarding the control logic, randomization takes place in relation to the Exponential Moving Average (EMA) factor, the controller gains and the action scale.

The EMA factor α is randomly selected from a range of $[0.025, 0.1]$, while the action scaling coefficient is randomized by $\pm 25\%$, therefore the actions translation varies from a minimum of 1.6cm to a maximum of 2.5cm. The rotation randomization based on a factor of $\pm 29\%$, referring to the maximum allowable rotation value of $5.5°$, which leads to a rotation limit varying in the range of $[4.3, 7.1]°$.

The control dynamics randomization also extends to the proportional gains ($\boldsymbol{K}_p^{pos}, \boldsymbol{K}_p^{rot}$) of the impedance controller. The gains are, indeed, randomized with a 50% chance of being multiplied, or divided, by a factor of 1.41. This consequently affects the derivative control gains since they are calculated based on the proportional values.

4.5 Noise Injection

In order to take into account for the non-ideality of sensor measurements, a manual noise injection process is applied. Contrary to the randomization, that takes place at the environment reset at the beginning of each episode, noise injection happens at every physical step. The trained agent, therefore, always operates with noisy measurements, representing a key enabler for real deployment.

TCP pose Noise The TCP position $p^{tcp} \in \mathbb{R}^3$ and orientation $q^{tcp} \in \mathbb{H}$ are perturbed as follows:

$$\tilde{p}^{tcp} = p^{tcp} + \epsilon_{pos}, \quad \epsilon_{pos} \sim \mathcal{N}(0, \sigma_{pos}^2) \tag{4.13}$$

$$\tilde{q}^{tcp} = q^{tcp} \otimes q_{noise}(\epsilon_{rot}), \quad \epsilon_{rot} \sim \mathcal{N}(0, \sigma_{rot}^2) \tag{4.14}$$

where $\otimes$ denotes quaternion multiplication and q_{noise} creates a small random rotation around an arbitrary axis.

Force Sensor Noise The 6-DoF force-torque sensor readings $F_{raw} \in \mathbb{R}^6$ are smoothed and then corrupted by additive Gaussian noise:

$$\tilde{F}_{obs} = \text{EMA}(F_{raw}) + \epsilon_{force}, \quad \epsilon_{force} \sim \mathcal{N}(0, \sigma^2_{force} I) \tag{4.15}$$

Table 4.1 summarizes the specific noise values.

Table 4.1: Noise Injection Parameters and Values

Parameter	Distribution	Value / Range	Update Freq.
TCP Position Noise	Gaussian	$\sigma = 0.25$ mm (0.00025 m)	Per Step
TCP Rotation Noise	Gaussian	$\sigma = 0.1^\circ$ (0.0017 rad)	Per Step
Force Sensor Noise	Gaussian	$\sigma = 1.0$ (Sim Units/N)	Per Step
Target Calibration Error	Uniform	± 1.0 mm ($[0.001]^3$ m)	Per Episode
Hand Init Position Noise	Uniform	± 2.0 cm (X,Y), ± 1.0 cm (Z)	Per Episode
Hand Init Rotation Noise	Uniform	± 0.39 rad ($\approx 22^\circ$)	Per Episode

4.6 Flexiv Rizon4s Adaptation

The FORGE Environment released by NVIDIA provides native support only for the Franka Emika Panda manipulator. Hence, to achieve the objectives of this research, the environment's internal logic was adapted to support the Flexiv Rizon4s. This section details the technical adjustments to the environment configuration required to switch the utilized manipulator, enabling for future reproducibility.

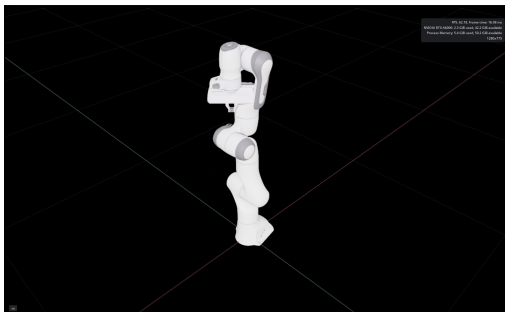
4.6.1 Robot USD Replacement

In NVIDIA IsaacLab, the physical and visual properties of the agents are defined via the Universal Scene Description File (USD), as detailed in Section 3.1.1. As a consequence, the environment's configuration was modified in order to replace the default Franka Emika Panda assets with the Flexiv Rizon4s.

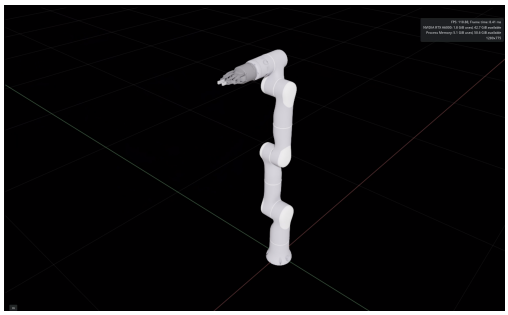
The official Rizon4s USD file, equipped with the GRAV gripper, was sourced directly from Flexiv github repository and is indeed utilized for this research. This asset defines the robot's URDF-derived kinematic properties, joint limits and visual representation.

As detailed in Section 4.4, the manipulator's initial configuration is determined by calculating an Inverse Kinematics (IK) solution from the fixed asset position. Such process enables to place the robot's gripper above the insertion peg. By

swapping the USD file of the manipulator, the IK solver automatically uses the Rizon4s specific 7-DOF kinematic equation rather then Panda's. This ensures that the environment reset logic executed at the start of each episode remains unchanged.



(a) Franka Emika Panda USD model.

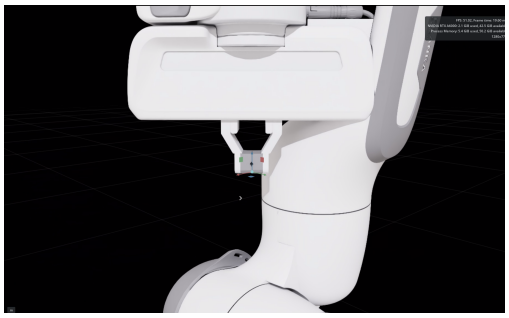


(b) Flexiv Rizon 4s USD model.

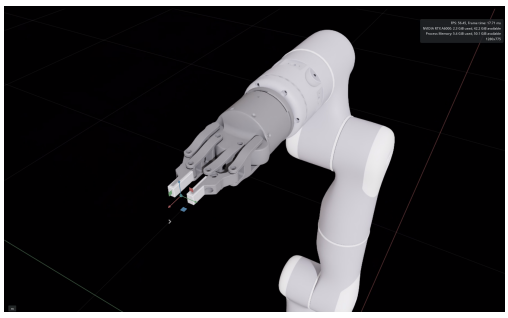
Figure 4.4: Rendered comparison of the robot assets within the Isaac Sim environment.

4.6.2 Tool Center Point (TCP) Frame Alignment

As described in Section 4.2.2, the environment makes an extensive usage of the pose of the TCP, in order to compute both the estimated force-torque observations F^{tcp} as well as the TCP pose p^{tcp}. To ensure consistency in these calculations, the coordinate frame hierarchy of the new asset had to match the expected interface of the FORGE environment.



(a) Franka Emika Panda: fingertip midpoint frame.



(b) Flexiv Rizon 4s: Custom midpoint frame.

Figure 4.5: Visual representation of the Tool Center Point (TCP) reference frames within the respective gripper geometries.

For this purpose, the Franka Emika Panda USD features a specific reference

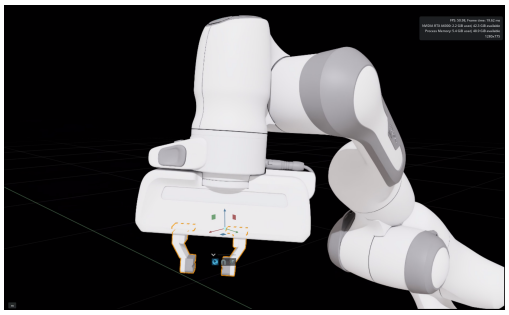
frame located at the geometric center of the gripper fingertips, as illustrated in Figure 4.5a. Reffered to as `fingertip_midpoint` serves as the reference frame for the TCP enabling for precise estimation of the gripper pose.

In the officially released by Flexiv, Rizon4s' USD file, an equivalent reference frame is missing. Hence, in order to preserve the control and observations logic, a custom USD Prim (Primitive) was inserted. Since the Flange reference frame is the only frame located exactly at the center of the gripper's fingers with respect to the X and Y axes, it was chosen as the starting base to create the new Prim. The newly inserted reference frame, named `fingertip_midpoint` for code compatibility, was thus established as a child of the robot's Flange frame

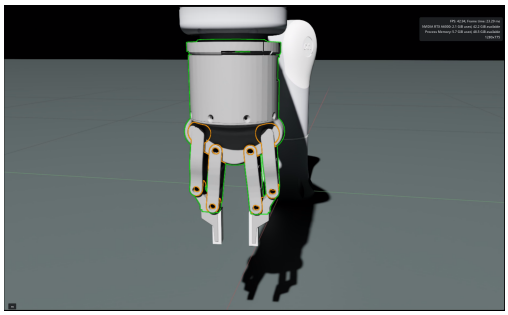
To account for the physical extension of the GRAV gripper, a translation of 0.19913 m was applied along the Flange's Z-axis. This modification ensures that the TCP pose is correctly represented and has been further reproduced in the deployment script.

4.6.3 Gripper Kinematics and Actuator Discrepancies

The main challenge encountered in adapting the FORGE environment for the Flexiv Rizon 4s, lies in the fundamental structural and kinematic differences between the end-effectors of the two manipulators. As exhibited by Figure 4.6a, the Franka Emika Panda features a parallel-jaw grpper actuated by two independent prismatic joints. Defining the translation of the fingers from the end-effector center point, these joints facilitate tight and secure grasping of the held asset.



(a) Franka Emika Panda: Dual prismatic gripper joints.



(b) Flexiv Rizon 4s: Complex revolute and mimic joint structure.

Figure 4.6: Comparison of the kinematic joint structures between the Franka Emika Panda and the Flexiv GRAV gripper.

The structure of the GRAV gripper integrated in Rizon4s USD file exhibits a significantly different kinematic chain, representing a challenging barrier for adaptation. It features a total of six joints of which only one, the `finger_joint`,

is actively actuated. The remaining five are *mimic joints*, unactuated joints whose positions are kinematically linked to a "leader" joint via a constant multiplier or offset.

These structural differences require for a strong adaptation of the environment configuration, in particular with respect to initialization logic of the environment, happening at the start of each episode.

While for the control logic it is sufficient to only actuate the `finger_joint`, the low-level interaction featured by Direct Workflows implemented by FORGE (see Section 3.1.2), makes the initialization of the mimic joints a challenging task. The initial state of every joint, mimics included, must be explicitly defined and physically consistent. Hence assigning non-feasible values where mimic positions contradict the leader joint's state leads to immediate physics collapse, making the simulation unable to proceed.

Discovering the right values is a manual time-consuming activity, which has to be conducted through trial and error. To ensure a stable and secure grasp of the gear at the start of each episode, the following state values were established for the GRAV gripper:

- **Actuated Joint:** `finger_joint`.
- **Mimic Joints (Positive Correlation):**
  - `left_inner_knuckle_joint`
  - `right_inner_knuckle_joint`
  - `left_outer_knuckle_joint`
  - `right_outer_knuckle_joint`

  These joints are set to match the state of the leader joint.
- **Mimic Joints (Negative Correlation):** `left_outer_finger_joint` and `right_outer_finger_joint` are set to the inverse (negative) value of the leader joint.

A further challenge in switching the manipulator, is represented by the different modalities of the joints of the grippers. While the Panda's joints are translational, the GRAV gripper features a revolute joint where the aperture is defined by an angular displacement (in radians) rather than a linear offset (in meters). Whereas the Panda's finger joints initialization can be simply derived by dividing by two the gear's radius, Flexiv Rizon4s required precise calibration to determine the exact angular value for a stable grip. Through iterative testing within NVIDIA IsaacSim simulation, a value of 0.122 radians was identified as the optimal setpoint for the gear meshing task.

Finally, the environment's actuator configuration was streamlined by removing the redundant second gripper actuator entry, reflecting the Rizon 4s' single-input actuation model.

4.7 Agent Configuration

The reinforcement Learning agent is trained using the **Proximal Policy Optimization (PPO)** algorithm. Being a model-free on-policy gradient method, it has been selected for its robust stability in continuous control tasks and its efficiency in avoiding destructive policy updates via its clipping mechanism. Such stability is vital for high-precision assembly, where slight oscillations in the control signal can prevent successful gear meshing.

The implementation utilizes the publicly available `rl_games` library, which provides a high-performance GPU-accelerated implementation suitable for the massive parallelism IsaacLab allows for. For this research, training was conducted using 512 **simultaneous environments**, ensuring that each policy update is informed by a significant and diverse batch of on-policy interaction data.

4.7.1 Network Architecture

The neural network structure derives from the two primary challenges the FORGE environment faces: The robust feature extraction for contact-rich environments as well as the facilitation of zero-shot Sim-to-Real transfer under spatial and dynamic uncertainty.

Asymmetric Asynchronous Actor-Critic In order to maximize learning efficiency while maintaining a deployable policy, an **Asymmetric Asynchronous Actor-Critic**(A3C) architecture is utilized. This paradigm involves feeding different inputs to the Actor and Critic networks during the training phase:

- **The Actor (Policy Network):** The Actor receives only the noisy, partial observation vector Ω (see Section 4.2.2). This restriction ensures that the trained policy generalizes directly to the physical robot without requiring ground-truth state data.
- **The Critic (Value Network)**: The Critic, instead, receives the full *privileged state* $\mathcal{S}$ available only within the simulation. This allows the Critic to learn a more accurate estimation of the Value Function $V(s)$ to guide the actor effectively. This reduces gradient variance and provides the Actor with a more stable "signal" for policy improvement.

Recurrent Policy (LSTM) Since the FORGE environment involves significant randomization and contact-rich interactions, a single frame of observation would be insufficient to estimate the system state. To address this challenge, both the Actor and Critic networks are augmented with a Long Short-Tern Memory (LSTM) layer. The LSTM cells enable to store temporal information, allowing the agent to encode the history of its previous observations and actions.

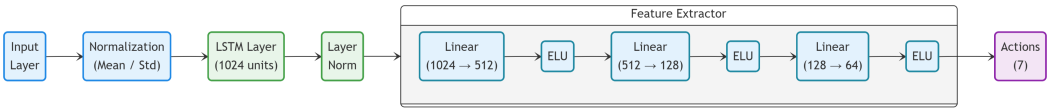


Figure 4.7: Schematic representation of the agent's neural architecture.

The data flow within the policy network, illustrated in Figure 4.7, follows a structured pipeline:

- **Input normalization:** The raw observation network is normalized to zero mean and unit variance.
- **LSTM layer:** The normalized vector is processed by an LSTM block with 1024 hidden units, which maintains the temporal context.
- **Feature Extractor:** The LSTM output is fed into a Multi-Layer Perceptron (MLP) consisting of three hidden layers of sizes $[512, 128, 64]$. These layers utilize Exponential Linear Unit (ELU) activation functions to facilitate efficient gradient flow.
- **Action Output:** The final 64 latent features are projected through a linear layer to output the 7-dimensional action space (6 control deltas and 1 success prediction).

4.7.2 Hyperparameters and Training Dynamics

The training process is governed by a specific set of hyperparameters. A critical metric utilized to monitor and regulate the optimization process is the **Kullback-Leibler (KL) Divergence**. In the context of Reinforcement Learning, the KL divergence acts as a statistical measure quantifying the difference between the policy distribution prior to an update ($\pi_{\theta_{old}}$) and the updated distribution (π_θ) (see Section 2.2.5). Mathematically, for the stochastic Gaussian policies learned by the agent, it is defined as the expected logarithmic difference between the old and new action probabilities:

$$D_{KL}(\pi_{\theta_{old}} \| \pi_\theta) = \mathbb{E}_{s_t, a_t \sim \pi_{\theta_{old}}} \left[\log \frac{\pi_{\theta_{old}}(a_t|s_t)}{\pi_\theta(a_t|s_t)} \right] \tag{4.16}$$

In order to maintain training stability, the learning configuration employs an adaptive learning rate scheduler. This mechanism dynamically modulates the optimizer's step size (α) based on the measured D_{KL}. If a policy updates results in a value exceeding the threshold of 0.008, then the scheduler reduces the learning rate for subsequent iterations.

The specific hyperparameter configuration utilized for the training runs is summarized in Table 4.2. Notable values include the high discount factor ($\gamma = 0.995$), which encourages the agent to prioritize long-term success, and the large number of parallel environments, which provides the statistical diversity necessary for robust convergence.

Table 4.2: Hyperparameter configuration for the PPO agent implementation in `rl_games`.

Hyperparameter	**Value**
Number of Parallel Environments	512
Horizon Length (T)	128
Mini-batch Size	16384 (512×32)
Mini-epochs (K)	4
Discount Factor (γ)	0.995
GAE Parameter (λ)	0.95
Initial Learning Rate (α)	1.0×10^{-4}
KL Threshold (D_{KL})	0.008
PPO Clip Range (ϵ)	0.2
Max Gradient Norm	1.0
Entropy Coefficient	0.0

Chapter 5

Validation Techniques and Metrics

This chapter provides a comprehensive overview of the validation techniques chosen to prove the efficacy of the trained agent. A rigorous assessment is fundamental to quantify the agent's proficiency in high-precision assembly and its capacity for generalization across different execution environments.

The validation pipeline assesses the agent's performance across multiple levels, starting from training performance to physical deployment. The following sections detail how the evaluation scenarios are constructed and the statistical metrics utilized to measure the trained agent's efficiency in completing the Gear Meshing task.

5.1 Training Metrics and Convergence Analysis

In order to measure the effectiveness of the training phase of a Reinforcement Learning (RL) agent, the primary indicator utilized is the reward signal. A progressive increase in the reward value across training iterations indicates an improvement in the policy's performance, thereby demonstrating that the agent is successfully learning to accomplish its assigned task. Therefore, in this research the primary indicators of a successful training procedure are the cumulative reward trajectories and the mean success rate.

Cumulative Reward Analysis To quantify the learning progress, the cumulative reward, calculated according to the weighted sum of alignment, force-thresholding, and regularization terms detailed in Section 4.2.3, is monitored across all training iterations.

Aggregate Success Rate While the reward provides a gradient for optimization, the most tangible indicator monitored is the success rate. This metric is defined as the moving average of correct task completion mediated over the 512 parallel training environments. Obtained once the held gear is fully inserted, high success rates in training are the prerequisite for progressing to Sim-to-Sim and Sim-to-Real deployment.

5.2 Deployment and Cross-Domain Validation

The deployment phase represents the core of this research, serving as a practical validation of whether Reinforcement Learning is a valuable methodology for high-precision contact-rich tasks as the Gear Meshing. In order to rigorously evaluate the capabilities of the trained agent, the deployment phase is structured in two consecutive phases. This structured approach ensures a gradual transition away from the controlled simulation environment toward real-world operating conditions.

In the details, the deployment procedure consists of:

1. **Sim-to-Sim deployment:** This intermediate step involves the re-creation of the task environment within an high-fidelity replica built in IsaacSim. This stage serves as a necessary benchmark to validate the policy's performance in a standalone inference environment, accounting for the latencies introduced by the ROS2 communication bridge without the added stochasticity of physical hardware.

2. **Sim-to-Real deployment:** The deployment on the Felxiv Rizon4s manipulator represents the final and hardest testing scenario. Transitioning from physics simulation within IsaacSim, it introduces uncertainty due to non-idealities of real-world measurements as well as it requires precise replication of the coordinate transformations applied in simulation.

For both of the deployment stages, a script for policy exporting has been developed. The utility represents one of the major contributions of this work, since it decouples the RL agent from the `rl-games` training environment, serializing it into a standalone format compatible with `PyTorch` for external inference. To ensure reproducibility, its main characteristics are detailed in subsequent sections.

Furthermore, the following sections detail the validation metrics utilized to quantify performance during the deployment phase, as well as the experimental configuration for both the virtual and physical deployment scenarios.

5.2.1 Policy Export Pipeline for RL-Games

A fundamental enabler for deploying an RL agent is the ability to decouple the learned policy from its training environment. This requires to export the policy model as a standalone serialization file (e.g., `.pt`) to be later integrated into external inference scripts via a selected library (e.g., `PyTorch`). While NVIDIA IsaacLab provides native support for exporting policies trained with other libraries (such as `rsl_rl`), a standardized pipeline for exporting trained models from the **rl-games** library was notably absent at the time of this research. Therefore, a dedicated export utility was developed. Being one of the main thesis contribution, it bridges the gap between high-throughput training in `rl-games` and industrial deployment.

The export system is implemented through a modular utility script, which is integrated into the evaluation workflow. The implementation addresses three primary requirements:

Neural Network Encapsulation: Since the trained agent uses Asymmetric Asynchronous Actor-Critic (A3C) with Recurrent Neural Networks (RNN), a wrapper class `ActorMLP` was implemented. This module isolates the Actor's Multi-Layer Perceptron (MLP) and the Gaussian mean output layer (μ) from the Critic network and the RNN hidden states. This ensures that the exported model represents a clean interface for inference, specifically targeting the transformation of latent features into actionable commands.

Multi-Format Serialization: The pipeline utilizes **TorchScript JIT** tracing and the **Open Neural Network Exchange (ONNX)** format. By providing a representative dummy input (e.g., a tensor of size 1×1024 corresponding to the latent feature dimension), the model's computational graph is traced and serialized.

Metadata Synchronization: To ensure the policy remains interpretable during deployment, the utility includes a configuration exporter. This generates a standardized YAML descriptor containing critical environment hyperparameters, such as observation and action space dimensions and the simulation time step (dt).

The export process is triggered during the execution of in-environment testing script. Once the `Runner` object restores the agent from a trained checkpoint, the export script is invoked. This captures the state of the model after initialization but prior to the simulation loop, guaranteeing that the exported weights precisely match the specific checkpoint being evaluated. This contribution provides a reproducible pathway for deploying IsaacLab agents trained with the **rl-games** framework into production-ready environments.

5.2.2 Deployment Script Architecture and Logic

The deployment script serves as the bridge between the Reinforcement Learning policy and the low-level industrial control hardware. It is responsible for orchestrating the real-time communication with the robot, sensor data processing, policy inference and the subsequent translation of actions into joint-space control trajectories. Shared by both the Sim-to-Sim and Sim-to-Real deployments, its correct implementation is fundamental since it must guarantee parity between the training environment and the physical world to mitigate the Sim-to-Real gap. Even marginal errors in the script's logic would result in the agent failing to complete the task in the simulated or real scenario, despite the performances obtained during training and in-environment testing.

System Architecture

The deployment script is composed of two main components:

- **Task Orchestrator (`run_assembly_task.py`):** This main script primarily handles data transformation of both in-bound and out-bound data streaming, hence it manages the ROS2 communication to store and transform the proprioceptive data coming from the robot. Furthermore it manages a critical sensor taring procedure, data logging/plotting logic and terminating condition.

- **Policy Controller (`assembly.py`):** This component represents the "brain" of the deployment procedure. By processing the data streamed by the Rizon4s robot, it builds the observation vector needed to execute policy inference. Moreover, to translate policy actions in meaningful control signals for the manipulator, it replicates the control logic detailed in Section 4.3 and adopted in the training phase.

Communication and Coordinate Transformation

Deploying a Reinforcement Learning policy requires abandoning the training environment provided by NVIDIA IsaacSim and IsaacLab frameworks. From a proprioceptive sensing perspective, this is translated in the sensor data no longer being generated within a simulator, but measured and streamed by the physical manipulator. In the field of robotics, the state-of-the-art framework for streaming sensor data and enabling inter-process communication is ROS2, whose main characteristics are described in Section 3.2.1.

To facilitate high-frequency interaction, the script implements a ROS2 node using the `rclpy` client library, subscribing to two main topics:

- `flexiv_arm/joint_states`: Through which the values of the joint variables of the robot are transmitted. This data is fundamental for computing the target joint trajectory to be applied after each policy inference step.
- `flexiv_robot_states`: This topic steams the information related to flange pose and the external contact forces acting on the manipulator's tool. Additional details regarding the structure of the broadcasted messages are provided in Appendix B.

A critical requirement for consistency is the alignment of the Tool Center Point (TCP) reference frame. As established in Section 4.6.2, the policy expects observations relative to a custom TCP frame located between the gripper fingertip. Hence, to ensure consistency with the training environment, the TCP pose is calculated from the Flange pose streamed by the HESPER controller, applying a translation of 0.1990 m along the local Z-axis to derive the actual TCP pose. Transformation is computed through the `pinocchio` library using the URDF of the robot that includes the custom TCP frame.

Dynamic Sensor Taring

Early deployment experiments revealed a non-negligible bias in the force sensor measurements for both the simulated and real environments. Specifically, the sensor measures a torque even though the tool is not in contact with any external element. This bias varies at each deployment experiment and significantly differs from simulation and reality, likely due to the methods exposed by IsaacSim in order to emulate the contact forces. The variability of this measurement errors required a dynamic taring procedure, executed at the onset of each trial. The manipulator remains stationary for the first 20 timesteps while the external wrench is recorded. The mean bias $\bar{F}_{bias}$ is computed and subsequently subtracted from all future observations:

$$F_{obs,t} = F_{raw,t} - \bar{F}_{bias} \tag{5.1}$$

This ensures that $F_{obs} \approx 0$ when the TCP is not in contact with the assembly environment, preventing the policy from reacting to phantom forces.

Policy Inference and Observation Encoding

The inference of the policy is the process of providing the Reinforcement Learning agent with the observation vector in order to compute the corresponding action commands. To ensure consistency with the training configuration, observations are supplied to the policy at a frequency of *15 Hz*.

The observation vector is built from the telemetry data streamed by the manipulator and it's composed of:

1. **Relative Position (p_{rel}):** The Cartesian offset between the TCP and the target assembly peg.

2. **TCP Orientation (q):** This represents the orientation quaternion of the TCP. To ensure consistency with training observations, all the components are inverted in sign while the first and last components are set to 0.

3. **Velocity Estimation ($\hat{v}^{tcp}$):** Represents the linear and angular velocities of the TCP. Despite being streamed by the manipulator itself and to maintain consistency with the simulator's computation, they are obtained via the the first derivative of the pose over the interval $\Delta t = 1/15\,\text{s}$.

4. **Force Observations ($\hat{F}^{tcp}_{smooth}$):** This represents the external forces measured at the TCP. In addition to the taring procedure and to faithfully replicate training logic, the actions are smoothed. As detailed in Section 4.4, at training time the Exponential Moving Average factor is set at 0.25. In the deployment script the same value is adopted to ensure consistency.

5. **Force threshold (τ_f):** During training the force threshold used to penalize the agent for excessive force exerted is randomly selected within a range of [5,10] (see Section 4.2.3). To grant a gentle behavior of the agent to avoid damaging the manipulated assets, during deployment the threshold is set to a constant 5.25 N, providing a conservative bound within the range seen during training.

Action Transformation

The raw actions calculated by the agent undergo a post-processing procedure to ensure uniformity with the control logic adopted during training and described in Section 4.3. In particular, the entire action vector, except for the last component corresponding to the success predictor, is smoothed according to Equation 4.12. While during training the EMA smoothing factor is randomly selected within the range [0.025, 0.1], in the deployment phase it is set to a constant value of 0.5. To constraint rotation about the Z-axis only, the rotational deltas along the x and y axes are set to zero, thereby replicating the rotation logic adopted during training.

To compute the target TCP positions to be transmitted to the manipulator, the deltas calculated by the policy, originally expressed in the reference frame of the insertion peg, are transformed into the word coordinate system. Subsequently, the computed targets are subject to the same scaling and clipping logic adopted in training, hence the position deltas are multiplied by a factor of $\sigma_{pos} = 0.05$ and then clipped to enable a maximum translation of $\pm 2cm$. The rotational deltas are instead clipped to allow for a maximum rotation of $\pm 5.56°$.

Finally, the last component of the action vector, the success predictor, undergoes a scaling procedure that is not applied during training. In fact, in order to extract a success probability within the [0,1] range, the raw policy output, originally bounded in [−1,1], is linearly rescaled.

Robot control logic and terminating condition

Once the policy inference is executed, the output is sent to the robotic manipulator in the form of joint trajectories to be followed by the robot. The transformed actions produced by the policy can be interpreted as a Cartesian twist that is later mapped to joint velocities through Damped Least Squares inverse kinematics:

$$\dot{\mathbf{q}} = J^T(JJ^T + \lambda^2 I)^{-1}\mathbf{v}^{cmd},$$

where J is the geometric Jacobian and λ is a damping coefficient set to 0.05. The geometric Jacobian of the manipulator is computed through the `pinocchio` library by importing the manipulator's URDF. Joint positions are updated via discrete-time integration:

$$\Delta\mathbf{q} = \dot{\mathbf{q}}\Delta t, \qquad \mathbf{q}_{k+1} = \mathbf{q}_k + \Delta\mathbf{q},$$

where $\Delta\mathbf{q}$ represents the incremental joint displacement applied at each control step. In order to grant a safe and gentle behavior of the robot, the $\Delta\mathbf{q}$ is set to 0.0035. The sequence of joint configurations implicitly defines the executed trajectory in joint space.

In order to control robot, the computed trajectories are streamed by a publisher ROS2 node through the `joint_trajectory` topic.

5.2.3 Deployment metrics

To evaluate the effectiveness of the deployment experiments, a set of dedicated performance metrics was collected. The deployment phase corresponds to the actual execution of Gear Meshing task; therefore the evaluation is intentionally decoupled from Reinforcement Learning specific indicators (e.g., reward signal), rather it focuses on task-level performance measures such as success achievement, exerted force and execution time.

In particular, for each experiment and for both simulated and real deployment, the following metrics were collected:

- **Success (S):** This metric is directly related to the success prediction encapsulated into the trained agent's actions. It is defined as a boolean value equal to 1 if the agent's success prediction overcomes the threshold established during

deployment. Hence, a positive value indicates that the *agent estimates that the task has been completed correctly.*

- **False Positive (FP):** This additional boolean serves to distinguish whether a Success value equal to 1 really corresponds to a correct task completion. Early deployment experiments highlighted the necessity of this indicator. It is set to 1 if the success prediction overcomes the threshold set, but the task has not been physically completed.
- **Path length (L):** Measured in centimeters, indicates the distance that the Tool Center Point has traveled during the deployment. Smaller values indicate that the manipulator followed a more efficient trajectory.
- **Maximum Force Exerted (F_{max}):** Measured in Newton, it represents the highest recorded value of exerted force during the deployment procedure. Given the gentleness required for high-precision insertion tasks like Gear Meshing, lower values are desirable.
- **Cumulative Exerted Force (F_{cum}):** Also measured in Newton, is calculated as the integral over time of the recorded forces. While the maximum exerted force may be recorded for a limited time (e.g due to the TCP getting stuck), the cumulative force, instead, records the overall gentleness of the agent throughout the entire deployment experiment.

5.2.4 Deployment Scenarios

To rigorously evaluate the deployment performance across multiple initial conditions, three distinct deployment scenarios (hereafter also referred to as *seeds*) had been developed. As extensively described in previous sections, the primary factor determining the task feasibility is the initial relative transformation between the TCP and the insertion peg . Specifically, the agent was trained to perform the Gear Meshing task within a workspace bounded by a ± 20 mm translation along the X and Y axes, with a vertical (Z-axis) approach offset ranging between 35 mm and 45 mm.

In order to prove the generalization capabilities of the trained policy, the deployment scenarios are designed to evaluate the performance at different offsets between the TCP and the insertion target position. For all scenarios, the vertical offset was maintained at a constant 42 mm. This value was intentionally selected for its high difficulty, as it resides near the maximum vertical limit encountered during training (45 mm), thereby testing the agent's ability to maintain precision at the extreme of its approach range.

The displacements in the two-dimensional plane for the three deployment seeds, instead, are defined as follows:

- **Scenario 0:** Initial offsets of 8 mm and 1 mm along the X and Y axes, respectively.. This represents the easier of the analyzed seeds, since the offsets rely in the core of the training distribution,

- **Scenario 1 (X-Axis Boundary):** Initial offsets of 19 mm on the X-axis and 1 mm on the Y-axis. The aim of this seed is to test the trained agent near to the maximum acceptable X-Axis displacement.

- **Scenario 2(Y-Axis Boundary/Multi-Axis):** Initial offsets of 8 mm on the X-axis and 19 mm on the Y-axis. Representing the hardest test case, it validates the agent to the maximum Y axis displacement, while simultaneously managing significat offset on the X-axis.

Figure 5.1 exhibits a graphical representation of the target insertion points for these three scenarios relative to the TCP position and the training boundary. Although the constant Z-axis displacement is omitted for visual clarity, it remains a consistent constraint across all trials.

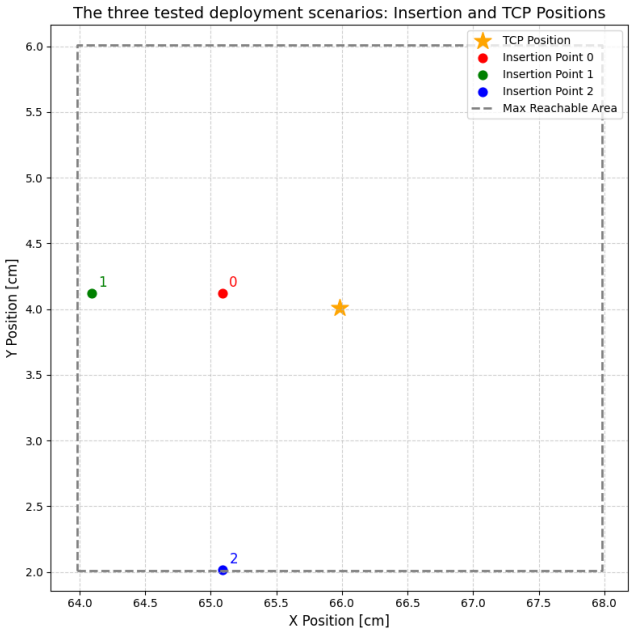


Figure 5.1: Spatial distribution of the three deployment test scenarios relative to the TCP initialization point and the maximum training boundaries.

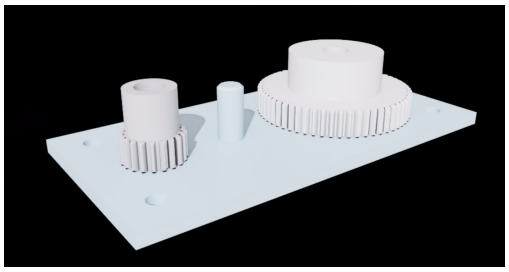
5.2.5 Simulation-to-Simulation Deployment

The simulation of the deployment procedure represents a fundamental step in a Reinforcement Learning pipeline. The purpose of this phase is twofold: First, it supplies a safe environment to develop the deployment script without endangering the robotic arm and the manipulated assets. Second, it acts as an intermediate benchmark to validate the policy's transferability when decoupled from the training framework's internal wrappers Hence, for the purpose of this research, an high-fidelity replica of the training environment has been created within the IsaacSim framework.

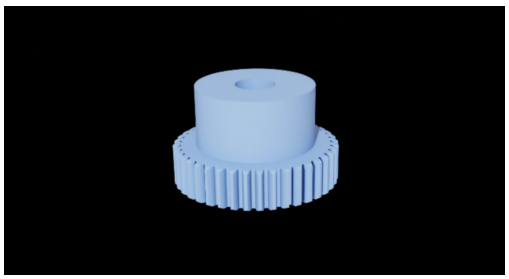
Unlike the training phase where IsaacLab orchestrates physics, policy updates and the manipulator control, the simulated deployment only handles the rigid-body physics and contact dynamics. In addition, a replica of the ROS2 communication of the real robot is simulated, thus ensuring complete compatibility with the deployment script described in Section 5.2.2.

Scenarios building

To create the three simulated deployment scenarios described in Section 5.2.4, a replica of the training scene has been created in IsaacSim. The scene has been manually composed using the exact same USD files employed during training, including the modified Flexiv Rizon 4s USD (Section 4.6). Figure 5.2 illustrates the high-fidelity renders of the fixed base with the two flanking gears and the medium sized gear to be inserted. These components are hereafter referred to as the *fixed asset* and the *held asset*, respectively.



(a) Fixed assets: Assembly base and flanking gears.



(b) Held asset: Medium sized gear to be inserted.

Figure 5.2: 3D renders of the task-relevant assets within the Isaac Sim environment.

For each of the created scenarios, the TCP is initialized at the constant position, in centimeters, of `(65.98, 4.01, 15.99)` in the word reference frame. In order to locate the gripper in the aforementioned coordinates, the 7 joint variables of the

Rizon4s are determined by inverse kinematics, resulting in the following joint configuration (in degrees): $q_{init} = [-27.46, -45.05, 52.05, 92.91, -39.24, 28.07, -150.73]$. Lastly, the revolute joint governing the GRAV gripper's aperture has been manually tuned to the value of 5.194° to ensure a tight grasp of the held gear.

Figure 5.3 represents the resulting manipulator configuration, within the first of the three deployment scenarios built in IsaacSim.

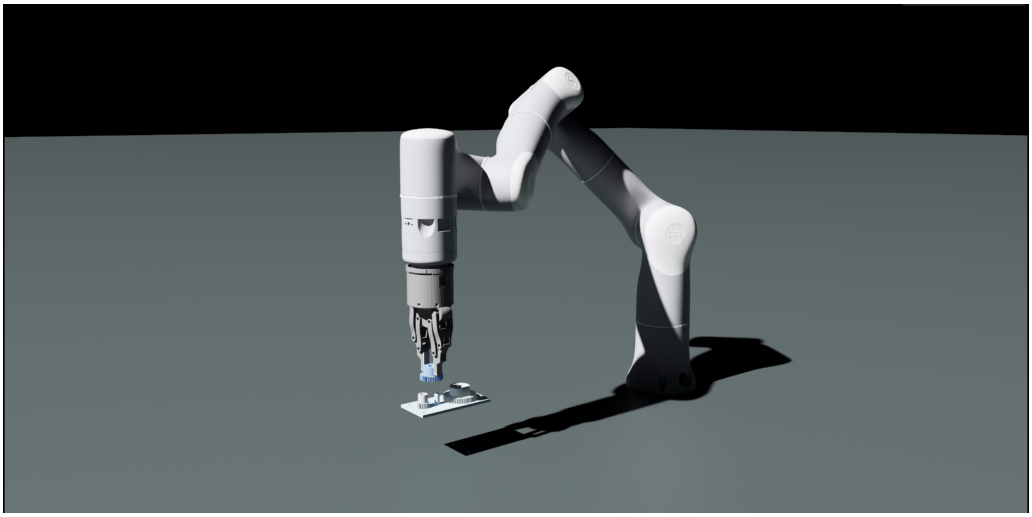


Figure 5.3: Simulated deployment environment showcasing the manipulator configuration at initialization.

In order to create the three of the deployment scenarios, three different USD files had been created. For each of the three the only difference relies on the position of the fixed asset. As exhibited in Figure 5.4, the variations in planar displacement provide a variegated set of initial conditions, testing the agent's ability to localize the insertion peg and manage contact forces under varying approach vectors.

Simulated Robot State Acquisition and Telemetry Streaming

In order to build the observation dictionary required by the trained agent, the deployment script relies on the proprioceptive information streamed by the robotic manipulator. The streaming of this information is natively handled by the Flexiv Rizon4s ROS2 driver and obtained by connecting via Ethernet to the robot. IsaacSim, on the contrary, requires a custom communication bridge.

This interface is implemented via a programmatically generated ROS2 Action Graph, which manages the high-frequency exchange of simulation states and control commands in the IsaacSim scene. The graph is constructed via a Python script executed within the framework's script editor which automates the instantiation

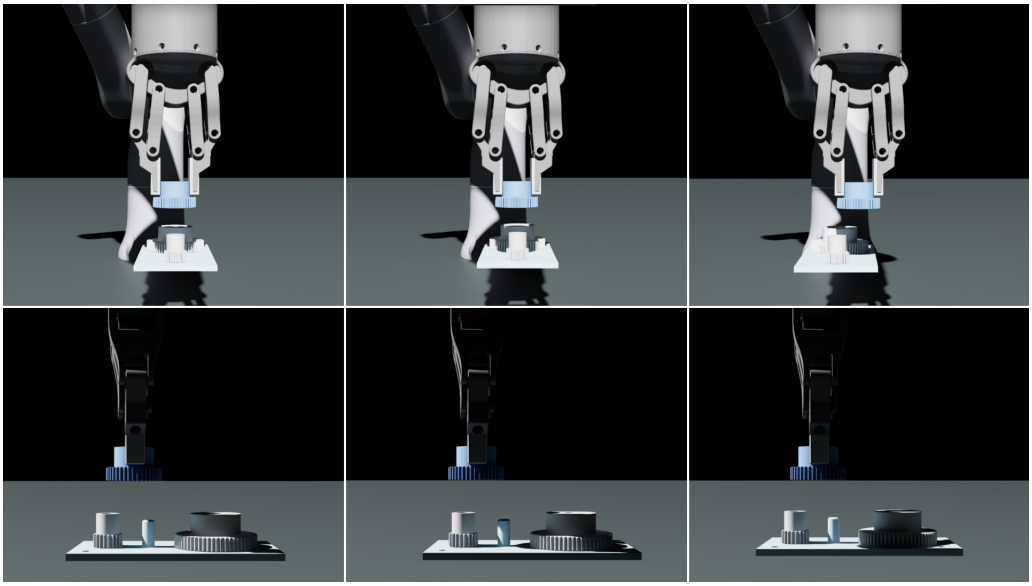


Figure 5.4: Visual comparison of the three simulated deployment scenarios, highlighting the varying initial offsets.

of the necessary communication nodes. In particular, the script implements the following core functionalities:

- **Action Graph Generation**: It creates an `omni.graph` structure that links simulation events (`OnPlaybackTick`) to the ROS2 bridge. Specifically, it instantiates `ROS2PublishJointState` and `ROS2PublishTransformTree` for outbound telemetry, while a `ROS2SubscribeJointState` node is configured to receive the target joint configurations $\mathbf{q}_{target}$ from the deployment script.
- **Physics-Based Wrench Extraction:** To replicate the Rizon4s force sensing capabilities, the script attaches the `PhysxArticulationForceSensorAPI` to the flange joint (specifically the *flange_to_gripper* prim). This allows for the extraction of raw wrench data directly from the physics solver.
- **Spatial Transformation and Publication:** Since the solver provides joint forces in the local articulation frame, a physics-step callback is implemented to perform a world-frame transformation. The resulting world-frame force and torque are published as a `geometry_msgs/WrenchStamped` message.

Middleware Adaptation and Force Sensing Discrepancies

To ensure seamless compatibility between the IsaacSim replica and the deployment pipeline utilized for the physical robot, an intermediate ROS2 adaptation

layer was developed. This middleware serves as a translation bridge between the generic simulation outputs and the manufacturer-specific communication protocols. The `SimAdapter` node aggregates the joint positions (q), velocities ($\dot{q}$), and the world-frame external wrench into the unified `flexiv_msgs/RobotStates` message described in Appendix B.

Furthermore, a significant discrepancy was identified regarding the force sensing interface. During training the `get_link_incoming_joint_force()` function offered by IsaacLab is used to obtain the applied force-torque measurements. IsaacSim, contrary, doesn't expone the same API, hence for the Sim-to-Sim deployment the `get_measured_joint_forces()` is utilized. Comparative analysis revealed that the forces reported by the latter exhibited magnitudes significantly larger than those observed during training or in real-world experimentation. To maintain the integrity of the observation space and ensure the policy remains within its learned distribution, a scaling layer was implemented. This adjustment normalizes the simulated wrench magnitudes to align with the physical characteristics of the real robot, thereby mitigating the risk of policy divergence during the gear meshing task.

5.2.6 Simulation-to-Reality Deployment

This section details the hardware configuration and procedural steps required for the physical deployment of the Reinforcement Learning policy. In particular, it delineates the robot connection procedure and the replication of the three deployment scenarios.

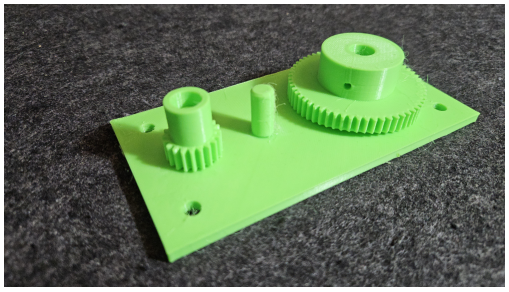
Hardware Interfacing and Networking

In order to establish the connection between the robot and the real-time Linux machine (see Section 3.2.3) running the deployment script and the Flexiv HESPER control unit, a private Ethernet connection is established. The connection is managed via the Flexiv ROS2 drivers. This setup facilitates bidirectional communication: the workstation receives high-frequency proprioceptive telemetry (joint states, Cartesian poses, and external wrench data) while publishing joint-space trajectory commands generated by the deployment script's inverse kinematics solver.

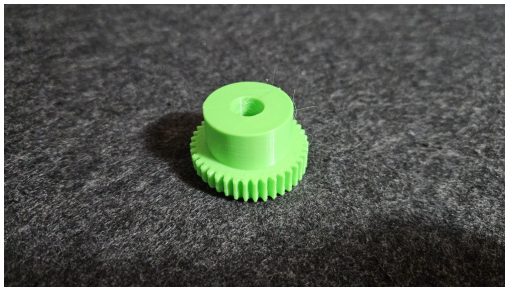
Assets 3D printing

In order to replicate the contact dynamics and clearances of the Gear Meshing task, the relevant components were fabricated via 3D printing. The geometries were derived directly from the USD files utilized during the simulation phase to ensure geometric parity.

These physical components, illustrated in Figure 5.5, maintain the stringent 0.5 mm diametrical clearance, presenting a significant challenge for the policy's force-guided alignment strategy.



(a) 3D-printed fixed asset: Insertion peg and flanking gears.



(b) 3D-printed held asset: Medium-scale gear.

Figure 5.5: Physical realization of the manipulated assets.

Grasp Execution and Kinematic Initialization

In contrast to the training and simulated deployment where the robot's initial state is defined by joint-space assignments in the USD file, the setup of the real robot starting pose requires a deterministic and safe procedural initialization.

Since the world coordinate frame in the training environment coincides with the base frame of the robotic manipulator, exploiting the robot's high-resolution proprioceptive encoders for precise measurements represents the best choice. In order to position the TCP to a specific value in the world coordinate frame derived from simulation (p^{tcp}), the manipulator was programmatically moved via the proprietary Flexiv Element Studio.

The software installed on the robot's teach pendant allows for direct control of the manipulator. Flexiv provides the capability to define robot projects, which consist of a set of customizable motion and control primitives executed sequentially.

Figure 5.6 exhibits a screenshot of the Flexiv Elements Studio interface featuring all the robot projects created to reproduce the three deployment scenarios. To set the manipulator to the initial pose, the `A-MoveToStart` and `A2-CloseGripper` projects are executed. The first one utilizes a point-to-point (PTP) motion primitive to command the 7-DOF arm, ensuring that the policy inference begins at the exact coordinates defined for the three different seeds. The second project, conversely, executes a force-controlled closing primitive of the GRAV gripper to grant a tight grasp of the held asset.

Figure 5.7 illustrates the real robot initial condition.

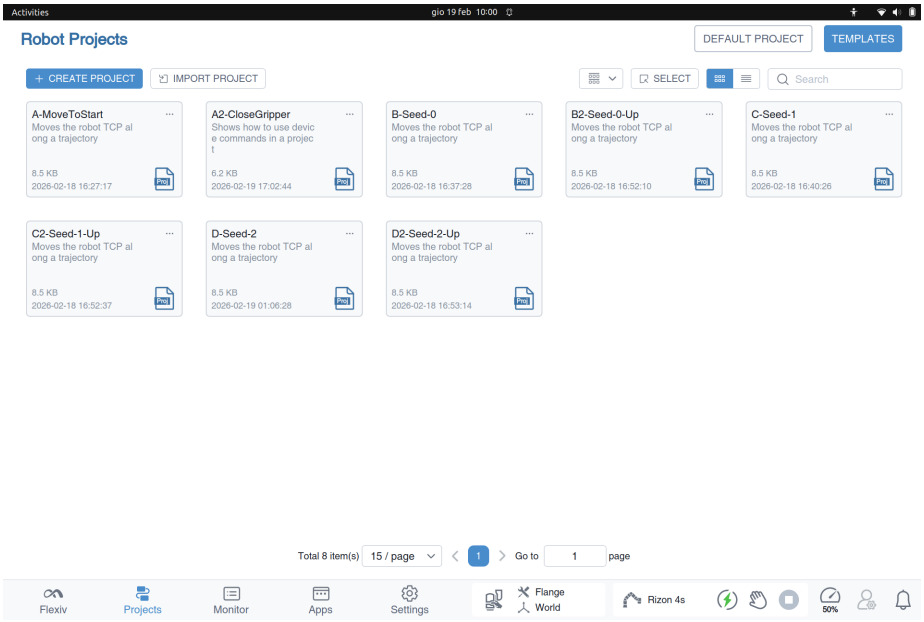


Figure 5.6: Interface of Flexiv Elements Studio showing the modular primitives utilized for standardized robot initialization.

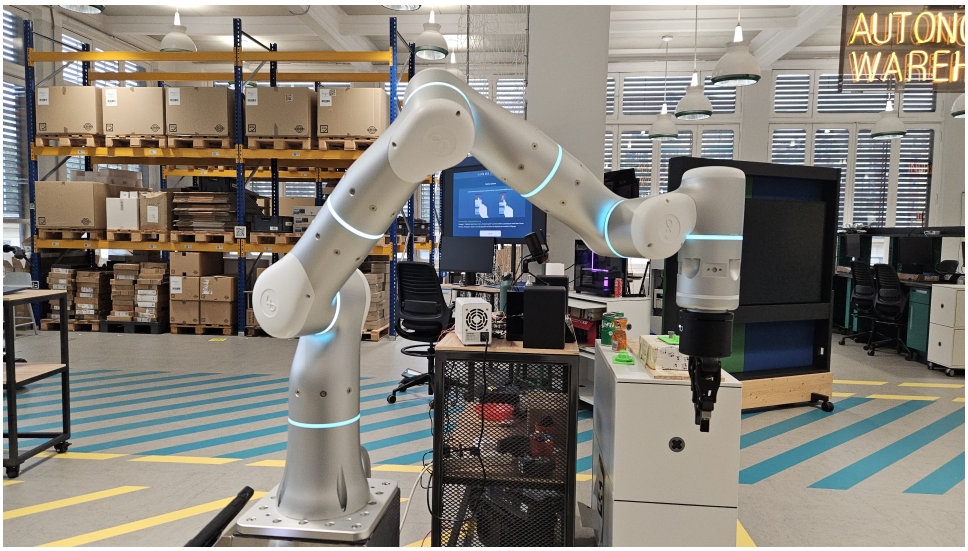


Figure 5.7: Physical manipulator configuration at the deployment starting point, mirroring the Sim-to-Sim initialization pose.

Calibration and Fixed Asset Positioning

Replicating the three deployment scenarios with high fidelity presents a significant challenge. The primary difficulty encountered relies in the accurate placement of

the fixed base in the real world, requiring sub-millimeter alignment. Since the entire deployment pipeline operates without external vision or depth sensors, the agent's ability to minimize the relative error p_{rel} depends entirely on the accuracy of the predefined target coordinates in the source code. While the agent is trained with stochastic observation noise to enhance robustness, the elimination of systematic offsets during physical setup is a prerequisite for successful zero-shot transfer.

To achieve this precision, the Rizon 4s was utilized as a high-precision measurement tool. By exploiting the robot's integrated encoders, the physical assets were collocated using a two-stage calibration procedure within Flexiv Elements Studio. As depicted in Figure 5.6, for every of the deployment seeds two different projects were created.

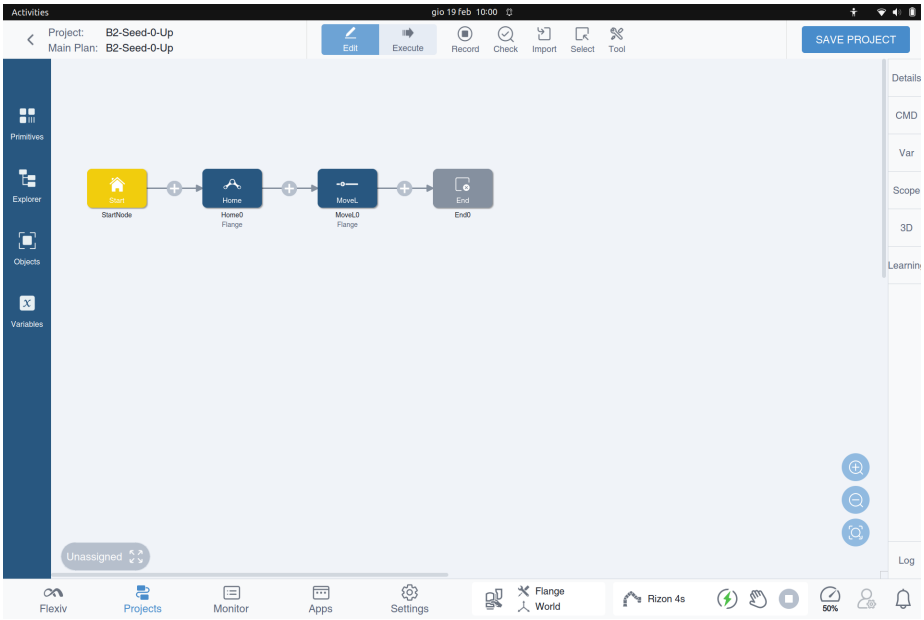


Figure 5.8: Structure of one of the calibration projects in Flexiv Elements Studio, utilizing the `MoveL` node for deterministic Cartesian positioning.

Figure 5.8 illustrates the structure of the projects, composed of four `Nodes`. While the first and last node mark, respectively, the start and the end of the project, the core is represented by the `MoveL` node. It implements a PTP motion primitive to command the manipulator. It is important to note that the `MoveL` primitive calculates trajectories relative to the robot's Flange frame. Therefore, the 19.913 cm offset representing the GRAV gripper extension was manually integrated into the Z-axis target values to ensure the resulting TCP pose matched the simulation parameters exactly.

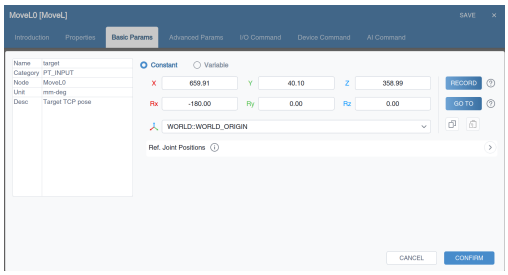
The calibration workflow for each deployment scenario follows a hierarchical

approach:

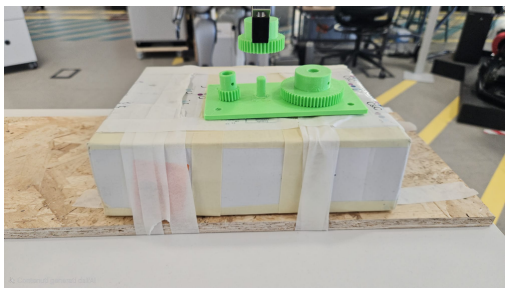
1. **Coarse Alignment:** A project is executed to position the held gear 40 mm directly above the theoretical insertion peg. This allows for the manual "rough" placement of the assembly base on the workspace table.

2. **Fine Alignment and Securing:** A second project slowly lowers the manipulator until the gear is perfectly meshed at the ground-truth goal coordinates. Once the physical alignment is verified, the assembly base is mechanically fixed to the workspace surface to prevent displacement during contact-rich interactions.

Figures 5.9 and 5.10 illustrate this procedure for seed number 0. Figure 5.9 depicts the initial approach height used for coarse positioning, while Figure 5.10 details the final insertion pose used to lock the fixed asset's coordinates in the world frame.

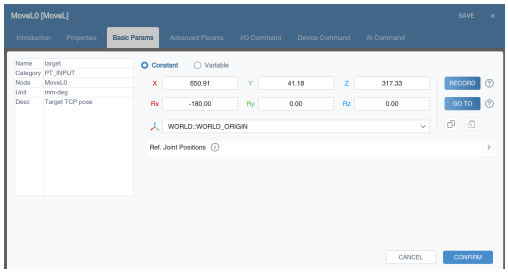


(a) Software target: 40 mm above.

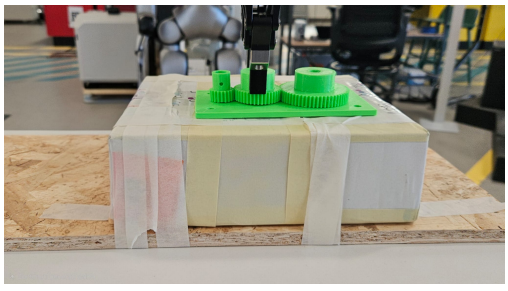


(b) Physical approach state.

Figure 5.9: Step 1 of the calibration procedure: Coarse spatial alignment above the insertion peg.



(a) Software target: Ground-truth goal.



(b) Physical goal state.

Figure 5.10: Step 2 of the calibration procedure: Validation of the insertion point before fixing the asset.

Overview of the deployment scenarios

By applying the aforementioned procedure, the three deployment scenarios described in Section 5.2.4 were reconstructed using the physical manipulator and 3D-printed assets.

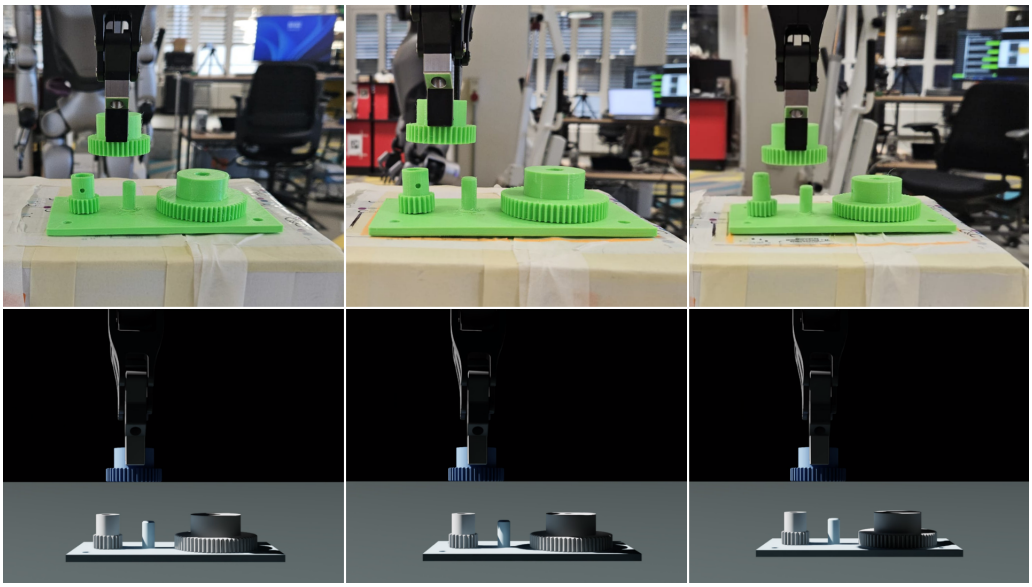


Figure 5.11: Visual comparison between the three simulated scenarios and their physical counterparts, highlighting the consistency of the experimental setup.

Figure 5.11 illustrates the visual comparison between the simulated and physical setups demonstrating a high degree of spatial fidelity. This high-fidelity replication ensures that any observed performance degradation in the Sim-to-Real phase can be attributed to dynamic non-idealities (friction, sensor noise, latency) rather than static positioning errors.

Chapter 6

Experimental Results

This chapter provides a comprehensive analysis of the experimental findings in performing the Gear Meshing task. The experimental protocols and observed metrics are established in Chapter 5, hence the focus of this section is to report and analyze the obtained results. The analysis encompasses the performance of the trained PPO agent across the training, simulated deployment and physical realization phases. This ensures to determine wether the training procedure has been correctly executed and then appropriately translated to the deployment pipeline.

The analysis of the deployment procedure, in particular, follows a two-fold approach:

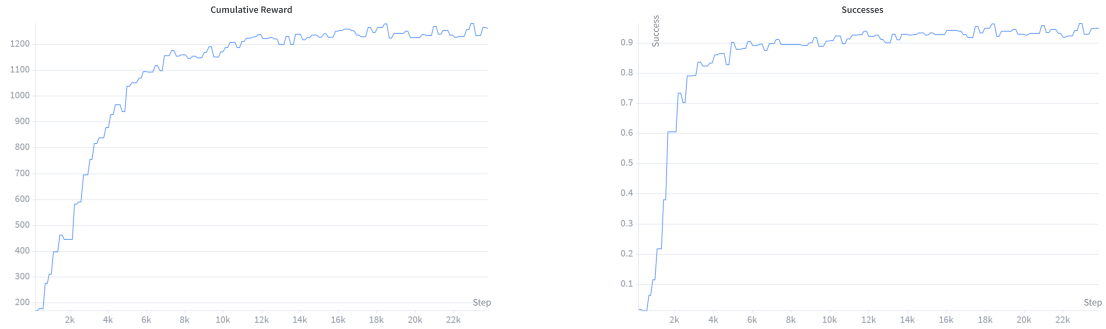
- **Quantitative Analysis:** A statistical comparison of the recorded metrics (e.g., success rates, force profiles, and path efficiency) across the three deployment scenarios.
- **Qualitative Analysis:** An investigation into failure modes and behaviors observed during contact-rich interactions, providing insights into the robustness of the learned policy.

6.1 Training Phase Convergence and Performance

A fundamental prerequisite for the zero-shot deployment this research aims to perform is that the training phase has been completed correctly. Given the complexity of the Sim-to-Real gap, it is essential that the policy achieves high performance and stability within the controlled training environment before transitioning to physical hardware.

As detailed in Section 5.1, the indicators utilized in this research to evaluate the efficiency of the training phase are the cumulative reward signal and the success

rate, both averaged over the 512 parallel agents.



(a) Cumulative reward per iteration.

(b) Success rate per iteration.

Figure 6.1: Training phase progression: Trends in reward accumulation and task completion.

Figure 6.1 illustrates the results of the training phase in terms of cumulative reward and successes obtained. The two depicted trajectories exhibit a similar trend, showcasing a rapid increment in the first third of the training duration followed by a stabilization phase where the policy achieves asymptotic convergence..

The cumulative reward reached a peak value of 1,281.05. While the absolute magnitude of the reward is an arbitrary term depending on the design of the reward function (see Section 4.2.3), the observed monotonic increase serves as a fundamental indicator that the agent is successfully optimizing the objective function. For the specific case of the Gear Meshing task, this translates to the agent learning to insert the gear while prioritizing alignment and minimizing the exerted force.

On the other hand, a far more concrete measure is represented by the the success rate, representing the percentage of the trained environment reaching the success state (i.e. the gear is inserted in the peg). Figure 6.1b exhibits not only a positive trend of the average success increasing over iterations, but most importantly shows that the maximum reached value is equal to 96.4%. This implies that, due to the significant randomization of initial poses and dynamic properties within the 512 environments, the agent developed a highly robust strategy for the gear meshing task. The small remaining error margin (3.6%) is typical for high-precision tasks under extreme randomization, often representing "edge cases" where the initial configuration is kinematically unfavorable.

In conclusion, the high success rate and stable reward convergence demonstrate the effectiveness of the training procedure, implying that the agent has reached a sufficient level of maturity to proceed to the next and far more complex phase of the deployment.

Table 6.1: Aggregate performance metrics for Sim-to-Sim deployment across 30 total trials.

Seed ID	Success (S) [%]	False Positive (FP) [%]	Path Length (L) [cm]	Maximum Force ($F_{\max}$) [N]	Cumulative Force (F_{cum}) [N]
0	100	20	6.45	30.39	39.21
1	100	10	6.40	28.44	44.28
2	100	100	6.22	42.56	38.89

6.2 Zero-shot Deployment

The zero-shot deployment represents the ultimate goal of this research: determining whether Reinforcement Learning policies, trained exclusively in a synthetic environment, can generalize to high-precision, contact-rich assembly tasks without fine-tuning. The details of the adopted validation techniques, in terms of collected metrics and evaluated scenarios, are illustrated in Section 5.2.

In order to ensure statistical significance of the observed results and to mitigate eventual outliers or "lucky" runs, each of the three deployment scenarios was evaluated over 10 different trials. Every trial was initialized from the exact same spatial coordinates and is performed until reaching the termination condition. Termination is defined by the agent's success predictor, rescaled to a $[0, 1]$ range, exceeding the confidence threshold of 0.99. This requirement implies that the deployment continues up until the agent is mathematically certain it has correctly performed the insertion task. A total of 60 independent runs were, thus, conducted.

It is important to highlight that the core of this research is the deployment on the physical manipulator, since it enables to determine whether Reinforcement Learning represents a valuable choice for real-world assembly task. Altough, experiments within the Sim-to-Sim scenario serve as a critical diagnostic layer, isolating software-level discrepancies from hardware-level non-idealities.

To ensure parity within the Sim-to-Sim and Sim-to-Real experiments the same deployment code is employed, with the only exceptions detailed in Section 5.2.

6.2.1 Simulation-to-Simulation Deployment

The Sim-to-Sim deployment represents the first step of the deployment validation. Success in this domain validates the decoupling of the agent from the training framework and confirms the fidelity of the ROS 2 middleware and control logic.

Quantitative Analysis Table 6.1 illustrates the aggregate results, averaged across 10 runs per scenario, of the conducted experiments.

The data highlights how, for each run and seed, the agent overcame the confidence threshold of 0.99. However, this perceived success did not always translate in successful insertion as evidenced by the False Positive (FP) rates, reaching, respectively, 20% and 10% for the first two scenarios. Most notably, Scenario 2 recorded a rate of 100%, meaning that not even once the agent effectively performed the task. As described in Section 5.2.4, this deployment seed was chosen to test the agent in an extremely hard scenario, involving simultaneous maximum Y and Z-axis displacement as well as half of the maximum X-axis offset.

To evaluate the efficiency of the learned movements, the observed path length (L) is compared against the theoretical Euclidean minimum (L_{min}) required for insertion:

$$L_{min} = \sqrt{\Delta x^2 + \Delta y^2 + \Delta z^2} \tag{6.1}$$

The calculated L_{min} values for Scenarios 0, 1, and 2 are approximately 4.26 cm, 4.58 cm, and 4.71 cm, respectively. In Scenario 0, the agent followed a trajectory roughly 51% longer than the optimal path (6.45 cm vs. 4.26 cm), which corresponds to the necessary search and alignment motions required to mesh the gear teeth. Notably, in Scenario 1 it followed a trajectory 39% longer than the optimal path. This result, together with the decreased False Positive Rate, represents a notable finding since for this seed the initial distance between the TCP and the target of insertion is increased.

Regarding the force profiles, the recorded maximum forces (F_{max}) reached values between 28 N and 43 N. This values represent the transient impulse forces occurring at the moment the held asset faces the insertion peg and the flanking gears, resulting in a "search" phase for correct alignment. The F_{cum} recorded values over all scenarios are likely due to the agent exerting sustained effort to overcome alignment friction.

Despite the 100% failing rate in the Scenario 2, the recorded metrics represent a promising result allowing to proceed further with the physical deployment.

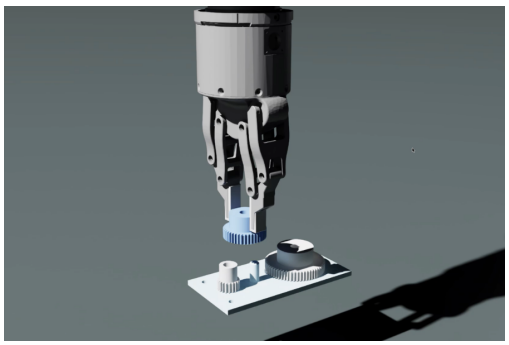
Qualitative Success Analysis and Contact Dynamics While the quantitative analysis provides a statistical foundation for evaluating the policy, a qualitative analysis of the agent's behavior is essential for characterizing the quality of the learned assembly strategy. This observation is particularly critical for the real-world assembly scenario, thus allowing to determine whether the trained agent's policy transfers to the physical scenario.

Figure 6.2 illustrates a representative successful execution within the simulated Scenario 0, divided into three fundamental phases: The initial condition, the first contact within the insertion peg and the held gear and, lastly, task completion.

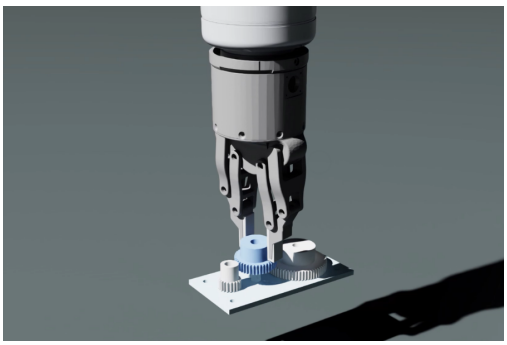
The quantitative analysis clearly exhibited that the manipulator is capable of correctly inserting the gear, although is important to observe the transition between the aforementioned phases, the first contact particularly.

As shown in Figure 6.2b , when the gear faces the peg, a critical phenomenon happens. The exerted contact forces cause the gear to physically slip within the GRAV gripper's fingertips. Despite this undesired behavior, agent's robustness still allows to compensate for this positional shift and complete the insertion. This comes from the reactive behavior of RL techniques, opening to a broad scenario of implementations robust to unmodeled dynamics and unforeseen disturbances.

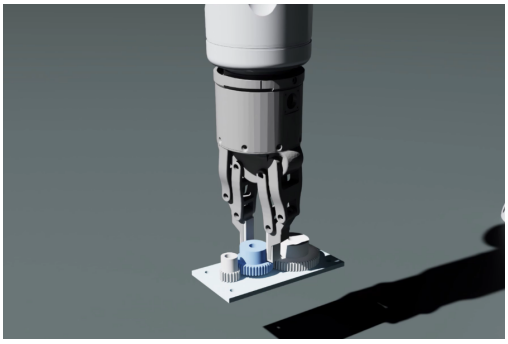
Although the agent still manages to complete the task, it is evident that this behavior is not desirable and constitutes the most probable failure cause for failing experiments. The root cause of this slipping is likely to address to the static nature of the grasp command. In the adopted pipeline, the gripper's closure is initiated at the beginning of the insertion experiment without an active force-closing loop to maintain the grasp under high external wrenches. This behavior presents a significant risk for the physical implementation. In the real-world applications such slipping could lead to the gear being ejected from the gripper or becoming permanently jammed at an unrecoverable angle.



(a) Phase 1: Initial approach.



(b) Phase 2: Contact-induced slippage.

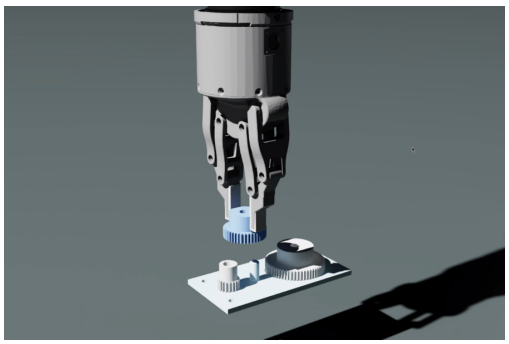


(c) Phase 3: Successful insertion.

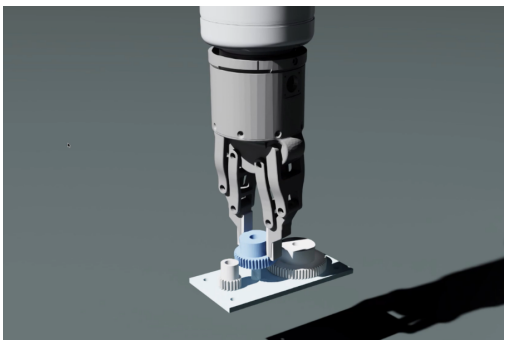
Figure 6.2: Example of a successful deployment episode in simulation.

Analysis of Notable Failure Modes Rather then the successful episodes, a rigorous analysis of the failure modes is essential for identifying the weaknesses of the trained agent as well as potential improvements.

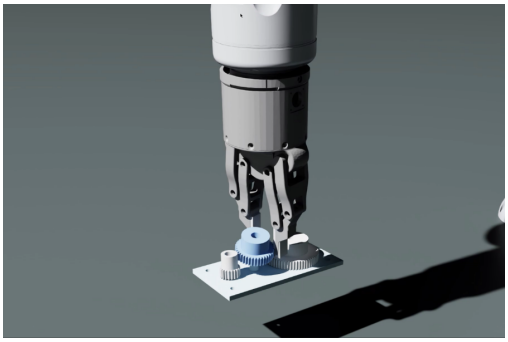
Figure 6.3 illustrates the three phases of a failing experiment conducted within the simulated scenario number 1. As already noted in the successful trials, and depicted in Figure 6.3b, the same slippery behavior is observed. However, in this analyzed example, the agent fails in adapting to this unforeseen disturbance, hence it does not succeed in inserting the gear. Therefore, as exhibited in Figure 6.3c, the held asset is ejected from the GRAV gripper's fingers before the insertion depth can be reached.



(a) Phase 1: Initial approach.



(b) Phase 2: Contact-induced slippage.



(c) Phase 3: Loss of grasp and task failure.

Figure 6.3: Example of a failed deployment episode in simulation.

Lastly, we report in Figure 6.4 an example of failing termination for the simulated seed number 2.

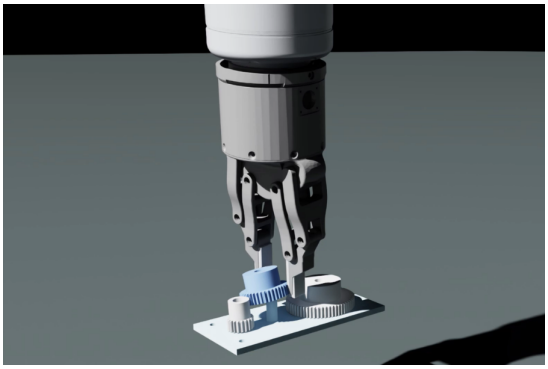


Figure 6.4: Scenario 2: Example of the failing mode

6.2.2 Simulation-to-Reality Deployment

The deployment on the Flexiv Rizon4s manipulator constitutes the ultimate objective of this research. It validates the agent's capability to operate within a non-deterministic environment characterized by sensor noise and complex frictional interactions.

Following the promising benchmark gathered in the Sim-to-Sim phase, the agent was subject to the same three deployment scenarios replicated in the real world.

Quantitative Analysis

The Sim-to-Real deployment performances are summarized in Table 6.2 and record a result significantly exceeding the expectations, particularly regarding the policy's robustness to boundary-case initializations.

The physical manipulator, indeed, achieved a 100% ground-truth success rate in Scenarios 0 and 1, hence indicating perfect zero-shot transfer for nominal and X-axis boundary tests.

Most notably in Scenario 2, which resulted in a 100% False Positive rate in the Sim-to-Sim validation phase, the physical robot successfully performed the task in 6 out of the 10 executed runs, highly overperforming the simulated deployment. As in the simulated experiments, for every run including the failures, the agent's success prediction overcame the threshold set, hence generating a 100% success rate.

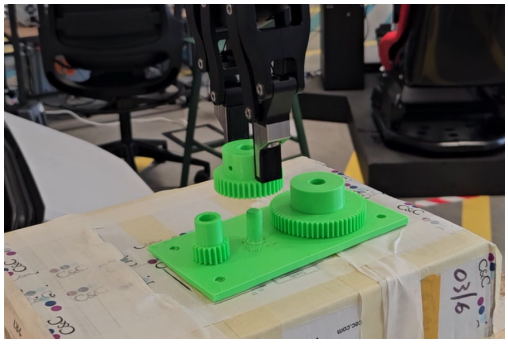
As observed in simulation, the followed trajectories exceed the theoretical minimum by, respectively, 58.68%, 55.6% and 37.57% registering similar performance when compared to the Sim-to-Sim benchmark. Interestingly, Scenario 2 recorded the lowest path overhead (37.57%), likely because the agent either quickly reached a successful state or entered a jammed condition (False Positive) early in the episode, thereby truncating the trajectory.

Seed ID	**Success** (S) [%]	**False Positive** (FP) [%]	**Path Length** (L) [**cm**]	**Maximum Force** ($F_{\max}$) [**N**]	**Cumulative Force** (F_{cum}) [**N**]
0	100	0	6.76	38.91	39.00
1	100	0	7.13	44.78	46.25
2	100	40	6.48	37.71	35.92

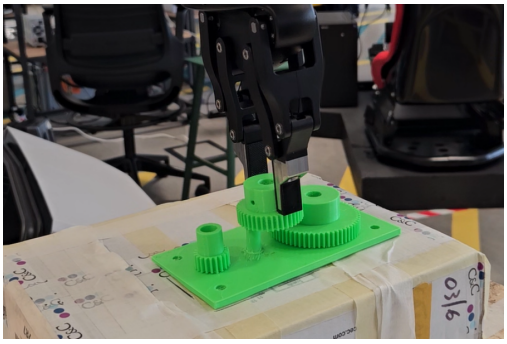
Table 6.2: Aggregate performance metrics for Sim-to-Real deployment across 30 physical trials on the Flexiv Rizon4s.

The recorded forces are consistent with simulated trials, with peaks registered between 37 N and 45 N. The F_{cum} values demonstrate that while the peaks are high, the agent maintains a generally gentle approach, with cumulative forces remaining stable across scenarios.

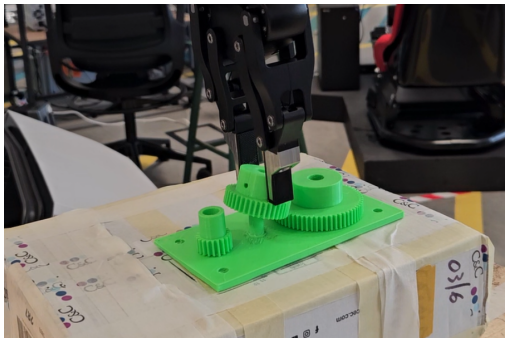
Qualitative Success Analysis and Contact Dynamics



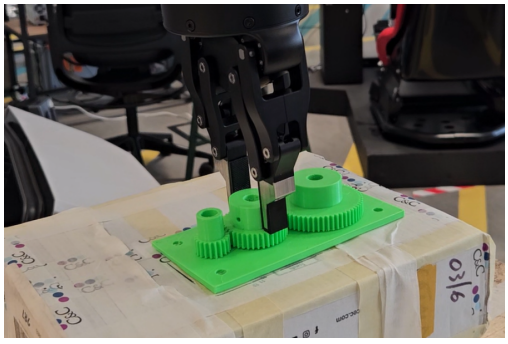
(a) Phase 1: Initial approach.



(b) Phase 2: Spatial misalignment.



(c) Phase 3: Force-guided recovery.

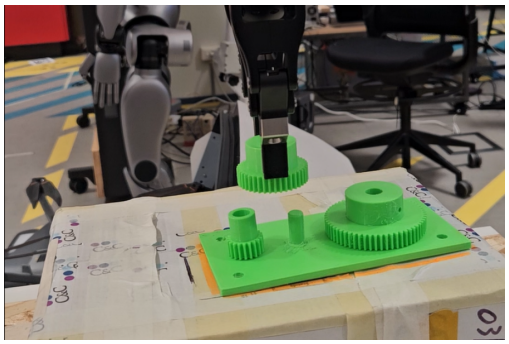


(d) Phase 4: Successful insertion.

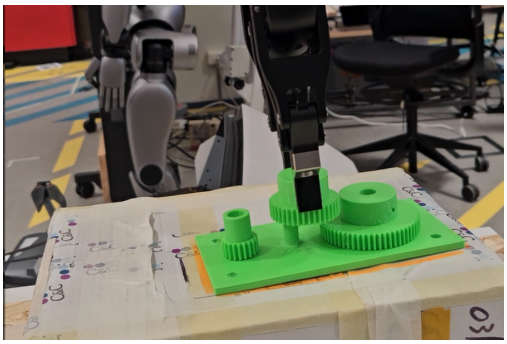
Figure 6.5: Example of a successful deployment episode on the real manipulator.

Qualitative observation confirms that the notable collected metrics are to be addressed to a high degree of robustness of the agent to unforeseen circumstances. The learned policy, indeed, excels in corrective behaviors emerged when facing spatial uncertainties that would cause standard scripted controllers to fail.

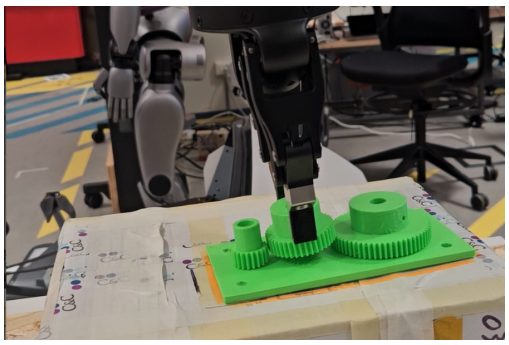
Figure 6.5 illustrates a successful deployment execution in Scenario 0. As evident from Figure 6.5b, the first approach of the robot to the insertion peg is characterized by an evident misalignment of the hole, likely due to measurement errors or the fixed base shifting because of the previous experiments. Rather than becoming stuck or triggering a safety stop, the agent utilizes the real-time force observations to recover from the incorrect placement, thus re-centering the held asset above the peg (see Figure 6.5c). Lastly, the insertion is successfully completed without even requiring teeth alignment (Figure 6.5d)



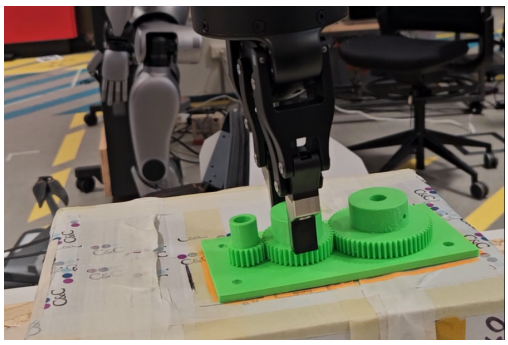
(a) Phase 1: Initial approach.



(b) Phase 2: Teeth misalignment.



(c) Phase 3: Teeth alignment.



(d) Phase 4: Successful insertion.

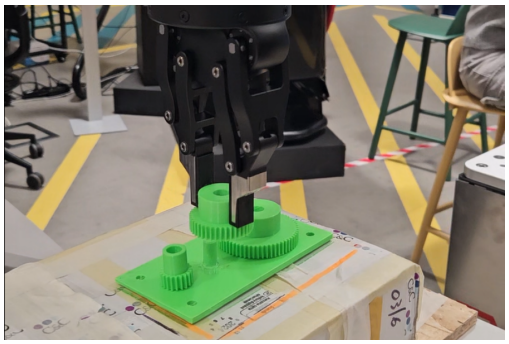
Figure 6.6: Success trial in Scenario 1 showcasing emergent rotational alignment strategies for meshing.

A second and more sophisticated adaptive behavior is shown in Figure 6.6, representing a successful insertion trial within the deployment seed number 1. As shown in Phase 3, once the agent successfully centers the gear hole above the

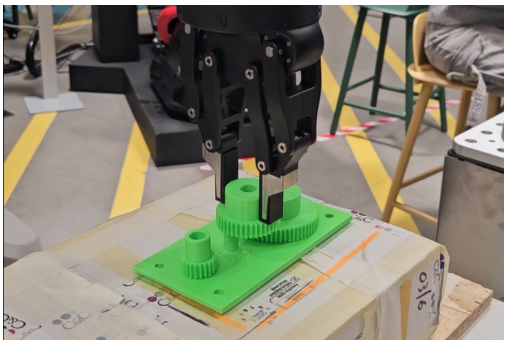
insertion peg, the complete insertion is hindered by mechanical resistance due to the angular misalignment of the gear teeth. To overcome this "dead-lock" state, the policy executes a corrective rotational maneuver around the Z-axis (Phase 4). This alignment allowed the agent to correctly complete the task, again showcasing high capabilities of adaptation.

Analysis of Notable Failure Modes

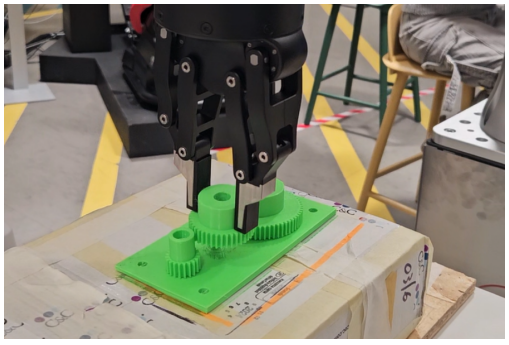
As this was the only test case where the agent failed to complete the Gear Meshing task, the failure modes analysis is restricted to the deployment Scenario 2. Analyzing these instances provides meaningful insight into the limitations of zero-shot transfer where the policy is pushed to the boundaries of its spatial training distribution. Figures 6.7 and 6.8 exhibit two different and relevant failure modes observed.



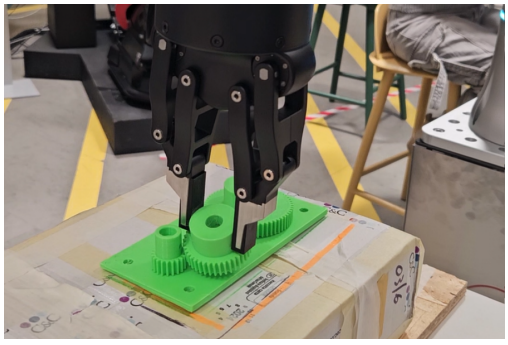
(a) Phase 1: Initial approach.



(b) Phase 2: Spatial misalignment.



(c) Phase 3: Sliding on the peg.

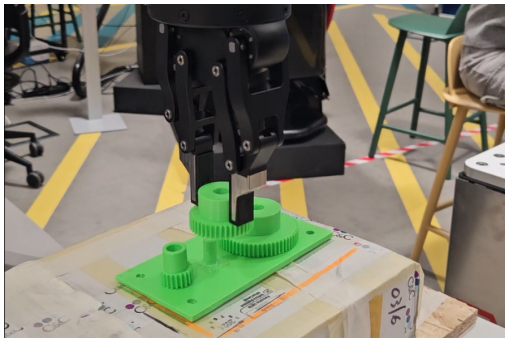


(d) Phase 4: Failure.

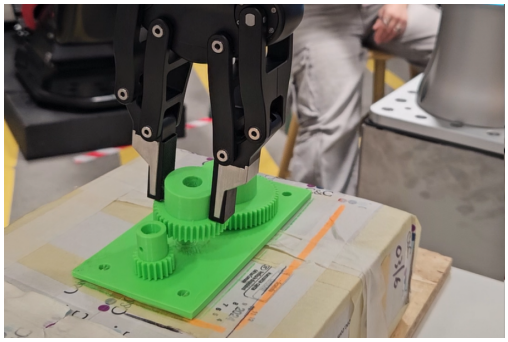
Figure 6.7: Failure mode in physical Scenario 2: Lateral sliding resulting in gear ejection from the workspace.

Phase 3 and 4 of Figure 6.7 exhibit a "slippery" behavior , this time different from what observed in simulation. During the failed physical trials, indeed, the gear remains securely held by the GRAV gripper, but lateral search maneuvers cause

the gear to slide off the rounded geometry of the insertion peg. This structural slippage results in the gear being deposited to the side of the assembly base (see Figure 6.8b, thus bringing the task execution into an unrecoverable state.



(a) Phase 1: Initial approach.



(b) Phase 2: Faliure.

Figure 6.8: Failure mode in physical Scenario 2: Gear jamming against flanking assets resulting in a False Positive termination.

Another failing method observed is depicted in Figure 6.8. This modality, contrarily, involves the held asset becoming physically stuck against the smaller of the two flanking gears. In this state the agent is unable to recover and complete the task. However, a "False positive" is triggered, hence the success prediction value overcomes the threshold.

6.2.3 Comparative Analysis

As clearly established by the gathered results, the Sim-to-Real trials, surprisingly and significantly, overcame the Sim-to-Sim experiments. This is a definitive finding of this research that deserves an analysis of its root cause.

Since in both of the deployment domains the same policy and deployment code are utilized, the reasons for this non-intuitive divergency must be fundamentally rooted in the modeling of the high-fidelity simulated replica of the deployment environment, rather than the ROS2 communication or policy export procedure..

The quantitative and qualitative evidence points to two primary drivers for this divergence:

- **High-Fidelity Force Correspondence:** As discussed in Section 5.2.5, early experiments of Sim-to-Sim deployments highlighted some discrepancies in the force measurements which required not only the taring procedure to be executed, but also a manual rescaling of the force magnitudes. The success in zero-shot transfer to the real world, although, suggests that the different API (offered by IsaacLab) used in the training procedure represents a more

accurate estimate of the real forces measured by the manipulator, contrarily to the API offered by IsaacSim and used for the Sim-to-Sim environment. The increased failure rate of the simulated trials, therefore, are caused by sensor-modeling discrepancies rather than a failure of the learned policy.

- **Grasp Stability and Physical Compliance:** The physical GRAV gripper provides a higher-impedance, more stable, kinematic constraint than is simulated counterpart. The simulator, indeed, often exhibited a "fingertip slippage", while the real-world grasp remained robust throughout all the conducted experiments. As previously analyzed, this was the root cause of some of the failures observed.

Despite the performance degradation observed in the Sim-to-Sim scenario, the results on the real manipulator are exceptionally compelling. Achieving 100% success in Scenarios 0 and 1, and 60% in the extreme boundary case (Scenario 2), demonstrates that Reinforcement Learning is capable of managing sub-millimeter tolerances (0.5 mm clearance) without the aid of external vision systems or depth cameras. By relying solely on proprioceptive and force-torque feedback, the agent has developed a generalized, tactile-driven strategy that is robust enough to handle the complex, contact-rich dynamics of industrial gear assembly.

6.3 Additional Analysis: Robustness to Different Geometries

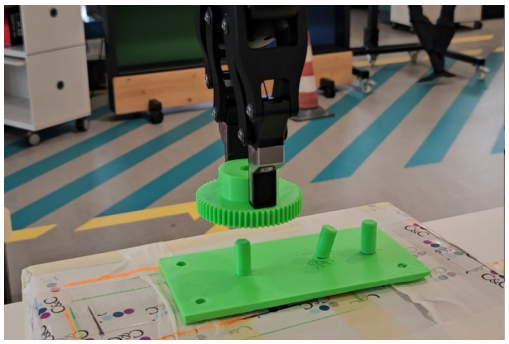
The primary Gear Meshing task addressed in this research is considered the most demanding phase of the assembly sequence. Indeed, inserting the middle gear while the two flanking gears are already in place represents a challenging objective, mostly, due to the simultaneous alignment of the central gear with two pre-positioned flanking gears. However, this additional analysis comes from the desire to push Reinforcement Learning adaptation capabilities to their limits.

This Section evaluates the learned agent to the completely unseen scenario of assembling the two flanking gears, exhibiting respectively a bigger and smaller radius. This test, conduceted only on the real manipulator, aims to determine if the agent has learned a universal "insertion and alignment" primitive rather than a geometry-specific trajectory.

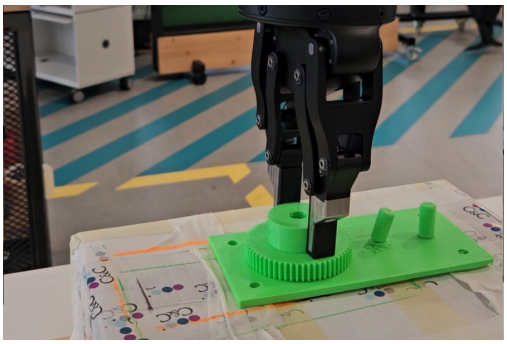
A complete industrial application would involve an automated end-to-end sequence (localization, grasping, and sequential assembly). However, Since the aim of this experiment is to analyze the robustness of the policy to different geometries, the gears were manually placed into the GRAV gripper and located to the exact same spatial conditions of the analyzed experiments (Seed 0) to perform the deployment procedure.

As illustrated in Figure 6.9, the agent's performance again significantly exceeded expectations.. Even though the geometries of the manipulated assets are different from the medium size gear it was trained to insert and despite the absence of the lateral constraints typically provided by flanking gears, both of the trials were completed successfully.

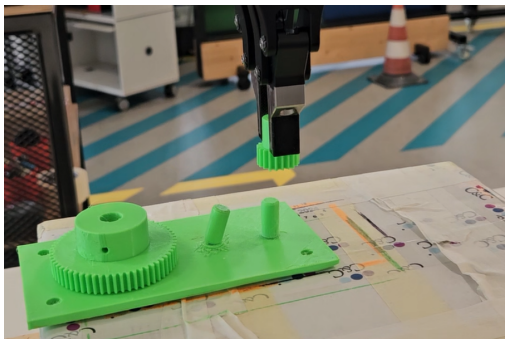
This finding is particularly significant because it demonstrates that the policy has internalized a robust force-guiding strategy. The agent does not rely on a fixed "memory" of the medium gear's dimensions; instead, it dynamically adapts its corrective movements based on real-time contact forces.



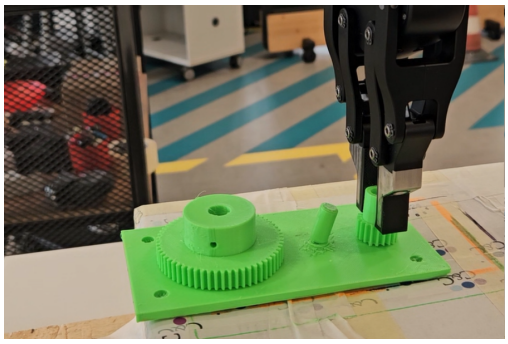
(a) Large gear: Initial approach.



(b) Large gear: Insertion.



(c) Small gear: Initial approach.



(d) Small gear:Insertion.

Figure 6.9: Empirical analysis of policy generalization across heterogeneous gear geometries.

Chapter 7

Conclusions

This research entailed an entire end-to-end Reinforcement Learning pipeline with a clear objective: evaluating the efficacy of autonomous agents in high-precision contact-rich industrial assembly. A field of application usually demanded to traditional robotic control. The project successfully bridged the gap between state-of-the-art simulation and physical realization, focusing on the challenging task of gear meshing with sub-millimeter clearances.

By leveraging the `NVIDIA Isaac Lab` framework, a robust environment was architected through the precise formulation of a Partially Observable Markov Decision Process (POMDP). The resulting policy achieved near-perfect convergence in training , thus serving as solid foundation to proceed to the next steps of the research.

In order to prove Reinforcement Learning agent's suitability for real-world applications, the work proceeded to decouple the agent from its learning framework, stepping towards the real manipulator deployment. For the purpose, the Flexiv Rizon4s manipulator was adopted, an industry leader robotic arm in contact-rich and force-sensing applications . In order to safely evaluate the results of transitioning from training to the real world, an intermediate Sim-to-Sim validation step was introduced. Meticulously developed to create an high-fidelity replica of the real scene, the Simulated deployment represented a safe developing and testing environment as well as a benchmark to decide whether or not to proceed with the subsequent step: the Sim-to-Real deployment.

To replicate the training scene, the deployment scenarios were faithfully reproduced in the real world, mirroring the initial manipulator configuration and 3D printing the manipulated assets. Each of the followed steps are detailed in this thesis, thus allowing for reproducibility of the observed findings.

The physical deployment far exceeded initial performance expectations. Through quantitative metrics and qualitative behavioral analysis, the agent demonstrated that RL is not only capable of completing high-precision tasks typically reserved for

classical control theory but also exhibits superior robustness to spatial uncertainties and mechanical non-idealities.

The encouraging results exhibited in this work open a broad scenario for Reinforcement Learning applications in the assembly domain. These findings pave the way for further research aimed at refining and extending RL approaches, particularly through the evaluation of agents in increasingly complex and realistic task scenarios.

7.1 Summary of Contributions

The **first** contribution of this research is that it represents the first documented application of a zero-shot RL policy for gear meshing task on the Flexiv Rizon4s manipulator. While academic literature is rich with experimental studies conducted on robots from different and more widely adopted vendors (i.e. Franka or Universal Robots), this work represents the first (at the time of writing) end-to-end research of a Reinforcement Learning policy zero-shot transfer on this robotic arm. As such, it establishes an initial performance benchmark for RL-driven assembly tasks using this adaptive manipulator, providing a reference point for future research and comparative evaluations.

Second contribution is represented by the first, at the time of writing, example of exporting policies trained within the `rl-games` library to real world applications. While the adopted framework is widely used in simulation, its decoupling from the training environment and export to a standardized PyTorch compatible format is not publicly available. This work of thesis not only proves feasibility, but details the steps undertaken to achieve this objective.

Third, this thesis provides a comprehensive overview of a complete pipeline for contact-rich high-precision Reinforcement Learning tasks. Detailing all the fundamental steps of training, simulated validation and transfer to the real world, this thesis encompasses the entire workflow. By detailing the technologies, methodologies, challenges and developed solutions, it enables for streamlined reproducibility of the conducted experiments.

7.2 Future Work

Although the trained agent has proven not only the capability to complete the assigned task in nominal conditions, but also an high level of adaptation and robustness to unforeseen conditions and disturbances, the work can, certainly, undergo an optimization and strengthening process.

7.2.1 Integration of Multi-Modal Perception

The current policy's reliance on purely proprioceptive data is a solid proof of its robustness. The target position is established before starting the deployment procedure, while the TCP pose and velocity as well as the contact forces are computed from the data collected by the proprioceptive sensors of the manipulator.

While representing, certainly, a core strength since it does not require any additional hardware overhead, the possibility exploiting external data would represent a clear benefit in this scenario. The difficulties introduced by the absence of vision or depth sensors were mitigated in the deployment scenarios' configuration by establishing a rigorous measuring procedure. However, integrating external sensors would remove the need for precise measurements performed before the deployment; instead the target position could be inferred in real-time. Moreover, external cameras would enable the agent to understand if it reached a dead-lock state, thus allowing for reset and recover.

7.2.2 Full Assembly Sequence Automation

This research focuses on the most challenging part of the assembly pipeline: The insertion of the middle gear while the two flanking gears are already in position. Despite this was the only scenario the agent has seen in the training phase, the conducted assembly experiments on the two remaining gears proved the policy robustness to unforeseen geometries. Such finding naturally suggests to implement the whole assembly pipeline as a state machine of pick-translate-assembly phases to be executed sequentially. This implementation, however, would benefit from external sensors integration as previously discussed.

7.2.3 Advanced Learning Paradigms

The adopted training environment already represents a robust choice for Zero-Shot real world transfer. The introduced randomization concerning both geometries, control parameters and physics properties proved the possibility of training an agent able to perform well under the uncertainties faced in real-world applications. However, especially if training aims to an even more complex task to be performed, more advanced learning techniques could be investigated. For example, Imitation Learning modalities like Behavior Cloning from human demonstrations could be well suited for this type of task.

In conclusion, the discussed extensions could further improve the already promising results observed in this research, thus realizing an even more mature and potentially industrial-ready application.

Appendix A

Software Environment

This section describes the software stack adopted for the training and deployment of the proposed system. Due to the significantly different requirements in terms of computational load, dependencies, and runtime constraints, two distinct environments were maintained: one dedicated to training, and one optimized for deployment to execute the deployment script. This separation ensured reproducibility, stability, and a streamlined deployment pipeline.

A.1 Training Environment

The training and simulation environment was designed to support large-scale reinforcement learning experiments, physics-based simulation, and rapid prototyping.

Languages and Package Managers

- Python 3.11.13
- Pip 25.2
- Conda 25.5.1

Deep Learning and Numerical Libraries

- PyTorch 2.7.0+cu128 (CUDA 12.8, cuDNN 9.7.1)
- NumPy 1.26.0
- SciPy 1.15.3
- scikit-learn 1.7.1

Simulation and Robotics

- Isaac Sim 5.0.0.0 (main modules: `isaacsim-app`, `isaacsim-core`, `isaacsim-rl`, `isaacsim-ros2`, etc.)
- Isaac Lab 2.2.1

Reinforcement Learning Frameworks

- RL-Games 1.6.1
- `skrl` 1.4.3
- Gym 0.23.1, Gymnasium 1.2.0

A.2 Deployment Environment

Languages and Package Managers

- Python 3.10.19
- Pip 25.2
- Conda 25.5.1

Deep Learning and Runtime Libraries

- PyTorch 2.9.1+cu128 (CUDA-enabled inference)
- NumPy 2.2.6

Robotics and Communication

- pinocchio 3.8.0
- ROS2 Jazzy (runtime and motion planning packages)
- Fleviv ROS2 Jazzy 1.8

Appendix B

Flexiv ROS2 messages

This appendix reports the structure and semantic meaning of the `RobotState.msg` ROS 2 message officially provided by *Flexiv Ltd.* as part of the Robotic Development Kit (RDK). The message is used to publish the complete runtime state of a Flexiv robotic manipulator, including kinematic, dynamic, thermal, and force-related information.

All joint-related quantities refer to the seven degrees of freedom of the Flexiv robot.

B.0.1 Message Fields Description

- **header** (`std_msgs/Header`) Standard ROS 2 message header containing timestamp and reference frame information.
- **robot_timestamp** (`builtin_interfaces/Time`) Timestamp generated by the robot controller, representing the internal system time at which the measurements were acquired.
- **q** (`float64[7]`) Measured joint positions obtained from the link-side encoders, expressed in radians.
- **theta** (`float64[7]`) Measured joint positions obtained from the motor-side encoders, expressed in radians.
- **dq** (`float64[7]`) Measured joint velocities obtained from the link-side encoders, expressed in radians per second.
- **dtheta** (`float64[7]`) Measured joint velocities obtained from the motor-side encoders, expressed in radians per second.
- **tau** (`float64[7]`) Measured joint torques, expressed in newton-meters.

- **tau_des** (`float64[7]`) Desired joint torques commanded by the control system.
- **tau_dot** (`float64[7]`) Numerical time derivative of the measured joint torques.
- **tau_ext** (`float64[7]`) Estimated external joint torques caused by interactions with the environment.
- **tau_interact** (`float64[7]`) Estimated interaction torques resulting from contact dynamics.
- **temperature** (`float64[7]`) Measured joint temperatures obtained from embedded thermal sensors.
- **tcp_pose** (`geometry_msgs/PoseStamped`) Measured pose of the Tool Center Point (TCP), expressed in the world reference frame.
- **tcp_vel** (`geometry_msgs/AccelStamped`) Measured TCP velocity expressed in the world reference frame.
- **flange_pose** (`geometry_msgs/PoseStamped`) Measured pose of the robot flange, expressed in the world reference frame.
- **ft_sensor_raw** (`geometry_msgs/WrenchStamped`) Raw force–torque sensor measurements expressed in the flange reference frame.
- **ext_wrench_in_tcp** (`geometry_msgs/WrenchStamped`) Estimated external wrench applied at the TCP, expressed in the TCP reference frame.
- **ext_wrench_in_world** (`geometry_msgs/WrenchStamped`) Estimated external wrench applied at the TCP, expressed in the world reference frame.
- **ext_wrench_in_tcp_raw** (`geometry_msgs/WrenchStamped`) Unfiltered external wrench applied at the TCP, expressed in the TCP reference frame.
- **ext_wrench_in_world_raw** (`geometry_msgs/WrenchStamped`) Unfiltered external wrench applied at the TCP, expressed in the world reference frame.

Appendix C

Reward Function Hyperparameters

This appendix provides the specific weights and coefficients used in the unified reward function. These values represent the configuration of the training environment and are essential for the reproducibility of the results discussed in Section 4.

Table C.1: Hyperparameters and weights for the unified reward function.

Category	Parameter	Value
Geometric	w_{base}	1.0
	w_{coarse}	1.0
	w_{fine}	1.0
	N	4
	$offset$	0.15
Regularization	w_{mag}	0.0
	w_{grad}	0.1
	w_r	0.001
	τ_{pos}	0.002
	τ_{rot}	0.097
Prediction	w_p	1.0
Safety	w_f	0.05
Sparse	w_e	1.0
	w_s	1.0

Table C.2: Squashing function coefficients for multi-resolution alignment.

Resolution	**Scale** (a)	**Power** (b)
Baseline	$a_0 = 5$	$b_0 = 4$
Coarse	$a_1 = 50$	$b_1 = 2$
Fine	$a_2 = 100$	$b_2 = 0$

Bibliography

[1] Bruno Siciliano, Lorenzo Sciavicco, Luigi Villani, and Giuseppe Oriolo. *Robotics: modelling, planning and control*. Springer, 2009 (cit. on p. 1).

[2] Schulman et al. «Proximal policy optimization algorithms». In: *arXiv preprint arXiv:1707.06347* (2017) (cit. on pp. 2, 13, 17).

[3] Zhehua Zhou, Jiayang Song, Xuan Xie, Zhan Shu, Lei Ma, Dikai Liu, Jianxiong Yin, and Simon See. «Towards building AI-CPS with NVIDIA Isaac Sim: An industrial benchmark and case study for robotics manipulation». In: *Proceedings of the 46th international conference on software engineering: software engineering in practice*. 2024, pp. 263–274 (cit. on pp. 2, 19, 20).

[4] Mayank Mittal et al. «Isaac lab: A gpu-accelerated simulation framework for multi-modal robot learning». In: *arXiv preprint arXiv:2511.04831* (2025) (cit. on pp. 2, 21).

[5] Flexiv Ltd. *Rizon 7-Axis Robotic Arm | Flexible Motion Solutions for Industrial Automation*. `https://www.flexiv.com/products/rizon`. Accessed: 2026-02-08. 2026 (cit. on p. 2).

[6] Michael Noseworthy et al. «Forge: Force-guided exploration for robust contact-rich manipulation under uncertainty». In: *IEEE Robotics and Automation Letters* (2025) (cit. on pp. 2, 21, 30, 31).

[7] Bingjie Tang, Michael A Lin, Iretiayo Akinola, Ankur Handa, Gaurav S Sukhatme, Fabio Ramos, Dieter Fox, and Yashraj Narang. «Industreal: Transferring contact-rich assembly tasks from simulation to reality». In: *arXiv preprint arXiv:2305.17110* (2023) (cit. on pp. 2, 30).

[8] 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)*. 2017, pp. 23–30 (cit. on pp. 2, 16).

[9] 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 (cit. on p. 2).

[10] International Organization for Standardization. *Robotics — Vocabulary*. Tech. rep. ISO 8373:2021. Geneva, CH: International Organization for Standardization, 2021. URL: https://www.iso.org/standard/75539.html (cit. on p. 5).

[11] KUKA. *Automation in the Automotive Industry*. https://www.kuka.com/en-de/industries/automotive. Accessed: 3 March 2026. 2026 (cit. on p. 6).

[12] Daniel E Whitney. «Quasi-static assembly of compliantly supported rigid parts». In: (1982) (cit. on p. 6).

[13] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018 (cit. on p. 7).

[14] 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 (cit. on p. 7).

[15] Richard Bellman. *Dynamic Programming*. Princeton University Press, 1957 (cit. on p. 10).

[16] OpenAI. *Part 2: Kinds of Reinforcement Learning Algorithms*. Accessed: 3 March 2026. 2018. URL: https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html#citations-below (cit. on p. 12).

[17] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. «Policy gradient methods for reinforcement learning with function approximation». In: *Advances in neural information processing systems* 12 (1999) (cit. on p. 13).

[18] 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 (cit. on p. 14).

[19] Wenshuai Zhao, Jorge Peña Queralta, and Tomi Westerlund. «Sim-to-real transfer in deep reinforcement learning for robotics: a survey». In: *IEEE Access* 8 (2020), pp. 113712–113736 (cit. on p. 15).

[20] Jiayuan Gu, Fanbo Xiang, Xuanlin Li, Zhanjia Ling, Liu Liu, Tongzhou Mu, Li Yi, and Hao Su. «ManiSkill2: A Unified Benchmark for Generalizable Manipulation Skills». In: *International Conference on Learning Representations (ICLR)*. 2023 (cit. on p. 17).

[21] NVIDIA Corporation. *NVIDIA Isaac Lab — Robot Learning Framework*. https://developer.nvidia.com/isaac/lab. Accessed: 2026-02-08. 2026 (cit. on pp. 21, 22).

[22] Flexiv Ltd. *ROS 2 Bridge — Robotic Development Kit Documentation*. 2026. URL: `https://www.flexiv.com/software/rdk/manual/ros2_bridge.html` (visited on 02/09/2026) (cit. on p. 24).