

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Teaching Robot Manipulation Tasks Using Extended Reality

Guilherme Miguel Monteiro da Costa



Mestrado em Engenharia Eletrotécnica e de Computadores

Supervisor: Prof. Marcelo Petry

June 30, 2026

Resumo

A indústria tem passado por várias transformações tecnológicas, desde a mecanização inicial até à digitalização associada à Indústria 4.0. Mais recentemente, a Indústria 5.0 tem colocado maior ênfase na colaboração entre pessoas e sistemas inteligentes, procurando desenvolver soluções industriais mais centradas no ser humano, adaptáveis e acessíveis.

Apesar desta evolução, o ensino de tarefas de manipulação robótica continua a ser um processo pouco flexível e dependente de conhecimento especializado. Muitos métodos de programação ainda exigem o uso de teach pendants, programação manual ou competências técnicas que nem sempre estão disponíveis no contexto industrial. Além disso, tarefas ensinadas de forma convencional tendem a adaptar-se mal a alterações no ambiente, como mudanças na posição dos objetos ou pequenas variações nas condições de execução. Para responder a estes problemas, a programação de manipuladores robóticos deve tornar-se mais intuitiva, reutilizável e adaptável.

Esta dissertação propõe uma metodologia para ensinar tarefas de manipulação robótica com recurso a realidade estendida e a ambientes de simulação. A abordagem permite recolher um pequeno número de demonstrações humanas, estruturar essas demonstrações em subtarefas, gerar novos exemplos em simulação e treinar políticas de aprendizagem por imitação para execução autónoma. A utilização de uma interface baseada em realidade estendida permite ainda captar informação espacial relevante, como o movimento das mãos do operador, e torna o processo de ensino mais acessível a utilizadores sem formação avançada em robótica.

O processo de trabalho proposto demonstra a viabilidade de combinar a recolha de um pequeno número de demonstrações através de realidade estendida, a geração artificial de dados para treino baseada em simulação e a aprendizagem por imitação numa única aplicação. Partindo de 10 demonstrações humanas por tarefa, recolhidas através de teleoperação em realidade estendida, o sistema gera 1000 demonstrações sintéticas por tarefa utilizando o Isaac Lab Mimic, produzindo um dataset com diversidade suficiente para treinar políticas de "behaviour cloning". A framework foi avaliada em quatro tarefas de pick-and-place em simulação, alcançando uma taxa de sucesso global de 93%. O sistema resultante suporta decomposição de tarefas modular e execução autónoma de políticas num ambiente de gémeo virtual, identificando também limitações atuais e direções futuras relacionadas com a robustez da orquestração.

Abstract

Industry has undergone several technological transformations, from early mechanization to the digitalization associated with Industry 4.0. More recently, Industry 5.0 has placed stronger emphasis on collaboration between people and intelligent systems, aiming to develop industrial solutions that are more human-centered, adaptable, and accessible.

Despite this evolution, teaching robotic manipulation tasks remains a relatively rigid process that often depends on specialized knowledge. Many programming methods still require teach pendants, manual programming, or technical skills that may not be available in industrial settings. In addition, conventionally taught tasks often adapt poorly to environmental changes, such as variations in object position or small changes in execution conditions. To address these limitations, robot programming must become more intuitive, reusable, and adaptable.

This dissertation proposes a methodology for teaching robotic manipulation tasks using augmented reality and simulation environments. The approach collects human demonstrations, structures them into subtasks, generates additional examples in simulation, and trains imitation-learning policies for autonomous execution. The use of an extended-reality-based interface also makes it possible to capture relevant spatial information, such as the operator’s hand movement, and supports a teaching process that is more accessible to users without advanced robotics expertise.

The proposed workflow demonstrates the feasibility of combining immersive demonstration collection, simulation-based data augmentation, and imitation learning within a unified robot teaching pipeline. Starting from 10 human demonstrations per task collected through extended reality teleoperation, the system generates 1000 synthetic demonstrations per task using Isaac Lab Mimic, producing a sufficiently diverse dataset for training behaviour cloning policies. The framework was evaluated on four pick-and-place tasks in simulation, achieving a global success rate of 93%. The resulting system supports modular task decomposition and autonomous policy execution in a digital twin environment, while also identifying current limitations and future directions related to orchestration robustness.

UN Sustainable Development Goals

The United Nations Sustainable Development Goals (SDGs) provide a globally recognised framework to guide actions towards a fair, resilient, and sustainable future for all. They comprise 17 goals that address major challenges such as poverty, inequality, climate change, environmental degradation, and the promotion of peace and justice. Within this framework, the present dissertation is particularly aligned with SDG 7 (Affordable and Clean Energy) and SDG 8 (Decent Work and Economic Growth).

This work proposes an Extended-Reality-based, Digital-Twin-driven framework for programming and executing robot tasks, with a focus on improving flexibility, safety, and usability in industrial environments. By enabling intuitive task specification through Augmented Reality and leveraging digital replicas of the human-robot cell, the system supports more efficient deployment and reconfiguration of automation. This contributes both to a more reliable and sustainable use of energy-intensive robotic infrastructure (SDG 7) and to safer, higher-quality working conditions and productivity for operators (SDG 8).

The specific Sustainable Development Goals considered in this dissertation are:

SDG 7 Ensure access to affordable, reliable, sustainable and modern energy for all.

SDG 8 Promote sustained, inclusive and sustainable economic growth, full and productive employment and decent work for all.

Table 1 summarises how the proposed system relates to selected targets within these goals and indicates example performance indicators that can be used to assess its contribution.

SDG	Target	Contribution	Performance Indicators and Metrics
7	7.1	The proposed architecture facilitates more reliable operation of robotised production cells by supporting intuitive configuration through Extended Reality and monitoring through the Digital Twin, indirectly contributing to access to automation resources.	Percentage of Human-Robot Collaboration cells operated with the proposed framework; reduction in programming time due to configuration or programming errors leading to improvement in overall equipment effectiveness.
	7.2	By lowering the effort required to (re)program industrial robots for new products and layouts, the system helps increase the practicality of production lines	Number of reconfigurations per year enabled without expert programming; reduction in commissioning time for new tasks; share of production volume handled by reconfigurable robotic cells.
	7.3	The combination of Digital Twin-based validation and task-level optimisation reduces unnecessary robot motions, contributing to higher energy efficiency at cell level.	Reduction in energy consumption per completed task; decrease in redundant robot movements; improvement in cycle time without loss of quality.
8	8.2	Intuitive, code-free task specification and visual feedback through Extended Reality support higher labour productivity by enabling operators without specialised programming skills to configure and adapt robot behaviour.	Number of tasks programmable by non-expert users; reduction in training time for new operators; increase in tasks executed per shift per operator.
	8.5	The framework promotes safer and more practical robot programming by allowing task verification and adjustment in the virtual twin before physical execution, reducing physical strain and exposure to hazardous situations.	Reduction in near-miss or collision reports in HRC cells; decrease in time spent in hazardous zones during programming; subjective improvement in perceived workload and safety (questionnaires).

Table 1: Contribution of the dissertation to selected UN SDG

Acknowledgements

I would like to express my sincere gratitude to my supervisors, Marcelo Petry and João Martins, for their invaluable guidance, continuous support, and constant availability throughout the development of this dissertation. Their expertise, constructive feedback, and encouragement were fundamental to the successful completion of this work.

I would also like to thank my family, especially my parents and my sisters, for their unconditional support, encouragement, and belief in me throughout my academic journey. Their motivation has always been one of the main driving forces behind my achievements.

I am deeply grateful to my girlfriend, Telma de Azevedo, for her unwavering support, patience, and encouragement throughout this journey. Whether during the moments when everything progressed as expected or during the more challenging periods when nothing seemed to work, her understanding and confidence in me were a constant source of motivation.

A special word of appreciation goes to Daniel Martins, Luís Leitão, Guilherme Osório, and Rui Martins for their friendship and for making the long hours spent both inside and outside the laboratory far more enjoyable. Their companionship throughout this semester greatly contributed to making this demanding period a memorable one.

Additionally, I would like to thank Tiago Pintado, João Ribeiro, Diogo Sousa, Eduardo Rocha, João Mendes, Gonçalo Rodrigues, Hugo Fonte, Tiago Correia, and Rafael Félix for their friendship and companionship throughout the years of university that ultimately led to this milestone. Sharing this journey with them made the challenges and successes of these years all the more meaningful. Finally, I would like to express my heartfelt thanks to everyone who has been part of this journey, including those I have not mentioned individually. Every gesture of support, encouragement, and companionship, no matter how small, has been meaningful and has contributed, directly or indirectly, to the completion of this dissertation.

<Guilherme Costa>

“You have exactly the same number of hours per day that were given to Helen Keller, Pasteur, Michelangelo, Mother Teresa, Leonardo da Vinci, Thomas Jefferson, and Albert Einstein”

H. Jackson Brown, Jr.

Contents

Acknowledgements	vii
1 Introduction	1
1.1 Context	1
1.2 Motivation	2
1.3 Objectives	3
1.4 Dissertation Structure	3
2 Background and Literature Review	5
2.1 Background	5
2.1.1 Reinforcement Learning	5
2.1.2 Imitation Learning	6
2.1.3 Digital Twins	8
2.1.4 Extended Reality	10
2.2 Literature Review	12
2.2.1 Robot Teaching through Demonstration	12
2.2.2 XR-Based Robot Programming Interfaces	13
2.2.3 Digital-Twin-Supported Learning and Simulation	15
2.2.4 Planning, Perception, and Adaptation to Changing Environments	16
2.2.5 Integrated XR-DT Robot-Teaching Systems	17
2.2.6 Findings	21
2.3 Chapter Summary	23
3 Methodology	25
3.1 Constraints and Assumptions	25
3.2 Proposed System Design	27
3.2.1 Conceptual Architecture	29
3.2.2 System Components and Information Flow	30
3.3 Methodological Limitations	32
3.4 Chapter Summary	33
4 Implementation	37
4.1 From Methodology to Implementation	38
4.2 Overall System Architecture	40
4.3 Task Definition in Isaac Lab	42
4.4 XR Teleoperation and Human Demonstrations	44
4.5 Dataset Annotation and Synthetic Demonstration Generation	46
4.6 Behaviour Cloning Policy Training	47

4.7	DT Extension and Runtime Execution	48
4.8	Policy Orchestrator and Task Sequencing	49
4.9	Integrated Workflow and Reproducibility	52
4.10	Chapter Summary	53
5	Results and Evaluation	55
5.1	Evaluation Setup	55
5.2	Evaluated Tasks	57
5.3	Demonstration and Dataset Generation Results	59
5.4	Behaviour Cloning Training Configuration	60
5.4.1	Training Convergence	60
5.5	Individual Policy Performance in the Training Environment	62
5.6	Individual Policy Performance in the Digital Twin Environment	63
5.7	Multi-Policy Sequential Execution	64
5.8	Failure Analysis	66
5.8.1	Orientation-Dependent Failures	66
5.8.2	Recovery Behaviour	67
5.8.3	Execution Timeout	67
5.8.4	Failure Distribution Across Tasks	68
5.9	Deployment Time and Reproducibility	68
5.10	Discussion	69
5.11	Chapter Summary	71
6	Conclusions and Future Work	73
6.1	Summary of Contributions	73
6.2	Main Findings	74
6.3	Limitations	76
6.3.1	System Limitations	76
6.3.2	Evaluation Limitations	77
6.4	Future Work	78
	References	81

List of Figures

2.1	Diagram of interactions between real and virtual environments	9
3.1	Conceptual architecture of the proposed robot-teaching system.	30
3.2	Workflow from XR demonstration to trained policy execution in the DT.	31
4.1	Implemented system architecture connecting XR teaching, synthetic data generation, policy training, and DT execution with policy orchestration.	41
4.2	Simulation environment for the developed tasks.	43
4.3	XR demonstration recording pipeline using Meta Quest 3, CloudXR, Isaac Lab teleoperation, and HDF5 dataset storage.	45
4.4	Dataset annotation, synthetic demonstration generation, and behaviour cloning training pipeline.	47
4.5	DT UI for object management, policy selection, and queued task execution. . . .	50
4.6	Software flow of the DT orchestrator for filtering executable policies and associating task-relevant prims with the current scene.	51
5.1	Motor components used in the evaluation: axis, top bearing, top housing, and bottom housing.	56
5.2	Training metrics over 2000 epochs for Top Bearing task	61

List of Tables

1	Contribution of the dissertation to selected UN SDG	vi
2.1	Classification of references by main research topics	20
4.1	Computational environment and software versions used in the implementation. . .	41
4.2	Description of the fields used in the policy naming convention.	50
5.1	Evaluation criteria and corresponding measurements.	58
5.2	Robomimic behaviour cloning training configuration.	61
5.3	Comparison of policy performance at the 1500-epoch and 2000-epoch checkpoints (50 executions per task).	62
5.4	Policy evaluation results in the Isaac Lab training environment (50 executions per task).	63
5.5	Policy evaluation results in the DT environment (50 executions per task).	63
5.6	Comparison of policy success rates between the Isaac Lab training environment and the DT runtime environment.	64
5.7	Two-policy chain evaluation results (50 executions per chain).	65
5.8	Three-policy chain evaluation results (50 executions per chain).	66
5.9	Failure attribution by task in the DT environment (50 executions per task).	68
5.10	Approximate time required to add and train the developed manipulation tasks . .	69

List of Acronyms

AI	Artificial Intelligence
AR	Augmented Reality
BC	Behaviour Cloning
CAD	Computer Aided Design
DMP	Dynamic Motion Primitive
DRL	Deep Reinforcement Learning
DT	Digital Twin
FIFO	First In, First Out
HMD	Head-Mounted Display
HRC	Human-Robot Collaboration
HRI	Human-Robot Interaction
IL	Imitation Learning
LfD	Learning from Demonstration
LLM	Large Language Model
MR	Mixed Reality
PbD	Programming by Demonstration
RL	Reinforcement Learning
SUS	System Usability Scale
UI	User Interface
USD	Universal Scene Description
VR	Virtual Reality
XR	Extended Reality

Chapter 1

Introduction

This chapter introduces the research context, motivation, objectives, and structure of the dissertation. It first situates the work within recent developments in Industry 4.0 and Industry 5.0, with particular attention to the role of Extended Reality (XR) in robotics. It then explains the problem addressed in this dissertation: the need for more accessible and adaptable methods for teaching robotic manipulation tasks.

1.1 Context

Industrial development is often described through four main revolutions: mechanization through steam and water power, mass production enabled by electricity and assembly lines, automation supported by electronics and computers, and, more recently, Industry 4.0, characterized by cyber-physical systems, connectivity, data exchange, and smart factories [1]. Since the early 2000s, Industry 4.0 has promoted the integration of information technologies into manufacturing systems, enabling cyber-physical production systems and smart factories in which production systems, components, and people communicate through networks and production processes can become increasingly autonomous. Building on this technological evolution, Industry 5.0 places stronger emphasis on human-centered design, diversity, inclusivity, accessibility, equity, sustainability, resilience, and closer human-machine collaboration [2, 3]. These paradigms are particularly relevant to this dissertation because they increase the need for flexible, adaptable, and user-friendly robotic systems in modern manufacturing. They create production environments that are more flexible and customized, but also more complex, since robots are increasingly expected to operate in changing workspaces, collaborate with human operators, and adapt to variations in products, tasks, and production requirements. As a result, effective Human-Robot Interaction (HRI) becomes essential: operators must be able not only to supervise robotic systems, but also to understand, teach, correct, and reconfigure robot behaviour in an intuitive way. This raises important challenges in adaptability, usability, and accessibility, especially when robotic systems are intended to be used by non-expert users in dynamic industrial settings.

These challenges motivate the use of interaction technologies that can reduce the gap between human intention and robotic execution. In particular, **XR** technologies are relevant because they can make robotic systems more understandable and easier to interact with by presenting digital information directly in the physical workspace. Instead of requiring users to interpret robot behaviour through abstract code, external screens, or low-level interfaces, **XR** can overlay trajectories, target poses, task instructions, warnings, and feedback in the operator’s field of view [4, 5]. In robotic manipulation, this is especially important because teaching and programming robots remain difficult tasks for non-expert users [6, 7]. By allowing users to demonstrate, visualize, and refine robot behaviour through spatially grounded interaction, **XR**, Digital Twin (**DT**)s, and Artificial Intelligence (**AI**) can help connect human intention with robotic execution [8, 9].

1.2 Motivation

Robotic manipulation remains a demanding area of industrial automation. Despite significant technical progress, programming and deploying robotic manipulators often still requires specialized personnel. This makes it difficult for companies to respond quickly to changing production requirements [6].

A major reason for this difficulty is the rigidity of conventional programming approaches. Many systems still depend on manual coding, teach pendants, or task-specific configuration. These methods provide limited support for adapting a learned motion to new object positions, different workspaces, or slightly modified tasks [6, 10]. As a result, even small changes in the production environment can require substantial reprogramming, which limits the scalability and adaptability of robotic systems [4, 8].

These limitations are particularly relevant in the context of Industry 5.0, where industrial systems are expected to support human-centred design, inclusivity, and effective Human-Robot Collaboration (**HRC**) [3, 11]. If operators without advanced robotics expertise are to participate more directly in robot teaching, the technical barriers associated with programming and interaction must be reduced. Improving usability is therefore not only a matter of convenience; it is also necessary for making robotic systems more practical in dynamic industrial environments [9, 12].

Several emerging technologies can help address this problem. Head-Mounted Display (**HMD**)s allow immersive visualization of robotic workspaces, helping users observe, understand, and guide robot behaviour more naturally [8, 10]. **DT**s provide virtual representations of robots and their environments, supporting safe experimentation, rapid testing, and transfer from virtual programming to physical execution [5, 13]. **XR** technologies, also support interaction methods such as gesture-based demonstrations, spatial annotations, and real-time trajectory visualization [4, 7].

Together, these tools create an opportunity to rethink how robotic manipulation tasks are taught, validated, and executed. A teaching method that is visually grounded, user-friendly, and adaptable can support the principles of Industry 5.0 while addressing the practical limitations of traditional robot programming.

1.3 Objectives

This dissertation proposes a methodology for intuitive programming of industrial manipulators using an **HMD** and its **XR** capabilities. The goal is to allow a user to teach a manipulation task, such as picking up an object and placing it in a target location, while enabling the learned task to adapt to similar scenarios with variations in object position or environment configuration.

The specific objectives are:

- Design an **XR**-based interface and workflow that allows operators to teach manipulation tasks directly in the simulated workspace.
- Develop a **DT** representation of the robot and its environment to support task teaching, refinement, and validation before physical execution.
- Implement a software framework that integrates demonstration collection, task learning, dataset generation, policy training, and task execution.
- Evaluate the proposed methodology in terms of usability, task execution performance, and adaptability to variations in object pose and environmental conditions in a controlled simulation setting.

1.4 Dissertation Structure

The remainder of this dissertation is organized as follows.

Chapter 2 presents the background and literature review. It introduces the main concepts needed to understand the proposed approach, including robot manipulator kinematics, path planning, Reinforcement Learning (**RL**), **DTs**, virtual environments, and extended reality. It also reviews related work on Learning from Demonstration (**LfD**), **XR**-based robot programming, **DTs**, and **HRC**.

Chapter 3 presents the proposed methodology. It explains the conceptual workflow of the approach and the rationale behind each design decision, focusing on what each stage contributes to the proposed system. The chapter describes how the robot-teaching process is organized, including **XR**-based demonstration collection, subtask decomposition, simulation-based data augmentation, Behaviour Cloning (**BC**) policy learning, and modular task orchestration.

Chapter 4 describes the implementation details of the proposed system, focusing on the specific tools, software modules and data formats. It explains how the methodology was translated into a working technical framework, including the hardware configuration, the **XR** teleoperation interface, the information presented to the operator through the **HMD**, the task environment structure, the demonstration recording pipeline, IsaacLab Mimic data generation, Robomimic policy training, and the naming-convention-based orchestrator.

Chapter 5 presents the results of evaluating the proposed system. It includes quantitative metrics on policy success rates, the effect of dataset augmentation, robustness to object position variations, multi-task orchestrator performance, and training dynamics.

Chapter 6 presents the conclusions, limitations, and future work. It summarizes the main contributions of the dissertation, discusses the current constraints of the implementation, and identifies directions for further development, including physical validation and broader task coverage.

Chapter 2

Background and Literature Review

This chapter introduces the technical concepts needed to understand the proposed system, including **RL**, Deep Reinforcement Learning (**DRL**), Imitation Learning (**IL**), **BC**, **DTs**, virtual environments, and **XR**. After that, it reviews related work on robot teaching, **LfD**, **XR**-based robot programming, **DT**-supported learning and simulation, planning and adaptation, and integrated **XR-DT** robot-teaching systems. The reviewed works are then classified according to their main research topics, allowing the chapter to identify the gaps and limitations that motivate the methodology proposed in this dissertation.

2.1 Background

The proposed robot-teaching system depends on several connected concepts, each of which supports a specific part of the methodology. Robotic learning methods provide mechanisms for adapting robot behaviour from data. Within this broader area, **RL** and **DRL** are relevant as general background because they show how robots can learn through interaction with an environment, although the methodology developed in this dissertation does not rely on reward-based policy optimization. Instead, **IL** and, more specifically, **BC** are central to this work because they allow manipulation policies to be learned from demonstrations. **DTs** and virtual environments provide the simulated representation in which tasks can be demonstrated, tested, augmented, and executed. Finally, **XR** provides the human-facing interaction layer through which users can teach and inspect robot behaviour in a more spatial and intuitive way.

2.1.1 Reinforcement Learning

Many recent approaches to robot adaptation use **RL** or methods derived from it. **RL** is a machine-learning method in which one or more agents learn to make decisions by interacting with an environment and by receiving rewards that evaluate the quality of their actions [14]. Through repeated interaction, the agent learns a policy that maps states to actions in order to maximize long-term cumulative reward. **RL** problems are commonly modelled as Markov Decision Processes, where the next state depends on the current state and action [15].

Classical **RL** is often limited by tabular or low-dimensional representations. Robotics, however, frequently requires high-dimensional state information, such as images, force signals, robot states, or complex sensor data. **DRL** addresses this limitation by using deep neural networks to approximate value functions or policies, making it possible to handle high-dimensional decision problems that would otherwise be difficult to solve. In robotics, **DRL** has been applied to motion planning, grasping, obstacle avoidance, and manipulation in dynamic environments [15]. Its main advantage is that it can support adaptation to changing conditions and learn behaviours that generalize across related scenarios.

Training **DRL** policies directly on physical robots is usually impractical. It requires large amounts of interaction data, can damage hardware during exploration, and makes repeated trial-and-error expensive. For this reason, much of the training is performed in simulation, where large numbers of interactions can be executed quickly and safely before transferring the learned policy to the physical system [16]. In this dissertation, **RL** is therefore used mainly as contextual background for learning-based robotics. The implemented learning approach is not based on reward design or interactive exploration, but on learning from demonstration data through **IL** and **BC**.

2.1.2 Imitation Learning

IL methods aim to learn robot behaviour directly from demonstrations provided by a human operator, an expert controller, or another source of task examples. Instead of defining a reward function and allowing the robot to discover a behaviour through trial-and-error, as in **RL**, **IL** uses examples of the desired behaviour as the main source of supervision. This makes **IL** particularly relevant for robot-teaching systems, because the user can communicate the intended task by demonstrating it rather than by manually specifying low-level commands, waypoints, or control rules [17, 18].

IL is closely related to **LfD**. In **LfD**, the robot learns from examples of a task provided by a teacher. These examples may be collected in several ways, such as kinesthetic guidance, teleoperation, motion capture, or demonstration in a virtual environment. Although the terms are sometimes used with slightly different emphases, **LfD** usually refers to the broader robot-teaching paradigm, while **IL** refers more specifically to the learning algorithms that infer a policy from the demonstration data. In this dissertation, this distinction is useful because the proposed system includes both aspects: a teaching workflow for collecting demonstrations, and a learning stage where those demonstrations are converted into executable manipulation policies.

A demonstration is typically stored as a trajectory, that is, a sequence of observations and actions recorded over time. In robotic manipulation, observations may include information such as robot joint positions, end-effector pose, gripper state, object pose, target pose, or other task-relevant variables. Actions correspond to the commands that should be executed by the robot, such as desired end-effector displacements, joint commands, or gripper opening and closing commands. The learning objective is therefore to estimate a policy that maps observations to actions in a way that reproduces the demonstrated task behaviour. This formulation is well aligned with manipulation tasks, where the same high-level objective, such as picking and placing an object, can be represented through a sequence of state-action pairs.

One of the most common and direct **IL** approaches is **BC**. In **BC**, policy learning is formulated as a supervised learning problem. The demonstrations provide input-output examples, where the input is the observation and the output is the action executed by the demonstrator. The policy is then trained to predict the demonstrated action for each observed state. This makes **BC** comparatively simple to implement, compatible with offline datasets, and suitable for workflows where demonstrations are collected first and policy training is performed afterwards. For this reason, **BC** is widely used as a baseline and as a practical method for learning manipulation policies from demonstration datasets [18, 19].

However, **BC** also has important limitations. First, the performance of the learned policy depends strongly on the quality, consistency, and diversity of the demonstrations. If the demonstrations contain imprecise motions, ambiguous behaviours, or insufficient variation, the learned policy may reproduce those limitations. Second, **BC** suffers from distribution shift. During training, the policy observes states from the expert demonstrations. During execution, however, the robot acts according to its own predictions. Small prediction errors can move the robot into states that were not present in the training data, causing further errors to accumulate over time [20]. This is especially relevant in robotic manipulation, where small deviations in grasp pose, object position, or approach trajectory can determine whether a task succeeds or fails.

This limitation is important for the proposed workflow because the quality of the learned policy depends not only on the number of demonstrations, but also on how well those demonstrations cover the variations that may occur during execution. If the policy is trained only on a small set of similar trajectories, it may reproduce those examples accurately during training while still failing when the object pose, initial robot configuration, approach direction, or execution conditions differ from the demonstrated cases. In this situation, the robot may enter states that are valid for the task but poorly represented in the dataset, making the policy more vulnerable to accumulated errors.

For this reason, the diversity of the demonstration dataset becomes a central requirement in **BC**-based robot teaching. Additional demonstrations, controlled variations of the task, and simulation-based dataset expansion can expose the policy to a broader range of states and actions while preserving the intended manipulation behaviour. This does not eliminate distribution shift completely, but it can reduce the dependency on a small number of original trajectories and make the later evaluation of robustness more meaningful.

Several strategies can reduce the limitations of **BC**. One possibility is to collect more demonstrations, especially demonstrations that cover different object positions, initial robot states, and task variations. Another possibility is to use interactive methods, such as dataset aggregation, where the policy is iteratively corrected with additional expert data in states induced by the learned policy [20]. In the context of this dissertation, the relevant point is that the training dataset must include enough variation for the learned policy to be evaluated beyond the exact trajectories originally demonstrated.

For the proposed methodology, **IL** and **BC** are therefore central because they provide the link between intuitive task teaching and autonomous policy execution. The user demonstrates the task, the demonstration is stored as trajectory data, additional examples can be generated in simulation,

and a policy is trained to reproduce the demonstrated behaviour. This avoids the need for reward engineering and reduces the programming burden for non-expert users, while still allowing the robot to execute the task autonomously within the represented environment. At the same time, the known limitations of **BC**, particularly sensitivity to dataset quality and distribution shift, motivate the use of simulation, dataset augmentation, and controlled evaluation in the remaining chapters.

2.1.3 Digital Twins

DTs are virtual representations of physical objects or systems that can be used to represent, analyse, monitor, or simulate the behaviour of their real-world counterparts [14]. In their strongest form, **DTs** are continuously synchronized with a physical system through real-time data, allowing the virtual representation to reflect the current state, performance, and condition of the real system. In robotics, this may include information about the robot geometry, kinematics, dynamics, sensors, actuators, workspace, objects, and interactions with the environment.

However, the term **DT** is used with different levels of complexity in the literature. Some **DTs** focus mainly on real-time monitoring and data synchronization, while others emphasize simulation, visualization, task validation, or virtual commissioning. This distinction is important for the present dissertation. The system proposed in this work does not implement a fully synchronized industrial **DT** connected to a physical production cell. Instead, it uses a simulated digital representation of the robot, objects, and task environment to support robot teaching, policy learning, validation, and execution. Therefore, the **DT** concept is used here in an operational and simulation-oriented sense.

A **DT** does not necessarily require a rich graphical or three-dimensional environment. Some **DTs** may be based mainly on analytical models, sensor data, or monitoring dashboards. However, in robot teaching and manipulation tasks, including a virtual environment as part of the **DT** is especially useful because it provides spatial information about the robot, objects, target poses, obstacles, and workspace constraints. In this dissertation, the virtual environment is considered an operational representation layer of the **DT**, since it is the space where demonstrations can be interpreted, tasks can be tested, and learned policies can be executed.

Figure 2.1 illustrates the interaction between real and virtual environments in a general **DT** architecture. In this type of architecture, the real environment performs the physical task, produces data, and communicates with other devices. The virtual environment supports planning and simulation by translating information from the physical system into a simulated representation. This makes it possible to test trajectories or task behaviours in the virtual space before applying them to a real robot. In the scope of this dissertation, the emphasis is placed on the virtual side of this architecture, as a controlled environment for robot teaching and evaluation.

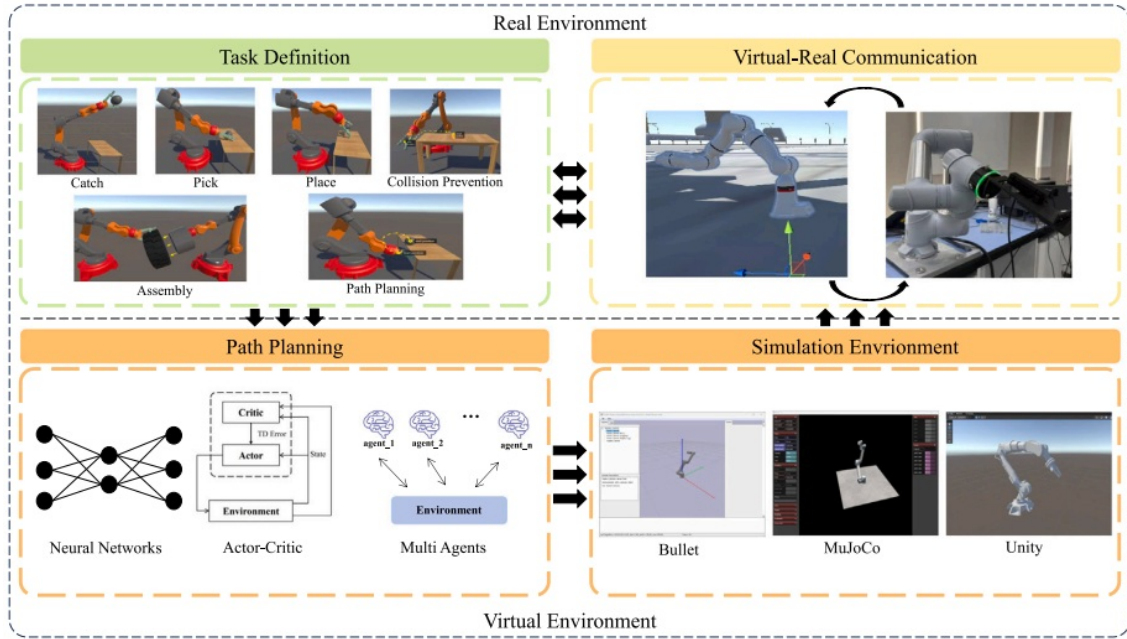


Figure 2.1: Diagram of interactions between real and virtual environments

In robotics, a digital representation may model not only the geometry of a robot but also its kinematics, dynamics, control interfaces, end-effector state, object poses, and relevant task constraints. When a physical robot is involved, synchronization with sensor data can allow the **DT** to monitor operational parameters, detect anomalies, or support predictions about performance and wear [5]. These capabilities are important in the broader field of industrial **DTs**, particularly for monitoring and maintenance. In this dissertation, they are treated as general background rather than as implemented features.

For robot teaching, the most relevant role of the **DT** is to provide a safe and controllable environment for training, validation, and execution. When the **DT** includes a three-dimensional virtual environment, it can represent the robot, nearby objects, target locations, and workspace constraints. This provides an operational space where learning methods, such as **IL** or **BC**, can be trained or evaluated before any physical deployment is considered. When realistic physics engines, accurate object models, and sensor assumptions are included, the virtual representation can support testing of grasping strategies, trajectory generation, collision checking, path planning, and task sequencing [21]. It also supports rapid prototyping because changes to layout, objects, or task specifications can be tested without modifying a physical setup.

A simulation-oriented **DT** also helps address a major bottleneck of learning-based robotics: data collection. Training and testing directly in the physical world can be slow, expensive, and potentially unsafe, whereas simulation allows repeated executions under controlled conditions [22]. Simulation can also be used to vary task parameters such as object positions, initial robot states, friction, or sensor assumptions, which can improve the diversity of the data available for learning [23]. This is particularly relevant for **BC**-based workflows, where the learned policy depends strongly on the coverage and quality of the demonstration dataset.

Digital representations can also make robot programming more accessible. Operators can inspect a simulated robot and its task environment before executing or validating behaviours. This is useful for tasks involving constrained workspaces, uncertain object poses, or operation near humans. When combined with **XR**, the virtual representation can help non-expert users understand robot motion, reachability, collisions, and task constraints, reducing the technical barrier to robot programming [8]. In this context, the **DT** acts as a bridge between the learning pipeline and the human-facing interaction interface.

Despite these advantages, **DTs** and simulation-based digital representations also introduce limitations. A complete real-time **DT** requires accurate models, calibration, sensor integration, and continuous synchronization with the physical system, all of which can be technically demanding. Even when the focus is simulation rather than full synchronization, the validity of the results depends on the accuracy of the robot model, object models, physics simulation, and task assumptions. Differences between the simulated environment and a physical setup may affect policy performance if the system is later transferred to real hardware. For this reason, the use of the **DT** concept in this dissertation should be understood as a controlled digital representation for teaching, learning, visualization, and evaluation, rather than as a complete real-time industrial twin.

2.1.4 Extended Reality

XR refers to technologies that enhance or replace the user’s view of the world by presenting digital information in real, virtual, or combined environments [24, 25]. It is an umbrella term that includes Augmented Reality (**AR**), Mixed Reality (**MR**), and Virtual Reality (**VR**). In the context of this dissertation, **XR** is used as the broader term because the proposed approach relies on immersive visualization and spatial interaction capabilities rather than on a single strict category of reality technology.

VR replaces the user’s view of reality with a fully virtual world. **AR**, by contrast, overlays virtual elements onto the real world, supporting a combined view of physical and digital information [6, 25]. **MR** also combines physical and digital elements, but places stronger emphasis on interaction between the user and virtual objects anchored in the real environment [24]. For example, a user could manipulate a virtual representation of a robot end effector or target pose to define the intended task spatially. In practice, robot-teaching systems may combine characteristics from these categories, which is why the broader term **XR** is useful.

The relevance of **XR** for robot teaching comes from the technical characteristics of immersive interfaces, especially when they are implemented through **HMDs**. These devices and related spatial interfaces can provide stereoscopic visualization, six-degree-of-freedom head tracking, hand or controller tracking, depth perception, a user-centred field of view, and interactive three-dimensional feedback. Some systems may also use passthrough or mixed-reality visualization, allowing the user to perceive physical surroundings while interacting with digital information. Together, these characteristics allow information to be presented spatially in relation to the robot, the objects, and the task environment, rather than only through abstract code, external screens, or low-level interfaces [4, 26, 27].

In robotic manipulation, these capabilities are useful because they support a more spatial form of robot teaching. The operator can observe the robot and the scene from an embodied perspective, inspect the relation between the end effector, objects, and targets, and understand task constraints more directly. Spatial input can also make it more natural to control or guide the end effector, define target poses, and collect demonstrations, because the interaction is closer to how humans reason about motion and object manipulation. In addition, XR can be used to visualize trajectories, poses, task instructions, warnings, and workspace constraints directly in relation to the task environment. This reduces the abstraction required by traditional programming interfaces, where the user must often reason in terms of numerical coordinates, robot-specific commands, or separate two-dimensional displays [4, 26, 27].

In robot-teaching applications, XR can function as a bridge between the human operator and a digital or simulated representation of the robotic task. Rather than interacting only with low-level code, teach pendants, or external two-dimensional displays, the operator can use immersive visualization and spatial input to observe the robot, inspect the scene, define task-relevant poses, or provide demonstrations. The simulated or digital environment can then provide a controlled space where the taught behaviour is interpreted, tested, and refined before deployment or evaluation. In this broader sense, XR is not a complete programming solution by itself, but an interaction layer that can make robot teaching more intuitive while still depending on the surrounding learning, validation, and execution pipeline [4, 26, 27].

At the same time, XR does not solve the robot-programming problem by itself. The quality of the interaction depends on the accuracy and stability of the sensing and display pipeline. Hand or controller tracking may be affected by occlusions, lighting conditions, limited precision, jitter, or fast user movements. When visual content is streamed from an external workstation, additional latency may also be introduced, which can affect the correspondence between the user's motion and the displayed robot response. Field of view limitations, ergonomics, headset weight, and prolonged use can also influence user comfort and demonstration quality. These aspects are particularly important for LfD, because the quality of the collected demonstrations directly affects the quality of the trained policy [26, 27].

For this reason, spatial data collected through XR should be treated as an estimate of the user's intended motion rather than as a perfect measurement. Camera-based hand or body tracking, for example, estimates the position and orientation of the user's hands or body from sensor observations, and this estimate may contain noise or temporary errors. As a result, XR should be understood as one component of a broader robot-teaching pipeline, together with simulation, dataset generation, policy learning, and task validation. Its role is to make teaching more intuitive and spatially grounded, while the remaining stages of the pipeline are still needed to structure demonstrations, train policies, evaluate robustness, and identify execution limitations [4, 27].

For Industry 5.0, XR is relevant because it supports human-centered interaction. It can reduce the learning curve for operators, make robot behaviour easier to understand, and provide access to spatial information that would be difficult to obtain through traditional interfaces. For these reasons, XR is a key component of human-centered robot-teaching approaches, acting as an interface

between the user, the digital task representation, and the learning-based task execution workflow [4, 26].

2.2 Literature Review

The literature search was guided by the following query: robots AND ("robot programming" OR "imitation learning" OR "learning from demonstration") AND ("augmented reality" OR "mixed reality" OR "extended reality"). This query was selected to identify work that addresses how robots are programmed or taught, including through imitation learning, using augmented, mixed, or extended reality interfaces. The query was applied in IEEE Xplore and returned 133 results.

The reviewed works can be organized around a common problem: how to make robot programming more intuitive, safe, and adaptable for users without requiring expert-level knowledge of robot control. To make this discussion clearer, the review is structured from individual components towards more integrated systems. It first considers demonstration-based learning methods, independently of the interface used to collect or specify the task. It then examines XR-based interfaces as the human-facing layer for robot programming and task specification, before turning to simulation and DT-supported approaches as infrastructure for training, testing, and validation. The review then discusses works concerned with robustness, planning, perception, and adaptation to changes in the task environment. Finally, it considers integrated robot-teaching systems that combine several of these dimensions, such as demonstration, immersive interaction, simulation, learning, and task execution.

2.2.1 Robot Teaching through Demonstration

Early approaches to robot programming were closely connected to inverse kinematics and trajectory specification. These methods required detailed knowledge of the robot model and the mathematical formulation behind its motion. Although inverse kinematics remains essential for manipulator control, recent work has focused on making programming more accessible to users without specialized robotics training.

One widely studied alternative is LfD, which is the paradigm in which robots learn a task policy from examples provided by a human teacher [28]. LfD can support IL and, in some systems, can also be combined with RL to reproduce or refine human demonstrations. Recent work on Gaussian-process-based LfD [28], Dynamic Motion Primitive (DMP)s [29], and trajectory imitation with RL [30] shows that probabilistic and generative models can encode variation across demonstrations and generalize to new goals. However, these methods usually assume structured demonstrations and do not by themselves solve collision avoidance or adaptation to moving obstacles.

Demonstrations in LfD can be collected through kinesthetic guidance, where the operator physically moves the robot end effector along the intended trajectory, as in Lauretti et al. [29], or through motion-capture systems such as Xsens MVN, as in Arduengo et al. [28]. Markerless motion-retargeting systems extend this idea by mapping human movement to robot motion

through a **DT**, using low-cost cameras instead of instrumented suits [31]. These systems can reduce hardware cost and support more natural demonstrations, but they also introduce challenges such as pose-estimation jitter, difficult human-to-robot joint mapping, and calibration errors that may produce unsafe trajectories [31].

Traditional **LfD** methods, including teach pendants and kinesthetic guidance, also tend to produce mainly imitative behaviour. As a result, new tasks often require new demonstrations, and robustness to environmental changes remains limited. More advanced frameworks, such as TS-RIL (Two-Stage Robot Imitation Learning) [32], separate the problem into trajectory learning and obstacle avoidance, showing that reliable deployment in cluttered environments often requires combining **LfD** with planning modules. TS-RIL [32] shows that explicit obstacle-avoidance layers can improve safety, but this comes with additional system complexity and new failure modes when perception or planning is not aligned with the demonstration space.

Classical **LfD** and motion-retargeting pipelines often assume that the execution conditions remain close to those observed during demonstration. This is not a limitation of a single approach, but a broader challenge in demonstration-based robot teaching: the learned behaviour depends on how the task, the robot state, and the environment are represented during training and execution. Some recent systems address this issue by adding richer state representations and curriculum or residual **RL**. For example, **DT**-assisted 3C assembly learning decomposes demonstrations collected in a multimodal virtual environment into reusable skills and refines them with **RL** and domain randomization before transfer to the real system [33]. These works show that combining demonstrations with simulation, structured task representations, and additional adaptation mechanisms can improve robustness beyond direct trajectory reproduction. At the same time, adaptation to substantially changed scenes, unforeseen obstacles, or dynamic human co-workers remains difficult and generally requires perception, scene updating, and planning mechanisms that go beyond the demonstration policy itself.

2.2.2 XR-Based Robot Programming Interfaces

To improve accessibility, a growing body of research has explored **XR** as a way to provide more intuitive interfaces for robot programming and interaction. However, **XR**-based robot programming is not a single type of approach. The reviewed systems use **XR** for different purposes, including visualization and feedback, waypoint-based programming, teleoperation, and demonstration collection. These categories are related, but they have different implications for the type of user input required, the flexibility of the resulting robot behaviour, and the amount of learning or planning needed after the interaction.

One common use of **XR** is visualization. In this case, the interface mainly helps the user understand the robot, the task, or the environment, but does not necessarily define the complete robot program by itself. Early work on **AR**-based robot programming showed that overlaying Computer Aided Design (**CAD**) models and virtual assembly sequences on the real workspace allows operators to assemble components virtually and derive robot programs without relying on traditional teach pendants [6]. Similarly, surveys on **AR** for human-robot collaboration and

collaborative robots in manufacturing report that **XR** interfaces can improve task understanding, reduce training time, and support flexible reconfiguration in Industry 4.0 and 5.0 settings. At the same time, these surveys identify gaps in safety, standardization, and real-world validation [4, 9, 12]. In this type of system, the main contribution of **XR** is to make robot state, task information, trajectories, or constraints easier to interpret.

Beyond visualization, **XR** can also support waypoint-based or rule-based task specification. Here, the user actively defines elements of the robot program by placing poses, selecting objects, drawing paths, or specifying task rules in space. Yigitbas et al. [34] proposed DART (Digital Augmented Reality Twin), a framework that combines **XR** and **DTs** to replace traditional web-based interfaces with more immersive forms of human-robot interaction. The reported findings indicate improved user satisfaction and task comprehension when using **XR**. However, the programming process took longer than with a standard web interface and still required some programming literacy from participants [34]. This suggests that **XR** alone does not guarantee accessibility. Even when spatial interaction is available, users may still need to reason about program structure, task logic, coordinate frames, or the sequence of robot actions.

Subsequent work has addressed these issues through end-user programming and multimodal interaction within **XR**. MARCER (Multimodal Augmented Reality for Composing and Executing Robot Tasks) [35], for example, combines an Large Language Model (**LLM**)-based natural-language interface with **XR** visualization, allowing non-experts to specify trigger-action rules using speech and in-situ holograms. This enables hands-free programming without code, but the system depends on external motion-capture infrastructure and remains vulnerable to **LLM** logic errors and to the limited expressiveness of predefined trigger-action primitives [35]. Similarly, RoboART [36] and the production-line **XR** tool presented by Fridman et al. [37] show that immersive interfaces can allow artists or operators to define trajectories through gestures and holographic waypoints. However, these approaches often support only simple motion primitives and provide limited support for path constraints, collision handling, or formal usability evaluation in industrial settings. In these systems, **XR** reduces the abstraction of robot programming, but the resulting behaviour can still be rigid if the specified waypoints or rules are not combined with planning, perception, or adaptation mechanisms.

Immersive teleoperation represents another use of **XR**. In teleoperation systems, the user does not simply place waypoints or inspect a visual overlay, but controls the robot or its end effector more directly through an immersive interface. This can make interaction more natural because the operator can guide the robot using spatial motion, hand input, controllers, or haptic feedback. Teleoperation can also support supervision and correction during execution. However, it introduces different constraints from waypoint-based programming: the quality of the interaction depends on tracking accuracy, latency, correspondence between user motion and robot motion, and, when available, the fidelity of force or haptic feedback.

When **XR** is used for demonstration collection, the goal is not only to define a single trajectory, but to collect examples that can later be used to train a reusable policy. Integrating **XR** into **LfD** interfaces is therefore a promising way to reduce cognitive load beyond manual waypoint pro-

gramming. **XR**-based teleoperation and imitation-learning pipelines such as HiLoTEL [38] and Haptic-ACT [39] show that immersive interfaces with haptic feedback can collect high-quality demonstrations while reducing operator effort and contact forces. These systems also allow human intervention or supervision during execution [38, 39]. However, the quality of the recorded trajectories becomes central: tracking noise, latency, task complexity, and operator inconsistency may affect the dataset and therefore the learned policy. These systems are also often evaluated on relatively simple pick-and-place tasks, and challenges remain in scaling to long-horizon manipulation, handling irreversible actions safely, and maintaining stable performance under latency and sensor noise [38, 39].

Overall, **XR**-based systems show that intuitive spatial interfaces can lower the barrier to robot programming, but their contribution depends on how **XR** is used. Visualization-oriented systems mainly improve understanding and feedback. Waypoint- or rule-based systems allow users to specify robot behaviour more directly, but may still produce rigid programs. Teleoperation systems provide more continuous control, but are sensitive to latency, tracking, and embodiment issues. Demonstration-collection systems can support learning reusable policies, but depend strongly on demonstration quality and dataset coverage. This distinction is important because **XR** should not be treated only as a generic interface layer. Its role in robot teaching depends on whether it is used to visualize information, specify tasks, control the robot, collect demonstrations, or support an integrated pipeline that also includes learning, planning, and environment representation.

2.2.3 Digital-Twin-Supported Learning and Simulation

DTs and simulation environments are increasingly used to reduce the cost, risk, and physical effort associated with robot teaching. In the literature, the term **DT** may refer to fully synchronized physical-virtual systems, but it is also often used more broadly for digital representations that support simulation, visualization, training, and validation. This distinction is important in the context of this dissertation, where the **DT** concept is used mainly in a simulation-oriented sense rather than as a continuously synchronized replica of a physical robot cell. By reproducing the robot, workspace, objects, and relevant constraints in a virtual representation, these systems allow demonstrations, trajectories, and policies to be tested in controlled conditions before any physical deployment is considered. This is particularly important for learning-based methods, where repeated interactions or large datasets may be required.

Wang et al. [40] presented a related strategy for bridging demonstration and execution by integrating **XR**, a **DT**, and demonstration-based teaching for welding tasks. Other **DT**-assisted systems combine **XR** demonstrations with skill learning and **RL** refinement in a high-fidelity virtual environment before transfer to the real robot [33]. In the welding system by Wang et al. [40], a handheld controller served as the reference for the robot’s welding tip, while an **XR**-based **DT** analyzed welder behaviour and synchronized operations with the physical robot. These examples represent stronger physical-virtual **DT** architectures than the one considered in this dissertation,

but they are relevant because they show how virtual representations can support safer demonstration, analysis, and task validation. They also illustrate that the benefits of **DT**-supported teaching depend on the fidelity of the representation and on how well it captures the relevant task conditions.

These approaches improve safety and operator comfort, but they still inherit some limitations of **LfD**: trajectories can remain rigid, generalization is bounded by the demonstrated data, and adaptation depends on the information available in the virtual or digital representation. This point should be understood as a general challenge in **DT**-supported robot teaching rather than as a solved problem in the present work. In simulation-oriented systems, such as the one considered in this dissertation, this challenge appears mainly as a dependency on the accuracy of the simulated scene, object models, task assumptions, and policy evaluation conditions.

Studies on industrial robot **DT** collision detection and **DTs** for collaborative robots also show that high-fidelity virtual models can improve simulation and fault detection, while highlighting the difficulty of maintaining accurate real-time synchronization and standardized modelling methods across platforms [13, 41, 42]. **XR**-based **DT** frameworks for sim-to-real transfer show that hybrid physical-virtual setups can accelerate training and evaluation of **RL** policies, while still facing challenges in domain randomization, parallel simulation, and synchronization between the **DT** and the physical world [43]. These works are useful for positioning the broader field, but their synchronization and physical deployment components should be distinguished from the more limited use of a digital representation as a simulation, training, and validation environment.

Therefore, **DTs** are valuable not only as visualization tools, but also as infrastructure for training, validation, and safer deployment. In this dissertation, however, this value is explored primarily through a simulated representation of the robot task environment. The focus is on using the digital environment to support demonstration processing, dataset generation, policy learning, task execution, and evaluation under controlled conditions, rather than on implementing a complete real-time **DT** connected to a physical robotic cell.

2.2.4 Planning, Perception, and Adaptation to Changing Environments

A recurring challenge across demonstration-based, **XR**-based, and **DT**-based systems is the ability to adapt robot behaviour when the execution environment differs from the demonstration or programming environment. If a robot only reproduces demonstrated trajectories or follows manually specified waypoints, its behaviour may fail when object poses change, obstacles appear, or the workspace configuration is modified. For this reason, several works integrate planning, collision avoidance, perception, or dynamic scene representations into the robot-teaching process. In this dissertation, these topics are reviewed mainly as related work and future-facing context, because the implemented system is evaluated in simulation, with controlled scenes, known objects, and task-relevant poses available from the simulated environment.

Dogangun et al. [44] proposed RAMPA, which couples **XR**-based trajectory specification with path planning and a digital representation of the workspace. The system allows the user to draw or specify trajectories through an immersive interface while using planning mechanisms to generate

executable robot motion. Soares et al. [10] similarly showed that operators without programming experience can program collaborative and industrial robots using an XR-based interface and a robotics back end. These works are relevant because they reduce the abstraction of robot programming and show how spatial interfaces can be combined with planning. However, they also illustrate that intuitive trajectory specification does not, by itself, solve robustness to changing environments.

Extensions of this line of work have incorporated collision checking and trajectory adaptation so that generated paths can avoid obstacles instead of remaining fixed. This is important because a trajectory that is valid in the demonstration scene may become invalid when the workspace changes. Nevertheless, planning-time collision checking and trajectory deformation usually assume that the relevant obstacles and object poses are already known to the system. In dynamic environments, robust execution would require continuous scene updating, perception, prediction, and re-planning. These capabilities are important for real-world deployment, but they are beyond the scope of the system implemented in this dissertation.

The need for richer scene representations is also reflected in communication-efficient XR systems. Liu et al. [45], for example, introduced a Gaussian Splatting mixed-reality approach in which the environment is represented in a compact form that can be rendered from different viewpoints without continuously streaming full-resolution video. Such representations can support more efficient visualization and robot-environment understanding in bandwidth-constrained settings. At the same time, they introduce their own limitations, such as sensitivity to changes in lighting or geometry and dependence on the quality of the reconstructed model.

Other motion-retargeting and teleoperation systems use markerless pose estimation, object detection, or scene reconstruction to maintain a more informative representation of the robot workspace. Ng et al. [31] show how a digital representation can support motion retargeting and robot-environment synchronization within an XR interaction loop. These approaches indicate that providing the robot with information beyond the demonstrated trajectory can improve adaptability and reduce some limitations of classical LfD. However, they also depend on perception quality, calibration, and scene consistency.

Overall, planning, perception, and adaptive scene representations are essential for robust robot teaching in changing environments. For the present dissertation, they are important mainly because they define the boundary between what the proposed workflow addresses and what remains open. The work developed here focuses on controlled variations in object pose and scene configuration within simulation, rather than on continuous environment monitoring, perception of unknown objects, or automatic re-planning in dynamic physical workspaces.

2.2.5 Integrated XR-DT Robot-Teaching Systems

In this section, integrated robot-teaching systems are understood as approaches that combine more than one of the dimensions discussed in the previous subsections: immersive interaction, digital or simulated task representations, demonstration-based programming, teleoperation, learning, adaptation, or task execution. The level of integration varies considerably across the literature. Some

works use **XR** mainly to visualize or manipulate virtual robot representations, while others use digital representations to support collaboration, demonstration collection, or end-user programming. This makes the section useful not only as a continuation of the literature review, but also as a comparison with the works that are closest to the scope of this dissertation.

Several systems combine **XR** with spatial demonstration or trajectory definition. In these approaches, the user interacts with a holographic or virtual representation of the robot, end effector, or task trajectory instead of programming the physical robot directly. Ngui [46] developed an **XR**-based virtual representation of a robotic arm that users can program by guiding a holographic end effector along the desired trajectory. This improves safety, particularly for manipulators equipped with cutting tools, because demonstration can be performed through the robot's **DT** rather than the physical robot. Hubel [7, 8] similarly showed that users can manipulate holographic robots to reach Cartesian frames without code, reducing the mental workload associated with traditional programming interfaces. Femenia [47] explored **XR**-based programming by demonstration, where experts generate trajectories in a virtual painting workspace and transform them into executable programs for collaborative robots. In these systems, **XR** and **DTs** can capture motion parameters such as pressure, orientation, and speed. These approaches are relevant because they improve safety and spatial intuition during programming, but the resulting behaviours often remain closely tied to specific demonstrated trajectories, task geometries, or controlled environments. This is not a limitation unique to those works; it is a broader challenge for demonstration-based robot teaching, including the system developed in this dissertation.

Other integrated systems use **XR** and **DTs** primarily to support collaboration, supervision, or coordination. Perez [48] and Semeraro [49] present systems where immersive interfaces or digital representations support coordination in collaborative or multi-robot settings, allowing operators to remain separated from hazardous environments while still supervising complex tasks. User studies in this area suggest that these interfaces can improve subjective acceptance and coordination in multi-user or multi-robot scenarios. Karpichev [50] also argues that **XR**-enhanced human-in-the-loop manipulation should not only act as a front end, but should mediate the learning process by combining human judgement, machine learning, and **DT** simulation. These works show the value of immersive and digital representations for situational awareness, coordination, and safety. However, their main focus is usually not the full learning pipeline from demonstration recording to reusable policy training and execution. In comparison, the present dissertation is less focused on multi-user or multi-robot coordination and more focused on connecting immersive teaching, simulation-based data generation, **BC** policy learning, and task execution within one controlled workflow.

A closely related direction uses immersive teleoperation to improve the collection of demonstrations for learning. HiLoTEL [38] and Haptic-ACT [39] show that combining **XR** interfaces with improved demonstration capture can increase success rates, reduce operator fatigue, and improve safety. These systems also show that haptic feedback and immersive supervision can improve the quality of the interaction and allow intervention during execution. They are close to the logic of this dissertation because **XR** is used not only for visualization or waypoint defi-

dition, but also as a way to obtain demonstrations that can later support learning. At the same time, they often depend on specific hardware configurations, are commonly evaluated on limited manipulation tasks, and still face challenges related to latency, bandwidth constraints, irreversible actions, and generalization beyond the demonstrated cases [38, 39]. These issues are relevant to the present work as well: the proposed system also depends on the quality of the collected demonstrations and is evaluated in simulation with controlled scenes, known objects, and no real-world perception loop.

End-user programming and repair systems address a different aspect of the same problem: making robot behaviour easier to specify, inspect, and modify by non-specialists. MARCER [35], for example, combines natural-language interfaces with XR visualization, allowing non-experts to specify trigger-action rules using speech and in-situ holograms. This can lower the programming barrier, but the resulting behaviour depends on predefined primitives, symbolic task structures, environment tracking, and the correctness of language interpretation. ARConstrained-LfD [51] demonstrates how XR-based visualization and constraint editing can support long-term skill maintenance and in-situ repair of learned behaviours, emphasizing transparency and user-guided adaptation. These works address accessibility and adaptation in important ways, but their mechanisms are different from training reusable manipulation policies from demonstration datasets. They are therefore complementary to the direction followed in this dissertation rather than direct replacements for it.

Table 2.1: Classification of references by main research topics

Reference	IL	RL	DT	XR	HRI	Motion Planning
[29]	X					X
[28]	X					X
[32]	X					X
[30]	X	X				X
[31]	X		X	X		X
[33]	X	X	X			X
[6]	X			X		
[4]				X	X	
[12]					X	
[9]			X	X	X	
[34]	X		X	X		
[35]	X			X		
[36]	X			X		
[37]				X		
[8]	X		X	X		
[7]	X		X	X		
[47]	X		X	X		
[48]			X	X	X	
[49]					X	
[50]				X	X	
[40]			X	X		
[41]			X			X
[13]			X		X	
[42]			X			
[44]	X		X	X		X
[10]	X			X		
[45]			X	X		
[43]	X	X	X	X		
[38]	X		X	X		
[39]	X			X		
[52]		X				X
[14]		X	X			
[53]			X	X		
[46]	X		X	X		
[51]	X			X		
[54]		X				X

Table 2.1 summarizes the works reviewed in this chapter by classifying them according to

their main research topics. The table highlights that many works combine two or three relevant dimensions, such as **XR** with **DTs**, **XR** with **IL**, or **DTs** with learning and simulation. However, fewer works address the full chain from intuitive demonstration collection to dataset expansion, policy learning, and task execution within a single workflow. The goal of the table is not to rank the works, but to make visible which combinations of topics are common and where integration gaps remain.

This comparison clarifies the position of the present dissertation. The proposed work does not claim to solve all open problems in integrated **XR**-based robot teaching. In particular, it does not implement continuous monitoring of a real environment, automatic re-planning around unforeseen obstacles, real-time perception of unknown objects, or deployment on a physical robot. The evaluation is performed in simulation, with controlled scenes and known task objects. These constraints are shared with many proof-of-concept systems in the literature and should be understood as part of the current scope of the work.

Within this scope, the dissertation addresses a specific integration gap. It connects immersive demonstration, a simulated digital task environment, synthetic dataset expansion, **BC**-based policy learning, and modular task execution in one controlled workflow. In doing so, it differs from works focused mainly on spatial trajectory definition, waypoint programming, teleoperation hardware, collaborative coordination, physical **DT** synchronization, or symbolic end-user programming. The remaining challenges identified in the literature, especially dynamic perception, real-world deployment, safety validation, robust adaptation to unseen environments, and long-horizon manipulation, are therefore treated as limitations and future research directions rather than as solved components of the proposed system.

2.2.6 Findings

The classification presented in Table 2.1 provides a high-level view of how the reviewed literature is distributed across the main topics relevant to this dissertation. The table should be interpreted as an overview rather than as a precise measure of each paper’s contribution. An “X” indicates that a topic is present in a work, but it does not show whether that topic is central or peripheral, whether the system was evaluated in simulation or on physical hardware, whether the learned behaviours are reusable, or whether the system supports dataset augmentation and modular execution. For this reason, the table is used to identify broad patterns and gaps, not to rank the reviewed works.

One visible pattern is that **LfD**, **IL**, and Programming by Demonstration (**PbD**) are strongly represented in the literature. This confirms that demonstration-based robot teaching remains one of the main strategies for reducing the need for traditional robot programming. However, demonstrated trajectories usually require additional processing before they can be executed safely or reused in different task conditions. Depending on the system, this may involve trajectory smoothing, constraint handling, collision checking, planning, dataset expansion, or policy learning. This is directly relevant to the present dissertation, because the proposed workflow does not treat demonstrations as final robot programs, but as data that can be structured, augmented, and used to train manipulation policies.

A second pattern is that **XR** appears frequently, but with different roles across the reviewed works. Some systems use **XR** mainly for visualization and feedback, others for waypoint or rule-based programming, others for teleoperation, and others for demonstration collection. These roles have different implications. Visualization can improve task understanding, but may not define executable behaviour. Waypoint-based programming can reduce the abstraction of robot programming, but may still produce rigid trajectories. Teleoperation can make demonstration more natural, but depends on tracking accuracy, latency, and operator consistency. Demonstration-collection interfaces are closer to the objective of this dissertation, but their usefulness depends on the quality and coverage of the recorded data.

A third pattern concerns **DTs** and simulation. Many works use digital representations to support visualization, validation, monitoring, or safer testing. However, the meaning of **DT** varies substantially across the literature. Some works use the term for synchronized physical-virtual systems, while others use it for simulation-oriented digital environments. This distinction is important for the present dissertation, which uses the **DT** concept in a simulation-oriented sense. The contribution is therefore not a fully synchronized industrial **DT**, but a controlled digital task environment for teaching, data generation, policy execution, and evaluation.

RL also appears in several reviewed works, especially where simulation is used for policy refinement, adaptation, or sim-to-real transfer. In this dissertation, however, **RL** is not the main learning method. The central learning mechanism is **IL**, more specifically **BC**. Therefore, **RL** is treated as important background and as a related direction in robot learning, but not as the core methodological contribution of the work.

The reviewed literature also highlights the importance of **HRI** and accessibility. Many **XR**-based systems are motivated by the goal of reducing programming effort, improving task understanding, and making robot programming more accessible to non-specialists. This motivation is shared by the present dissertation. However, the dissertation does not perform a detailed user study on workload, acceptance, collaborative safety, or ergonomics. Its main contribution is technical: the integration of immersive teaching, simulated task representation, dataset generation, **BC**-based policy learning, and modular task execution in a controlled simulation workflow.

The comparison also clarifies the limits of the proposed work. The objective is not to solve all remaining problems in **XR**-based robot teaching, such as dynamic perception, physical deployment, real-time **DT** synchronization, robust adaptation to unknown environments, or long-horizon manipulation. Instead, the dissertation focuses on a narrower contribution: connecting immersive demonstration collection, simulation-based dataset expansion, **BC** policy training, and modular policy execution in a simulated digital environment. Within this scope, the system aims to improve adaptation to controlled variations in object pose and scene configuration within simulation, while treating broader real-world robustness as a limitation and future research direction.

Overall, the reviewed integrated systems show that **XR** and **DTs** can make robot teaching safer, more intuitive, and more transparent. However, most systems emphasize either interaction, visualization, teleoperation, or task specification, rather than a complete workflow that transforms immersive demonstrations into augmented datasets, trains reusable manipulation policies, and ex-

ecutes them modularly in a simulated **DT** environment. This gap motivates the workflow proposed in this dissertation.

2.3 Chapter Summary

This chapter introduced the main technical concepts and related work that support the proposed robot-teaching system. The background section presented **RL**, imitation learning, **BC**, **DTs**, and **XR** as the foundations for learning from demonstrations, training safely in simulation, and providing intuitive interaction interfaces.

The literature review showed that current research already explores many of these components individually, but that complete integration remains limited. The classification analysis showed that **XR** improves interaction, **DTs** support simulation and validation, and **LfD/IL** methods reduce the need for explicit programming. However, these elements are not always combined into reusable workflows that support learning, controlled adaptation, and accessible task teaching at the same time.

Based on these findings, this dissertation focuses on an **XR**-supported robot-teaching workflow that combines operator demonstrations, subtask decomposition, simulation-based data generation, **BC**-based policy learning, and modular task orchestration. The goal is to provide a teaching process that is more accessible to non-specialists and less rigid than traditional trajectory replay.

Chapter 3

Methodology

This chapter presents the methodology followed in the development and evaluation of the proposed system for teaching robot manipulation tasks using XR. The chapter begins by defining the main constraints and assumptions that shape the methodology. It then presents the conceptual system design, including the main modules of the proposed workflow and the information flow between them. Afterward, the test scenarios and operating conditions used to assess the system are described, followed by the main methodological limitations of the work.

3.1 Constraints and Assumptions

This dissertation is developed under a set of constraints, assumptions, and design decisions that define the scope of the proposed methodology. Constraints refer to the conditions that bound the work, assumptions describe the conditions under which the methodology is expected to operate, and design decisions correspond to architectural choices made to achieve the objectives of the dissertation. Together, these elements establish the conditions under which the proposed workflow is developed and evaluated.

The first major methodological decision is that the complete system is developed and evaluated in simulation. This choice allows the teaching, training, and execution workflow to be assessed under controlled and repeatable conditions, focusing the evaluation on the proposed methodology rather than on hardware-specific factors. Within this environment, the methodology evaluates whether demonstrations can be collected through the XR interface, whether the collected demonstrations can be augmented to increase dataset diversity, whether behavioural cloning policies can be trained from the resulting demonstrations, and whether these policies can reproduce manipulation tasks under varying simulated conditions. The simulation environment also enables the execution and orchestration of learned manipulation policies within a controlled workflow, providing a consistent environment for evaluating the proposed approach.

The XR component assumes the use of a head-mounted display as the user interaction device. Through this interface, the user is able to visualize the robot and the workspace, perceive the position of objects and task-related elements within the virtual environment, and interact with

the scene during the teaching process. The **HMD** is also used to support robot teleoperation, where user inputs are transformed into robot actions to guide the end-effector and generate task demonstrations. This design decision was adopted because an **HMD** provides spatial information, immersive visualization, and interaction capabilities that can reduce the effort required from non-expert users when compared with conventional robot programming approaches. The methodology assumes that this interaction provides sufficient information to generate demonstrations suitable for training, while the detailed implementation of the interface and its technical specifications are addressed in Chapter 4.

The **DT** is used as the simulated execution and visualization environment for the proposed workflow. In this dissertation, it is not synchronized with a physical robotic platform. Instead, it provides a virtual representation that can be used to inspect the scene, manage task execution, and run previously trained manipulation tasks. This design decision was adopted because the dissertation focuses on the teaching, training, and execution workflow rather than on real-time synchronization with physical hardware. Therefore, the **DT** supports the objective of providing a reusable environment where trained policies can be selected, executed, and evaluated after the teaching stage.

The proposed methodology assumes that manipulation tasks take place in static and controlled scenes. The objects involved in the tasks are known in advance, and their poses are assumed to be available from the simulation environment. In an application using the real cell, the object poses would need to be obtained via a perception and object detection component; hence, these problems are not treated as separate methodological problems. This assumption was adopted to isolate the learning-from-demonstration problem from the perception problem, allowing the evaluation to focus on whether demonstrations can be transformed into reusable manipulation policies. Similarly, deformable objects and dynamic environments with moving obstacles are not considered. This assumption allows the methodology to focus on robot teaching, dataset generation, **LfD**, and task reproduction rather than on perception, scene reconstruction, or dynamic re-planning.

The methodology further assumes that the demonstration dataset is generated from a limited number of user demonstrations collected through the **XR** teaching interface. To increase the variability of the training data, these demonstrations are augmented synthetically. The objective of this augmentation process is to improve the coverage of the state space within the defined task distribution, while reducing the effort required from the operator by limiting the number of manually collected demonstrations needed. The augmentation process introduces controlled variations in task conditions, such as object poses, target poses, initial robot poses, trajectory variation, and noise. However, synthetic augmentation is designed to increase diversity within the defined task distribution and does not replace demonstrations of fundamentally different strategies or tasks. Therefore, the quality and representativeness of the original demonstrations remain important for the learning process.

The methodology assumes that the generated and augmented demonstrations provide sufficient information to train a model capable of reproducing the target manipulation behaviours. The learning process is based on **IL**, specifically **BC**, where a policy is trained to reproduce the demon-

strated manipulation behaviours from the collected and augmented dataset. The exact model architecture, training configuration, and implementation details are implementation-specific and are therefore described in Chapter 4. At the methodological level, the important assumption is that the collected demonstrations contain enough information to represent the intended task and that synthetic augmentation can improve the diversity of the training data.

The proposed methodology adopts a task-level policy representation, where each trained model corresponds to one reusable manipulation skill. This design decision was adopted to improve modularity and reusability, allowing individual learned behaviours to be combined into larger manipulation operations. In this context, a task refers to the overall manipulation objective taught to the robot, such as pick-and-place, object positioning, or an assembly-like operation. A subtask represents an internal phase of a task, such as approaching an object, grasping, moving, or releasing it. The proposed policies are not explicitly trained to represent individual subtasks; instead, each policy learns the complete demonstrated task behaviour, which may internally contain multiple motion phases.

This distinction allows the system to separate task learning from task orchestration. While a trained policy represents a reusable skill for a specific manipulation task, a larger operation can be obtained by combining multiple task policies into a sequence. The selection and sequencing of policies are handled at the execution level rather than learned by an individual policy. This choice improves modularity, reduces the complexity of each training problem, simplifies validation, and makes the resulting policies easier to reuse.

Overall, these constraints, assumptions, and design decisions define the conditions under which the proposed methodology is evaluated. Within this scope, the simulation environment makes it possible to assess the complete workflow, including demonstration collection through the XR interface, synthetic dataset augmentation, behavioural cloning training, and the execution of learned manipulation policies under controlled and repeatable conditions. The next section discusses the methodological limitations resulting from these choices and how they influence the interpretation of the obtained results.

3.2 Proposed System Design

The proposed system is designed as a modular robot-teaching workflow that combines XR interaction, LfD, and a DT-based simulation environment. The main objective of the design is to allow a non-expert user to teach manipulation tasks through intuitive teleoperation, while keeping the learning and execution components separated from the interaction interface. This separation makes the system easier to extend, since new tasks can be taught, trained, stored, and later selected for execution within the DT.

At a high level, the system is composed of three main modules: the XR teaching module, the learning and training module, and the DT/simulation module. This modular division was selected to keep teaching and execution independent from each other, which makes the workflow easier to debug and extend. The XR teaching module is responsible for allowing the user to visualize

and teleoperate the robot during task demonstration. The learning and training module transforms the collected demonstrations into an augmented dataset and uses this dataset to train task-specific policies through IL. The DT/simulation module provides the environment where trained policies can later be selected, executed, and observed.

The workflow begins with a user wearing a head-mounted display and interacting with the robot through an XR interface. During this stage, the user sees the robot and the relevant workspace elements, and teleoperates the robot to demonstrate a predefined manipulation task. The task definition, including the relevant objects, scene elements, and success conditions, is assumed to exist before the demonstration stage. This design choice ensures that the success criteria, relevant objects, and recorded dataset fields are already defined before data collection begins. It also avoids requiring the user to specify the logical structure of the task during demonstration. The role of the user is therefore not to program the task through code, but to provide examples of how the robot should perform the task.

During each demonstration, the system records the information required to describe the robot behaviour and the state of the task environment. This recorded information is composed by the actions applied at each step (which are the deltas of the joints between the previous step and the current one), the poses of the relevant objects, the pose of the robot end effector, and the robot joint positions and velocities. These recorded demonstrations form the initial dataset for the learning process. These data fields were selected because they capture both the robot state and the task-relevant scene state, providing the information needed for a policy to reproduce the demonstrated behaviour under varied conditions.

The initial demonstration dataset is then augmented synthetically to increase the diversity of training examples. The augmentation process introduces variations in object poses, target poses, initial robot poses, trajectory execution, and noise. The purpose of this stage is to improve the coverage of the state space within the defined task distribution, reduce the dependence on a large number of manually collected demonstrations, lower the effort required from the operator, and provide a more time-efficient way to obtain diverse training data. The augmented dataset preserves the same type of information as the original demonstrations, but represents additional variations of the taught task within the defined task distribution. These augmented demonstrations are intended to improve coverage around the original task distribution, but they do not represent fundamentally new manipulation strategies or unseen object categories.

The learning and training module uses the augmented dataset to train a policy for the demonstrated task. In the proposed design, each trained model corresponds to one reusable manipulation skill, where a skill represents a policy capable of executing a specific task under the conditions represented in the training data. In this context, a task refers to the overall manipulation objective taught to the robot, such as pick-and-place, object positioning, or an assembly-like operation. A subtask represents an internal phase of a task, such as approaching an object, grasping, moving, or releasing it. The proposed policies are not explicitly trained to represent individual subtasks; instead, each policy learns the complete demonstrated task behaviour, which may internally contain multiple motion phases.

This distinction allows the system to separate task learning from task orchestration. While a trained policy represents a reusable skill for a specific manipulation task, a larger operation can be obtained by combining multiple task policies into a sequence. For example, a complex operation may require several learned skills to be executed in a predefined order, where the selection and sequencing of policies are handled at the execution level rather than learned by an individual policy. This choice improves modularity, reduces the complexity of each training problem, simplifies validation, and makes the resulting policies easier to reuse.

This design choice is important to ensure the modularity and ease of use of the system. For example, a more complex operation such as disassembling a motor and placing its components into a container can be interpreted as a sequence of simpler manipulation tasks. Each component removal or placement operation can be represented as a learned task policy, and the User Interface (UI) in the DT can then be used to select and execute the appropriate policies in sequence. In this way, the system aims to support reusable task skills and policy orchestration rather than one rigid program for a single complete operation. This avoids the need to train one large policy for an entire long-horizon operation, which would be harder to train, debug, reuse, and validate.

After training, the resulting policy is made available to the DT/simulation module. The DT contains a representation of the robot cell and provides a UI through which trained tasks can be selected and executed. Using the DT as the execution environment provides greater control over the scene and creates a clearer path toward future transfer to a physical robot cell. When a task is selected, the corresponding policy is loaded and run in simulation. The simulated scene is updated during execution, allowing the user to observe the behaviour of the robot and the effect of the selected task on the environment.

3.2.1 Conceptual Architecture

The conceptual architecture of the proposed system is shown in Figure 3.1. The architecture is organized around the separation between task teaching, policy training, and task execution. This separation allows the XR interface to focus on demonstration collection, while the DT focuses on scene representation and execution of previously trained tasks. This prevents the teaching interface from becoming responsible for task orchestration and helps keep each component focused on a specific role.

The XR teaching module is the entry point of the workflow. It provides the user with a spatial interface for observing the robot and controlling it through teleoperation. The purpose of this module is to make the demonstration process intuitive by allowing the user to interact with the robot in a visual and task-oriented way. Instead of specifying coordinates, commands, or trajectories manually, the user demonstrates the desired behaviour by moving the robot in the virtual environment. This choice supports the objective of intuitive programming, since the user provides examples of the desired motion rather than constructing a robot program explicitly.

The learning and training module receives the demonstration data produced during the teaching stage. This module is responsible for organizing the recorded demonstrations, generating augmented variants, and training a task policy through IL. The output of this module is a trained

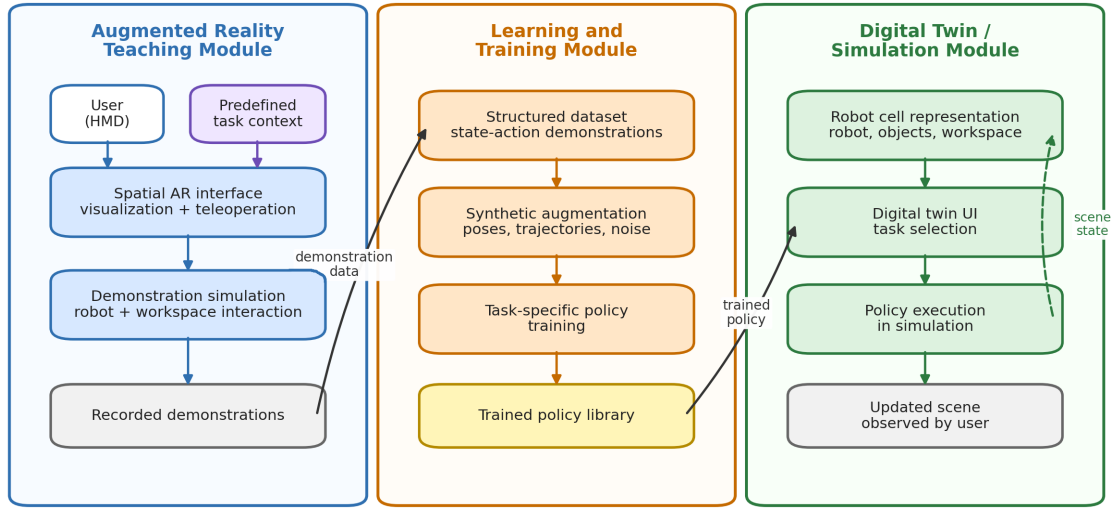


Figure 3.1: Conceptual architecture of the proposed robot-teaching system.

model associated with a specific manipulation task. Once trained, the model becomes part of the set of available task policies that can be executed from the **DT** interface.

The **DT**/simulation module represents the robot cell, including the robot, the relevant objects, and the task environment. In the proposed methodology, the **DT** is not used as the teaching interface itself. Instead, it acts as the environment where trained policies can be selected and executed. The **UI** associated with the **DT** allows the user to choose a trained task, run the corresponding policy, and observe the updated state of the simulated scene.

The proposed task-teaching workflow, illustrated in Figure 3.2, consists of three main stages: demonstration, learning and training, and policy execution. During the demonstration stage, the operator performs the task through the **XR** teaching interface, which provides the required input to teleoperate the robot within the demonstration simulation. The resulting robot and environment states are recorded and provided to the learning and training module. In this stage, the recorded demonstrations are organized, augmented, and used to train a task policy through **IL**. The resulting trained model is then integrated into the set of available task policies. During the execution stage, the user selects a trained task from the **DT** interface, which loads and executes the corresponding policy within the simulated robot cell, allowing the updated state of the environment to be observed.

3.2.2 System Components and Information Flow

The information flow between the system components describes how the data generated during task demonstration is progressively transformed into a trained task policy and later used for task execution. While the previous section described the role and organization of each system component, this section focuses on the information exchanged between them and how this information is processed throughout the workflow.

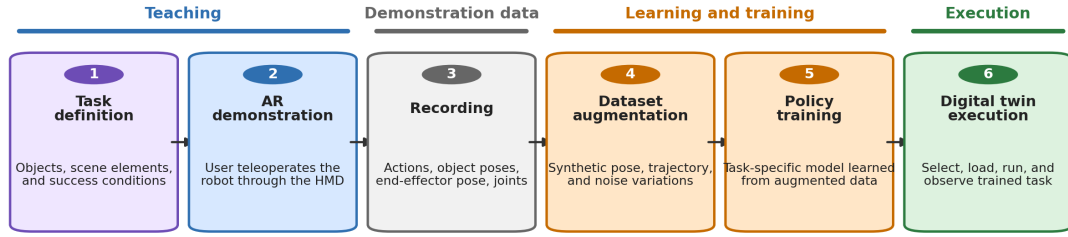


Figure 3.2: Workflow from XR demonstration to trained policy execution in the DT.

First, the **XR** teaching interface sends the operator's hand or control input to the demonstration simulation, where this input is used to teleoperate the robot. The input provided by the user is converted into robot commands within the simulation environment, allowing the operator to guide the robot through the execution of the desired manipulation task. During this process, the simulation records the resulting robot and environment states during the task execution. This recorded information includes the actions applied at each step, relevant object poses, end-effector pose, joint positions, and joint velocities. These elements form the demonstration trajectory, representing the relationship between the task state and the robot actions performed by the operator.

The recorded demonstrations are then passed to the dataset generation stage. At this stage, the original demonstrations are converted into a structured dataset suitable for training. The same data structure is preserved during augmentation, so that both original and synthetic demonstrations contain equivalent types of information. Keeping the data structure consistent simplifies the transition between demonstration recording, dataset augmentation, and policy training. The augmented dataset therefore contains a broader set of task examples while remaining compatible with the learning module.

The learning and training module receives the augmented dataset and uses it to train a task policy through **IL**. The policy uses the recorded state and action information to learn how the demonstrated manipulation behaviour should be reproduced. In this process, the relationship between the observed task state and the corresponding robot actions is used to estimate the behaviour required to perform the task. Once training is complete, the policy is stored as a task-specific model that can be made available to the **DT** environment.

During execution, the **DT UI** allows the user to select one of the trained task policies. The selected policy is then executed within the simulated robot cell, where it receives the current state of the robot and environment and generates the corresponding robot actions. As the policy executes, these actions are applied to the simulated robot, updating the robot motion and scene state in the **DT**. This creates a closed execution workflow in which taught tasks can be reused without requiring the user to repeat the teaching process each time. This is important for the intended modular workflow, where a learned task policy should remain available as a reusable component after the initial teaching and training stages are completed.

This design supports a modular and reusable approach to robot programming. The user teaches simple manipulation behaviours through **XR** teleoperation, the system transforms these demonstrations into trained task policies, and the **DT** provides an interface for selecting and executing the learned tasks. As a result, the proposed system separates demonstration, learning, and execution while maintaining a coherent workflow for intuitive robot programming.

3.3 Methodological Limitations

The methodology adopted in this dissertation has several limitations that define the scope of the conclusions that can be drawn from the proposed system. These limitations result from the assumptions and design decisions established in Section 3.1 and are mainly related to the simulation-based evaluation, the controlled structure of the manipulation tasks, the assumptions regarding the environment, and the dependence of the learning process on the quality of the generated demonstrations and datasets.

The first limitation is related to the use of simulation as the evaluation environment. Although simulation provides a controlled and repeatable environment for assessing the proposed workflow, the obtained results only demonstrate the feasibility of the teaching, learning, and execution pipeline under simulated conditions. Factors associated with physical robotic deployment, such as hardware calibration errors, actuator limitations, sensor noise, communication delays, mechanical tolerances, and real-world contact dynamics, are not directly evaluated. Therefore, the results cannot be interpreted as evidence of successful deployment on physical robotic platforms, and further validation would be required to assess sim-to-real transfer and real-world performance.

Another limitation concerns the use of the **DT** as a simulated execution environment rather than as a real-time synchronized representation of a physical robotic cell. The proposed methodology demonstrates how a **DT** can support the selection, execution, and observation of trained policies within a controlled environment. However, the results do not provide conclusions regarding real-time physical-virtual synchronization, continuous updates from physical sensors, or operation in an industrial **DT** scenario. These aspects would require additional integration with physical hardware and real-time data sources.

The manipulation tasks considered in this dissertation are intentionally limited to simple and reusable task skills. This design choice enables the evaluation of modular task policies and allows learned behaviours to be reused as components of larger operations. However, the conclusions obtained from these tasks are limited to the types of behaviours represented in the training data. A policy trained for a specific manipulation task is not expected to automatically generalize to different object types, substantially different task conditions, or fundamentally different manipulation strategies without additional demonstrations and training.

The methodology also assumes static and controlled scenes, where the objects involved in the tasks are known in advance and their poses are available from the simulation environment. As a consequence, the evaluation does not provide conclusions regarding the autonomy of the system in scenarios where perception, object detection, pose estimation, or scene reconstruction

are required. Similarly, dynamic workspaces, moving obstacles, deformable objects, and complex human-robot interaction scenarios are not assessed. These aspects represent additional challenges that would need to be considered for deployment in less structured environments.

The quality and representativeness of the demonstrations constitute another limitation of the proposed approach. Since the learning process depends on demonstrations collected through **XR** teleoperation, the resulting policies are influenced by the quality of the user demonstrations, the accuracy of the interaction interface, and the ability of the user to perform consistent task executions. Demonstrations that do not adequately represent the desired behaviour may negatively affect the resulting policy. Although synthetic augmentation is used to increase the diversity of the training dataset, it cannot fully compensate for missing task strategies, different interaction patterns, or behaviours that are not represented in the original demonstrations.

The dataset augmentation process also limits the interpretation of the generalization capabilities of the trained policies. While variations in object pose, target pose, initial robot pose, trajectory execution, and noise can improve the coverage of the defined task distribution, they do not represent completely new tasks or unseen manipulation strategies. Therefore, the obtained results demonstrate the ability of the methodology to improve policy learning within the defined task distribution, but they do not demonstrate unrestricted generalization beyond the conditions represented in the training data.

Finally, although the proposed system aims to reduce the complexity of robot programming for non-expert users, conclusions regarding usability, workload, and industrial adoption must be interpreted according to the scope of the performed evaluation. The methodology establishes a workflow intended to simplify task teaching through **XR** interaction and demonstration-based learning, but a complete assessment of user acceptance and industrial applicability would require dedicated user studies involving representative operators and realistic industrial scenarios.

Overall, these limitations define the scope of the conclusions that can be drawn from this dissertation. The proposed methodology demonstrates that **XR** teleoperation, demonstration-based learning, dataset augmentation, and **DT**-based execution can be combined into a coherent simulation-based robot-teaching workflow. However, further work is required to validate the approach on physical robots, integrate perception and dynamic scene understanding, evaluate real-time physical-virtual synchronization, and assess the robustness of the system in more complex industrial environments.

3.4 Chapter Summary

This chapter established the methodological foundation of the proposed robot-teaching system by defining the assumptions, design decisions, and evaluation conditions that guide the development of the proposed approach. The methodology is built around the idea of transforming human demonstrations into reusable robot manipulation skills through a workflow that combines **XR** interaction, learning from demonstration, synthetic data generation, and **DT**-based execution. Rather

than focusing on explicit robot programming, the proposed approach aims to allow users to specify desired behaviours through examples and subsequently transform these examples into policies that can be reused within the simulated robot cell.

The decision to develop and evaluate the system in simulation provides the controlled environment required to assess this complete workflow. By avoiding the additional complexity introduced by physical hardware at this stage, the methodology can focus on the interaction between demonstration collection, dataset generation, policy learning, and task execution. This allows the different components of the proposed system to be evaluated under repeatable conditions, while also establishing a foundation for future validation on physical robotic platforms.

Within this workflow, **XR** plays a central role by changing how the robot is taught. Instead of requiring the user to manually define trajectories or robot instructions, the user can demonstrate the desired behaviour through direct interaction with the virtual environment. This choice supports the objective of reducing the programming barrier for non-expert users, allowing robot tasks to be specified through demonstrations rather than through conventional programming approaches.

This demonstration-based approach, however, introduces a related challenge: obtaining sufficient, diverse, and high-quality training data. For this reason, the methodology incorporates synthetic dataset generation and augmentation to extend the information provided by a limited number of user demonstrations. By introducing controlled variations within the task distribution, the training process can benefit from a broader set of examples while reducing the effort required from the operator during the teaching stage. At the same time, the quality of the resulting policies remains dependent on the original demonstrations, since augmentation can improve data diversity but cannot replace missing behaviours or fundamentally different task strategies. In this context, the use of an **HMD** supports the collection of richer demonstrations by providing a more intuitive and spatial interaction mechanism compared with conventional input methods, potentially improving the quality of the data collected during the teaching stage.

The organization of the learning process around task-level policies further supports the objective of creating reusable manipulation skills. Instead of training a single model for complete long-horizon operations, the methodology allows individual tasks to be learned independently and later combined through policy orchestration. This approach improves modularity and facilitates the construction of more complex operations from previously learned behaviours, while keeping each learning problem focused on a specific manipulation objective.

The final stage of the workflow is supported by the **DT**, which provides the environment where trained policies can be selected, executed, and observed. By separating the teaching interface from the execution environment, the methodology maintains a clear distinction between generating new skills and reusing existing ones. This separation supports a more modular workflow, where learned policies remain available after training and can be integrated into future task executions.

Overall, the methodology defined in this chapter establishes a structured approach for moving from human demonstration to reusable robot manipulation policies. The combination of **XR**-based teaching, synthetic data generation, task-level learning, and **DT**-based execution provides the basis for the implementation and evaluation of the proposed system. At the same time, the

assumptions introduced throughout the methodology define the scope of the conclusions that can be drawn, particularly regarding physical deployment, perception, and operation in less structured environments. The following chapters build on this foundation by describing the implementation of the proposed workflow and the evaluation of its performance under the defined experimental conditions.

Chapter 4

Implementation

This chapter describes how the methodology proposed in Chapter 3 was implemented as a simulation-based robot-teaching system. The goal of the implementation was not only to connect existing tools, but to build a modular architecture for teaching robotic manipulation tasks through XR, learning from demonstration, generating additional training data, and executing trained behaviours in a DT.

The implementation was guided by the main objective of the dissertation: to make robot teaching more accessible while preserving a reusable and adaptable representation of learned manipulation tasks. This required an implementation in which the user demonstration, the training dataset, the learned policy, and the runtime execution environment remained connected through consistent task information. For this reason, the system was designed around three separated stages. The first stage allows a user to demonstrate a task through XR teleoperation and store the resulting trajectories in a structured dataset format. The second stage uses the recorded demonstrations to generate a larger training dataset and trains a BC policy. The third stage executes trained policies in a DT environment, where policies can be filtered according to the current scene and combined into longer task sequences through an orchestrator embedded in the DT.

A central concern in the implementation is that each tool or artifact should serve a specific architectural requirement, rather than appearing as an isolated software choice. Isaac Lab¹ was selected as it provides a common task definition shared across demonstration recording, synthetic data generation, training, evaluation, and execution. HDF5² was adopted as the storage format for demonstration datasets, as the learning pipeline required a structured approach for storing complete trajectories with observations, actions, and scene states. Isaac Lab Mimic³ was integrated to address the scalability limitations associated with collecting a large number of manual XR demonstrations. Robomimic [55] was incorporated to provide the required framework for training imitation-learning policies from demonstration datasets. Finally, the Isaac Sim⁴ extension and the

¹<https://isaac-sim.github.io/IsaacLab/main/index.html>

²<https://www.hdfgroup.org/solutions/hdf5/>

³https://isaac-sim.github.io/IsaacLab/main/source/overview/imitation-learning/teleop_imitation.html

⁴<https://docs.isaacsim.omniverse.nvidia.com/latest/index.html>

embedded policy orchestrator were implemented to support runtime execution, scene-dependent policy selection, and task sequencing inside the **DT**.

The chapter follows this implementation logic by first presenting the design considerations that shaped the system and then detailing the role of each component in the complete workflow. Section 4.1 explains how the methodology was translated into concrete implementation considerations, while Section 4.2 presents the overall architecture and the artifacts exchanged between the main components. Since the task definition is the element that connects demonstration recording, dataset generation, training, and execution, Section 4.3 then describes how manipulation tasks were formalized in Isaac Lab.

The following sections describe the main stages of the pipeline. Section 4.4 presents the **XR** teleoperation and demonstration recording process, and Section 4.5 explains how the recorded demonstrations are annotated and expanded through synthetic demonstration generation. Section 4.6 then describes how the augmented dataset is used to train behaviour cloning policies. After the training stage, Section 4.7 presents the **DT** extension used for runtime execution, and Section 4.8 describes how the policy orchestrator filters and sequences trained policies according to the current scene. Finally, Section 4.9 summarizes the complete workflow from task definition to policy execution, and Section 4.10 closes the chapter by linking the implementation to the evaluation presented in Chapter 5.

4.1 From Methodology to Implementation

The methodology proposed in Chapter 3 requires an implementation capable of connecting user demonstrations, task learning, dataset generation, and digital-twin-based execution. Each decision was motivated by the need to preserve consistency between the different stages of the pipeline while keeping the system modular enough to support reusable manipulation behaviours.

A central requirement of the implementation was to keep each manipulation task consistent across the complete pipeline. The task demonstrated by the user had to be the same task used for synthetic data generation, policy training, evaluation, and runtime execution. If these stages were described independently, differences in observations, actions, object definitions, reset conditions, or success criteria could lead to inconsistencies between the dataset and the execution environment. For this reason, each task was implemented once in Isaac Lab and reused throughout the pipeline. This design reduces duplicated task logic and ensures that the same scene, robot, gripper, objects, observations, actions, events, reset conditions, and success condition are shared across the implementation.

With this common task representation in place, the next implementation concern touches on how the robot should learn from the task. Since the goal of the dissertation is to make robot teaching more accessible, the system was designed to allow the user to demonstrate the task by doing it instead of programming. This naturally leads to a **LfD** paradigm, where robot behaviours

are learned directly from user demonstrations rather than being manually programmed. A **RL** approach would require defining reward functions and training through exploration, introducing additional complexity and shifting the focus away from learning directly from user demonstrations. Therefore, the implemented methodology uses **IL**, more specifically **BC**. This choice allows the policy to learn a direct mapping from observations to actions using recorded demonstrations as supervised training data.

However, relying only on manually recorded **XR** demonstrations would make the teaching process too demanding as it requires a large number of demonstrations. A learning system based exclusively on human demonstrations would require the user to repeat a task execution many times in order to introduce variability in the demonstrations such as object-pose variation. To avoid this limitation, the implementation separates human demonstration from training dataset generation. This way, a small set of human demonstrations is used as a structural template for synthetic demonstration generation, preserving the task structure provided by the user while increasing the amount and diversity of data available for training.

Once the training dataset is produced and the policy is trained, the system must also support reuse. The user should not need to repeat the demonstration recording, dataset generation, or training stages in order to execute the same behaviour again. For this reason, training is separated from runtime execution, and each learned behaviour is stored as a task-specific policy checkpoint. These checkpoints can later be loaded by the **DT** environment as reusable behaviour modules, making trained policies persistent. Focusing on the scope of this dissertation, the **DT** environment is a simulated representation that is not synchronized with a real cell, however, we will refer to it as the **DT** of the cell since the real robot integration would be a modular step posterior to this implementation that would not compromise the developed work.

Finally, the runtime stage requires more than simply loading one trained policy at a time. A single trained policy is useful for executing one manipulation behaviour, but it is not sufficient for composing longer task sequences involving different objects or task steps. Training a new policy for every possible long-horizon sequence would reduce modularity and would require additional demonstrations for each complete task. Therefore, it was implemented a policy orchestrator inside the **DT** to connect the current scene state with the set of available trained policies, filter executable policies, and execute selected policies in sequence. Together, these design considerations define the main structure of the implemented system: a shared task representation, a demonstration-based learning process, a dataset generation stage, a checkpoint-based policy representation, and a runtime orchestration layer.

The Robomimic training configuration was defined for the **BC** policies used in the system. Finally, the **DT** extension, its **UI**, the YAML configuration files used to define and load task policies, and the policy orchestrator were implemented to support runtime object management, policy filtering, and task sequencing.

4.2 Overall System Architecture

The implemented system is organized around three main implementation components: the **XR** teaching module, the learning and training module, and the **DT** execution environment. Within the **DT** execution environment, a policy orchestrator is implemented as an internal component responsible for selecting and coordinating the execution of trained policies. In this structure, the HDF5 dataset is not treated as a separate module, but as the output artifact produced by the demonstration recording process and consumed by the following learning stages.

These implementation components refine the three conceptual modules defined in Chapter 3. The **XR** teaching module described in the methodology corresponds to the **XR** teaching component, which provides the user interaction layer and records the demonstrations. The trajectories produced during this process are stored in HDF5 format because the following imitation-learning stages require structured trajectory data containing observations, actions, and scene states. The learning and training module is implemented through the synthetic data generation and training component, which uses the recorded demonstrations to generate additional data and train **BC** policies. Finally, the **DT**/simulation module is implemented as the **DT** execution environment, where the policy orchestrator manages policy filtering and task sequencing.

The communication between components is defined through the artifacts exchanged along the pipeline. The training configuration defines how the policy is learned from those datasets, and the resulting checkpoint stores the trained policy for later execution. At runtime, the YAML configuration file connects the **DT** interface to the available objects and policy folders, while the Universal Scene Description (**USD**)⁵ assets provide the scene elements to use in simulation.

In this architecture, the collected dataset is stored in HDF5 format, which preserves the generated trajectories in a structure compatible with the following learning stages. From this point, the synthetic data generation and training component annotates and, using the original demonstrations as a baseline, creates and uses the augmented dataset to train a **BC** policy. Once training is complete, the trained policy checkpoint becomes available to the **DT** where the execution environment loads the scene representation, objects, and trained policies, while the policy orchestrator uses the current scene state to determine which policies can be executed and allows the user to choose in which order they should run. This flow of information, from user demonstration to HDF5 file, policy training, **DT** execution, and orchestration, is represented in Figure 4.1.

The **XR** device used was a Meta Quest 3, as it provides the interface required for spatial interaction and hand-based teaching in simulation. However, due to the computational requirements of running the physics simulation and rendering the virtual environment, the Meta Quest 3 is not used to execute the simulation itself. Instead, Isaac Sim and Isaac Lab run on a workstation with the required computational resources, and NVIDIA CloudXR⁶ is used to stream the rendered simulation to the **HMD**. This architecture allows the workstation to perform the simulation and physics computation, while the user interacts with the virtual environment through the Meta Quest 3. The

⁵<https://openusd.org/release/index.html>

⁶<https://www.nvidia.com/pt-br/design-visualization/solutions/cloud-xr/>

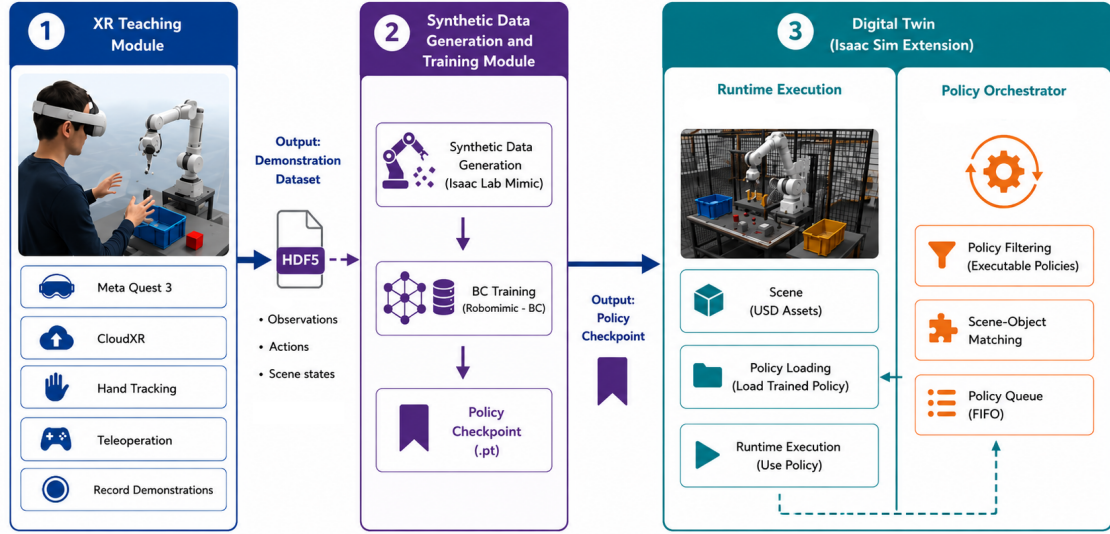


Figure 4.1: Implemented system architecture connecting XR teaching, synthetic data generation, policy training, and DT execution with policy orchestration.

hand-tracking and interaction data collected by the **HMD** are transmitted back to the workstation and used to control the robot during the teleoperation process. All of this information, including the hardware specifications and software versions used in the implementation, is summarized in Table 4.1.

Table 4.1: Computational environment and software versions used in the implementation.

Component	Specification
GPU	NVIDIA GeForce RTX 3080Ti
CPU	Intel i9-11900F
RAM	32 GB
Operating System	Ubuntu 22.04
Isaac Sim	5.1.0
Isaac Lab	2.3.0
Robomimic	0.4.0
CloudXR	6.1.0
Python	3.11.14

The learning workflow is implemented within the ecosystem composed by Isaac Lab, Isaac Lab Mimic, and Robomimic. Isaac Lab defines the task and records the demonstrations in HDF5 format. Isaac Lab Mimic uses those demonstrations for annotation and synthetic generation. IsaacLab then trains the Robomimic **BC** policy from the generated dataset.

After training, execution is handled by an Isaac Sim extension that implements the **DT** runtime. This implementation consideration keeps the runtime interface inside the simulation environment, instead of separating policy execution into an external script disconnected from the scene. The

extension provides the interface for loading objects, selecting configuration files, displaying executable policies, and executing queued task sequences. Within this same **DT** runtime, the policy orchestrator acts as the link between the trained policies and the objects present in the scene.

4.3 Task Definition in Isaac Lab

The task definition is one of the central components of the implementation. Its purpose is to formalize how a robotic manipulation task exists inside the simulated environment and to provide a common structure for recording, generation, training, evaluation, and execution. This common structure is necessary because the same task must remain consistent across all stages of the pipeline.

Isaac Lab was used for this role as it provides a task-based structure where the relevant elements of a manipulation problem can be defined together. The task definition specifies the scene, robot, gripper, manipulated object, target, observations, actions, events, reset conditions, and success condition. These elements determine what the user is supposed to interact with, what the dataset stores, what the policy uses as input, and what the runtime system executes. Importantly, all of these elements are configurable within the Isaac Lab task script, which means that adapting the system to a new robot, object, workspace layout, or success criterion requires changes in a single place rather than across multiple independent components.

For this case, it was created four scripts, one for each object, of a pick and place task. The environment remains the same across these tasks, while the manipulated object changes. This direction was implemented due to the objective lying on evaluating reusable object-specific manipulation behaviours within a controlled simulated cell. Keeping the robot and workspace fixed reduces the development effort required for the different tasks, since the workspace configuration only needs to be defined once and can be reused across multiple manipulation tasks within the same environment. The same structure could also be extended to different robots, scenes, objects, or task variants.

The scene definition specifies the simulated robot cell. In this implementation, the simulated platform consists of a KUKA LBR iiwa manipulator equipped with a Robotiq 2F-85 two-finger gripper. The workspace includes a table, the manipulated object, and the target container. These elements are represented through Isaac Sim assets and are loaded as part of the task environment. The complete simulated cell, including the robot, gripper, workspace, manipulated object, and target container, is illustrated in Figure 4.2.

The robot and gripper definitions specify the simulated manipulation platform and the actuation model used during teleoperation and policy execution. The same robot model is used during demonstration recording, dataset generation, training, and **DT** execution, ensuring that the learned actions remain consistent with the robot configuration defined in the task.

The object definitions specify which part is manipulated in each task. In the implemented cases, the main difference between the four task scripts is the object to be picked and placed. Object poses are part of the recorded state information and are also relevant for the policy input.

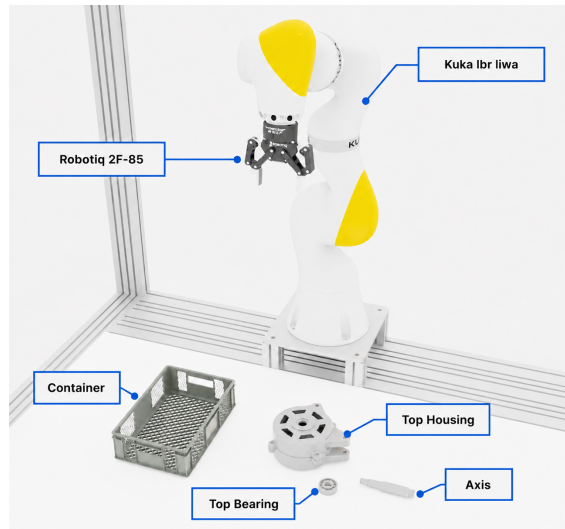


Figure 4.2: Simulation environment for the developed tasks.

Although only object changes were explored in the implemented tasks, the same task-definition structure could support changes to the robot, scene, workspace layout, or target configuration.

The observation definition specifies the information available to the policy. In this implementation, only state-based policies were explored. The observations include the object pose, target pose, end-effector pose, robot joint positions, and robot joint velocities. Although another approach based on image observations could be used, where the policy learns directly from visual information obtained from the environment, the direction taken in this implementation relied on state-based observations to avoid introducing the additional complexity associated with visual perception. Image-based approaches typically require longer training processes and additional considerations related to processing and learning from visual inputs, which were not explored in this dissertation. State-based observations provide a more direct representation of the information needed for the pick-and-place behaviours considered in this dissertation.

The action definition specifies how teleoperation input or policy output is applied to the robot. In the implemented tasks, the policy issues target commands, depending on the observations, to execute the manipulation task. This action representation was used because the demonstrations are collected through end-effector-oriented teleoperation and the policy must reproduce both the arm motion and the grasping/releasing behaviour.

The event and reset definitions determine how the environment is initialized or modified between episodes. These definitions control the initial conditions under which demonstrations are recorded and synthetic dataset generation is performed. Reset conditions also support repeatable evaluation by allowing the task to start from controlled scene configurations. This is particularly important for [II](#), because the distribution of initial conditions affects the trajectories available in the dataset and the situations encountered by the trained policy. The specific reset distributions used in the implemented tasks are configurable within the task script and are discussed in the context of the evaluation presented in [Chapter 5](#).

The success condition defines when the task is considered complete. Since all implemented tasks are pick-and-place tasks, the success condition is triggered when the manipulated object is inside the container within a defined spatial threshold and the gripper is open. During policy execution, this condition determines when a policy can stop and the system can move to the next queued task.

4.4 XR Teleoperation and Human Demonstrations

The purpose of this stage is to allow a user to teach a task through demonstrations rather than through manual coding, teach pendants, or predefined waypoints. In this implementation, the **XR** interface provides the teaching layer, while Isaac Lab provides the simulated task environment and recording mechanism.

The user wears the Meta Quest 3 and connects to the Isaac Lab simulation through NVIDIA CloudXR. This streaming approach allows Isaac Sim and Isaac Lab to run on the workstation, where simulation and physics computation are executed, while the rendered environment is streamed to the **HMD**. The **XR** interface allows the user to visualize and interact with the simulated environment through the **HMD**, either using a fully virtual representation or a passthrough view of the physical surroundings. In this implementation, the passthrough mode was used during teleoperation, providing visibility of the user’s surroundings while interacting with the simulated robot. Since the robot and workspace are entirely simulated, this mode does not provide spatial alignment with a physical robot or environment, but only changes the visual background in which the simulation is displayed.

Hand-tracking input from the **HMD** is mapped to robot control through Isaac Lab’s teleoperation and retargeting components. The retargeting mechanism maps the pose of a configurable part of the operator’s hand (either the wrist or the palm) to the pose of the robot’s end effector. When the operator moves their hand, the inverse kinematics target point updates and the simulated robot end effector follows the same motion. When the operator closes their hand, the gripper closes accordingly. The specific hand link used for retargeting and any additional offset applied to the mapping are configurable within the Isaac Lab task script, making the teleoperation behaviour adjustable to different tasks or operator preferences without modifying the core system.

The contribution of this stage is therefore the integration of Isaac Lab’s **XR** teleoperation capability into a demonstration-based learning workflow. The task environments were created and configured so that the user-generated motions could be recorded with the observations, actions, and scene states required for **IL**.

If a demonstration does not go as intended, the operator can discard it and restart the recording without it being included in the dataset. This capability is important for keeping the demonstration quality consistent, since **BC** learns directly from the provided trajectories and poor demonstrations propagate into the trained policy. The dataset annotation stage described in Section 4.5 also provides an additional opportunity to discard demonstrations that are structurally inconsistent before synthetic generation begins.

The demonstration recording process produces the dataset used by the rest of the learning pipeline. This dataset preserves complete task executions as time-indexed sequences of observations, actions, and timestamps, rather than only storing isolated states or final outcomes. For **IL**, each recorded sample maintains the correspondence between the observations available at a given time step and the action applied by the demonstrator. The recorded data remain directly compatible with the subsequent stages of the workflow, namely dataset annotation, synthetic demonstration generation, and behaviour cloning training.

For this reason, the recording stage required a structured dataset format capable of storing multiple demonstrations, each containing synchronized observations, actions, and scene states. Since Isaac Lab and Robomimic both use HDF5 as their native demonstration format, it was used consistently throughout the pipeline. HDF5 supports hierarchical storage, handles large numerical arrays efficiently, and allows the same dataset structure to be used for recording, generation, and training without conversion steps.

In this implementation, the recorded information includes object pose, target pose, end-effector pose, robot joint positions, robot joint velocities, and gripper commands. These data are sufficient for the **BC** policy to learn a mapping from state observations to actions. The complete demonstration recording workflow, from user interaction through the **XR** interface to HDF5 dataset storage, is illustrated in Figure 4.3.

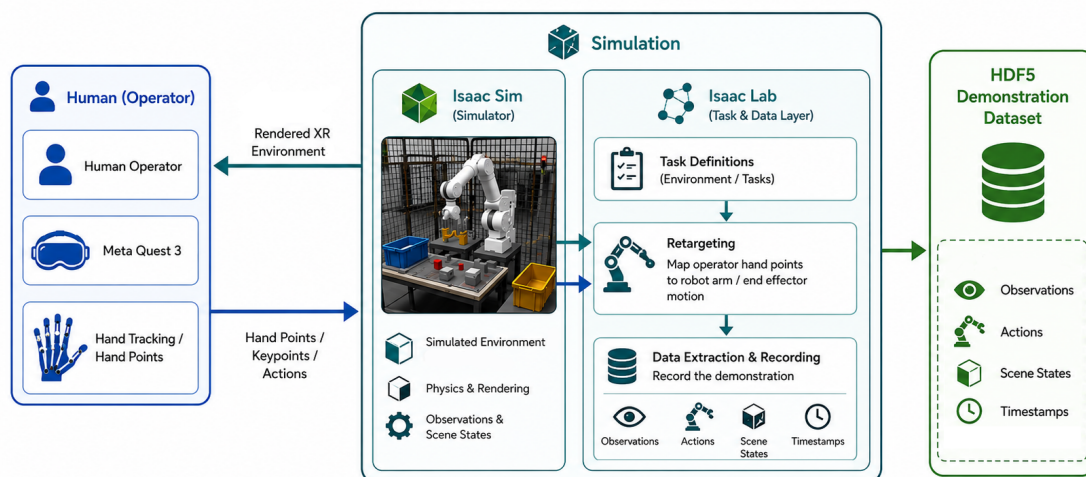


Figure 4.3: XR demonstration recording pipeline using Meta Quest 3, CloudXR, Isaac Lab teleoperation, and HDF5 dataset storage.

This stage has an important consequence for the rest of the system. Since **BC** learns directly from demonstrations, the quality of the recorded demonstrations directly influences the quality of the learned policy.

4.5 Dataset Annotation and Synthetic Demonstration Generation

The dataset annotation and synthetic generation stage was introduced to reduce the amount of manual demonstration required from the user. A purely manual approach would require many XR-recorded demonstrations to cover different object poses and task variations. This would make the system less practical and would conflict with the objective of accessible robot teaching. The implementation therefore uses a small set of human demonstrations as the basis for generating a larger and more varied training dataset, reducing the amount of manual demonstration required from the user.

The input to this stage is the original HDF5 dataset recorded through XR teleoperation. Before generating additional demonstrations, the user demonstrations are annotated and divided into subtasks. This annotation process identifies the internal subtask boundaries of the task by delimiting the different phases of the manipulation behaviour. For the implemented pick-and-place tasks, these subtasks correspond to the main phases of the operation, which are the grasping of the object and its release in the target position. This subtask definition provides the information required by Isaac Lab Mimic to transform and reuse meaningful parts of the original demonstrations during the generation process. The annotated user demonstrations are then used with Isaac Lab Mimic to generate additional demonstrations, applying the original task structure to different task variations. The original XR demonstrations therefore provide the basis for the generated dataset, while Mimic allows this structure to be expanded without requiring the user to manually record every variation.

Like the Isaac Lab task scripts described in Section 4.3, the Mimic configuration is defined as part of the task setup and describes the subtask structure of each manipulation behaviour. This configuration specifies how the task is divided into meaningful segments and provides the information required to identify the transitions between them in the demonstrations. For example, in the implemented pick-and-place tasks, the behaviour is represented through subtasks such as approaching the object, grasping it, transporting it to the target container, and releasing it. Using this subtask representation, Isaac Lab Mimic can identify relevant segments of the original user demonstrations and generate new demonstrations by applying variations while preserving the overall task execution logic. Therefore, when a new manipulation task is introduced, the required adaptation is mainly the definition of its subtask structure and corresponding task information, while the general generation process remains unchanged.

The annotation process identifies meaningful subtask segments inside each demonstration. For the implemented manipulation tasks, these segments correspond to the general phases of a pick-and-place operation, which in this implementation are: grasping the object and releasing it in the container. Although the final trained policy is used as a task-level model, this subtask structure is required during generation because Mimic operates by transforming and recombining annotated demonstration segments. Demonstrations that are structurally inconsistent with the expected phase sequence can be identified and discarded at this stage before generation begins.

After annotation, synthetic demonstrations are generated from the original data. As shown in Figure 4.4, the generated dataset preserves the task structure of the original demonstrations while

introducing variation according to the task configuration. The resulting augmented HDF5 dataset provides a larger set of demonstrations that is then used during behaviour cloning training.

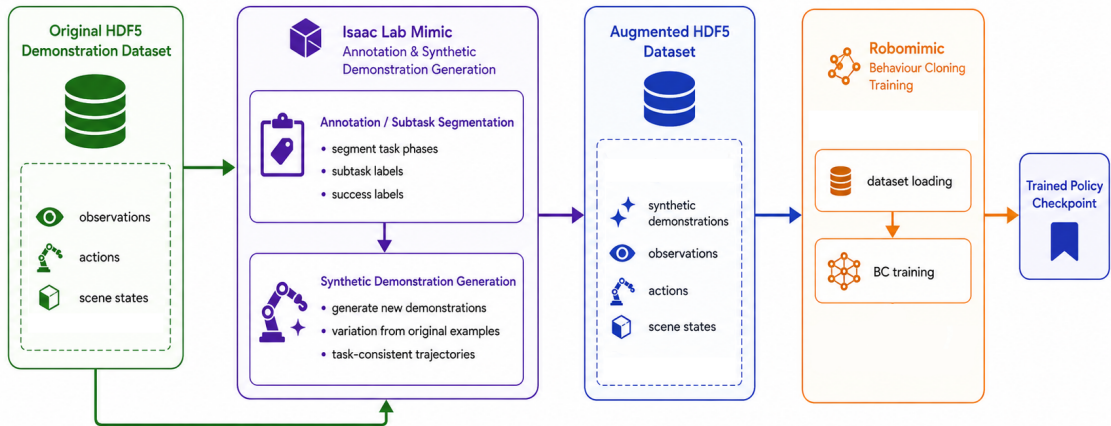


Figure 4.4: Dataset annotation, synthetic demonstration generation, and behaviour cloning training pipeline.

The main implication of this design is that the system does not rely only on the exact trajectories recorded by the user. Instead, it uses the demonstrations to define the structure of the task and then expands that structure into a larger training set while still using human demonstrations as the source of task knowledge.

This stage also has limitations. Synthetic demonstrations are constrained by the quality of the original demonstrations, the correctness of the annotations, and the variation allowed by the task configuration. If the original demonstrations do not capture important parts of the task, or if the generated variations do not represent meaningful task conditions, the resulting training dataset may still be limited. These limitations are considered in the evaluation of the trained policies in Chapter 5.

4.6 Behaviour Cloning Policy Training

The behaviour cloning training stage uses the augmented demonstration dataset as input and trains a reusable task policy. Its role is to learn a direct mapping from observations to actions using supervised learning.

As the methodology relies on **LfD**, **BC** was selected as the **IL** technique to implement. From the dataset generated, the policy learns to imitate the demonstrated action. In the context of this implementation, this is important because the task knowledge is provided through demonstrations rather than through reward signals.

The input to this stage is the augmented HDF5 dataset generated from the original demonstrations. Each file contains the observations and actions required for training. The observations include the state information defined in the Isaac Lab task: object pose, target pose, end-effector pose, robot joint positions, and robot joint velocities. The actions correspond to the target poses and gripper commands in the generated dataset for each time step.

Robomimic was used to train the **BC** policies. This framework was selected because it is designed for **IL** from demonstration datasets and supports the type of HDF5 trajectory data produced by the previous stages. In this implementation, Robomimic is not the architectural contribution itself, but the training framework used to implement the selected imitation-learning method. The Robomimic training configuration used for these policies was defined as part of the implementation.

The output of training is a policy checkpoint that stores the learned model parameters and represents the trained behaviour independently from the training process. This representation allows the learned policy to be stored, reused, and loaded later by the runtime execution environment. Each checkpoint corresponds to a specific task, robot, and object configuration.

4.7 DT Extension and Runtime Execution

The **DT** extension was implemented to provide the runtime environment for trained policies. While Isaac Lab is used for task definition, recording, generation, and training, the **DT** extension is used after training to load the scene, manage objects, select policies, and execute learned behaviours. This separation was necessary because runtime execution has different requirements from training. At runtime, the system does not need to generate demonstrations or optimize a policy; it needs to load existing behaviours and apply them to the current simulated scene.

The **DT** was implemented as an Isaac Sim extension. This implementation choice was driven by the requirement for the runtime interface to operate directly within the simulation environment, providing access to the scene, robot, objects, and policy execution pipeline. An external script could execute policies, but it would provide less direct interaction with scene management and user selection, while limiting the future interaction with a real robot. By implementing the runtime layer as an Isaac Sim extension, the system can combine object loading, policy visualization, queue management, and execution control in the same environment where the **DT** is displayed.

The extension contains the simulated robot cell and a **UI** for interacting with the scene. Through this interface, the user can load the scene, add and remove objects, select a YAML configuration file, view system messages, inspect executable policies, add policies to the execution queue, and execute the queued tasks. This interface was implemented to make the trained behaviours accessible at runtime without requiring the user to manually run scripts or edit code for each execution.

The scene is composed of simulated **USD** assets, which is the format used for asset representation, in Isaac Sim to describe and load simulated objects, robots, and environments. Using **USD** assets allows the extension to manage the robot, gripper, table, manipulated objects, and

target elements consistently within the Isaac Sim scene. This also makes the object-loading mechanism modular, because new assets can be added to the simulated scene without changing the core execution logic.

The YAML configuration file connects the **DT** interface with the available resources. The file contains the standardized object name and the path to the policies related with this object. This configuration layer separates resource definitions from the extension implementation, allowing the runtime system to access the required policy checkpoints and object assets without hard-coding their locations. The YAML format was used since it provides a human-readable format for defining paths to policy folders and **USD** assets. This makes the system easier to update when new policies or objects are added. The YAML file defines the path to the folder containing trained policy checkpoints and lists the available objects together with the paths to their **USD** assets.

The policy folder stores the checkpoints generated during the training stage. These checkpoints are not executed automatically. Instead, the **DT** uses the orchestrator to determine which policies are compatible with the current scene configuration. This avoids exposing behaviours that cannot be executed because the required object is not present in the scene. Once a policy is selected for execution, the system loads the checkpoint and runs it in the simulated scene.

During runtime execution, the policy receives the current state observations from the simulation and outputs the corresponding robot action. The simulation state is updated at each step according to the robot action and the physics of the environment. The policy continues to execute until the task success condition is reached or until the execution is interrupted by a failure or termination condition which is a piece dropping off the table. This runtime interaction between scene management, policy selection, and queued execution is illustrated in Figure 4.5, which shows the **DT UI** used to manage objects, select executable policies, and control task execution.

4.8 Policy Orchestrator and Task Sequencing

The policy orchestrator is one of the main implementation contributions of the system. It addresses the problem that individual trained policies usually represent short manipulation behaviours, while practical tasks often require sequences of actions involving different objects or task steps. A single **BC** policy can learn to perform one pick-and-place behaviour, but it does not by itself decide which behaviour should be available in a given scene or how several behaviours should be ordered. Training a single policy for every possible long-horizon sequence would reduce modularity and require new demonstrations for each complete task, while adding complexity to the scripts and increasing generation and training times. The orchestrator provides a simpler alternative: it allows independently trained policies to be filtered, selected, and executed in sequence, making it possible to develop simpler and more reusable tasks.

The orchestrator connects the current state of the **DT** scene with the available policy checkpoints. Without this filtering mechanism, the user could select policies that cannot be executed correctly, leading to avoidable runtime failures.

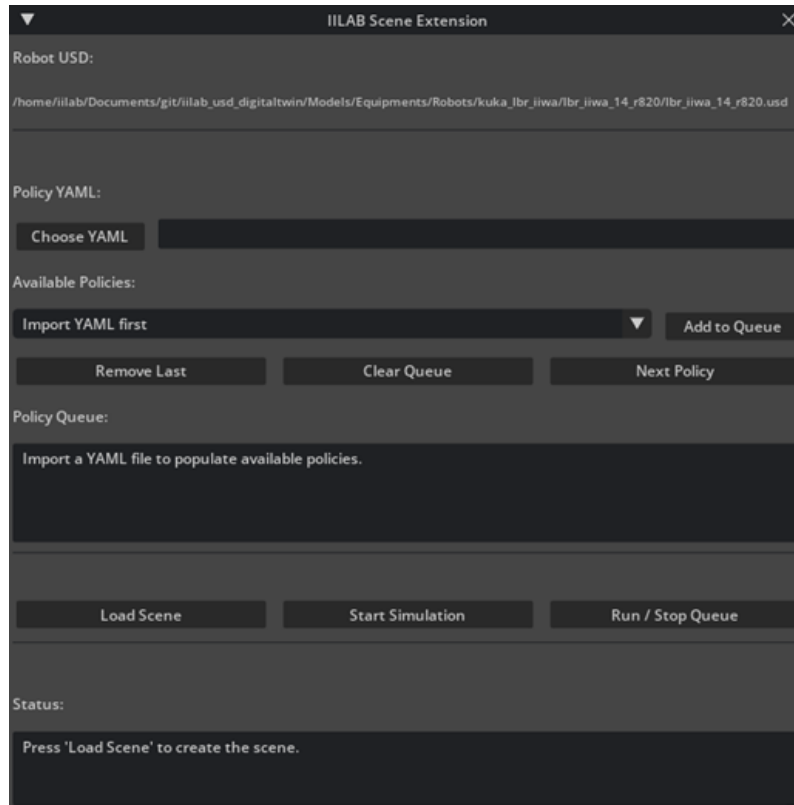


Figure 4.5: DT UI for object management, policy selection, and queued task execution.

The filtering logic is based on parsing the names of the trained policies and comparing them with the names of the objects currently present in the scene. This approach was adopted to provide a lightweight mechanism for associating policies with their task requirements, without introducing additional complexity through a database or a complete symbolic planning representation. The naming convention defined in Equation 4.1 encodes the task type, robot name, and object name, and when necessary, an additional model-specific field. This makes the policy checkpoint itself carry the information needed by the orchestrator to decide whether it is executable.

$$\langle \text{task_type} \rangle * \langle \text{robot_name} \rangle * \langle \text{object_name} \rangle _ \langle \text{optional_info} \rangle \quad (4.1)$$

Table 4.2: Description of the fields used in the policy naming convention.

Field	Description
task_type	Type of manipulation task
robot_name	Robot used during policy training
object_name	Manipulated object associated with the policy
optional_info	Additional model-specific information

In practice, the scene may contain multiple instances of the same object type, distinguished by an index suffix (for example, Axis1 and Axis2). To handle this, the orchestrator does not require an exact match between the full scene prim name and the policy name. Instead, it checks whether the

object name encoded in the policy name appears as a substring of any prim name currently loaded in the scene. A policy associated with Axis is therefore considered executable if any prim whose name contains Axis is present, regardless of the instance index. If no matching prim is found, the policy is not shown in the interface dropdown, preventing the robot from trying to execute a policy that is not possible to run. This association is not limited to deciding whether the main manipulated object is present. During execution, the policy may also depend on additional scene prims that are part of its observation space, such as target objects, containers, or other task-relevant references defined in the task. The orchestrator therefore also links the prims expected by the selected policy to the corresponding prims currently present in the **DT** scene. In this way, the same filtering and association logic connects the trained policy not only to the object it manipulates, but also to the contextual scene elements required for execution. The complete filtering and prim-association process is represented in Figure 4.6.

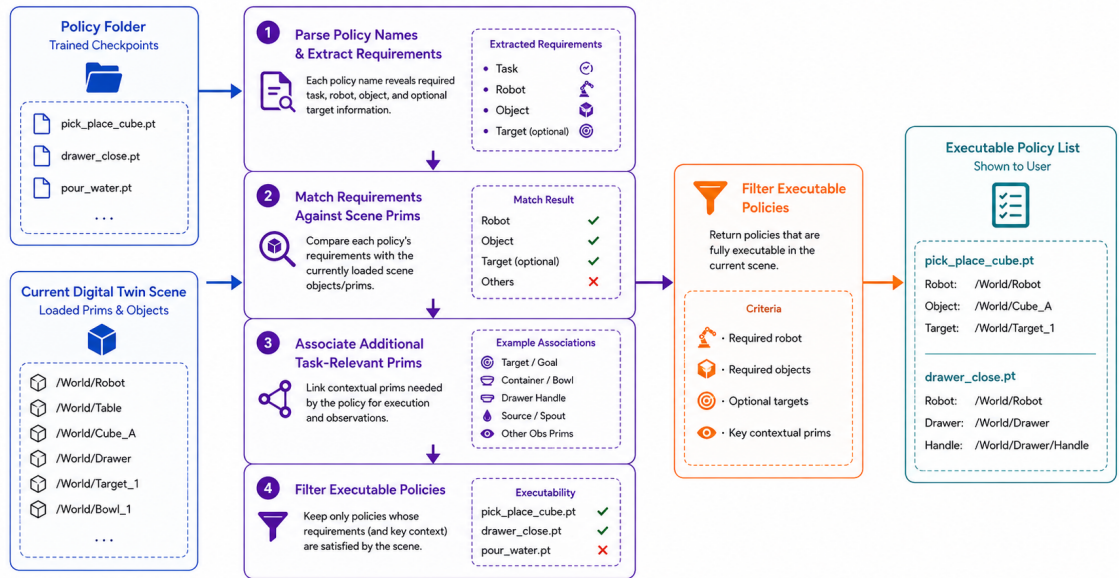


Figure 4.6: Software flow of the DT orchestrator for filtering executable policies and associating task-relevant prims with the current scene.

After filtering, the user can add executable policies to a First In, First Out (**FIFO**) execution queue. A queue-based mechanism was implemented since the runtime objective is to execute policies in a clear and predictable order. **FIFO** execution is simple, transparent, and sufficient for the implemented workflow, where the user defines the intended sequence by adding policies to the queue. This avoids the complexity of a full task planner while still allowing several learned behaviours to be composed into a longer task sequence, while preserving the role of the human in the process.

When execution begins, the first policy is loaded and run in the **DT** scene. The policy receives observations and outputs robot actions until a termination condition is reached. In the implemented workflow, the main successful termination condition is the task success condition. Since

the implemented tasks are pick-and-place tasks, success is reached when the manipulated object is inside the container and the gripper is open. After a successful execution, the system proceeds to the next policy in the queue.

The orchestrator also needs to handle cases where execution cannot continue normally. If a policy does not reach the success condition within the allowed execution horizon, a timeout stops the execution. For the implemented tasks, this timeout is set to 15 seconds, which provides a large enough window for each of the simple pick-and-place tasks to complete successfully while also allowing some time for the policy to recover from a suboptimal intermediate state. At the same time, it avoids an excessively long wait in cases of clear failure. After a timeout occurs, the queue either advances to the next policy or retries the current one from the beginning, depending on the configured behaviour. The timeout value is configurable in the extension source code and can be adjusted for tasks with different complexity or execution time requirements.

The current orchestrator should not be interpreted as a complete task planner. It does not reason about complex preconditions, object state changes, causal dependencies, or alternative plans. Its purpose is more specific: it checks which learned behaviours are applicable to the current scene, associates the required task prims with their corresponding scene prims, and allows the user to compose the resulting executable policies into a sequence. This makes the learned policies more reusable while keeping the runtime system modular and understandable.

4.9 Integrated Workflow and Reproducibility

The complete workflow integrates the components described in the previous sections. This section summarizes the implementation as a sequence of connected inputs and outputs rather than as a set of installation or execution instructions. Its purpose is to show how the system can be reproduced and how information moves from task definition to runtime execution.

The workflow begins with the definition of a manipulation task in Isaac Lab. This task definition specifies the scene, robot, gripper, objects, observations, actions, events, reset conditions, and success condition. The same task definition is then used to record demonstrations through **XR** teleoperation. This ensures that the user demonstrates the same task structure that will later be used for generation, training, and execution.

The recorded demonstrations are stored in an HDF5 dataset. This dataset is annotated to identify the subtask structure required by Isaac Lab Mimic. The annotated dataset is then used to generate additional synthetic demonstrations. The resulting augmented dataset is used by Robomimic to train a **BC** policy. After training, the policy is stored as a checkpoint in the policy folder.

To make the policy available at runtime, the **DT** configuration is updated. The YAML configuration file defines the policy folder and the available object assets. When the **DT** extension is loaded, the user can spawn objects in the scene, inspect the policies that match the current scene configuration, add executable policies to the queue, and run the selected sequence.

The workflow can therefore be summarized as follows:

1. Define the manipulation task in Isaac Lab.

2. Record **XR** demonstrations using the Meta Quest 3 and CloudXR.
3. Store the demonstrations in an HDF5 dataset.
4. Annotate the dataset according to the required subtask structure.
5. Generate synthetic demonstrations from the annotated data.
6. Train a **BC** policy using the augmented dataset.
7. Store the trained policy as a checkpoint.
8. Update the YAML configuration used by the **DT**.
9. Load the **DT** scene and available object assets.
10. Filter executable policies through the orchestrator.
11. Execute selected policies through the **FIFO** queue.

One practical advantage of this structure is that adding a new task to the system is a well-bounded activity. For a developer already familiar with the Isaac Lab and Isaac Sim APIs, writing the task script and the corresponding Mimic annotation script for a simple pick-and-place task takes approximately one hour. Collecting the required demonstrations through the **XR** interface takes around ten minutes. Synthetic data generation via Mimic and **BC** policy training with Robomimic together require approximately thirty minutes, depending on the number of generated demonstrations and training epochs. In total, a new manipulation task can be taken from definition to a deployable policy in approximately two hours. This is a direct consequence of the modularity of the task-definition layer: because all pipeline stages share the same task script, there is no need to separately configure the recording environment, the generation environment, and the execution environment for each new task.

After this checkpoint has been produced, the workflow moves to the runtime layer of the system. The trained behaviour is made available through the **DT** extension, where the user can load the scene, manage objects, inspect executable policies, and execute selected behaviours through the policy queue.

4.10 Chapter Summary

This chapter presented the implementation of the proposed simulation-based robot-teaching system. The chapter was structured around the architectural decisions required to realize the methodology, or the isolated use of software tools.

The implementation uses a consistent Isaac Lab task definition because the same manipulation task must be shared by demonstration recording, synthetic data generation, behaviour cloning training, evaluation, and **DT** execution. All task-specific parameters — including observations, reset distributions, success condition thresholds, and teleoperation retargeting configuration — are

concentrated in this single script, making the system straightforward to extend to new tasks. **XR** teleoperation through the Meta Quest 3 and CloudXR provides the demonstration interface, giving the operator the option between a fully virtual scene and an **XR** view overlaid on the physical environment. HDF5 datasets store the observations, actions, and scene states required for **IL**, using the same format natively expected by Isaac Lab and Robomimic and avoiding any conversion between stages. Isaac Lab Mimic is used to annotate demonstrations and generate additional synthetic examples, with task-specific Mimic scripts that encode the subtask structure of each manipulation behaviour. Robomimic is used to train **BC** policies from the augmented datasets and to store the resulting behaviours as reusable checkpoints.

The **DT** runtime was implemented as an Isaac Sim extension because trained policies must be executed in an interactive simulated environment where objects, assets, and policies can be managed. YAML configuration files connect the runtime interface to the available policy folders and **USD** assets, allowing the system to be updated without hard-coding each object or checkpoint. The policy orchestrator is embedded in the **DT** and filters policies according to the objects present in the scene using a substring-matching naming convention. It then executes selected policies through a **FIFO** queue with a configurable timeout. This allows simple learned behaviours to be reused and combined into longer task sequences while preserving modularity.

Overall, the implementation realizes the proposed methodology as a modular workflow for teaching, training, storing, and executing robot manipulation policies. The chapter also defined the system boundaries and limitations that are relevant for evaluation. The next chapter builds on this implementation by presenting the results obtained from the trained policies and the **DT** execution workflow.

Chapter 5

Results and Evaluation

This chapter presents the evaluation of the system implemented in Chapter 4. The evaluation focuses on analysing whether the developed workflow effectively supports the complete cycle of robot teaching, dataset generation, policy training, and policy execution within the **DT** environment.

The evaluation was designed to assess the system from both a learning-performance perspective and a workflow-integration perspective. On the one hand, the trained manipulation policies must be able to execute the target pick-and-place tasks with a success rate of at least 90%. On the other hand, the system should reduce the manual effort required to teach new tasks and should provide a reusable execution structure through the **DT** and policy orchestrator. Therefore, the evaluation does not focus only on the final success rate of the trained policies, but also on the process required to obtain and execute those policies.

The chapter is organized around five main aspects. First, it analyses the demonstration and dataset generation process, with particular attention to the reduction of manual teaching effort achieved by combining **XR** demonstrations with synthetic data generation. Second, it evaluates the training convergence behaviour observed across all tasks. Third, it evaluates the performance of the trained **BC** policies on four pick-and-place tasks involving motor components, both individually and through multi-policy sequential execution. Fourth, it analyses the main failure modes observed during evaluation. Finally, it discusses the implications of the obtained results and their relation to the task design and training process.

5.1 Evaluation Setup

The evaluation was conducted using the implemented simulation-based teaching and execution pipeline. The same general workflow was applied to all evaluated tasks. First, a task was defined in Isaac Lab. Then, a small number of 10 human demonstrations was recorded through the **XR** teleoperation interface. These demonstrations were augmented through Isaac Lab Mimic, and the resulting dataset was used to train a Robomimic **BC** policy. Finally, the trained checkpoint was loaded into the **DT** extension and executed through the runtime policy execution system. The

computational environment and software versions used in the evaluation are the same as those described in Table 4.1.

The evaluation considered four pick-and-place tasks based on components of a motor assembly that can be seen in Figure 5.1: the axis, the top bearing, the top housing, and the bottom housing. These tasks were selected because they share the same general manipulation structure, while still involving different object geometries, target poses, and grasping conditions. Therefore, they provide a suitable test case for evaluating whether the implemented workflow can be applied consistently across multiple manipulation tasks with similar high-level objectives.

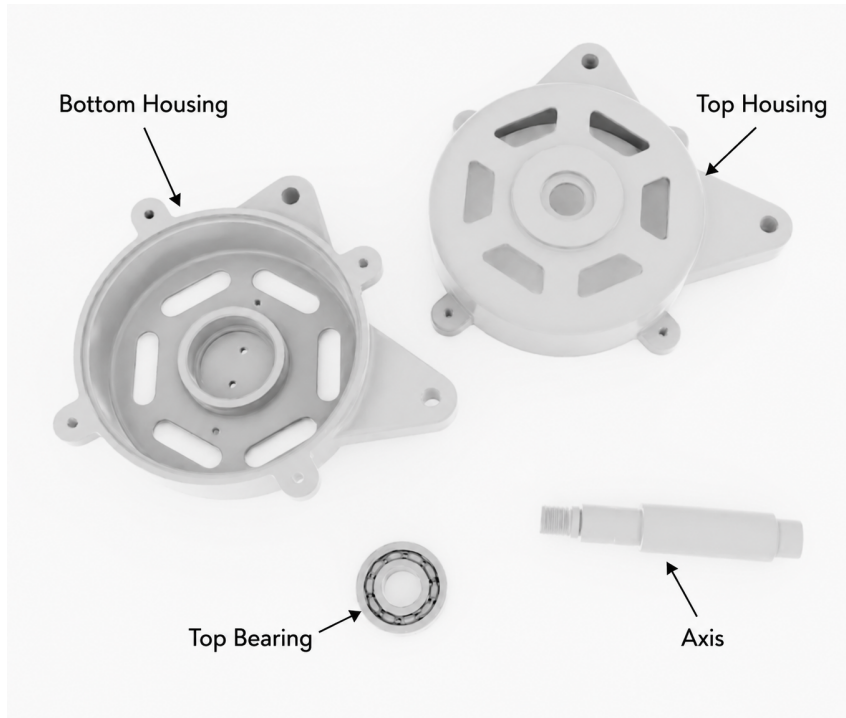


Figure 5.1: Motor components used in the evaluation: axis, top bearing, top housing, and bottom housing.

Each task was evaluated under two distinct conditions. First, policies were tested in the Isaac Lab training environment to verify that the learned behaviour was consistent with the demonstrations. Second, policies were evaluated in the **DT** runtime environment, which serves as the target execution platform and introduces potential distribution shifts from the training conditions. The results reported in the following sections distinguish between these two evaluation contexts whenever both are available.

The robot control frequency was set to 60 Hz, and each execution had a timeout of 15 s this was an empiric decision, but motivated by metrics such as allowing enough time for the task to run correctly while still allowing recovery time in case an initial attempt fails. An execution was terminated when one of three conditions was met: (1) the manipulated object was correctly placed in the assigned target container and the gripper was open (success), (2) the 15 s timeout

was reached (timeout failure), or (3) the object’s z -coordinate dropped below -0.2 m, indicating that the object was lost or fell outside the workspace (failure termination).

The object pose variation used during evaluation followed the same distribution as the one used during synthetic data generation. For all evaluated tasks, the object’s initial position was sampled uniformly within $x \in [0.4, 0.6]$ m and $y \in [0, 0.4]$ m, with a fixed z -position. The only orientation variation was the yaw, sampled uniformly from $[-1, 1]$ in quaternion representation. The evaluation used the same pose variation ranges as the synthetic data generation, meaning that the evaluation conditions were drawn from the same distribution as the training data. No separate test set with out-of-distribution poses was used as the range configured equalled the simulated cell’s tabletop encapsulated by the robot’s workspace limits. Each of the 50 executions per task used newly sampled random poses, ensuring that the evaluation tested the policy across the configured variability rather than on a fixed set of conditions.

The evaluation criteria considered in this work are summarized in Table 5.1, covering the effort required for demonstration and dataset generation, the execution performance of the trained policies, the functionality of the DT runtime and policy orchestrator, and the reproducibility of the overall workflow. The reported results measure robustness within the task-defined distribution of object poses and scene configurations. Therefore, they should not be interpreted as evidence of generalization to arbitrary object poses or substantially different scene layouts, but rather as a validation of the implemented workflow under the conditions explicitly represented during task configuration and synthetic data generation.

5.2 Evaluated Tasks

The evaluation considered four object-specific pick-and-place tasks involving components of a motor assembly: the axis, the top bearing, the top housing, and the bottom housing. In all cases, the task followed the same high-level objective. The robot was required to pick the specified component from its initial pose, transport it to the corresponding target container, and release it after placement.

A task execution was considered successful when the manipulated component was correctly placed in the target container and the gripper released the object. Specifically, an execution was classified as successful when all of the following conditions were simultaneously met: (1) the absolute distance between the object centre and the container centre was less than the container half-dimension in each axis (0.08 m in x , 0.05 m in y , and 0.05 m in z); (2) the gripper was open, as reported by the simulator; and (3) the execution completed within the 15 s timeout. Only position was considered for the placement criterion; no angular tolerance was enforced. The container dimensions were 0.16 m in x , 0.1 m in y , and 0.1 m in height. Object positions and gripper states were extracted automatically from the simulator at each control step, and the success or failure of each execution was determined automatically without visual inspection. Collisions with the container or table during transport or placement were not counted as failures, as long as the object ended up correctly placed and the gripper was open.

Table 5.1: Evaluation criteria and corresponding measurements.

Evaluation aspect	Objective	Measurement
Demonstration and dataset generation	Evaluate the reduction of manual teaching effort achieved through synthetic data generation	Number of human demonstrations, number of generated demonstrations, and time required for dataset generation
Policy execution	Evaluate whether the trained policies can reliably perform the target manipulation tasks	Success rate over 50 execution attempts per task with newly sampled random poses, considering successful grasp, transport, placement, and release within the 15 s timeout
Multi-policy sequencing	Evaluate whether multiple trained policies can be chained for sequential task execution	Success rate over 50 execution attempts per policy chain, with analysis of error propagation and positioning accuracy
DT runtime and orchestrator	Evaluate the integration of trained policies into the execution environment	Policy discovery, scene compatibility filtering, prim association, queue execution, and timeout handling
Workflow reproducibility	Evaluate the effort required to create and train new manipulation tasks	Time required for task setup, demonstration recording, data generation, and policy training

Executions in which the object was missed, dropped during transport, not placed within the container bounds, not released, or not completed within the 15 s timeout were considered failures. Additionally, executions in which the object's z -coordinate dropped below -0.2 m, indicating that the object fell outside the workspace, were terminated as failures. This success definition was applied consistently to all four evaluated components, ensuring that the policies were compared using the same evaluation criterion. Although the objective and success condition were identical across the four tasks, the tasks were evaluated separately because each manipulated component introduced differences in geometry which can affect the difficulty of the pick-and-place operation.

For this reason, the evaluation reports the performance of each trained policy separately. Each policy was evaluated through 50 independent executions in the training environment and then 50 independent executions in the **DT**, resulting in a total of 400 evaluation trials across the four manipulation tasks, 200 for each environment. Performance was measured using the task success rate, where an execution was considered successful when the manipulated component was correctly grasped, transported to the target location, and released within the allowed execution time. This evaluation procedure makes it possible to compare how the same teaching, data generation, training, and execution workflow performed across the four motor components while still accounting for the task-specific difficulty introduced by each object.

5.3 Demonstration and Dataset Generation Results

A central objective of the implemented workflow was to reduce the amount of manual demonstration required to train a new manipulation policy. For each of the four evaluated tasks, 10 human demonstrations were collected through the **XR** interface. These demonstrations were then used as the basis for synthetic data generation with Isaac Lab Mimic, producing 1000 demonstrations per task. The same procedure was applied to the axis, top bearing, top housing, and bottom housing tasks.

A direct comparison against a dataset composed entirely of human demonstrations could be estimated as follows: if a human operator takes approximately 10 minutes to collect 10 demonstrations, collecting 1000 demonstrations would require around 1000 minutes. Although these demonstrations would likely have higher quality due to being directly performed by the user for each reset, generating such a dataset would require a substantially greater amount of time compared to the 10 minutes required for the initial demonstrations, while also preventing the dataset generation process from running autonomously in the background as the operator performs other tasks. Moreover, manually generating an equivalent dataset would require a significantly larger human effort, contradicting the objective of reducing user involvement during task teaching and minimising deployment time for a given policy. Therefore, the evaluation focuses on determining whether a small set of 10 human demonstrations, augmented through Mimic, is sufficient to train policies capable of reliably executing the target manipulation tasks.

All demonstrations used for dataset generation, as well as the resulting datasets, the trained policies, their validation, and the execution tests reported in this chapter, were performed by the

author.

5.4 Behaviour Cloning Training Configuration

The policies were trained using Robomimic with a **BC** configuration based on a recurrent policy. The same training configuration was used for the four evaluated tasks, with only the task name, dataset path, and object-specific observation entries adapted to the corresponding manipulated component. This design direction was intentional, since it allowed the evaluation to focus on the effect of the task and object differences while keeping the learning setup consistent.

The policy was trained using the Robomimic **BC** model configured with a recurrent LSTM-based architecture, a sequence length of 10, a recurrent horizon of 10, two recurrent layers, and a hidden dimension of 512. The recurrent structure was used to account for the temporal nature of the manipulation behaviour, since successful pick-and-place execution depends not only on the current state but also on the progression of the grasping, transport, and placement sequence. The action distribution was represented using a Gaussian Mixture Model output head with five modes. Training used the Adam optimizer with an initial learning rate of 3×10^{-4} , a batch size of 256, and 2000 training epochs. The loss function was based on an L2 action prediction loss with weight 1.0.

The observations used by the policy were low-dimensional state observations. For all evaluated tasks, the observation keys included the robot joint positions, joint velocities, end-effector pose, object pose, target container pose, and the corresponding action sequence. The object pose entry was adapted according to the manipulated component in each task, due to the different prim name, while preserving the same overall observation structure across all policies. This ensured that the policies received equivalent types of information across tasks, as well as standardizing the I/Os used in the **DT**.

The main Robomimic training parameters are summarized in Table 5.2. Policy performance was evaluated both by rollout evaluations after training and through post training execution in the **DT**. This evaluation strategy directly reflects the behaviour of the implemented runtime system.

5.4.1 Training Convergence

Training convergence was monitored through TensorBoard across all four tasks. Figure 5.2 presents the training metrics over the 2000 training epochs, including the action prediction loss, log-likelihood, learning rate, and policy gradient norm. Across all tasks, the training loss decreases rapidly during the first 200–300 epochs before gradually converging, while the log-likelihood increases steadily throughout training. The learning rate remains constant, whereas the policy gradient norm decreases initially before increasing progressively during the later stages of training.

The observed behaviour was consistent across all four tasks, which is expected given their similar complexity and identical training hyperparameter configurations. Although the optimization objective continued to improve until the end of training, the increase in policy gradient norm after approximately epoch 1500 suggests that policy updates became progressively larger during the

Table 5.2: Robomimic behaviour cloning training configuration.

Parameter	Value
Algorithm	Behaviour Cloning
Policy model	Recurrent policy with LSTM
Sequence length	10
RNN horizon	10
RNN hidden dimension	512
RNN layers	2
Policy output	Gaussian Mixture Model
GMM modes	5
Optimizer	Adam
Initial learning rate	3×10^{-4}
Batch size	256
Training epochs	2000
Loss function	L2 action prediction loss
Dataset format	HDF5 generated dataset
Observation type	Low-dimensional state observations
Validation during training	Disabled
Rollouts during training	Disabled

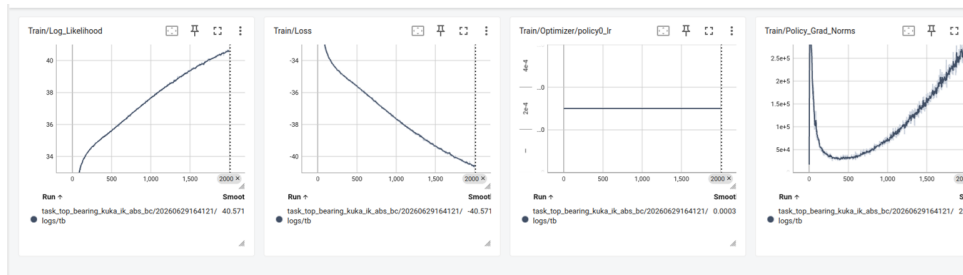


Figure 5.2: Training metrics over 2000 epochs for Top Bearing task

later stages of optimization. On its own, this behaviour does not demonstrate overfitting; however, it motivated a comparison between checkpoints from different stages of training.

To investigate the effect of prolonged training, policies from the 1500-epoch and 2000-epoch checkpoints were evaluated on each assembly task. As shown in Table 5.3, the 1500-epoch checkpoint consistently achieved higher success rates across all four tasks, improving the overall success rate from 86% to 93%. In addition to the quantitative results, qualitative observations during policy rollouts indicated that trajectories generated by the 2000-epoch policies were noticeably less smooth and exhibited greater motion instability than those produced by the 1500-epoch checkpoints.

These findings suggest that extending training beyond approximately 1500 epochs does not improve the policy’s ability to generalize to evaluation rollouts, despite the continued improvement of the training objective. This behaviour is consistent with overfitting or diminished generalization performance following prolonged optimization. Consequently, all subsequent evaluations presented in this work were conducted using the 1500-epoch checkpoints.

Table 5.3: Comparison of policy performance at the 1500-epoch and 2000-epoch checkpoints (50 executions per task).

Task	1500 Epochs	Success Rate	2000 Epochs	Success Rate
Axis	48/50	96%	45/50	90%
Top bearing	45/50	90%	43/50	86%
Top housing	45/50	90%	40/50	80%
Bottom housing	48/50	96%	44/50	88%
Total	186/200	93%	172/200	86%

5.5 Individual Policy Performance in the Training Environment

After training, each 1500-epoch policy was first evaluated through 50 executions in the Isaac Lab training environment to verify that the learned behaviour was consistent with the demonstrated task. These results, presented in Table 5.4, confirm that all four policies learned the target manipulation behaviour from the generated datasets. The training environment evaluation used the same task configuration, reset distributions, and success criteria as the demonstration recording and synthetic data generation stages.

These results indicate that the implemented workflow was able to produce reliable manipulation policies from a small number of human demonstrations when combined with synthetic data generation. Since each policy was trained from only 10 manually recorded demonstrations, the obtained success rates support the objective of reducing manual demonstration effort while still producing policies capable of completing the target manipulation tasks.

Table 5.4: Policy evaluation results in the Isaac Lab training environment (50 executions per task).

Task	Successful	Success rate
Axis	48	96%
Top bearing	45	90%
Top housing	45	90%
Bottom housing	48	96%
Total	186	93%

The difference between the best-performing tasks and the remaining tasks was relatively small, with a difference of 6 percentage points between the 96% success rate of the axis and bottom housing policies and the 90% success rate of the top bearing and top housing policies. This suggests that the same training and execution pipeline was effective across all evaluated components, but that object-specific manipulation difficulty still influenced the final performance.

5.6 Individual Policy Performance in the Digital Twin Environment

The same four policies were subsequently evaluated through 50 executions each in the **DT** runtime environment. The **DT** environment serves as the target execution platform for the implemented system and introduces potential distribution shifts from the training environment due to differences in the runtime execution pipeline. These results are presented in Table 5.5.

Table 5.5: Policy evaluation results in the DT environment (50 executions per task).

Task	Successful	Success rate
Axis	47	94%
Top bearing	45	90%
Top housing	40	80%
Bottom housing	48	96%
Total	180	90%

The overall success rate in the **DT** environment was 90%, compared to 93% in the training environment. This small reduction is consistent with the expected effect of minor distribution shifts between the two evaluation contexts. The axis, top bearing, and bottom housing policies maintained comparable performance across both environments, with differences of 2 percentage points or less.

The most notable difference was observed in the top housing policy, which achieved 90% in the training environment but only 80% in the **DT**. This degradation is attributed to the specific manipulation conditions of the top housing task. The top housing was trained with the object placed on top of a support structure, because its geometry does not allow the gripper to grasp it from the side and the object requires sufficient height to prevent the robot arm from colliding with

the table. This configuration occasionally led to the object falling from the support during resets in the **DT** environment, resulting in variations in pitch and yaw orientation that were not represented in the training data. When the top housing assumes an unexpected orientation, the policy is unable to recover, as these orientation variations were not encountered during the training process. This effect is discussed further in Section 5.8. Table 5.6 presents a side-by-side comparison of the success rates obtained in both evaluation environments.

Table 5.6: Comparison of policy success rates between the Isaac Lab training environment and the DT runtime environment.

Task	Training env.	DT env.	Difference
Axis	96%	94%	−2%
Top bearing	90%	90%	0%
Top housing	90%	80%	−10%
Bottom housing	96%	96%	0%
Overall	93%	90%	−3%

5.7 Multi-Policy Sequential Execution

Beyond individual policy execution, the implemented orchestrator supports chaining multiple policies in sequence, where upon a policy finishing execution, the next policy is loaded automatically. This capability is essential for assembly workflows that require multiple components to be manipulated in a defined order. All possible two-policy combinations and a representative set of three-policy chains were evaluated through 50 executions each.

Table 5.7 presents the results for all two-policy chains. Under the assumption that policy failures are independent, the expected success rate for any two-policy chain is the product of the individual success rates. For example, the axis to top bearing chain would be expected to achieve $0.94 \times 0.90 = 84.6\%$ under independence. The observed results consistently fall below this expectation, with an additional failure rate of approximately 4–8 percentage points depending on the specific pair.

The results reveal two distinct performance tiers. Chains composed exclusively of the axis, top bearing, and bottom housing achieve success rates of 76–82%, while chains that include the top housing drop to 66–70%. This gap is consistent across all tested orderings and reflects both the lower individual success rate of the top housing policy and the additional orientation-related failures that arise when this task is executed after another policy has completed. Within each tier, the ordering of tasks had a limited effect, with variations of at most 2 percentage points between reversed pairs.

The additional reduction beyond independent error propagation is attributed to two factors: error propagation between sequential tasks and reduced positioning accuracy for policies executed

Table 5.7: Two-policy chain evaluation results (50 executions per chain).

Policy chain	Successful	Success rate
Axis → Top bearing	40	80%
Top bearing → Axis	41	82%
Axis → Bottom housing	39	78%
Bottom housing → Axis	39	78%
Top bearing → Bottom housing	38	76%
Bottom housing → Top bearing	38	76%
Axis → Top housing	34	68%
Top housing → Axis	33	66%
Top bearing → Top housing	33	66%
Top housing → Top bearing	33	66%
Bottom housing → Top housing	35	70%
Top housing → Bottom housing	34	68%

after the first. When a policy completes its execution, the robot arm returns to a default home position before the next policy begins. However, this default position does not provide the subsequent policy with the same initial conditions it encountered during training, where each episode reset the arm to the same default pose. The first policy in a chain benefits from this default position matching its training distribution, while subsequent policies must begin their approach from a state that may differ from their training conditions. This effect was consistently observed during evaluation: policies executed after the first in a chain demonstrated increased difficulty in reaching the correct grasp position, although once contact with the object was established, the manipulation behaviour proceeded normally.

This observation aligns with the task design used throughout this work. During training, each episode reset the environment with the arm returning to the same default home position. Consequently, the policies learned to approach objects from a fixed initial arm configuration rather than learning position-invariant approach behaviour. Introducing greater variability in the initial arm position during training would be expected to improve the policy’s generalization to different starting configurations and reduce this specific source of failure in chained execution.

Table 5.8 presents the results for three-policy chains.

The three-policy results follow the same pattern observed in the two-policy chains. Chains that avoid the top housing entirely achieve success rates of 70–74%, while those that include it fall to 54–58%. The ordering of tasks within a chain had a limited effect on the overall success rate, with variations of at most 2 percentage points across different orderings of the same component set.

Across all tested chain lengths, the results demonstrate that the compounding effect of error propagation and positioning difficulty grows with the number of sequential policies. Chains of two tasks remain above 66% for all tested combinations and chains of three tasks range from 54% to 74%. It is expected that chains of four chain tasks would reduce this success rate even further, considering the mentioned theories on the cause of the failures. These results highlight

Table 5.8: Three-policy chain evaluation results (50 executions per chain).

Policy chain	Successful	Success rate
Axis → Top bearing → Bottom housing	36	72%
Top bearing → Axis → Bottom housing	37	74%
Bottom housing → Axis → Top bearing	36	72%
Axis → Bottom housing → Top bearing	35	70%
Top bearing → Bottom housing → Axis	35	70%
Axis → Top bearing → Top housing	28	56%
Top bearing → Axis → Top housing	29	58%
Top housing → Axis → Top bearing	28	56%
Axis → Top housing → Bottom housing	27	54%
Bottom housing → Top housing → Axis	27	54%

the importance of addressing both the individual policy robustness and the chaining efficiency for applications that require multi-step manipulation sequences.

5.8 Failure Analysis

Although the individual policies achieved an overall success rate of at least 90%, failures were observed in all evaluated tasks. This section analyses the failure modes across both individual policy execution and multi-policy chaining, with particular attention to the factors that contributed to the reduced performance of the top housing task and to the behaviour observed during sequential policy execution. Three main failure modes were identified during evaluation: orientation-dependent failures, object displacement during transport, and execution timeout.

5.8.1 Orientation-Dependent Failures

The most significant failure mode, particularly for the top housing task, was the policy’s inability to recover from unexpected object orientations. When the top housing assumed a pitch or yaw orientation that deviated from the training distribution, the policy was unable to adapt its approach and complete the manipulation successfully. This behaviour was observed both in isolated execution and within policy chains.

This failure mode is directly related to the task design. The top housing task required the object to be placed on a support structure rather than directly on the table, because the object’s geometry does not allow the gripper to grasp it from the side and the object needs sufficient height for the robot arm to avoid collision with the table surface. During **DT** execution, the top housing occasionally fell from this support during environment resets, resulting in orientation variations that were not represented in the training data. Since the policy was only trained with the object in its expected upright pose on the support, it could not recover when the object assumed an unexpected orientation.

The other three tasks did not exhibit this failure mode to the same extent. The axis, top bearing, and bottom housing were placed directly on the table surface, which provided a stable base that limited orientation variation. These objects maintained consistent orientations across resets, and the policies were able to handle the trained orientation range reliably.

5.8.2 Recovery Behaviour

During successful executions, approximately 10% required some form of recovery action, such as re-grasping the object after it slipped from the gripper or recovering from a dropped object. In these cases, the policy demonstrated the ability to detect the failure condition and attempt a corrective action, ultimately completing the task successfully.

In contrast, many failed executions were characterized by the absence of this recovery behaviour. When the policy encountered a failure condition, such as an unsuccessful grasp or a perceived obstruction during motion, it often continued attempting the same behaviour without adapting its strategy. This frequently resulted in the arm becoming stuck in a repeated motion pattern or looping around a particular position until the execution timed out. These failures suggest that, although the policy can recover from some disturbances, it lacks a more general mechanism for recognizing when the current action sequence is no longer effective and switching to an alternative recovery strategy.

The top housing task exhibited fewer successful recoveries than the other tasks, which is consistent with the nature of its failures. In particular, the task was more sensitive to approach and placement errors, and failures often resulted in the arm entering unproductive motion patterns rather than initiating a corrective action. This indicates that recovery behaviour was dependent not only on detecting object-level failures, but also on recognizing when the overall motion execution had become ineffective.

5.8.3 Execution Timeout

Approximately 95% of all failed executions across all tasks ended due to the 15 s timeout rather than a catastrophic failure such as the object falling outside the workspace. In these timeout cases, the robot arm either failed to reach the object within the allowed time or became positioned in a configuration from which it could not progress toward the manipulation objective. In many timeout cases, the arm did not demonstrate movement patterns suggesting that additional execution time would have led to a successful outcome; the policy appeared to be in a state where it could not determine a productive next action.

This timeout-dominant failure pattern was consistent across all four tasks and was particularly prevalent in the top housing task due to the additional orientation variability. The timeout failures also accounted for most of the unsuccessful executions in the multi-policy chains, where the accumulated positioning difficulty and the absence of appropriate recovery behaviours led to the arm becoming non-productive before completing the full sequence.

5.8.4 Failure Distribution Across Tasks

The failure analysis reveals a clear relationship between task design and failure rate. Table 5.9 summarizes the failure attribution for each task in the DT environment.

Table 5.9: Failure attribution by task in the DT environment (50 executions per task).

Task	Failures	Orientation-related	Timeout (other)	Object lost
Axis	3	0	3	0
Top bearing	5	1	4	0
Top housing	10	7	3	0
Bottom housing	2	0	2	0

The top housing task accounts for the majority of orientation-related failures, while the remaining tasks fail primarily due to timeout in situations where the policy cannot complete the grasp or placement within the allowed time. The fact that no object lost failure condition was detected in the evaluation means the policies rarely or never demonstrate unstable behaviours that would cause the objects dropping off the table.

5.9 Deployment Time and Reproducibility

A practical goal of the implemented workflow was to make the addition of new tasks faster than a day and reproducible to many scenarios. Once the task structure is understood, the process of adding a new task follows a bounded sequence of steps: defining the task in Isaac Lab, recording demonstrations through the XR interface, generating synthetic demonstrations with Mimic, training the BC policy with Robomimic, and loading the resulting checkpoint in the DT.

For the evaluated tasks, collecting the required human demonstrations took approximately 1 minute per demonstration. Synthetic demonstration generation took on average approximately 1.8 seconds per synthetic episode, and policy training took approximately 1 second per epoch, with the implemented tasks. It is expected that more complex tasks require longer times in all of these metrics. These timings refer to the main operational stages after the task structure and required scripts are already available. After training, the policy checkpoint can be loaded and reused in the DT without repeating the demonstration, generation, or training stages.

These timings are summarized in Table 5.10. They represent the approximate time needed to add and train a new task once the system is already configured.

These results show that, after the system has been implemented and configured, a new manipulation task can be taken from demonstration recording to a trained policy in approximately 70 minutes, excluding task scripting and annotation scripting. When the time required to write the task script and the corresponding Mimic annotation script is included, the complete process takes approximately two hours for a simple pick-and-place task.

Table 5.10: Approximate time required to add and train the developed manipulation tasks

Stage	Approximate time
XR demonstration recording	10 minutes
Synthetic demonstration generation	30 minutes
Behaviour cloning training	30 minutes

This result is relevant from a usability perspective because it shows that most of the effort is concentrated in reusable configuration and automated processing stages rather than in extensive manual demonstration collection. The user is required to provide only a small number of demonstrations, while the remaining stages can be repeated in a structured way for each new task. This supports the claim that the implemented workflow provides a reusable structure for adding new object-specific manipulation behaviours.

5.10 Discussion

The results demonstrate that the implemented workflow can produce executable pick-and-place policies with a limited amount of human demonstration effort. Across the four evaluated tasks, 10 human demonstrations per task were sufficient to generate 1000 synthetic demonstrations and train policies that achieved an overall success rate of 90% in the DT execution environment.

This result supports the central motivation of the implementation: the user should not need to manually demonstrate a task hundreds or thousands of times in order to obtain a usable policy. Instead, the human provides a small number of demonstrations through the XR interface, and the system uses synthetic generation to expand those demonstrations into a dataset suitable for BC training. The obtained success rates suggest that this approach was sufficient for the evaluated pick-and-place tasks within the configured simulation variability.

The comparison between training environment and DT environment performance provides insight into the robustness of the trained policies. The axis, top bearing, and bottom housing policies maintained consistent performance across both environments, suggesting that these policies learned manipulation behaviours that generalize across minor distribution shifts. The top housing policy, however, showed a 10 percentage point reduction in the DT, which is attributed to the specific task design rather than to a general limitation of the workflow. The requirement to place the top housing on a support structure introduced orientation variability that was not present in the training data, and this variability directly impacted the policy’s ability to execute reliably.

The training convergence analysis reveals a systematic pattern across all four tasks: the loss destabilizes at approximately epoch 1500, suggesting that the policies may benefit from early stopping. The consistent behaviour across tasks confirms that this is a property of the training configuration rather than of individual task characteristics. This observation provides a clear direction for future optimization of the training process.

The multi-policy chaining results extend the evaluation from individual task execution to sequential task composition. Across all tested chain lengths, the observed chaining penalty of approximately 4–8 percentage points relative to independent error propagation is primarily attributed to the fixed initial arm position used during training. When the robot arm returns to its home position between tasks, the subsequent policy must approach the object from a configuration that may differ from its training distribution. This finding highlights the importance of including initial arm position variability in the training reset configuration, particularly for tasks intended to be used in sequential chains. Chains of two tasks maintained success rates above 66% for all tested combinations, while chains of four tasks dropped to 50–62%, with the ordering of tasks affecting the overall success rate by up to 12 percentage points.

The failure analysis provides actionable insight for improving the trained policies. The top housing task would benefit most from two specific improvements: including orientation variability (pitch and yaw) in the training reset configuration, and placing the object directly on the table surface or on a more stable support to reduce the frequency of unexpected orientations. These changes would address the primary source of failure without requiring changes to the training pipeline itself. Similarly, introducing initial arm position variability in the reset configuration would address the positioning difficulty observed during multi-policy chaining.

The results also show the value of using a shared Isaac Lab task definition across the pipeline. The same task structure supports demonstration recording, synthetic generation, training, and execution. This reduces duplicated configuration effort and makes the workflow easier to reproduce when a new task is added. The timing results reinforce this point, since the main stages after task definition can be completed in approximately 70 minutes for a new task, with the complete process taking approximately 2 hours when task scripting and annotation scripting are included.

Another important outcome is that the **DT** and orchestrator successfully provided a runtime layer for executing the trained behaviours. The orchestrator was able to discover available policies, filter them according to scene compatibility, associate task prims with scene prims, and execute selected policy queues.

At the same time, the evaluation has some limitations. First, the robustness evaluation was performed within the variability range configured in the Isaac Lab tasks. The results should therefore not be interpreted as evidence that the policies generalize to arbitrary object poses or substantially different scene configurations. Second, the work did not compare the generated datasets against a fully human-demonstrated dataset of equivalent size, because collecting such a dataset would require a large amount of manual effort. Third, the orchestrator was evaluated functionally rather than through a separate quantitative benchmark of long task sequences. Fourth, the Robomimic training configuration did not enable validation or rollout evaluation during training, so policy performance was measured through post-training execution rather than through training-time validation metrics.

Despite these limitations, the results indicate that the implemented system meets its main objective. It provides an integrated workflow for teaching, generating data, training policies, and executing them in a **DT**. The system was able to train four object-specific manipulation policies

from a small number of human demonstrations and execute them with high success rates in simulation. The evaluation therefore supports the proposed workflow as a practical approach for rapid development and deployment of manipulation policies in a **DT** environment.

5.11 Chapter Summary

This chapter evaluated the implemented workflow for teaching, dataset generation, policy training, and execution within the **DT** environment. Four pick-and-place tasks involving motor assembly components were considered: axis, top bearing, top housing, and bottom housing. For each task, 10 human demonstrations were recorded through the **XR** interface, expanded to 1000 synthetic demonstrations using Isaac Lab Mimic, and used to train recurrent LSTM-based **BC** policies with Robomimic. The trained policies were evaluated individually in both the Isaac Lab training environment and the **DT** runtime environment, as well as through sequential multi-policy execution.

The individual policies achieved an overall success rate of 93% in the Isaac Lab training environment and 90% in the **DT** environment. In the **DT**, the axis, top bearing, top housing, and bottom housing policies achieved success rates of 94%, 90%, 80%, and 96%, respectively. Multi-policy execution achieved success rates between 66 and 82% for two-policy chains and 54 to 74% for three-policy chains, depending on the selected task combination. The evaluation also showed that 10 human demonstrations were sufficient to generate datasets capable of training policies that successfully performed the evaluated manipulation tasks.

The obtained results demonstrate the effectiveness of the implemented workflow within the evaluated task configurations, however, they do not represent generalization to arbitrary object poses, different scene layouts, or more complex manipulation scenarios. The evaluation was limited to simulation-based execution and to the variability explicitly included during task configuration and synthetic data generation. Additionally, the multi-policy evaluation was performed on a limited number of task sequences, and no comparison was made against datasets composed entirely of human demonstrations of equivalent size.

Chapter 6

Conclusions and Future Work

This chapter concludes the dissertation by summarizing the main contributions, discussing the conclusions supported by the evaluation, identifying the main limitations of the current implementation, and outlining directions for future work. The conclusions should be interpreted within the scope defined throughout the dissertation: the system was implemented and evaluated as a simulation-based workflow for teaching, generating, training, and executing robotic manipulation behaviours in a controlled DT environment.

6.1 Summary of Contributions

This dissertation proposed and implemented a framework for teaching robotic manipulation tasks through XR-based teleoperation, simulation-based data generation, IL, and DT execution. The work was motivated by the need for more accessible and adaptable robot-programming methods in flexible industrial environments, where robotic manipulators are expected to be reconfigured without requiring extensive low-level programming expertise from the operator. In relation to the gap identified in the literature review, the main contribution of the dissertation is not a new learning algorithm in isolation, but an integrated and structured workflow that connects intuitive demonstration, synthetic dataset generation, policy training, and modular policy execution.

At the demonstration and training level, the framework provides an XR-based pipeline that allows the user to collect teleoperated demonstrations directly in the simulated workspace through the HMD interface. These demonstrations are then annotated with subtask boundaries and used as reference trajectories for Isaac Lab Mimic. Instead of requiring the user to manually provide a large dataset, the system uses the demonstrated subtask structure to generate synthetic demonstrations under different object configurations. The generated dataset is then used to train BC policies with Robomimic, which are subsequently deployed back into the simulated environment for execution.

This training workflow is connected to a DT-based runtime environment rather than being treated as a separate offline process. A shared Isaac Lab task definition supports demonstration

recording, synthetic generation, training, and execution, reducing duplicated configuration between stages. The **DT** therefore acts not only as a visualization environment, but also as the execution context in which trained policies are selected, configured, launched, and inspected.

To make the trained behaviours reusable in multi-piece pick-and-place scenarios, the dissertation also develops a modular orchestrator architecture. Since Isaac Lab Mimic relies on predefined subtask sequences, training a single policy for a complete multi-object task would reduce flexibility and make policy reuse more difficult. To address this limitation, the system trains one object-specific policy per piece and delegates the high-level task sequence to an orchestrator inside the Isaac Sim extension. The orchestrator discovers available policy checkpoints, filters executable tasks according to the objects present in the scene, associates **USD** scene prims with task prims through a naming convention, and manages a **FIFO** execution queue with configurable ordering and timeout-based failure handling.

The implemented workflow was then evaluated in four motor-component pick-and-place tasks: axis, top bearing, top housing, and bottom housing. For each task, 10 human demonstrations were collected through the **XR** interface and expanded to 1000 synthetic demonstrations using Isaac Lab Mimic. The resulting datasets were used to train recurrent **BC** policies with Robomimic, which were then evaluated in the **DT**. Across 200 executions, the system achieved 186 successful executions, corresponding to an overall success rate of 93%. The axis and bottom housing policies achieved 96% success rates, while the top bearing and top housing policies achieved 90% success rates.

Together, these contributions directly address the objectives defined in Chapter 1. The developed **XR**-based interface addresses the objective of enabling intuitive teaching of manipulation tasks through direct interaction with the simulated workspace. The **DT** representation provides the environment required for task definition, validation, and execution before physical deployment. The integrated software framework connects demonstration collection, synthetic data generation, policy training, and runtime execution into a single workflow. Finally, the evaluation of the trained policies and the orchestrator demonstrates the performance and practical applicability of the proposed methodology for manipulation tasks.

6.2 Main Findings

The results of this dissertation support the conclusion that the proposed workflow is feasible for rapidly creating executable manipulation policies from a small number of human demonstrations in simulation. In the evaluated tasks, 10 demonstrations per object-specific policy were sufficient to generate 1000 synthetic demonstrations and train policies that achieved success rates between 90% and 96%. The overall result of 186 successful executions out of 200 shows that the trained policies were generally reliable within the configured variability of the **DT** environment.

These results indicate that simulation-based demonstration generation can reduce the amount of manual demonstration effort required within the implemented pipeline. In this workflow, the

human demonstrations are not used as the complete training dataset. Instead, they provide structural examples of the desired behaviour, including the relevant subtask decomposition. Isaac Lab Mimic then uses these examples to generate a larger dataset over different object configurations. This shifts the emphasis from collecting a large number of human demonstrations to collecting a smaller number of clean, consistent, and well-structured demonstrations. However, this conclusion should be interpreted carefully: the dissertation did not compare the generated datasets against an equivalent dataset composed entirely of human demonstrations. Therefore, the results support a practical reduction of manual data-collection effort within the proposed workflow, but they should not be interpreted as a direct benchmark against all alternative data-collection strategies.

The same results also support the use of a modular per-piece policy architecture for the type of pick-and-place tasks considered in this work. Training one policy per object trades global optimization of the full manipulation sequence for modularity, reusability, and easier replacement of individual behaviours. For tasks in which the pieces are spatially independent without strict interdependence between actions, this trade-off is suitable. It allows each policy to function as a reusable skill that can be retrained or replaced without modifying the remaining policies.

At runtime, the **DT** orchestrator provides a useful layer for integrating trained policies into a complete task execution workflow. The orchestrator successfully identified available policies, filtered them according to scene compatibility, associated scene prims with task prims, and executed selected queues. Since the orchestrated execution preserved the same individual policy error rates observed during isolated policy evaluation, no additional source of execution error was observed at the orchestration level. This indicates that the runtime integration did not degrade the behaviour of the trained policies in the evaluated scenarios.

A further implication of the implementation is that structural conventions can reduce part of the operator’s configuration workload. The naming-convention-based association between scene objects and policy folders allows the system to infer which trained policies are compatible with the current scene, without requiring the operator to manually connect each object to a checkpoint. This supports extensibility: once a new object-specific task has been defined, demonstrated, generated, and trained, the resulting policy can be made available to the **DT** by placing the checkpoint in the correct folder structure. The timing results reinforce this point, since a new simple pick-and-place task can be taken from demonstration recording to a trained policy in approximately 70 minutes after the required task structure and scripts are available, or approximately two hours when task scripting and Mimic annotation scripting are included.

The demonstrated adaptability corresponds to robustness within the object pose ranges and scene conditions configured in simulation. It does not demonstrate unrestricted generalization arbitrary layouts, dynamic environments, or real-world physical execution. Similarly, the **XR** interface reduces the need for the operator to specify trajectories through code during demonstration collection, but the current pipeline still requires developer-level configuration for task definition, annotation scripting, training setup, and **DT** integration.

Overall, the dissertation demonstrates that combining **XR**-based demonstration, Isaac Lab Mimic data generation, Robomimic **BC** training, and **DT** orchestration is a practical foundation for

creating reusable manipulation behaviours in a controlled simulation environment. The main value of the work lies in the integration of these components into a coherent workflow, demonstrating how accessible human demonstrations, synthetic dataset generation, **IL**, and **DT** execution can be effectively combined to support the rapid development of manipulation policies while leveraging human expertise to teach robotic manipulation tasks.

6.3 Limitations

The identified limitations do not invalidate the proposed workflow, but they define the scope under which the obtained results should be interpreted. The achieved performance corresponds to a controlled simulation setting with known objects, predefined task structures, and explicitly configured variability. This distinction is important because the system combines several components often associated with broader autonomy, including **XR** teaching, **DTs**, synthetic demonstration generation, and learned control policies. In the present work, these components were integrated and validated as a functional prototype rather than as a fully autonomous or deployment-ready industrial system.

The limitations are organized according to their impact on the interpretation and practical applicability of the proposed methodology. The first group concerns system-level limitations, where the current implementation introduces constraints that affect deployment beyond the evaluated simulation environment. The second group concerns evaluation limitations, which define the boundaries of the conclusions that can be drawn from the reported results.

6.3.1 System Limitations

The most significant limitation is that the complete pipeline has been validated in simulation only. Although simulation provides a safe and repeatable environment for collecting demonstrations, generating datasets, and evaluating policies, it cannot fully reproduce the contact dynamics, actuator behaviour, calibration errors, sensor noise, visual appearance, and physical uncertainty of a real robotic cell. Consequently, the obtained success rates demonstrate reliability in the tested **DT** scenarios, but they do not guarantee equivalent performance when transferred to physical hardware.

A related limitation is the absence of perception-driven synchronization between the physical environment and the **DT**. In the current implementation, object poses are obtained from the simulated environment or manually configured. A deployment-ready version would require object detection and pose estimation to update the **DT** automatically from real sensor data. Without this capability, the current system cannot autonomously adapt to arbitrary object placements in a physical workspace.

Although the demonstration interface reduces the need for low-level programming during data collection, the complete workflow is not yet fully accessible to non-expert users. Adding a new

task still requires task scripting, annotation scripting, training configuration, and correct integration with the **DT** folder structure. Therefore, the current implementation reduces the barrier associated with demonstration collection, but does not eliminate all developer-level requirements from the complete task creation process.

Task completion detection is another limitation of the current implementation. The execution loop relies on the done signal provided by the terminating task’s Mimic environment definition. This signal represents the success condition encoded in the simulation task rather than an independent verification of the final scene state. If an object shifts, falls, or becomes unstable after the signal is triggered, the orchestrator may advance the execution queue without detecting the discrepancy. A more robust implementation would include a scene-level verification step after release, such as checking the distance between the placed object and the target container or verifying object stability after a short settling period.

Execution continuity is also limited by the pause introduced when policy checkpoints are loaded between consecutive tasks. For the evaluated pick-and-place scenarios, this pause is acceptable because the tasks are independent and do not require continuous physical motion. However, it would become a limitation in applications requiring uninterrupted execution, time-sensitive handoffs, or close physical coordination with a human operator.

Finally, the order-agnostic execution mode provides flexibility but currently relies on uniform random ordering. Although this demonstrates that the orchestrator can execute policies without a fixed predefined sequence, it does not optimize the execution order according to the current robot pose, object positions, or expected travel distance. For larger numbers of objects, this may result in inefficient motion and increased execution times.

6.3.2 Evaluation Limitations

The main limitation of the evaluation is the restricted task scope. The system was evaluated on four rigid-object pick-and-place tasks involving known motor components and fixed container targets. While these tasks represent a relevant class of industrial manipulation problems, they do not cover articulated objects, deformable objects, tool use, insertion tasks, contact-rich assembly, or tasks requiring long-horizon reasoning.

The robustness evaluation was also limited to the variability configured in the Isaac Lab tasks. The policies were tested under controlled object pose variations and scene conditions, but not under arbitrary disturbances, unseen objects, occlusions, dynamic obstacles, or substantially different layouts. Therefore, the reported success rates measure reliability within the designed simulation distribution rather than general-purpose manipulation robustness.

The evaluation did not include a baseline dataset composed of an equivalent number of fully human-demonstrated trajectories. Since collecting such a dataset would require substantial manual effort, the evaluation focused on the practical workflow enabled by synthetic demonstration generation. The results therefore demonstrate that the proposed pipeline can produce effective policies from few human demonstrations, but do not provide a direct comparison with alternative data-collection strategies.

The orchestrator was evaluated functionally rather than through a separate quantitative benchmark of long task sequences. The results demonstrate that it can discover policies, filter executable tasks, associate prims, manage the queue, handle timeouts, and execute policy sequences without introducing an observed additional source of error. However, the evaluation did not include a statistical analysis of long multi-object sequences with different queue lengths, ordering strategies, or accumulated failure probabilities.

Robomimic validation and rollout evaluation were not enabled during training. Policy performance was therefore measured through post-training execution tests rather than training-time validation metrics. While the final execution results remain meaningful, this limits the analysis of overfitting, convergence behaviour, and checkpoint selection during training.

Finally, the evaluation did not include a formal user study. Since one motivation of the dissertation is to make robot teaching more accessible, the current accessibility claim should be interpreted as a design-based claim rather than as an empirically validated usability result. It is supported by the design of the **XR** interface and the reduced manual demonstration requirement, but not by usability metrics such as System Usability Scale (**SUS**) score, time to first successful demonstration, number of discarded demonstrations, or subjective workload.

6.4 Future Work

Several directions for future development follow directly from these limitations.

Physical deployment and sim-to-real transfer. The first development needed is to deploy the trained policies on a physical cell. This would require calibration between the robot, sensors, and **DT** and evaluation of how contact-rich pick-and-place behaviours transfer from simulation to the real world. Domain randomization and fine-tuning with a small number of real demonstrations could be used to reduce the gap between simulated and physical execution.

Perception-driven **DT synchronization.** A deployment-ready version of the system should include a perception layer capable of automatically updating the **DT** from the real workspace. This would require the integration of RGB-D cameras for scene observation, object detection, and 6D pose estimation to determine the position and orientation of relevant objects. The estimated poses would then need to be transferred to the corresponding **USD** prims, ensuring that the simulated representation remains synchronized with the physical environment. Additionally, the perception pipeline should account for estimation uncertainty, perform continuous calibration or pose refinement between frames, and handle partial observations caused by object occlusions. Integrating these capabilities with the **DT** would enable the system to launch trained policies according to the actual state of the scene, monitor execution changes, and detect discrepancies between expected and observed object configurations.

Scene-level verification and runtime recovery. Future versions of the orchestrator should include independent post-placement verification rather than relying only on the task-internal done signal. After each policy execution, the **DT** could check whether the manipulated object is close to

the intended target and whether it remains stable after release. If verification fails, the orchestrator could retry the policy, call an alternative recovery behaviour, or replan the remaining queue.

Informed task ordering and online re-planning. The current order-agnostic mode could be improved with a lightweight nearest-neighbour heuristic based on the current end-effector position and the positions of unmanipulated objects, keeping the option to have a specified task order for scenarios without an order-agnostic solution. This would improve execution efficiency without requiring a full symbolic planner. In a more advanced version, the orchestrator could monitor policy outcomes during execution and dynamically reorder or replan the remaining tasks in response to failures or changes in the workspace.

Broader task classes and richer policy architectures. The framework should be extended beyond rigid-object pick-and-place tasks. Articulated objects, deformable objects, insertion tasks, and multi-step assembly sequences would test the limits of the current subtask definitions and may require more expressive policy architectures than standard **BC**. Evaluating these more complex tasks would clarify which parts of the workflow generalize and which require redesign.

Improved task authoring and configuration tools. Although the current workflow reduces manual demonstration effort, adding a new task still requires technical configuration. Future work could leverage higher-level authoring tools, such as IsaacLab Arena, for defining task objects, subtask boundaries, success conditions, and Mimic annotations directly through the **XR** or **DT** interface. While some engineering effort will likely remain necessary for task-specific implementation and low-level configuration, such tools could substantially reduce the amount of manual programming required, moving the system closer to enabling non-expert users to create new manipulation behaviours without modifying code.

Overall, future work should focus on closing the gap between the current simulation-based prototype and a robust physical system. The most important developments are physical validation, perception-based synchronization, runtime verification, recovery mechanisms, and improved task-authoring tools. These additions would transform the implemented workflow from a controlled **DT** prototype into a more complete system for practical robot teaching and execution in flexible industrial environments.

References

- [1] Peter P. Groumpos. “A Critical Historical and Scientific Overview of all Industrial Revolutions”. In: *IFAC-PapersOnLine* 54.13 (2021), pp. 464–471. DOI: [10.1016/j.ifacol.2021.10.492](https://doi.org/10.1016/j.ifacol.2021.10.492).
- [2] Maija Breque, Lars De Nul, and Athanasios Petridis. *Industry 5.0: Towards a Sustainable, Human-Centric and Resilient European Industry*. Tech. rep. European Commission, Directorate-General for Research and Innovation, 2021. DOI: [10.2777/308407](https://doi.org/10.2777/308407). URL: <https://op.europa.eu/en/publication-detail/-/publication/468a892a-5097-11eb-b59f-01aa75ed71a1/>.
- [3] Chengxi Li et al. “An AR-assisted Deep Reinforcement Learning-based Approach Towards Mutual-cognitive Safe Human–Robot Interaction”. In: *Robotics and Computer-Integrated Manufacturing* 80 (2023), p. 102471. DOI: [10.1016/j.rcim.2022.102471](https://doi.org/10.1016/j.rcim.2022.102471).
- [4] Christine T. Chang and Bradley Hayes. “A Survey of Augmented Reality for Human–Robot Collaboration”. In: *Machines* 12 (2024), p. 540. DOI: [10.3390/machines12080540](https://doi.org/10.3390/machines12080540).
- [5] Fei Tao et al. “Digital Twin and Big Data Towards Smart Manufacturing and Industry 4.0: 360 Degree Comparison”. In: *Industrial Internet of Things: Cybermanufacturing Systems*. 2019.
- [6] Sebastian Blankemeyer et al. “Intuitive Robot Programming Using Augmented Reality”. In: *Procedia CIRP* 76 (2018), pp. 155–160. DOI: [10.1016/j.procir.2018.02.028](https://doi.org/10.1016/j.procir.2018.02.028).
- [7] Niklas Hübel and Eric Guiffo Kaigom. “Codeless, Inclusive, and End-to-End Robotized Manipulations by Leveraging Extended Reality and Digital Twin Technologies”. In: *International Conference on Human System Interaction (HSI)*. 2024, pp. 1–6. DOI: [10.1109/HSI61632.2024.10613560](https://doi.org/10.1109/HSI61632.2024.10613560).
- [8] Niklas Hübel, Siegfried Müller, and Eric Guiffo Kaigom. “Immersive Assistance System for Intuitive Robot Programming Using Mixed-Reality and Digital Twin”. In: *IEEE International Conference on Digital Twins and Parallel Intelligence (DTPI)*. 2023, pp. 1–6. DOI: [10.1109/DTPI59677.2023.10365470](https://doi.org/10.1109/DTPI59677.2023.10365470).
- [9] Uqba Othman and Erfu Yang. “Human–Robot Collaborations in Smart Manufacturing Environments: Review and Outlook”. In: *Sensors* 23.12 (2023), p. 5663. DOI: [10.3390/s23125663](https://doi.org/10.3390/s23125663).
- [10] Inês Soares, Marcelo Petry, and António Paulo Moreira. “Programming Robots by Demonstration Using Augmented Reality”. In: *Sensors* 21.17 (2021), p. 5976. DOI: [10.3390/s21175976](https://doi.org/10.3390/s21175976).
- [11] Anne Weiss and Thomas Bauernhansl. “Cobots in Industry 4.0: Current Applications and Future Perspectives”. In: *Procedia CIRP* (2021).

- [12] Ali Keshvarparast et al. “Collaborative Robots in Manufacturing and Assembly Systems: Literature Review and Future Research Agenda”. In: *Journal of Intelligent Manufacturing* 35.5 (2024), pp. 2065–2118. DOI: [10.1007/s10845-023-02137-w](https://doi.org/10.1007/s10845-023-02137-w).
- [13] Ali Ahmad Malik and Alexander Brem. “Digital Twins for Collaborative Robots: A Case Study in Human–Robot Interaction”. In: *Robotics and Computer-Integrated Manufacturing* 68 (2021), p. 102092. DOI: [10.1016/j.rcim.2020.102092](https://doi.org/10.1016/j.rcim.2020.102092).
- [14] Li Wang, Hui Zhang, and Rong Chen. “Digital Twin-Driven Human–Robot Collaboration for Complex Assembly in Industry 5.0”. In: *Robotics and Computer-Integrated Manufacturing* (2026).
- [15] Kai Arulkumaran et al. “Deep Reinforcement Learning: A Brief Survey”. In: *IEEE Signal Processing Magazine* 34.6 (2017), pp. 26–38. DOI: [10.1109/MSP.2017.2743240](https://doi.org/10.1109/MSP.2017.2743240).
- [16] OpenAI et al. “Learning dexterous in-hand manipulation”. In: *The International Journal of Robotics Research* 39.1 (2020), pp. 3–20. DOI: [10.1177/0278364919887447](https://doi.org/10.1177/0278364919887447).
- [17] Brenna D. Argall et al. “A Survey of Robot Learning from Demonstration”. In: *Robotics and Autonomous Systems* 57.5 (2009), pp. 469–483. DOI: [10.1016/j.robot.2008.10.024](https://doi.org/10.1016/j.robot.2008.10.024).
- [18] Takayuki Osa et al. “An Algorithmic Perspective on Imitation Learning”. In: *Foundations and Trends in Robotics* 7.1–2 (2018), pp. 1–179. DOI: [10.1561/23000000053](https://doi.org/10.1561/23000000053).
- [19] Ajay Mandlekar et al. “What Matters in Learning from Offline Human Demonstrations for Robot Manipulation?” In: *Conference on Robot Learning (CoRL)*. 2021, pp. 1678–1691. URL: <https://arxiv.org/abs/2108.03298>.
- [20] Stéphane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell. “A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning”. In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. Vol. 15. Proceedings of Machine Learning Research. 2011, pp. 627–635.
- [21] Ilge Akkaya et al. “Solving Rubik’s Cube with a Robot Hand”. In: *arXiv preprint arXiv:1910.07113* (2019). DOI: [10.48550/arXiv.1910.07113](https://doi.org/10.48550/arXiv.1910.07113).
- [22] Yongkui Liu et al. “A digital twin-based sim-to-real transfer for deep reinforcement learning-enabled industrial robot grasping”. In: *Robotics and Computer-Integrated Manufacturing* 78 (2022), p. 102365. DOI: [10.1016/j.rcim.2022.102365](https://doi.org/10.1016/j.rcim.2022.102365).
- [23] Joshua Tobin et al. “Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2017, pp. 23–30. DOI: [10.1109/IROS.2017.8202133](https://doi.org/10.1109/IROS.2017.8202133).
- [24] Paul Milgram and Fumio Kishino. “A Taxonomy of Mixed Reality Visual Displays”. In: *IEICE Transactions on Information and Systems* E77-D.12 (1994), pp. 1321–1329.
- [25] Ronald T. Azuma. “A Survey of Augmented Reality”. In: *Presence: Teleoperators and Virtual Environments* 6.4 (1997), pp. 355–385. DOI: [10.1162/pres.1997.6.4.355](https://doi.org/10.1162/pres.1997.6.4.355).
- [26] Karthik Subramanian et al. “Supporting Human–Robot Interaction in Manufacturing with Augmented Reality and Effective Human–Computer Interaction: A Review and Framework”. In: *Machines* 12.10 (2024), p. 706. DOI: [10.3390/machines12100706](https://doi.org/10.3390/machines12100706).
- [27] Xian Wang, Luyao Shen, and Lik-Hang Lee. “A Systematic Review of XR-Enabled Remote Human-Robot Interaction Systems”. In: *ACM Computing Surveys* 57.11 (2025), p. 273. DOI: [10.1145/3730574](https://doi.org/10.1145/3730574).

- [28] Miguel Arduengo et al. “Gaussian-process-based robot learning from demonstration”. In: *Journal of Ambient Intelligence and Humanized Computing* (2023). DOI: [10.1007/s12652-023-04551-7](https://doi.org/10.1007/s12652-023-04551-7).
- [29] C. Lauretti, C. Tamantini, and L. Zollo. “A New DMP Scaling Method for Robot Learning by Demonstration and Application to the Agricultural Domain”. In: *IEEE Access* 12 (2024), pp. 7661–7673. DOI: [10.1109/ACCESS.2023.3349093](https://doi.org/10.1109/ACCESS.2023.3349093).
- [30] Yingjun Liu, Fuchun Liu, and Renwei Huang. “Supervised Optimal Control in Complex Continuous Systems with Trajectory Imitation and Reinforcement Learning”. In: *Scientific Reports* 15 (2025), p. 19479. DOI: [10.1038/s41598-025-04417-2](https://doi.org/10.1038/s41598-025-04417-2).
- [31] Jinhao Ng et al. “A System for Retargeting Human Motion to Robot with Augmented Feedback via a Digital Twin Setup”. In: *International Conference on Control, Automation and Robotics (ICCAR)*. 2024, pp. 95–100. DOI: [10.1109/ICCAR61844.2024.10569840](https://doi.org/10.1109/ICCAR61844.2024.10569840).
- [32] Yuming Ning et al. “TS-RIL: A Two-stage Robot Imitation Learning Framework with Motion Trajectory Learning and Obstacle Avoidance in Real-world Operating Scenarios”. In: *Robotics and Computer-Integrated Manufacturing* 97 (2026), p. 103111. DOI: [10.1016/j.rcim.2025.103111](https://doi.org/10.1016/j.rcim.2025.103111).
- [33] Fuchun Sun et al. “Digital-Twin-Assisted Skill Learning for 3C Assembly Tasks”. In: *IEEE Transactions on Cybernetics* 54.7 (2024), pp. 3852–3863. DOI: [10.1109/TCYB.2024.3368148](https://doi.org/10.1109/TCYB.2024.3368148).
- [34] Enes Yigitbas and Gregor Engels. “Enhancing Robot Programming Through Digital Twin and Augmented Reality”. In: *Hawaii International Conference on System Sciences (HICSS)*. 2023. DOI: [10.24251/HICSS.2023.818](https://doi.org/10.24251/HICSS.2023.818).
- [35] Bryce Ikeda et al. “MARCER: Multimodal Augmented Reality for Composing and Executing Robot Tasks”. In: *ACM/IEEE International Conference on Human–Robot Interaction (HRI)*. 2025, pp. 529–539. DOI: [10.1109/HRI61500.2025.10974232](https://doi.org/10.1109/HRI61500.2025.10974232).
- [36] Felipe Fronchetti et al. “RoboART: Artistic Robot Programming in Mixed Reality”. In: *IEEE Conference on Virtual Reality and 3D User Interfaces Workshops (VRW)*. 2024, pp. 1192–1193. DOI: [10.1109/VRW62533.2024.00390](https://doi.org/10.1109/VRW62533.2024.00390).
- [37] Shlomi Fridman, Shahar Berenson, and Naomi Unkelos-Shpigel. “Production Line Augmented Reality Application”. In: *IEEE International Requirements Engineering Conference (RE)*. 2025, pp. 580–581. DOI: [10.1109/RE63999.2025.00069](https://doi.org/10.1109/RE63999.2025.00069).
- [38] Amanuel Ergogo, Diego Dall’Alba, and Przemyslaw Korzeniowski. “HiLoTEL: Virtual Reality Robot Interface-Based Human-in-the-Loop Task Execution and Learning in the Physical World Through Its Digital Twin”. In: *IEEE Conference on Virtual Reality and 3D User Interfaces Workshops (VRW)*. 2024, pp. 681–682. DOI: [10.1109/VRW62533.2024.00137](https://doi.org/10.1109/VRW62533.2024.00137).
- [39] Kelin Li et al. “Haptic-ACT: Bridging Human Intuition with Compliant Robotic Manipulation via Immersive VR”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2025, pp. 1084–1091. DOI: [10.1109/IROS60139.2025.11247471](https://doi.org/10.1109/IROS60139.2025.11247471).
- [40] Qiyue Wang et al. “Digital Twin for Human–Robot Interactive Welding and Welder Behavior Analysis”. In: *IEEE/CAA Journal of Automatica Sinica* 8.2 (2021), pp. 334–343. DOI: [10.1109/JAS.2020.1003518](https://doi.org/10.1109/JAS.2020.1003518).

- [41] Yingying Du et al. “Industrial Robot Digital Twin System Motion Simulation and Collision Detection”. In: *IEEE International Conference on Digital Twins and Parallel Intelligence (DTPI)*. 2021, pp. 1–4. DOI: [10.1109/DTP152967.2021.9540114](https://doi.org/10.1109/DTP152967.2021.9540114).
- [42] Shimin Liu et al. “Digital Twin Modeling Method Based on Biomimicry for Machining Aerospace Components”. In: *Journal of Manufacturing Systems* 58 (2021), pp. 180–195. DOI: [10.1016/j.jmsy.2020.04.014](https://doi.org/10.1016/j.jmsy.2020.04.014).
- [43] Chinmay Samak, Tanmay Samak, and Venkat Krovi. “Mixed-Reality Digital Twins: Leveraging the Physical and Virtual Worlds for Hybrid Sim2Real Transition of Multi-Agent Reinforcement Learning Policies”. In: *IEEE Robotics and Automation Letters* 10.9 (2025), pp. 9040–9047. DOI: [10.1109/LRA.2025.3592085](https://doi.org/10.1109/LRA.2025.3592085).
- [44] Fatih Dogangun et al. “RAMPA: Robotic Augmented Reality for Machine Programming by Demonstration”. In: *IEEE Robotics and Automation Letters* 10.4 (2025), pp. 3795–3802. DOI: [10.1109/LRA.2025.3546109](https://doi.org/10.1109/LRA.2025.3546109).
- [45] Chenxuan Liu et al. “Communication Efficient Robotic Mixed Reality With Gaussian Splatting Cross-layer Optimization”. In: *IEEE Transactions on Cognitive Communications and Networking* (2025), pp. 1–1. DOI: [10.1109/TCCN.2025.3599522](https://doi.org/10.1109/TCCN.2025.3599522).
- [46] Isaac Ngui et al. “Extended Reality System for Robotic Learning from Human Demonstration”. In: *IEEE Conference on Virtual Reality and 3D User Interfaces Workshops (VRW)*. 2025, pp. 1304–1305. DOI: [10.1109/VRW66409.2025.00298](https://doi.org/10.1109/VRW66409.2025.00298).
- [47] Joan Lario Femenía et al. “Virtual Reality Capabilities for Robot Programming by Demonstration”. In: *IEEE International Conference on Engineering, Technology, and Innovation (ICE/ITMC)*. 2025, pp. 1–8. DOI: [10.1109/ICE/ITMC65658.2025.11106660](https://doi.org/10.1109/ICE/ITMC65658.2025.11106660).
- [48] Saul Alexis Heredia Perez, Hung-Ching Lin, and Kanako Harada. “Mixed Reality and Digital Twin Interface for Robot Collaborative Tasks”. In: *IEEE Conference on Virtual Reality and 3D User Interfaces Workshops (VRW)*. 2025, pp. 471–474. DOI: [10.1109/VRW66409.2025.00102](https://doi.org/10.1109/VRW66409.2025.00102).
- [49] Francesco Semeraro, James Leadbetter, and Angelo Cangelosi. “TriHRCBot: A Robotic Architecture for Triadic Human–Robot Collaboration Through Mediated Object Alignment”. In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2025, pp. 6703–6709. DOI: [10.1109/ICRA55743.2025.11128795](https://doi.org/10.1109/ICRA55743.2025.11128795).
- [50] Yehor Karpichev et al. “Extended Reality for Enhanced Human–Robot Collaboration: A Human-in-the-Loop Approach”. In: *IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)*. 2024, pp. 1991–1998. DOI: [10.1109/RO-MAN60168.2024.10731170](https://doi.org/10.1109/RO-MAN60168.2024.10731170).
- [51] Matthew B. Luebbbers et al. “ARC-LfD: Using Augmented Reality for Interactive Long-Term Robot Skill Maintenance via Constrained Learning from Demonstration”. In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2021, pp. 3794–3800. DOI: [10.1109/ICRA48506.2021.9561844](https://doi.org/10.1109/ICRA48506.2021.9561844).
- [52] Huihui Guo et al. “Recent Trends in Task and Motion Planning for Robotics: A Survey”. In: *ACM Computing Surveys* 55.13s (2023), pp. 1–36. DOI: [10.1145/3583136](https://doi.org/10.1145/3583136).
- [53] Samia Alghamdi, Hee Young Yoon, and Yeong-Tae Song. “Digital Twins in Education: Applications, Challenges, and Future Directions”. In: *IEEE/ACIS International Conference on Software Engineering Research, Management and Applications (SERA)*. 2025, pp. 225–232. DOI: [10.1109/SERA65747.2025.11154576](https://doi.org/10.1109/SERA65747.2025.11154576).

- [54] Y. Hu et al. “Motion Planning Algorithms for Intelligent Robots: A Survey”. In: *Journal of Intelligent Manufacturing* (2022), pp. 387–424. DOI: [10.1007/s10845-021-01867-z](https://doi.org/10.1007/s10845-021-01867-z).
- [55] Ajay Mandlekar et al. “What Matters in Learning from Offline Human Demonstrations for Robot Manipulation”. In: *arXiv preprint arXiv:2108.03298*. 2021.