HAW HAMBURG

MASTER THESIS
Luca Jedelhauser

PACT: A Framework for Predictive Task Validation and Adaptive Error Recovery in Robotic Systems

Faculty of Engineering and Computer Science
Department Computer Science

HOCHSCHULE FÜR ANGEWANDTE
WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Luca Jedelhauser

PACT: A Framework for Predictive Task Validation and Adaptive Error Recovery in Robotic Systems

Master thesis submitted for examination in Master´s degree
in the study course *Master of Science Informatik*
at the Department Computer Science
at the Faculty of Engineering and Computer Science
at University of Applied Science Hamburg

Supervisor: Prof. Dr. Stephan Pareigis
Supervisor: Prof. Dr. Tim Tiedemann

Submitted on: 23 May 2025

Luca Jedelhauser

Title of Thesis

PACT: A Framework for Predictive Task Validation and Adaptive Error Recovery in Robotic Systems

Keywords

Robotics, Digital Twin, Reinforcement Learning, Sim-to-Real Gap

Abstract

Robotic systems operating in unpredictable real-world scenarios require high levels of intelligence and adaptability. This thesis presents the PACT (Predictive Adaptive Collaborative Twin) framework, which tightly integrates a real robotic system with its Digital Twin, enabling a unified workflow for training, validation, and adaptive improvement, with reinforcement learning (RL) as the core source of intelligence. The framework's practical applicability is demonstrated on a simplified real-world grasping task, confirming PACT's ability to bring training, deployment, and adaptation together holistically, while also identifying areas for future improvement.

Kurzzusammenfassung

Robotiksysteme die in unvorhersehbaren Szenarien in der realen Welt agieren benötigen ein hohes Maß an Intelligenz und Anpassungsfähigkeit. In dieser Arbeit wird das PACT (Predictive Adaptive Collaborative Twin) Framework vorgestellt, das ein reales robotisches System eng mit seinem Digitalen Zwilling verknüpft und einen einheitlichen Workflow für Training, Validierung und adaptive Verhaltensverbesserung schafft. Dabei kommt Reinforcement Learning (RL) als zentrale Methode zum Entwickeln von intelligentem Verhalten zum Einsatz. Die praktische Anwendbarkeit des Frameworks wird anhand einer vereinfachten Greifaufgabe in der realen Welt demonstriert. Die Ergebnisse bestätigen die Fähigkeit von PACT, Training, Deployment und Adaption ganzheitlich zusammenzubringen, und weisen gleichzeitig Bereiche mit Verbesserungspotenzial auf.

Contents

List of Figures

List of Tables

1 Introduction

The following section provides essential background knowledge to support the understanding of the concepts presented in later chapters. The first part outlines the core concept of Digital Twins, ROS, and Reinforcement Learning. This section is intended to provide background information for readers who are not familiar with these areas and can be skipped by those who already have a general understanding. The second part outlines the problem addressed in this work, highlights its relevance within the field, and concludes by defining the main objectives of the thesis.

1.1 Theoretical Foundation

1.1.1 Digital Twins

The concept of a Digital Twin fundamentally refers to a digital informational representation of a physical object or system that can be linked to its physical counterpart during the entire lifecycle. In the idealized concept, any information that can be gained from investigating the real system can also be received from the Digital Twin in a more efficient way. They can also incorporate information exchange between them and the real entity, making it possible to maintain and update the twin throughout their lifespan. Digital Twins have been successfully applied across a wide range of industries and domains, from simple virtual replicas of individual components to complex systems such as manufacturing plants, aircraft, and spacecraft. [5][16]

1.1.2 ROS

The ROS (Robot Operating System) is an open-source collection of libraries and tools that aid developers in building robotic applications [28]. Its modular architecture and

wide range of supported devices make it well-suited for integrating physical robotic hardware and higher-level control logic. There are many different versions of ROS, organized into two main versions: ROS1 and ROS2, each containing several subversions. The last actively maintained version of the ROS1 distribution will reach its End-of-Life (EOL) this year. To keep this implementation relevant, ROS2 (Humble Hawksbill) is used in this project [27].

1.1.3 The Reinforcement Learning Concept

The concept of Reinforcement Learning (RL), although it has gained enormous popularity in the last two decades, is a fairly old concept [10]. It is one of the three basic categories of machine learning besides Supervised and Unsupervised Learning.

The core concept is that an acting entity (agent) learns an optimal strategy through interaction with an environment, using a trial-and-error approach. The agent observes the current state of the environment, chooses an action, and receives a state-dependent reward. This exchange of information is illustrated in Figure 1.1.

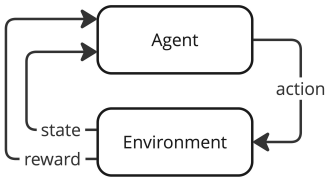

Figure 1.1: Basic RL agent environment interaction

The agent's objective is to develop a behavioral policy that maps states to actions in a way that maximizes the total cumulative reward over time. In modern approaches, this behavior is often learned using deep neural networks. RL training typically requires extensive interaction between the agent and the environment, especially when neural networks are used, resulting in the need to collect large amounts of experience data (samples). [40]

1.1.4 Reinforcement Learning Algorithms

RL has developed into a wide-ranging subject with a variety of different algorithms and approaches. With regard to this work, the most relevant core method categories are:

- Value-Based and Policy-Based
- Model-Based and Model-Free
- On-Policy and Off-Policy

Value- and Policy-Based Methods

Value-Based methods learn a value function estimating the future rewards of each action. The agent then selects actions based on the highest estimated value. These methods are effective for discrete action spaces, where a limited number of possible actions can be evaluated efficiently. [40]

A very well-known example of value-based RL is Q-Learning.

Policy-based methods directly learn a policy function, mapping states to actions, without the need for a discrete search. This makes them particularly useful for continuous action spaces where the number of actions is essentially infinite, such as the direct joint angle control used in this work. [40]

Common examples of policy-based algorithms are Proximal Policy Optimization (PPO), Soft Actor-Critic (SAC) or Trust Region Policy Optimization (TRPO).

Model-Based and Model-Free Methods

Model-based methods, as the name suggest, create an internal model that approximates the environment's dynamics. This model is used to anticipate future situations following chosen actions. These methods therefore emphasize planning, while model-free methods learn purely from trial-and-error interactions. [40]

Having a model can significantly reduce the samples needed from the actual environment. However, a good model accuracy for a complex environment is often very hard to achieve.

On-Policy and Off-Policy Methods

While on-policy methods always use the current version of the policy to interact with the environment, requiring an active interaction with the environment for each policy update, off-policy methods can use samples collected by either previous versions of the policy or even an entirely different one. [40]

This will become mostly relevant in this context in the form of off-policy methods that use replay-buffers, where past experiences are stored and sampled from to increase the sample efficiency.

1.2 Motivation & Problem Definition

1.2.1 Why Robots Should Be Intelligent

In the broader sense, robots date back many decades. Systems that move and function autonomously have always been fascinating to humankind. Their appearance ranges from mere science fiction to actual real-world applications that actively shape and improve our daily lives. Industrial robots in particular have transformed manufacturing by automating repetitive, hazardous, and labor-intensive tasks. However, their awareness of the environment in which they operate is generally very limited. Their movement trajectories are precisely predefined, and their workspaces are carefully designed to minimize variability, eliminating the need for adaptation or reasoning. Also, humans, as unpredictable agents, have to be separated from the effective range of these robots to ensure safety. [4]

This contrasts strongly with most real-world applications. Aside from isolated and standardized production lines, parameters change constantly, future states are highly unpredictable, and even familiar situations and behaviors can lead to unknown outcomes, presenting challenges for which the robot has no immediate solution. [32]

To actively participate in and adapt to complex real-world environments, robots need to have a significantly higher level of intelligence.

1.2.2 How to Make Robots More Intelligent

Machine learning methods show great potential in addressing these challenges, allowing agents to acquire robust strategies for managing diverse situations and tasks. As AI-driven robotic systems continue to become increasingly intelligent, they achieve progressively higher levels of autonomy, even in unpredictable environments. [6][32]

This, however, raises the question of what is meant by intelligence in this context. Perez et. al. define it in the following way: "This level of intelligence can be measured as the capacity of predicting the future, either in planning a task, or in interacting (either by manipulating or navigating) with the world."[32].

Despite many Digital Twin applications still lacking a strong connection between reality and virtuality [12], Digital Twins directly linked to the real world offer significant potential to hit exactly this spot. Especially when combined with machine learning, they can elevate robotic intelligence by enhancing the system's predictive and adaptive capabilities [15][17]. Maintaining an up-to-date virtual representation of the world enables safe exploration of actions, provides insights into how actions lead to future outcomes, and supports adaptation to changing conditions. This aligns closely with the need for robots to continuously adapt to new situations while leveraging predictive capabilities to make informed decisions.

However, prediction and adaptation alone are not enough. Robots must also be able to perceive and interact effectively with the real world to facilitate intelligent behavior. This is where machine learning methods show their strengths. Much like humans, robots rely on sensory feedback to understand their current state and guide their behavior. Without detailed external supervision, intelligent robots must learn from their own interactions to develop an understanding of how actions lead to specific outcomes. Reinforcement Learning is a promising candidate for solving exactly such challenges, where the learning agent adapts its actions based on evaluating the current state and predicting which actions will lead to better results. [6][38]

1.2.3 Robotic Reinforcement Learning: Curse and Blessing

In more theoretical applications, RL has made international headlines, beating human experts in all kinds of games [53][49][37].

However, in actual real-world applications like robotics, this performance is quite hard to replicate. Although much of what the classical industrial robot lacks in intelligence, ability to solve complex problems, and robustness to situational changes is addressed by RL, it also brings its own difficulties. Some are unique to the intersection of RL and robotics, while others are inherent to the technique itself. [55][20][11][2]

A fundamental conflict between Reinforcement Learning and robotics is the trial-and-error paradigm that RL follows. This comes with engaging in “bad” or suboptimal decisions to learn that certain trajectories lead to undesirable outcomes. While this is generally acceptable in purely computational experiments, the reality of physical robots is often unforgiving, and actions can lead to injury or damaged equipment. Also, running a large number of physical trials is usually very expensive, making the thousands or even millions of samples a RL agent needs to learn a behavior unfeasible in robotics. Therefore, researchers tend to use simulation environments to pretrain these algorithms. These environments are essentially replicas of the real world used as pretraining ground to move the training procedure off the physical platform, eliminating physical risks, and reducing wear on hardware. [35]

Once the agent has developed a sufficiently robust behavior in the simulation, transferring it to a real robot is the next step. However, this transition is generally not straightforward. Differences between the simulated and real-world dynamics (commonly referred to as the “Reality Gap” or "Sim-to-Real Gap") can degrade performance and require additional effort to eradicate. [35]

Thus, even though RL may initially appear to be the ideal approach for making robots highly intelligent and enabling human-like learning, achieving this goal comes with significant challenges. Overcoming these obstacles requires additional procedures that balance the need for learning efficiency, exploration, safety, and real-world applicability — a challenge that remains the core of developing real-world-RL.

1.2.4 Summary

Deploying robots in any complex real-world scenario requires them to be highly intelligent, both predictive and adaptive, to effectively handle dynamic and unpredictable environments.

Digital twins, closely connected with the real world, offer vast potential in elevating the intelligence of robotic AI agents by providing predictive information, allowing safe exploration of actions, and aiding with the validation and adaptation of strategies in dynamic environments.

Reinforcement Learning shows great capabilities for solving complex tasks with human-like learning and behavior, but relies on an extensive number of training iterations including dangerous states and often requires sophisticated support procedures to ensure stable and efficient convergence. Increased safety and faster learning are possible through simulated pre-training, also aligning well with the integration of a Digital Twin. Although this introduces one more challenge to be dealt with: the *Sim-to-Real Gap*.

1.3 Objective

This work aims to tightly couple a real robotic system with a Digital Twin to create a close collaborative relationship that paves the way to increase the robot's intelligence along the previously quoted definition.

The Digital Twin is intended to closely match the real setup and, through the framework, provide functionalities for prediction, interrupt detection, and behavior adaptation procedures. Both the real and simulated domains are aimed to be connected through a seamless data exchange, allowing states and commands to be transferred reliably between the two.

Further, the framework is built to integrate RL as the source of intelligence and also work as a pretraining simulator. Therefore, the simulated environment and the real robot control will be implemented as RL environments and follow the gymnasium standard to allow the seamless integration of reinforcement learning algorithms. To enhance RL pretraining performance, challenges associated with RL are addressed at their core by integrating support strategies directly into the framework.

This concept is presented as the *PACT (Predictive Adaptive Collaborative Twin)* framework: a cohesive system for training, validation, and real-time collaboration.

In order to provide a concrete use case for evaluating PACT and to address arising challenges in a practical manner, a complex real-world robotic task is selected: *grasping*. This task is not only a fundamental problem in robotics but also a challenging one,

requiring adaptability and predictive capabilities to solve safely and effectively. Grasping is treated here as a closed-loop continuous control problem, where the agent continuously adapts its behavior based on the latest observations from the environment.

However, the grasping task is only used as a means to an end to demonstrate the challenges and the proposed solution. To keep the training times and complexity manageable with the limited hardware resources available, the task and the environment will be strongly simplified and constrained.

The PACT framework is first introduced at a conceptual level and then implemented on a real robotic research platform. Therefore, a Digital Twin of the robot is created and connected to its real-world counterpart using the PACT concept. As the source of intelligence, RL is integrated into the framework and applied to address the grasping task. Following initial practical tests, both the suitability of the framework and the learning procedure are evaluated. Finally, limitations are discussed, and potential directions for future development are outlined.

2 Related Work

The related work is first reviewed with respect to collaborative Digital Twin implementations found in frameworks related to PACT, and followed by similar hardware, software, and grasping task setups.

2.1 Predictive Adaptive Collaborative Twin Frameworks

Early applications of Digital Twins in robotics treated simulation largely as a sandbox for training. Agents were trained extensively in simulators, using procedures like domain randomization to narrow the Sim-to-Real Gap, before being deployed on physical robots [42]. From there on the role of the Digital Twin largely ended. Recent trends shift towards more integrated couplings between twin and physical robot. An approach called *Dynamics-Aware Hybrid Offline-and-Online Reinforcement Learning* (H2O) brings imperfect samples collected from a simulation and real experience very close together has been proposed by Nui et al. [21]. However, the algorithm learns from an offline real dataset combined with online simulation roll-outs while tracking divergence from real dynamics. The real dataset here is fixed, while unlimited samples can be obtained from the simulation. So, contrary to PACT, it does not run a simulator in parallel to the real world, but holds a fixed replay buffer with real samples while running in the simulation.

Similar to that is the Dreamer concept which has also been applied to real-world robotic applications by Wu et al. [51]. Here instead of using a simulator, a world model is learned from real experiences that are continuously added to a buffer. The agent learns using "imagined roll-outs" in this world model in parallel to interacting with the real world. As in PACT the agent runs in the real domain but learns from the world model in parallel instead of switching from the real world to simulated runs as PACT does.

This real to sim switching behavior with the associated transfer of states was also addressed in more recent studies. Often a real-to-sim-to-real pipeline is discussed, which

shows strong parallels to PACT's idea of utilizing the sim by taking a snapshot of the current environment state and creating a virtual mirror of reality for efficient training before switching back. [43][31][52]

Some approaches also use a looping structure to iteratively go through the real-to-sim-to-real process [30]. But the main focus has yet to be placed on the looping training workflow itself.

In their real-to-sim-to-real approach, Zhu et al. mention "photorealistic and physically interactive "Digital Twin" simulation environments" applied to a legged locomotion task [58]. They successfully transfer objects and their position via mesh extraction on the basis of simple RGB images into a simulator which is then used to train parallel agents on a GPU. Their approach, however, conceptually differs from PACT in the zero-shot sim-to-real transfer. PACT aims to keep the simulation available during execution, to adaptively retrain and validate on critical situations. It does follow a zero-shot-like strategy, as there is a direct transfer to real after training, but PACT uses adaptive switch-backs in an attempt to improve policy adaptability and robustness.

A research paper that comes closest to the PACT concept was published by Sun et al. [39]. The proposed concept uses an interrupt-driven retrain framework by transferring real-world states from a depth camera to a simulation. It utilizes a Digital Twin as an initial pretraining ground as well as a monitoring, controlling, and optimization tool for self-improvement. As this shares basically the same core concept, this paper will be analyzed in more detail in the following.

According to the workflow illustrated by Sun et al., the state of the physical world is used to initially synchronize the Digital Twin with the real world. The Digital Twin will then attempt to perform the task in the simulation using a simulation pretrained policy. During this execution, collisions are detected and upon detection, the interrupt state is used to resume training the pretrained policy until convergence. Once the task is successfully and safely performed by the Digital Twin, this framework will perform a zero-shot sim-to-real transfer assuming the same outcome in the real world. During this execution, no collision tracking is mentioned.

At this point, the publication leaves some questions unanswered that this work aims to address. The Sim-to-Real Gap is initially mentioned as a research area but is not addressed regarding their own implementation and is essentially assumed to be non-existent. If and how issues after transferring their policy are detected or handled is not

further dealt with. Other topics such as sampling efficiency are addressed superficially, noting that resume training requires fewer training steps than initial pretraining, but are not discussed further. Also, the pretraining section is not mentioning any efficiency- or robustness-enhancing procedures.

PACT aims to continue the twin collaboration beyond the simulation, making it adaptive not only to interrupts within the digital representation but also to unseen and unpredictable situations in the real world. Furthermore, this work addresses known issues like sample inefficiency and sim-to-real transfer and integrates processes for mitigation early on into the architecture.

2.2 Deep Reinforcement Learning in Visual Robotic Grasping

To find research that can be compared to this implementation regarding the robotic platform and chosen task, the setup has to be dissected into its sub-components. Both together belong to a very specific sub-area, assigned to various overarching categories. Regarding the robotics part, it belongs to the category of robotic manipulation, which also covers grasping tasks. As visual information is used to detect the object to be grasped, it belongs to vision-based robotic manipulation. To control the grasping maneuver, one can use open-loop control, by predefining a fixed sequence of actions, or closed-loop control by continuously listening to sensor feedback and adjusting actions in real-time, which is used here. The robotic subcategory is therefore: vision-based closed-loop robotic manipulation.

The algorithm used to actually solve the task is a subfield of Reinforcement Learning that uses deep neural networks (also called "deep RL" or "DRL"), which is commonly used for autonomous robotic manipulation procedures due to their great capabilities, particularly for complex and high-dimensional problems. [41]

A term to classify this implementation precisely would therefore be “vision-based closed-loop deep Reinforcement Learning robotic manipulation”. This term was used to find research on matching setups using very similar approaches. However term "manipulation" was specified further to "grasping" as this made search results more specific, while still providing enough results.

The research was considered with the proviso that this work does not claim to solve the grasping task completely and robustly, but uses it to deal with the emerging challenges. Most of the mentioned publications on this topic go far beyond the scope of this paper. Nonetheless, many of the challenges addressed here also appear in advanced implementations of the scientific field.

Influential research was published by Zeng et al. [54] in a collaboration of Princeton University, Google and MIT. They were able to create behavior that forms synergies between pushing and grasping with a manipulator on tightly packed blocks. Their setup also consists of a six-joint UR5 robotic arm and a two-finger RG gripper together with an RGB-D camera which is, however, statically mounted on a tripod. The learning agent makes use of a model-free off-policy DRL algorithm known as Deep Q-Learning or Deep Q Network (DQN). Unlike many other approaches, which separate object detection and control with separate CNNs like YOLO [56][36][1], Zeng et al. follow an end-to-end approach, feeding the network with image data and directly returning primitive actions. These actions are, however, abstracted to e.g. pushing with one of 16 directions the network selects from. This differs from a genuine closed-loop concept, where feedback is continuously integrated during action execution. But once an action is fully executed, the system uses the most recent sensor data to select the next one, making it similar to closed-loop at least on a higher level.

Another impressive project has been done by Kalashnikov et al. in a joint work with an interdisciplinary team of Google, X-Team and the University of Berkeley. They follow a very similar approach with a robot manipulator, two-finger gripper, and a statically mounted camera but even dispense with the depth image functionality. To learn the grasping behavior they modified a DQN algorithm called QT-Opt, enabling the output of continuous actions and making it more stable and usable for large amounts of data. Notably, they use very sparse reward signals, only returning a value of one if the object is lifted and zero in any other situation. Nonetheless, the agent develops very intelligent emergent behaviors like probing occluded objects, repositioning them for better grasp success, or re-grasps when objects slip. To achieve this performance, however, they made use of massive amounts of real-world data. Seven robots were run for about 800 hours collecting samples over the course of four months. To boost the initial learning progress, a scripted exploration policy was used to guide the agent towards reasonable grasp actions.

A more recent study done by Zheng et al. [56] addresses the grasping problem on a much smaller scale, with a setup very close to the one used in this work. They use an RGB camera mounted on the manipulator above the two-finger end-effector. The grasping task is reduced to simple cube objects with different colors but otherwise equal properties. The detection and gripping of cubes was divided into two tasks, with tracking performed using YOLO v5 and gripping using DRL. The choice of the RL algorithm was made on the basis of an experimental performance comparison between three model-free variants. Two belonged to the on-policy (PPO and TRPO) and one to off-policy category (SAC). PPO was producing the best results for the given task and was, according to Zheng et al., able to reach a 100% grasping success rate in real physical experiments. However, their research was somewhat biased towards object recognition and less focused on the grasping procedure.

Generally, there seems to be a shift in studies performed in the last years towards setups that avoid massive data collection or the need for high-performance hardware by incorporating guidance strategies. For example, Chen et al. [1] make use of a two step process, where the objects are fixed in place at first, and after the agent is able to grasp them robustly, they are distributed randomly. There are also more sophisticated strategies like the GloCAL algorithm proposed by Kurkcu [13], which is able to automatically create a curriculum of subtasks that decreases training time with increased robustness of trained grasping policies.

3 Concept

This chapter introduces the core elements that make up the contribution of this thesis. Each component is presented at a conceptual level to provide a clear understanding of the underlying principles. The following implementation chapter will then build on this foundation by detailing how these concepts were realized in practice and how they interact within the overall system.

3.1 The PACT Framework

3.1.1 What is PACT?

At its core, PACT uses a simulated Digital Twin as an active companion that collaborates with a real-world robot controlled by machine learning. This twin runs in a GPU-capable simulation and can be instantiated. This means multiple copies can be created on demand with either identical or individual parameters and then be simulated in parallel. This allows to randomize efficiently around parameter values (mass, friction, actuator characteristics, etc.) or states and confront the agent with a variety of similar yet unique situations. Also, this enables RL training runs to collect dozens of samples at a time with a random parameter distribution. Further, this randomization is a common strategy to reduce the Sim-To-Real Gap and make policies more robust [42].

The Digital Twin serves as a training, validation, and prediction environment where the agent can act and get an estimation of what will happen in the real world as a result of its behavior. Unlike traditional simulation-based pretraining, PACT stays side by side with the robot beyond the first training procedure into the real-world execution.

Successful task execution is continuously checked and also dangerous or unwanted states are tracked and, upon encounter any domain, interrupts allow the real-time use of adaptation procedures, which can be validated with the Digital Twin before applying them

back to the real world. This increases the probability that the adapted behavior is actually able to handle or recover the critical situation in the real world gracefully. Therefore, once these situations are detected in the real world, the current state is captured using the available observation data and is then transferred to the Digital Twin.

Adaptation can take various forms, depending on the nature of the task and the available mechanisms. In this work, it will be evaluated whether reinforcement learning can adapt to such critical situations by continuing policy training with small randomizations around the interrupt state using parallelized training.

Once the new adapted behavior is able to solve the task in the simulation, the robot continues its execution with ongoing interrupt detection. This workflow is illustrated in Figure 3.1.

At the starting point of the flow, this framework presupposes a behavior that is capable to some extent. While it is not necessary that it has great performance, a very unstable and incapable behavioral policy will cause many switches back and forth between the real robot and the simulation. Also, the weaker the policy is, the more likely it is to maneuver the robot into states that require very complex recovery strategies.

Since the twin is already in a simulator that also supports efficient training, it can directly be used to pretraining the policy before the described workflow begins.

The red exit element at the top of Figure 3.1 should ideally never be reached, assuming the interrupt conditions are designed to cover all possible outcomes that do not lead to successful task completion. It serves as a fallback indicator for cases where the system configuration is incomplete or fails to account for certain failure states.

There is, however, one inherent pitfall with this looping structure. It assumes that the state of the real world can be represented accurately enough to resemble every problem in the simulation. As no training is done on the real hardware, a problem related inaccuracy in the process of replicating the situation in the simulation will make it impossible for the algorithm to solve it. In this case, the Digital Twin repeatedly comes up with solutions in the simulated world that are not helpful in reality. PACT then enters an infinite loop of retraining runs that always fail to validate in the real world.

There are two main potential causes for this effect:

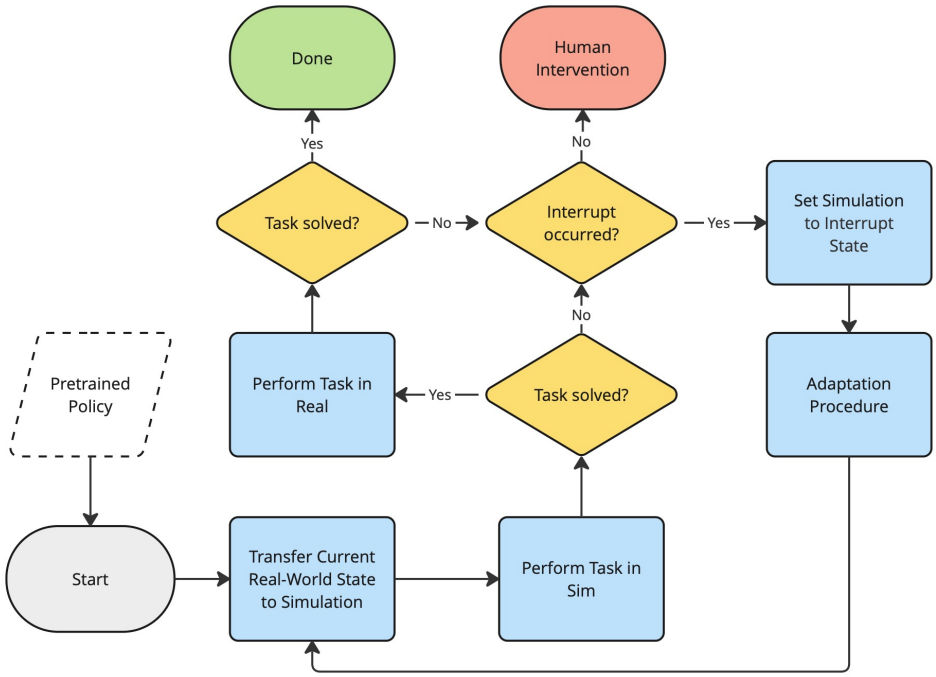

Figure 3.1: Workflow of the PACT Framework

The Sim-to-Real-Gap: Discrepancies between simulated and real-world physical parameters (e.g. friction, object dynamics, actuator characteristics) that alter the relationship between actions and observations and are not covered by domain randomization, making learned policies ineffective when transferred.

State Capture and Representation Errors: Capturing the current state of reality relies on sensors that introduce noise, as well as calculations to transform the raw data into state representations, adding another interference factor.

3.1.2 PACT Architecture and Implementation Concept

To realize PACT, the real robot and its Digital Twin have to be tied together closely. A cooperation between them presupposes that reality is interconnected with the simulation. However, the two parts are fundamentally different, not only in their design, but also in the way they are controlled and observed. Additionally, they have to coexist and be able to exchange information so that one can benefit from the knowledge of the other.

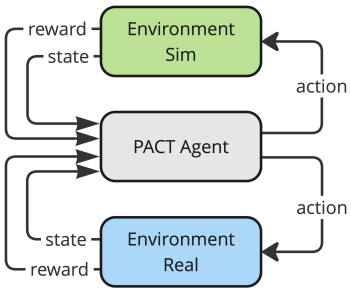

Figure 3.2: PACT RL agent environment interaction

The following concept makes use of increasing abstraction from the (simulated) hardware towards a central unit which then accesses both parts in a very similar fashion. It can send the same action command to both the simulation and the real robot and receive an observation in exactly the same format, regardless of which one of them it was sent to. At this level, both environments appear to the algorithm like a black box following the basic RL environment interaction concept shown in Figure 1.1, but extended by an additional environment as illustrated in Figure 3.2.

To achieve this layered abstraction, this implementation will start at the hardware level and then work towards the central unit. In Figure 3.3, a simplified architecture overview

is shown. In this view, the implementation would be a top-down approach. The necessary steps will be described thoroughly in the following.

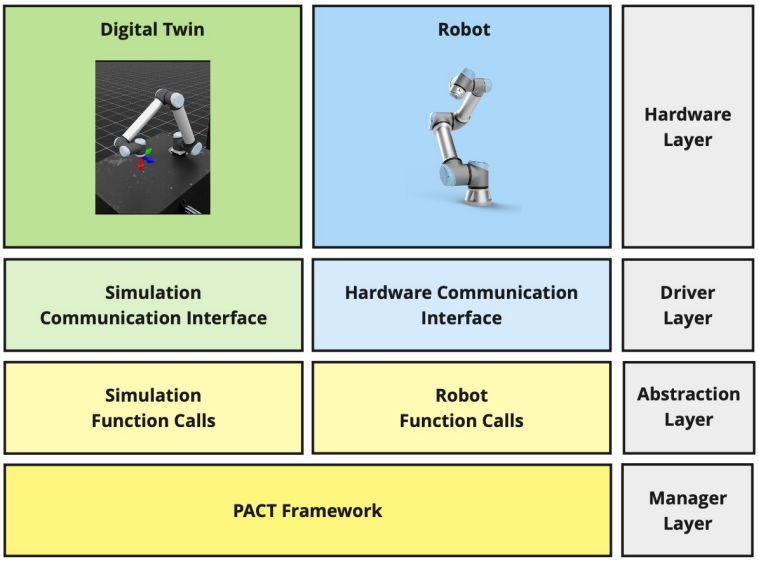

Figure 3.3: Simplified Architecture of the Development Platform

Controlling the (Simulated) Hardware

For actuators and sensors there has to be a "drivers level" which allows pushing action commands to the (simulated) hardware, as well as receiving the data collected by the sensors in a processable format. In the case of real hardware, this takes the form of a piece of software provided by the manufacturer, which in robotics typically exposes ROS interfaces such as topics, services, or action servers that allow sending commands and receiving state feedback. In simulation, these functionalities are usually integrated into the product as an interface that can be accessed programmatically.

Below that, a second layer has to use these drivers, essentially initiating all commands. This part needs to be developed with reinforcement learning in mind, since the algorithm has to finally send steering commands to and receive observations from the (simulated) robot using this interface layer. To comply with common standards, the best approach is to allow gymnasium-like function calls or ideally leverage existing gymnasium wrappers [44].

Making RL Training Possible

Having this foundational setup, the RL concept is the next logical step. Here, fundamental decisions have to be made about which control signals are used, which sensor information is important, what kind of reward structure should be used, and how complex or high-dimensional the action and observation space should be.

This is not only a question of RL design, but also of how the system can be boiled down from a highly complex robotic setup to a few numbers that are exchanged between the environment and the agent so that it can solve the task. The actions have to be designed in a way that ensures the agent can navigate precisely through the environment, but also allows that the algorithm quickly understands how to set the values to reach a goal.

The same goes for the observations. Ensuring that the system provides sensory information covering the key environment states is crucial to enable an RL algorithm to learn and solve the task effectively. But if the observations are too complex, the number of possible states in the environment becomes too large and the extraction of meaningful information gets increasingly difficult, which prevents the algorithm from learning effectively.

Since this is a complex topic of its own, this will be dealt with separately in section 3.4.

Incorporating Support Strategies into the Learning Concept

The simulation environment is where most of the common RL robotics challenges have to be addressed. As the twin is not only a source of information and validation, but initially used for pretraining and afterwards also for adaptive retraining procedures, all training enhancing strategies have to be incorporated here. That includes procedures for sim-to-real robustness, sample efficiency during training, and the encouragement of intelligent behavior. This will also be discussed further conceptually in subsection 3.4.2.

Laying the Foundations for PACT

Below all that there has to be a manager layer, the central part that makes use of the abstraction layer of either real or sim depending on the state in the flow. It has to hold all important states of both environments, track interrupts, and be able to make the correct decisions according to the PACT workflow.

With this architecture, the specifics of the real and simulated setup get aligned through the layers, so that the manager can seamlessly switch between real and sim without changing the way it communicates with the respective environment.

Pretraining the Policy

The next step is to pave the way towards actually using PACT for a real-world task.

To enter the PACT workflow with a capable policy, pretraining is conducted. This is also the first hands-on acid test of the learning environment. Here, mitigation strategies come into play, for the sake of efficiency itself, but also for the practical reason that the available hardware is limited and the training time would otherwise be prohibitively long. Driving an agent to converge to a global optimum is a science of its own. This work will therefore only scratch the surface but follow a best effort approach to create a sufficiently capable policy,

Putting PACT into Action

Once the policy is capable of performing basic task-solving behavior, the first sim-to-real transition is initiated, putting the robustness of the learned policy to the test. Then all elements of PACT can be assessed directly in practice with an actual physical platform. To evaluate the applicability, several factors will be reviewed which are listed below as key success factors of the framework.

3.1.3 Key Success Factors

- Is the Digital Twin an accurate replication of the real robot?
- Does the system capture real-world states with sufficient accuracy?
- Are interrupts detected reliably?
- Is the system able to switch correctly between the domains?
- Is the Digital Twin also capable to provide a sufficient pretraining ground?
- Can dynamic adaptative RL retraining make the policy more capable?

3.2 Baseline Hardware and Software

In the following sections, the basic components of the robotic setup and its Digital Twin are introduced together with the functionalities needed to realize the grasping task.

3.2.1 Real Robot Setup

The robotic setup is part of a mobile manipulator combination with a clearpath husky mobile base. However, in this work, no base movement is required and it will therefore be disabled entirely. The two main components of the manipulator are a UR5 robotic arm from Universal Robots and the attached RG6 gripper from OnRobot. Mounted on top of the gripper is an Intel RealSense D435 RGB-D camera. The specifics of the individual parts are outlined in the following.

UR5

The UR5 belongs to the Cobot category of robot manipulators, meaning it has built-in safety features enabling it to work within human range without safety barriers [47]. It can be equipped with a variety of different end-effectors, making it very versatile. Further, it is ROS2-compatible and therefore integrates well with state-of-the-art robotic setups. The six arm joints are numbered from the base to the tool with IDs (zero to five). The first three bigger joints can manage torques up to 150 Nm and the remaining three smaller ones limit at 28 Nm [45]. These are, however, maximum limits which should be avoided during operation.

Gripper

The RG6 is a two-finger gripper with a maximum gripping stroke of 150 mm and a maximum gripping force of 120 N. It has many advanced build-in capabilities like grip loss detection or width measurement. [26]

Unfortunately, its drivers are not as actively maintained as those of the UR5, and at the time of writing, there are no ROS2-compatible drivers available. This makes it difficult to fully utilize the functionalities of the gripper in this project, which heavily relies

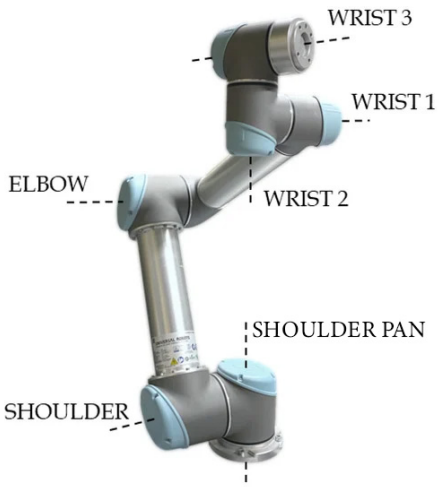

Figure 3.4: Joint names of the UR5 arm. Source: Adapted from [33]

Joint Name	ID	Max Torque (Nm)
Wrist 3	J5	28
Wrist 2	J4	28
Wrist 1	J3	28
Elbow	J2	150
Shoulder	J1	150
Base	J0	150

Table 3.1: UR5 joint names, IDs, and maximum torque values.

Figure 3.5: RG6 two finger gripper from OnRobot. Source: [26]

on ROS2. There are however workarounds to access basic functionalities, as discussed further in subsection 4.4.3.

Intel RealSense D435

The RealSense D435 is widely used and specifically well suited for robotic applications. It is a lightweight, wide range stereoscopic RGB-D camera that comes with a comprehensive software development kit (SDK). It can sense distances up to 10 m with up to 90 frames per second. Also, it features ROS2 wrappers that allow seamless integration of real-time depth information into existing modern ROS infrastructure. Due to its small size and low weight, it can be easily mounted on the tool-end of the UR5. [9]

Figure 3.6: Intel RealSense D435. Source: [9]

3.2.2 Hardware Control Strategy

The desired control procedure of the arm is a continuous closed-loop control. With the UR5 this can be achieved by either setting joint motor velocities using a velocity controller or directly setting angles with a position controller. For both approaches, there are driver implementations available through the official Universal Robots ROS2 Driver [46].

With regard to the Sim-to-Real Gap, control by absolute angle values seems more promising as this prevents angles from diverging due to e.g. delays in communication. But retrieving absolute angles directly from a neural network will produce unrealistically large changes. Thus, the network should only output small directional commands telling the joint motors in which direction (by providing a corresponding sign) they are supposed to adjust the position and in what magnitude. Afterwards, these commands are translated into absolute angles for the UR5 to move to. This also enables a stronger separation between states and actions than if the policy were to issue angles.

To allow smooth control, the UR5 has to set these angle values at a fixed rate, with the adjustments calculated according to the steering commands in real-time.

The angles are adjusted by adding a small angle delta $\Delta\phi$ at each update cycle. The size of the maximum angle delta per update is dependent on the maximum allowed linear velocity v_{max} per joint, the magnitude of the algorithm's steering command for this joint c_{joint}, and the update frequency f_{update}. Linear velocity is used because, in this context, the upper limit is often a safety-relevant factor, depending on the link length r, which sets how fast the outer end of any link (e.g. the end-effector) can move [48]. Since typically the desired speed of movement during the execution of this task is way below the maximum value, a scaling factor $\alpha_{vel} < 1$ is applied to adjust for practical speeds.

With normalized commands $c_{joint} \in [-1, 1]$, the delta is calculated as follows:

$$\Delta\phi = \alpha_{vel} \cdot c_{joint} \cdot \omega_{max} \cdot \Delta t \tag{3.1}$$

Above the angular velocity ω is used to provide an equation that is more intuitive, but in practice, to include the remaining described parameters, $\omega_{max} = \frac{v_{max}}{r}$ is substituted into the equation and Δt is replaced with $\frac{1}{f_{update}}$. This results in the final equation:

$$\Delta\phi = \alpha_{vel} \cdot c_{joint} \cdot \frac{v_{max}}{r \cdot f_{update}} \tag{3.2}$$

Due to the lack of ROS2 support, the gripper is forced to have a simpler control strategy. It can either open or close fully. This process is triggered by a boolean check of a single value gripper command, that sets the binary state $s_{gripper}$ by checking if the command $c_{gripper}$ is negative or zero for a closed state, else for an open state.

$$s_{gripper} = c_{gripper} > 0 \tag{3.3}$$

3.2.3 Cube Detection Algorithm

In this work, unlike in typical grasping implementations, the detection of the cube to be grasped is not implemented via a CNN like YOLO. For the sake of simplicity, the object detection problem is reduced to a color detection problem. The cube is set to have a very distinct color that clearly distinguishes it from its surroundings. This way, a simple threshold-based color mask can be applied to the RGB image of the camera, extracting areas with a certain range in color values. Then smaller chunks of pixels are filtered out to avoid detecting color noise, and the largest of the remaining areas is assumed to be

the cube. The position of this remaining area on the sensor plane is then used for further processing.

Using the detected cube position on the image, the distance z between camera and cube can be accessed from the RealSense depth image. To also get the 3D coordinates of the cube as a fixed reference, a backward perspective transformation is applied.

The base for this calculation is the more common forward perspective transformation [29]. The following equations are used to map a point in the 3D space onto a camera's image plane.

$$u = f_x \cdot \frac{X}{Z} + c_x \tag{3.4}$$

$$v = f_y \cdot \frac{Y}{Z} + c_y \tag{3.5}$$

Here, f_x, f_y, c_x and c_y are camera intrinsics that can be obtained directly from the RealSense via the SDK. The f values represent the camera's focal lengths in pixels, and c describes the principal point of the camera, which is usually in the center of the image. Both u and v are the 2D pixel coordinates on the image plane, originating in the top left corner. A visualization of these values is shown in Figure 3.7.

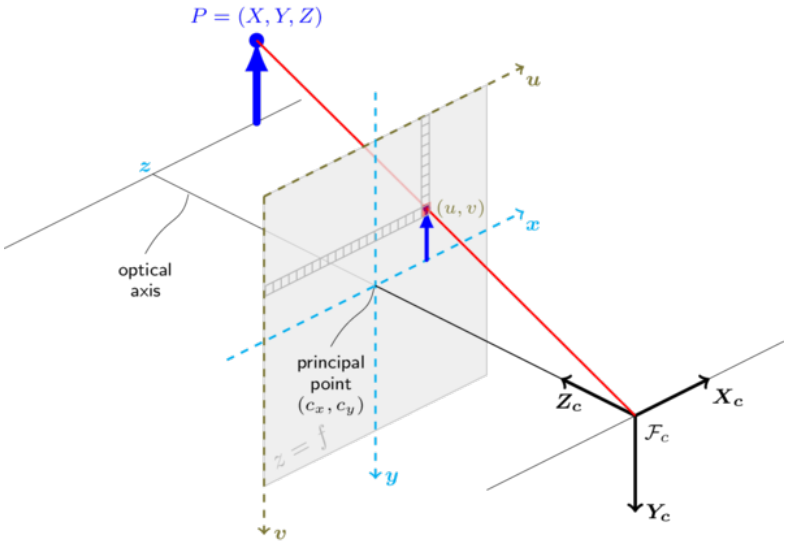

Figure 3.7: Perspective transformation model for a pinhole camera located at the origin F_c, capturing the point P in the 3D space, through the sensor plane visualized as the gray square. Source: [29]

But in this case, the transformation is turned around to calculate the 3D coordinates from the 2D sensor coordinates. As u, v and Z are detected by the RGB-D camera, this can be achieved by solving both equations for X and Y respectively.

$$X = \frac{u - c_x}{Z} \cdot Z \tag{3.6}$$

$$Y = \frac{v - c_y}{Z} \cdot Z \tag{3.7}$$

But the resulting position is relative to the dynamic position and rotation of the camera. To get coordinates from a static reference point, the calculated values are transformed to the coordinate frame of the environment's origin (called world frame in the following). This is done by applying a transformation, which consists of a rotation and a translation:

$$P\text{world} = R_{\text{cam}}^{\text{world}} \cdot P\text{cam} + T_{\text{cam}}^{\text{world}} \tag{3.8}$$

Here, P_{cam} represents the 3D point in the camera frame, while $R_{\text{cam}}^{\text{world}}$ defines the rotation matrix that transforms points from the camera frame to the world frame. The translation vector $T_{\text{cam}}^{\text{world}}$ describes the position of the camera origin relative to the world frame. Finally, P_{world} denotes the transformed 3D point in world coordinates.

3.2.4 Digital Twin

This section is about mirroring the real-world into the simulated domain, building an accurate Digital Twin of the robotic setup. Some parts of the concept can be applied across both domains, while others have to be developed from scratch. The cube detection algorithm as well as the calculation of $\phi\Delta$ are domain independent. The rest of the components need to be built along and adapted to their real counterparts.

Simulator

A fundamental decision that determines all subsequent steps is the choice of simulation software.

While there are many robotic simulators available, selecting the appropriate one depends on multiple factors, including computational performance, physics accuracy, integration with RL frameworks, and compatibility with real-world robotic systems.

For this project, the selection of NVIDIA IsaacLab provides significant advantages over more traditional simulators like MuJoCo, Gazebo or PyBullet.

IsaacLab goes back to a preview release called IsaacGym which was introduced by NVIDIA in 2021 [14]. It leverages GPU acceleration explicitly for high performance Reinforcement Learning with a capable physics backend [25]. It executes both the physics simulation and the RL training computations directly on the GPU, removing the need to constantly transfer data between CPU (for physics) and the GPU (for training RL). This concept allows massive parallelization of environments on a single GPU, shortening training times by two to three times compared to conventional simulators. The proposal demonstrates the performance by training complex robotic tasks and showcasing the accuracy of the physics engine with sim-to-real transfer.

IsaacGym has then been replaced by IsaacLab, which is a more refined development that aims to provide an easy-to-use, open-sourced interface for the development of custom robotic learning environments [24].

But IsaacLab is not one tool that does it all, it relies on two sublayers. IsaacSim resides directly below IsaacLab and is a robot simulation toolkit, with additional features like ROS support and import capabilities for robot description files. The base layer is NVIDIA Omniverse, which covers all the core functionalities like physics simulation and rendering. The resulting layered architecture inherits the capabilities from the lower layers and further adds functions on top that then make IsaacLab a great fit for robotic RL [24].

Simulating the Robot

In IssacLab, robots are defined as articulation assets [22]. They describe a collection of objects that are subject to physics calculations, connected by joints. Joints of articulations can be connected to actuator instances that are accessible through articulation function calls. Articulations not only allow replication of physical properties of a robot, but also of dynamics of motor control and steering commands.

Building robots can either be done from scratch or by using IsaacSim's tools to assemble them from existing parts. Some robot elements are available within IsaacSim libraries,

but it also features a robot description file importer tool for comprehensive compatibility to existing setups.

Simulating the Camera

IsaacLab supports multiple types of sensors. To replicate the RealSense D435, the camera sensor object is used. This sensor can be defined precisely by setting parameters like resolution, focal length, or frame rate. Besides classical RGB-rendering these sensors can also be set up to collect depth data. In this setup, both an RGB and depth camera are used to mimic the RealSense D435.

Usually, processing camera images in parallelized environments is a significant bottleneck due to the large bandwidth of the pixel data. More recent versions of IsaacLab alleviate this issue by using tiled rendering to collect all camera images and process them as one render product efficiently on the GPU in a single call. [23]

Controlling the Simulated Hardware

Although the calculation of $\Delta\phi$ is domain independent, the actuators are accessed very differently. Fortunately, in IsaacLab, by using the articulation asset, a lot of the heavy lifting is done internally, which abstracts the motor control to function calls that can be used to write either target positions or target velocities to the joints, much like the Universal Robots ROS2 Driver. This closely aligns both control procedures.

3.3 Task Setup

Since this work focuses on grasping a simple object based on color detection, the task setup is relatively straightforward. The object used is a cube, which provides a stable, non-rolling shape. While its orientation may vary, its geometric consistency makes it still suitable for this procedure. To have a strong separation from the background, the color of the cube is set to be red. To allow easier access to the cube for the arm, it is placed on a table. The cube has a side length of 50 mm – a third of the gripper's stroke. This makes it big enough for easy detection and at the same time very convenient to grasp with the RG6. The arm, which is attached to the mobile base (fixed in this setup) is placed close to the table at a distance that enables easy access to the cube. The RGB-D

camera is mounted on top of the end-effector so that the field of view is always directed to where the tool is pointing. The described concept is illustrated in Figure 3.8.

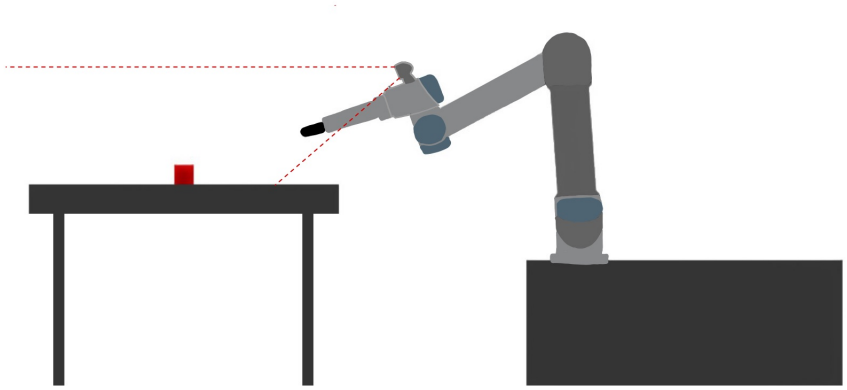

Figure 3.8: Task setup concept: Grasping a red cube from a table with a RG6 Gripper attached to a UR5 arm. The clearpath husky base is simplified to its physical boundaries. The RGB-D camera is attached to the tool link of the UR5 and its sensing field is sketched in red dashed lines.

3.4 Reinforcement Learning Architecture

As outlined in subsubsection 3.1.2, translating a real-world problem into a reinforcement learning framework involves several critical conceptual decisions. These decisions strongly influence the agent's learning behavior and can ultimately determine the success or failure of the entire setup.

3.4.1 Observation and Action Space

Observation Space

The observation space is designed to provide the agent with minimal yet sufficient information required to learn a grasping policy. For the agent to gain a basic understanding of its own position, all six joints are observed with their current angle. In addition to that, all individual directional velocities and torques are observed. This is necessary in order

to understand the direction of movement and the strain on physical parts. With this information, different operating conditions can be distinguished from each other. High torque and low velocity, for example, can indicate collision states or heavy payloads, or too high absolute torque values can help in telling the agent that it is operating the arm beyond its physical limit.

For the gripper, the available information is heavily restricted by the absence of compatible drivers. Data similar to those of the UR5 would also be of value here, however, as described in subsection 4.4.3, the goal opening and closing state as a binary value is what must be satisfied with.

Aside from its own state, the agent has to be informed about its surroundings and its relationship to them. The first info is the last known position of the cube in relation to the robot's base-link, detected by the algorithm described in subsection 3.2.3. Next, it observes if the cube has been grasped by detecting if the inner part of both gripper fingers touch the cube. One piece of information omitted for the sake of simplification is the position of the table. Accurately determining it would require an additional table detection algorithm, which lies beyond the scope of this work. To mitigate potential issues arising from this omission, the table is consistently placed in approximately the same relative position to the robot in the real-world setup and is subject to only low position randomization during training.

The next observation is a value which tells whether the cube is in sight or not, and if not, how long it's been since it was last seen. It is a variable that is set to zero whenever the cube is detected by the camera and increases towards a maximum of one after several steps have passed. When the cube is not within the visual range of the camera, it is assumed to stay in the same position. However, as an indicator of trustworthiness, the "age" of the position information is tracked. This will also help to create shaping rewards that encourage keeping the cube in sight.

The last three values are the distance to the cube as detected by the depth camera and the coordinates of the cube's centroid on the sensor plane of the camera. All of those will be set to minus one if the cube is out of sight.

The resulting observation space is of size 27 for a single environment.

A summary of the observation elements and their value range is shown in Table 3.2.

Category	Observation Variables	Value Range
Arm State	Joint positions J0-5	$[-2\pi, +2\pi]$
	Joint velocities J0-5	$[-3.14\frac{rad}{s}, +3.14\frac{rad}{s}]$
	Joint torques J0-2	$[-150Nm, +150Nm]$
	Joint torques J3-5	$[-28Nm, +28Nm]$
Gripper State	Gripper opening status	$\{0, 1\}$
Object Info	Cube position (as last seen)	3D coordinates $[-\infty + \infty]$
	Cube grasped?	$\{0, 1\}$
Perception	Cube visibility and data age	$[0, 1]$
	Cube depth value	$[-1, 10]$
	Position on sensor plane	Normalized 2D coordinates [-1,1]

Table 3.2: Observation space of the reinforcement learning agent with the respective theoretically possible ranges. The cube position is in practice limited by the clipped 10 m sensing range of the camera and is of course likely to stay on or in proximity of the table.

Action Space

The action space is linked to the joint angles. The agent can set 7 values between -1 and 1, used as joint deltas in each direction as described in subsection 3.2.2 and the gripper goal state explained in subsection 3.2.2.

The first six values are converted to $\Delta\phi$ values and used to calculate target joint angles. The gripper control strategy sets the gripper's goal state to closed when the action output of the network is negative or 0 and opened when it's positive. This is due to the limited access to the gripper's steering.

Action Variables	Value Range
Joint deltas J0-5	$[-1, +1]$
Gripper goal state	$[-1, +1]$

Table 3.3: Action space of the reinforcement learning agent with the respective value ranges.

3.4.2 Supporting Procedures for Training Efficiency and Robustness

In order to make the policy more robust and support learning overall, two common procedures are used. Domain randomization for robustness, and a combination of Curriculum Learning and Reward Shaping to support the learning process and increase sample efficiency. These procedures and how they are applied are outlined in the following.

Domain Randomization

To make the policy more robust to changes in general and to better prepare it for the sim-to-real transfer, domain randomization is used. Domain randomization is an extensive field of research of its own, and many advanced techniques have been developed over the years. Especially for RL in robotics, this is a very relevant topic due to the unavoidable presence of a Sim-to-Real Gap. [18] Here, noise from a parameter distribution will be applied to the simulation and the agent. This hinders the agent from adapting to a specific simulator setting and makes it more likely to adapt to the dynamics shift when transferred to the real world. The idea is that the real world appears to the agent as just one of the many variants of experiences it had due to randomization.

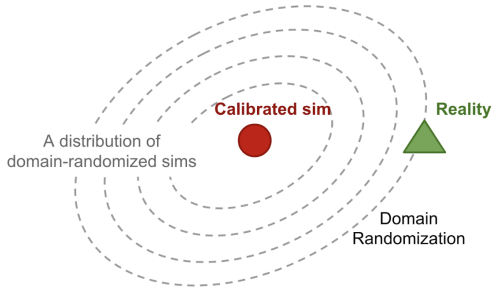

Figure 3.9: Concept of domain randomization for sim-to-real transfer. Source: [50]

Subject to this noise are: Action values, observation values, and physics parameters of the simulator.

Action and observation values are randomized with Gaussian noise that is added at every simulation step with a bias that is sampled after each episode reset. This ensures the noise is centered around the actual value, which is shifted by a small bias that changes every time one episode is done.

The physics parameters are sampled uniformly from ranges that encapsulate reasonable values. Subject to uniform noise sampling are: Joint stiffness and damping, physics material properties like friction coefficients both static and dynamic, and the elasticity coefficient for collisions.

Less focused on Sim-to-Real Gap robustness, the initial state of the environment is randomized as well, to avoid excessive dependence on a specific starting state and help the agent to handle different variations of arm poses in combination with cube positions.

Curriculum Learning

Curriculum Learning (CL) is a technique that aligns well with the human learning approach. Complex learning problems can often be sequenced into subtasks, each building on the skills that have been learned in the former ones. The learning agent can then transfer its learnings to the next step in this curriculum. There is no precise definition of CL, and consequently, there are many forms of that procedure. However, the best-known form is to organize tasks in a meaningful way that benefits the learning progress. [19]

Here to create a curriculum, an approach is used that modifies the reward function, creating different resulting subtasks. This way, the entire task setup, the simulator properties, and the environment dynamics can be left untouched, which makes it more straightforward to understand what these changes actually cause. Also, this enables increasing the task's difficulty step by step without eliminating the nature of the full task. Still, this has to be done with some care, since this changes what the agent perceives as "desirable behavior".

The resulting curriculum is divided into the following subtasks:

- Getting & keeping the cube within the visual range of the camera
- Approaching the cube
- Getting aware of and reducing load on the hardware during movement
- Grasping the cube

The subdivision is achieved through subsequently adding components to the reward function. The individual components are described in the shown order in the following subsection 3.4.3.

3.4.3 Reward Function

To highlight the CL structure of this reward function, each component and the effects on the behavior are discussed individually. The full reward function is described at the end of this section.

Keeping the Cube in Sight

The first reward function component is dependent on the presence of the cube within the vision field of the depth camera. Whenever the cube is out of sight, no reward is obtained. As soon as it enters the sensing range of the camera, a small constant reward is given at every step.

$$R_{\text{cube\_vis}} = \begin{cases} r_{\text{cube\_vis}}, & \text{if cube is visible} \\ 0, & \text{if cube is not visible} \end{cases} \tag{3.9}$$

Approaching the Cube

The next part aims to also reward approaching the cube. This is done by adding a second, distance-related reward component. To make it more attractive to move close to the cube, an exponential equation is used to calculate that reward. In order to avoid crashing into the cube the reward is only given until a reasonable distance for a grasp is reached.

$$R_{approach} = \begin{cases} \alpha_{approach} \cdot e^{-kd}, & \text{if d} > 0.2\text{m} \\ 0, & otherwise \end{cases} \tag{3.10}$$

The variable d represents the distance between the cube and the camera, while α and k are hyperparameters that determine what the maximum attainable reward per step is (α), in order to balance this reward with other reward values in the function, and how steep the incline of the reward is towards the cube position (k), deciding how strong the desire is to move closer to the cube.

An example of this graph as a function of the distance between cube and camera is visualized in Figure 3.10 without the distance condition.

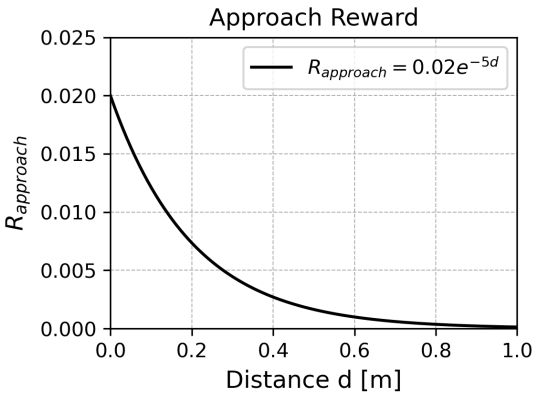

Figure 3.10: Equation used for the distance related reward with example values.

Reducing Load on the Hardware

Steering the arm towards a goal is already a solid control behavior. This means the agent has learned what joint values result in a desired overall movement. In contrast to the simulated hardware, however, the physical hardware can break or wear out depending on the load. Therefore, there must be a component in the reward function that includes this in the equation.

Until now, the curriculum consisted of rewards that encourage a certain behavior. The next step is to discourage unwanted behavior.

A simple solution would be to just penalize torque in general, analogous to the distance reward. Experiments showed that this slows down the movement of the arm drastically, as faster movement will always produce higher torques. What is actually desired is a penalty-free operating range and a penalized critical range that is close to the maximum allowed limit.

As shown in subsubsection 3.2.1, depending on the joints, the maximum torques of the UR5 range from 28 Nm to 150 Nm. The penalty-free range is set to be roughly 70% of the maximum. The remaining 45 or 8 Nm are penalized on a percentage basis.

Thus, at first it is calculated how much the current torque lies above the 70% limit denoted as $T_{critical}$ in Equation 3.11. As each joint is considered individually, all joint-specific values are dependent on the joint index J. Torques on the arm are measured in both directions, so negative values are also possible. This is included in the formula by calculating the absolute value.

$$T_{critical}(J) = max(0, |T(j)| - T_{max}(J) \cdot 0.7) \tag{3.11}$$

Once $T_{critical}$ is determined, it is set in relation to the remaining range from the penalty-free maximum to the maximum allowed value.

$$P_T(J) = min(\frac{T_{critical}(J)}{0.3 \cdot T_{max}(J)}, 1) \tag{3.12}$$

The total penalty is the sum of all joint penalties multiplied with a scaling factor α_{torque} used as a hyperparameter to balance the relation between penalty and reward.

$$P_{T\_total} = \alpha_{torque} \cdot \sum_{J=0}^{5} (P_T(J)) \tag{3.13}$$

Grasping the Cube

The last component is the main reward for the task itself, which is to grasp the cube. Here, the inner part of the gripper fingers, both left and right, have to touch the cube for a successful grasp. Just like with R_{cube_vis}, this reward value can either be 0 or a set value of r_{grasp}.

$$R_{\text{grasp}} = \begin{cases} r_{\text{grasp}}, & \text{if both fingers touch the cube} \\ 0, & \text{otherwise} \end{cases} \tag{3.14}$$

Full Reward Function

The resulting full reward function is the sum of all rewards minus the penalty:

$$R_{total} = R_{cube\_vis} + R_{approach} - P_{T\_total} + R_{grasp} \tag{3.15}$$

3.4.4 Algorithm and Hyperparameters

As this setup is very close to and of similar simplified nature as the work of Zheng et al. [56], where PPO was able to produce a good learning performance, this algorithm is also used here.

The IsaacLab framework comes with preimplemented example environments. One of them uses a simulated Franka Research 3 robotic arm to open a cabinet [3] with PPO by leveraging the RSL-RL library proposed by Rudin et al. [34]. The algorithm there is able to perform this task reasonably well with the given parameters. As the focus of this work is not on hyperparameter tuning, these parameters are used as the basic configuration, with some minor adjustments made on an empirical basis. To support reproducibility, Table 3.4 presents the applied values.

Hyperparameter	**Value**
Number of Steps per Environment	128
Initial Noise Standard Deviation	0.6
Actor Hidden Layers	[256, 128, 64]
Critic Hidden Layers	[256, 128, 64]
Activation Function	tanh
Value Loss Coefficient	1.0
Clipping Parameter (ϵ)	0.2
Entropy Coefficient	0.005
Learning Rate	2.5×10^{-4}
Discount Factor (γ)	0.99
GAE Lambda (λ)	0.95
Desired KL Divergence	0.01
Max Gradient Norm	1.5

Table 3.4: Selected hyperparameters for UR5 RL training

4 Implementation

In the following, the previously introduced components will be brought into practice by connecting them within the system and detailing their concrete implementation.

4.1 Tool & Architecture Implementation Overview

With the described components, Figure 3.3 can be specified more precisely. The extended visualization containing the described concepts and tools is shown in Figure 4.1. The bottom element is the PACT manager layer. It holds the network and supervises the training procedure. Also, performance tracking and interrupt detection happen here. Using this info, states are captured on demand that are then used to replicate real situations in the simulation, allowing the described switching behavior.

PACT accesses the simulation (shown on the left) through a gymnasium wrapper. This completely isolates the simulation specifics from the network and allows for the integration of a variety of sophisticated algorithms and training frameworks.

The Digital Twin resides within an NVIDIA IsaacLab entity, consisting of both an articulation and a virtual camera object. Also, the task setup is initialized here, but as these are simple passive instances, they will not be shown in this illustration. Both the articulation and the virtual camera are accessed through the use of a direct workflow environment. This is also where the calls through the gymnasium wrapper are translated into the IsaacLab-specific control functions. The environment accesses both the hardware control algorithm and the cube detection implementations that translate action commands to $\Delta\phi$ and pixel images to low-dimensional return values of the cube tracking procedure.

The real robot is also conceptually wrapped into a gymnasium environment, but since there is no environment entity that can be called like IsaacLab, this requires a custom

ROS implementation. This is done by a gymnasium to ROS translation node that is callable similarly to the RL environment interaction concept and then translates the function calls into ROS command messages that are sent to or received from the hardware through the drivers, essentially imitating the behavior of a gymnasium environment.

All implementations are written in Python to ensure consistency and seamless integration across components. In the following the realization of these components is described in more detail.

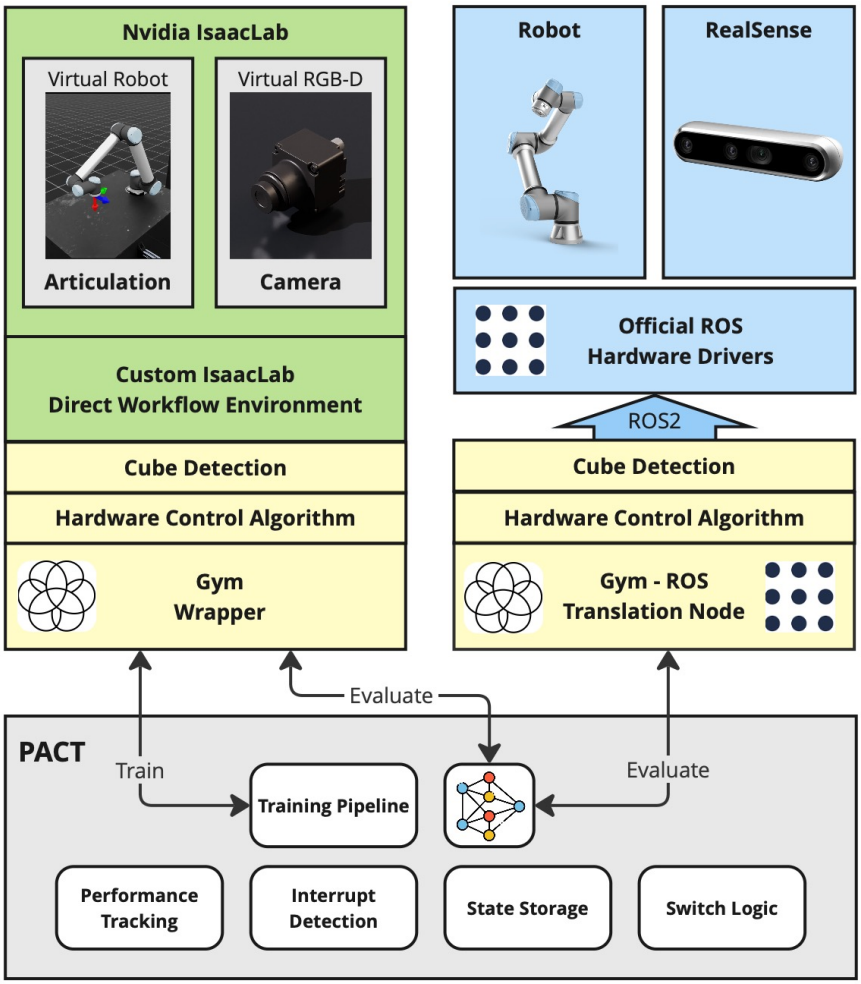

Figure 4.1: Tool and architecture implementation overview with the main components shown with their connections, showcasing PACT's interaction with both domains.

4.2 Shared Components for Grasping

The following two components are shared by both domains and are implemented independently of the execution context. As they have no domain-specific dependencies, the same implementation applies to both the real robot and the Digital Twin.

4.2.1 Hardware Control Algorithm

As the conversion algorithm from action commands to angles essentially is a single equation, the algorithm is imported into the gymnasium to ROS translation node and the IsaacLab environment directly. The v_{max} value is set to 20 $\frac{cm}{s}$ for both domains to have a relatively slow and safe speed limit when executed on the real robot. The update frequency is set to 120 hz to enable smooth control of the real arm and comprehensive physics stepping in the simulation. As the longest link between Elbow (J2) and Wrist1 (J3) is 44 cm, r is set accordingly.

4.2.2 Cube Detection Class

The cube detection consists of multiple calculations that are performed on the pixel data. To make them reusable and avoid boilerplate code, a Python class is constructed that is imported to the IsaacLab direct workflow environment and the gymnasium to ROS translation node.

The initial color detection is done using the Open Source Computer Vision Library (OpenCV). Both a lower and upper threshold are set for color values in the HSV color space to contain the red of the simulated and the real cube in a variety of different lighting scenarios.

Using these thresholds, the pixels lying within that range are separated and the areas of the resulting faces are calculated. Again, a threshold is used to filter out noise of small red areas within the image. Now the remaining values are used to calculate the centroid of the largest area present and extract the z value at that position. This process is illustrated in Figure 4.2. To reduce noise in the depth image, the median of a small area around the calculated position is used to estimate z.

Figure 4.2: Main three steps of the cube detection and centroid position calculation process illustrated from left to right.

The detection process is triggered by calling a function of the cube detector object with several arguments. Both images (RGB and depth) have to be passed, together with the rotation matrix, the translation vector of the camera, and the camera intrinsics. This ensures that the class is widely applicable by removing any hard-coded domain-dependent values from the class.

Using these values, the calculations described in subsection 3.2.3 are performed. The detector object also holds the last known position and a counter on how many times the position was returned without the cube being actively detected. This is the base for the data age element in the observation space. The data age is finally returned together with the cube position, the distance value, and the position coordinates of the centroid on the sensor plane.

Since training is also supposed to be performed with parallel environments, the function has to be able to extract the cube positions for multiple environments.

All these steps are shown as pseudocode in Algorithm 1.

4.3 Real Robot Setup

4.3.1 Connecting the Hardware

The hardware setup was inherited from previous projects, and no physical assembly was required in this work. But to ensure reproducibility, the setup and the connections between the physical components are documented here briefly. A simplified visualization of the connections is shown in Figure 4.3.

Algorithm 1 Extract Positions of Red Cubes in Camera Frames

Require: *rgb_images*, *depth_images*, *camera_translations*, *camera_rotations*, *camera_intrinsics*

Initialize lists for cube positions and metadata
for each environment **do**
    Convert *rgb_image* to HSV color space
    Apply red color and size thresholding to create binary mask *red_mask*
    Find largest surface in *red_mask*
    if no surface is found **then**
        Append last known position and metadata
        Increase data age
    else
        Set data age to 0
        Compute centroid (u, v) of the detected region
        Extract depth values within the detected region
        Compute median depth z at (u, v)
        Deproject (u, v, z) to 3D camera frame $\mathbf{p}_{cam}$
        Transform $\mathbf{p}_{cam}$ to world frame $\mathbf{p}_{world}$
        Append computed cube positions and metadata to lists
    end if
end for
    return cube positions in world frame, data age, cube distances, sensor positions

The UR5 and the RG6 share the same tool mount standard, which allows a straightforward connection by mounting the gripper's quick release connector to the tool flange of the UR5 and simply clipping in the gripper. Data transfer is realized through an M8 actuator cable connecting the RG6 to the UR5.

The UR5 is directly connected to a ROS master host system, residing in the mobile base, via Ethernet. The RGB-D camera is screwed on top of the gripper with a 3D-printed mounting plate and connected to the host with a USB-C cable. Like that, the host can access all hardware directly and simultaneously using ROS device drivers.

The simulation is running on another machine that shares its network with the host and can also communicate via ROS messages. Therefore, the host serves as the central node, directly interfacing with the components and managing data exchange with the machine running the Digital Twin via ROS.

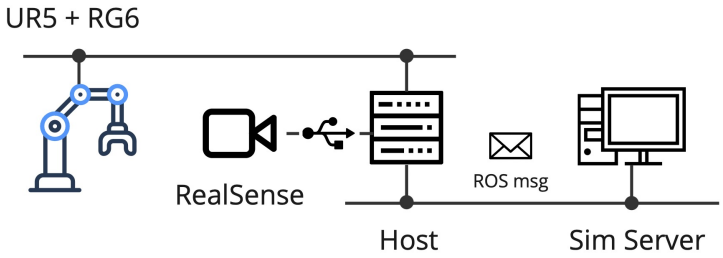

Figure 4.3: Hardware communication architecture

4.3.2 Hardware Drivers

Once the physical connection is established, there are two pieces of software necessary to connect the host to the hardware logically.

The first one is the Universal Robots ROS Driver [46]. It manages all communication between the host and the UR5. However, since it exposes publisher and receiver services to the ROS network, the simulation server can also access data from the UR5 and send control commands through the host.

The second one integrates the Intel RealSense SDK [7] with a ROS2 wrapper [8], enabling the camera to be accessed directly through ROS. Once the camera ROS2 node, that is included in the wrapper installation, is started on the host, the image data is published

to the ROS network at the specified frame rate and resolution, making it accessible to any connected device within the ROS network.

4.3.3 Gym-Compliant Robot Control with ROS

To enable gym-like control of the arm, the translation layer between the policy's action commands and the UR5 driver must be established. This ensures that the commands generated by the RL policy are properly interpreted and applied to the robotic system.

To achieve this, a custom ROS2 node is implemented, providing a callable function that accepts a seven-element list of normalized action commands c_{joint}, one for each arm joint, along with the gripper command $c_{gripper}$. When called, the node uses the hardware control algorithm to compute the corresponding joint angle updates $\Delta\phi$ and determine the target state of the gripper $s_{gripper}$. Simultaneously, it retrieves the current state of the arm and gripper. To execute the desired movement, the calculated $\Delta\phi$ values are added to the current joint angles, resulting in the target state of the joints. In order to apply these angles to the arm, the joint position controller of the Universal Robots ROS Driver is used. Since robotic manipulators are typically controlled using trajectory data, the default "joint_trajectory_controller" must be replaced with the "forward_position_-controller". This is done automatically during the startup of the node. At a frequency of $f = 120Hz$ the node then sends the target angles to the controller's dedicated ROS topic.

With this logic, an unintended effect emerges. When sending only zero action commands, the arm should theoretically maintain its position. In practice, however, a different behavior appears. Continuously retrieving the most recent joint angles and adding a zero delta results in the arm gradually drifting towards the gravitational pull. This drift occurs because small deviations accumulate over time, leading to unwanted movement.

To avoid this drifting effect, the node holds a ground truth state to witch the angle updates are added. This ground truth is compared against the actual angles, and if the difference is higher than a small threshold, the ground truth is updated to the real value. With this strategy, small shifts in angle values are not accumulated and the arm keeps its position stable, but if larger deviations appear, they are taken into account.

Since the Universal Robots ROS Driver only accesses the UR5 and not the RG6 gripper, usually a separate driver would be used to send commands to the tool. However, as

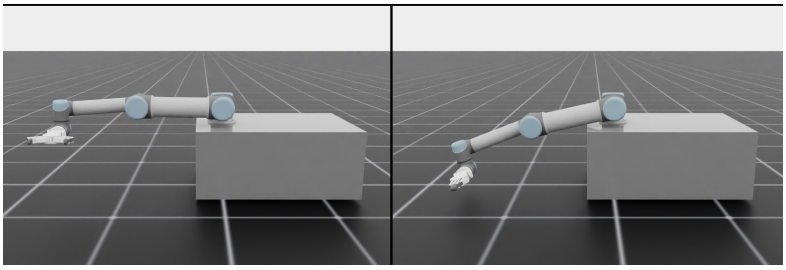

Figure 4.4: Gravitational drift when sending zero commands continuously.

described in the previous chapters, no up-to-date driver is available that supports ROS2. Therefore, a workaround is used to open and close the RG6. With the M8 connector, two digital output signals can be pushed through from the UR5 to the gripper. Setting them high or low triggers a closing or opening procedure. Although not as precise as the driver-supported movement, this allows to still use the tool's basic functionality.

4.3.4 Gym-Compliant Observations with ROS

Having the steering logic in place, the node also needs to return observations in a gym-conform format after every executed action command. This is done by communicating with the camera node of the Intel RealSense ROS2 Wrapper on the host. The gymnasium to ROS translation node subscribes to the topics sent by this camera node and therefore continuously receives the image data published by the node and stores the most recent images. When action commands have been successfully executed on the real robot, a function is called that hands over the most recent RGB and depth image to the cube detector Python class described in subsection 4.2.2 that in turn performs all steps necessary to determine the cube position and return the observational values. The gymnasium to ROS node then passes them through as the return value of the initial function call.

Like that, the policy can call a function of the node with its action commands and will get back the observation of the state after the execution of the command, following the gymnasium standard.

4.4 Digital Twin

4.4.1 Simulation Host Hardware

The machine running the simulator is a Dell OptiPlex 7080 with an Intel i7-10700 CPU and a NVIDIA RTX3070 GPU. While this meets the minimum requirements for Isaac Sim, the GPU can become a limiting factor when running multiple parallel environments, especially with active RGB and depth cameras.

4.4.2 Importing the UR5 in IsaacSim

IsaacSim includes pre-existing models of various common robotic manipulators, including the Universal Robots UR5, which significantly reduces the implementation effort required for simulation and integration. The arm can be directly imported into the IsaacLab RL environment as a Universal Scene Description (USD) file – the primary robot and environment description format of IsaacSim. There, the arm is already defined as an articulation and joint parameters are set.

To match the attachment height of the real UR5 that sits on a mobile base, the simulated UR5 is mounted to a simple block that has the size of the outer bounds of the base. Like this, self-collisions between the arm and base are also taken into account.

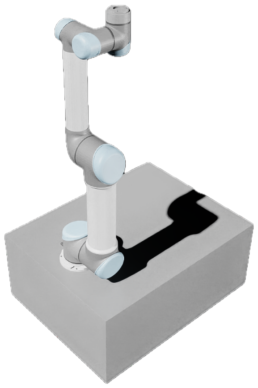

Figure 4.5: UR5 imported through IsaacSim mounted on top of a block, mimicking the disabled mobile base.

4.4.3 Building the Gripper in IsaacSim

Unlike the UR5, the RG6 gripper is not readily available in IsaacSim. However, since IsaacSim provides a tool for importing Universal Robotic Description Files (URDF), and such files exist for most robotic components, a URDF model is also available for the RG6. Although an official version is not provided by the manufacturer, a researcher from the University of Liverpool has published a URDF version of the gripper in a publicly accessible GitHub repository [57]. To ensure functionality in IsaacSim, the joint order had to be adjusted, and fundamental parameters such as upper and lower angle limits were defined. Additionally, the collision meshes interfered with proper finger closure and required modification. With some tweaking, the gripper is functional and controllable through IsaacLab. But as there is no underlying driver and the gripper is built with six moving joints, it has to be configured as a six-joint articulation, even though the joints are not intended to be actuated individually.

Figure 4.6: RG6 two finger gripper imported via IsaacSim and adapted from the URDF.

4.4.4 Adding a virtual RGB-D Camera

Adding the RGB-D camera to the simulation is a straightforward process. It can be instantiated during environment creation by defining its position, orientation, and placement within the entity hierarchy, ensuring the correct parent-child relationship. Like that it can be fixed in place precisely on top of the gripper at approximately the same position as in reality. As it is set as a child of the gripper, it will keep its relative position and move along with the tool. The camera's intrinsic parameters can also be set on creation, in order to match the real setup.

4.4.5 Assembling the Virtual Robot

Until now the UR5 and the gripper have been described as two separate robotic articulations. But like in reality, they should be controlled as one entity. Here, another tool of IsaacSim comes into play – the robot assembler. This tool allows the combination of two articulation entities by connecting them with a fixed joint. In this case, the fixed joint is set at the tool mount of the arm, resembling the screwed connection of the RG6's quick release connector on the real UR5.

Since the camera is not defined as an articulation, it doesn't require assembly and can exist as a separate entity that moves along with the tool. Sensors in IsaacSim per default don't have visual properties and are therefore not appearing in the rendering shown in Figure 4.7.

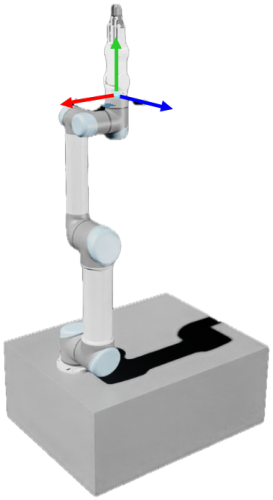

Figure 4.7: UR5 and RG6 assembled using the IsaacSim robot assembler tool. The RG6 is attached at the tool mount of the UR5 (mounting point marked at the coordinate origin).

4.4.6 Direct Workflow RL Environment

Reinforcement Learning environments can be created in different forms using IsaacLab. The most direct control over the environment is possible with the Direct Workflow RL Environment. Here, the entire code structure is less abstract, allowing a more holistic

implementation. Also, the reward function computation and observations can be defined precisely in this approach.

The environment class follows a structure similar to a gymnasium environment, featuring template implementations for core functions such as step, get_observations, and reset. It is tightly integrated with PyTorch, leveraging its JIT capabilities for efficient execution. With PyTorch, operations are primarily performed using tensors, enabling efficient computation and seamless handling of multiple parallel environments.

A function called setup_scene is used to dynamically build the environment. This is where the assembled robotic manipulator is loaded and placed into the simulation together with the camera and task objects. Loading all entities individually within this call, rather than assembling them as a single USD file, allows for precise control over the position, orientation, and hierarchical relationship of each entity. This enables the integration of real-world data to build a simulated environment that aligns with the actual state.

Additionally, when instantiating multiple parallel environments, randomization parameters can be passed to these objects, enabling random state distributions around the real-world state.

The Direct Workflow Environment also allows setting a global randomization configuration that modifies physics, articulation behavior, as well as observation and action values, for effective domain randomization. Since the reward function can be precisely defined and modified at runtime, CL and reward shaping procedures can be implemented directly into the environment.

4.4.7 Gym-Compliant Simulation Control with IsaacLab

Like on the real robot, the policy's action commands have to be converted into control commands for the simulated entities. A template function that comes in handy here is the pre-physics step. This function is called automatically with the action commands as arguments before applying actions to the simulation. It is used to perform all necessary action translation steps. First, all $\Delta\phi$ values are computed using the hardware control algorithm, and the corresponding target angles are determined by adding $\Delta\phi$ to the articulation's current angles while incorporating the drift correction.

Next, the gripper action command $c_{gripper}$ must be translated to interface with the six-joint gripper articulation constructed from the URDF. In principle, the continuous action command is mapped to target joint angles, where -1 corresponds to a fully closed state and 1 to a fully open state. Intermediate values result in a linear transition between the opening and closing states. But since on the real gripper this movement is initiated by setting the digital output and cannot be interfered with once initiated, the simulated control has to be adapted accordingly. Once a change in $s_{gripper}$ is detected, the simulated gripper transitions to the opposite state within $\approx$ 2s, mimicking the real behavior. During this period, any incoming gripper action commands are ignored. Once the target state is reached, the process repeats.

This way, the environment can be called with a seven-element list, and the prephysics step takes care of all conversion actions that modify the articulations accordingly.

4.4.8 Gym-Compliant Simulated Camera Observations

The observation processing of the simulation is equal to the real setup, as both cameras return an RGB pixel image together with a z value for every pixel. Only the format differs. The RealSense returns 3D arrays located on the CPU while IsaacLab returns a GPU pytorch tensor. Due to the lack of GPU support in OpenCV, the simulated camera data is currently processed on the CPU. However, this can potentially introduce a significant bottleneck in the training process. Future implementations should address this issue, but as the used hardware does not allow huge parallelization, the slowdown is rather small. Measurements show that the image processing in this setup, using 16 parallel environments, takes 4.5 ms which makes up $\sim$2.6 % of the total step time and is therefore negligible.

4.5 Connecting Sim & Real

For PACT to function, it requires a foundational connection between the simulation and the real world, enabling seamless access to both domains at runtime. This concept is illustrated in Figure 3.3 as the manager layer.

Due to the fact that the previous implementation steps abstracted all functionality into classes, ROS nodes, or function calls, the manager layer can now tie these components together.

The ROS functionality needs to run in the background continuously and cannot be called only on demand. Therefore, two Python threads are created that run the cube detection and the UR5 control, constantly sending joint angle commands and receiving joint states as well as camera images. On start, references to the running nodes are stored, so that the manager can pass action commands to, and receive joint states from the UR5 control while the current detected cube position can be accessed through the cube detection. The running nodes await the startup of the real robot and once messages are sent from the robot to the respective topics, they initiate their execution. To inform the user about the current state of the node, they push console output through to the manager script.

The IsaacLab Direct Workflow Environment is registered as a gymnasium environment and then created by the manager layer through gymnasium's standard make command. The IsaacLab environment is then callable with `env.step()` and `env.reset()` like a classical gymnasium environment.

Having the gymnasium-registered environment and the ROS nodes running and available, the manager layer can now decide where to send commands to and where to receive information from.

4.6 Incorporating the PACT Logic

After enabling access to both domains, the next step is to make use of that in a productive way. This means implementing the workflow as described in Figure 3.1.

The first essential capability of this workflow is the pretraining procedure that produces a robust policy capable of handling a wide range of states. Further, in this first implementation, RL training is also used in an attempt to adapt the policy to handle critical states upon interruption.

The initial training phase requires strong state and domain randomization, combined with techniques such as reward shaping and curriculum learning, to encourage broad generalization. In contrast, the adaptive retraining phases rely on reduced randomization to enable more focused, problem-specific learning.

But as the simulation is also used to validate the policy and predict what the result of certain behavior is in the real world by consolidating the simulation, there also needs to be a way to reduce variations between the runs.

Therefore, the environment is designed to operate in three distinct modes. The first is the pretraining mode, which utilizes curriculum learning and strong domain randomization to improve generalization. The second one is the retraining mode, which disables curriculum learning and applies reduced randomization to fine-tune the policy. And finally, the evaluation mode runs the policy without any randomization or learning-related strategies, providing a stable and replicable reference for assessment.

The second essential capability is the ability to transfer information from the real world into the simulation. This enables the system to replicate real-world states within the simulated environment for the purposes of validation and interrupt-driven adaptation.

The approach is to leverage the available information from the observation space, which by definition should contain all data necessary to fully describe the current state of the environment. In this way, no additional sensors are required, as the real-world state can be retrieved directly by requesting the observation data from the real environment through a gymnasium-compliant function call that is already implemented.

As the observation space contains all information about the joints, the entire robot can be replicated in its current state by setting the joint angles. The cube is placed in the simulation to its last seen position according to the observation data. This, of course, presupposes that it has been detected at least once at any point in time during the real-world execution.

With these features, the layer can set the environment mode and transfer the real-world state to the Digital Twin.

To further build on this functionality, the manager layer must be capable of initiating all run types (pretraining, adaptation, and validation) and tracking their execution. This includes monitoring whether the pretraining and adaptation phases have led to the desired improvements, and determining whether a validation run has resulted in successful task completion or interruption.

Since IsaacLab is designed for users to initiate training runs via predefined training files and supports only limited customization through a separate configuration files, this approach extracts the core functionality of the RSL-RL training script and integrates it

directly into the PACT framework. This allows for greater flexibility and tighter integration while maintaining the underlying training capabilities. Also, dynamic extraction of performance metrics of the training is made possible like that, which would otherwise only be written into a log file.

The detection of interrupts is performed by leveraging the observation data and defining trigger states. Like this, several possible issues can be detected. The only limitation is their detectability through the observation data. The utility of the individual interrupt, however, largely depends on the task at hand.

As an initial implementation, two testing interrupt conditions are integrated into the system. The first is triggered by high torques, which can indicate potential collisions, and the second activates after a timeout when too many steps have passed without a successful grasp. Naturally, these interrupts do not cover all possible unwanted states. Depending on the specific task and setup, additional interrupt conditions can be defined, as long as the observation data provides sufficient information to detect them. In this case both interrupts can be derived from the arm's torque values and the stepping function respectively.

With the ability to access both domains, change modes of the simulated environment, transfer real-world states to the Digital Twin and a fully integrated RSL-RL training procedure that returns performance metrics and interrupt information, the manager layer can transition through all states and checks of the PACT workflow.

4.7 Pretraining with RL

The initial training procedure will be directly included into the framework. Through configurable parameters, the PACT pretraining pipeline supports semi-automated Curriculum Learning when started in the corresponding mode. The configuration includes a boolean flag, which indicates that the workflow should start by pretraining a policy, either from scratch or from a preexisting model checkpoint file. Additionally, the level in the curriculum that should be used as the starting point is defined, so that customized runs are possible, depending on the initial situation. For each CL level, a performance threshold is set, telling PACT at which average episode reward this curriculum level is considered solved and the next one should be initiated. To avoid endless loops, two conditions are checked. The first one detects plateaus, meaning after a set period of time

a reward progress threshold has to be surpassed or PACT will continue with the next curriculum level. The second one absolutely limits the number of possible training steps per level. Both thresholds can be tuned in the configuration according to the use case.

The reward function is prepared in CL level structure in the environment definition, PACT switches levels by dynamically adding components to the function. The environment therefore has to order the reward components incrementally according to the CL levels. PACT then uses the modified RSL-RL training pipeline to retrieve performance metrics and traverse through the curriculum to optimize performance, while using the user-defined thresholds.

5 Results

5.1 Testing the System Components for Grasping

5.1.1 The Control Algorithm in Practice

First of all, it must be verified whether the calculation of $\Delta\phi$ and its following control procedures are suitable approaches.

The first tests are performed by simply sending constant steering commands and evaluating if the robot and its Digital Twin reliably follow the intended movement. The Digital Twin theoretically verifies the procedure as the joints accurately follow the sent commands. The articulation asset is behaving as expected and the function calls produce precise movement in the set directions.

The real robot executes the same joint commands and successfully reaches the intended positions, demonstrating that the concept transfers reliably to the real-world domain. However, the resulting trajectories show occasional stuttering during execution.

ROS provides command line tools to monitor the frequency of commands published on the network. Analyzing the steering commands published by the ROS node returns the configured value of 120 Hz. However, when tracking the frequency of the function calls that set the action commands, the frequency is significantly lower, averaging at about 10 Hz.

This behavior suggests that the stuttering is not caused by the control algorithm, and is therefore not a limitation of the steering concept itself. Closed loop joint position control is very time-sensitive and this setup is servoing joints in real time while processing images in-between, requesting action commands by a neural network and running a simulator on a machine with very limited resources.

Then more complex steering patterns are predefined and sent to both domains. The behavior matches the simpler commands, as the system directly follows the specified direction. Directional changes are executed immediately on both the Digital Twin and the physical robot. However, observing the UR5 in motion reveals that abrupt changes in direction are suboptimal, as they impose mechanical stress on the arm. The occasional stuttering on the real robot is also present in the second motion test.

5.1.2 Robustness of the Cube Detection

The next validation test evaluates whether the cube tracking algorithm is sufficiently accurate and reliable in estimating the cube's position. Aside from the joint data, which is accurately returned by the UR5, this is the most important observation made by the system. Without the knowledge of the cube's position, no intelligent behavior can be learned in this task.

To evaluate the accuracy of the cube position tracking, first the UR5 is positioned at distance to the cube and the detected cube area is marked in the RGB image and inspected visually. Then the arm is moved through a trajectory that is similar to the typical approach used for grasping, including strong perspective shifts, as well as translational and rotational changes of the tool-mounted camera. Throughout the sequence, the cube remains fixed in place, allowing for assessment of whether the calculated position remains stable despite substantial changes in the sensor's pose and position.

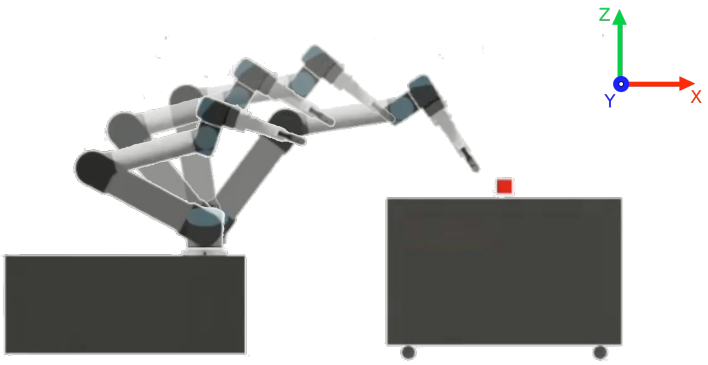

Figure 5.1: Illustration of the testing trajectory viewed from the right side of the scene. The starting position is located far from the cube, with the tool resting above the UR5 base. The goal position places the tool tip in close proximity to the cube.

To further challenge the algorithm, experiments under varying lighting conditions are conducted to assess its robustness to changes in the visual appearance of the scene.

Since the tracking approach is simplified, the results must be interpreted with this limitation in mind: the calculated position corresponds to a point on the cube's surface located at the centroid of its 2D projection. Depending on the cube's pose relative to the camera, this projection-based centroid position in the 3D space may shift slightly, even if the algorithm accurately calculates the position.

Simulation

In the simulation environment, the position is tracked very reliably. During this test, the cube is placed at the fixed position $[1.0, 0.0, 0.59]$ in the world frame, which is located in the center of the table's surface.

As shown in Figure 5.2, with normal lighting, the tracking shows only slight deviations from the ground truth and some noise. The y-axis is subject to the least deviation but is also mostly unaffected by the performed trajectory. The x-axis and z-axis values are moving up and down with an overall range of 1.5 to 2.0 cm. Shifts in that magnitude are unsurprising considering the effect of the moving centroid in this approach.

With respect to lighting changes, the simulation environment experienced little to no effect on tracking accuracy. To evaluate this, three test runs were conducted under distinctly different lighting conditions: one with significantly reduced brightness, one with standard illumination, and one with strong overexposure. Despite the extreme variations, the tracking performance remained consistent across all scenarios. As shown in Figure 5.3 the detection of the cube is very robust to different brightness values in the simulation.

However, given that the simulation environment lacks complex elements like oblique light incidence, textured surfaces, or visual clutter, and the only red object in the scene is the cube itself, this robustness is expected. Therefore, the findings offer only limited insight into the system's ability to handle visual variations of more complex real-world scenes.

Nevertheless, this experiment already shows the effectiveness of the algorithm and validates the approach under semi-optimal conditions.

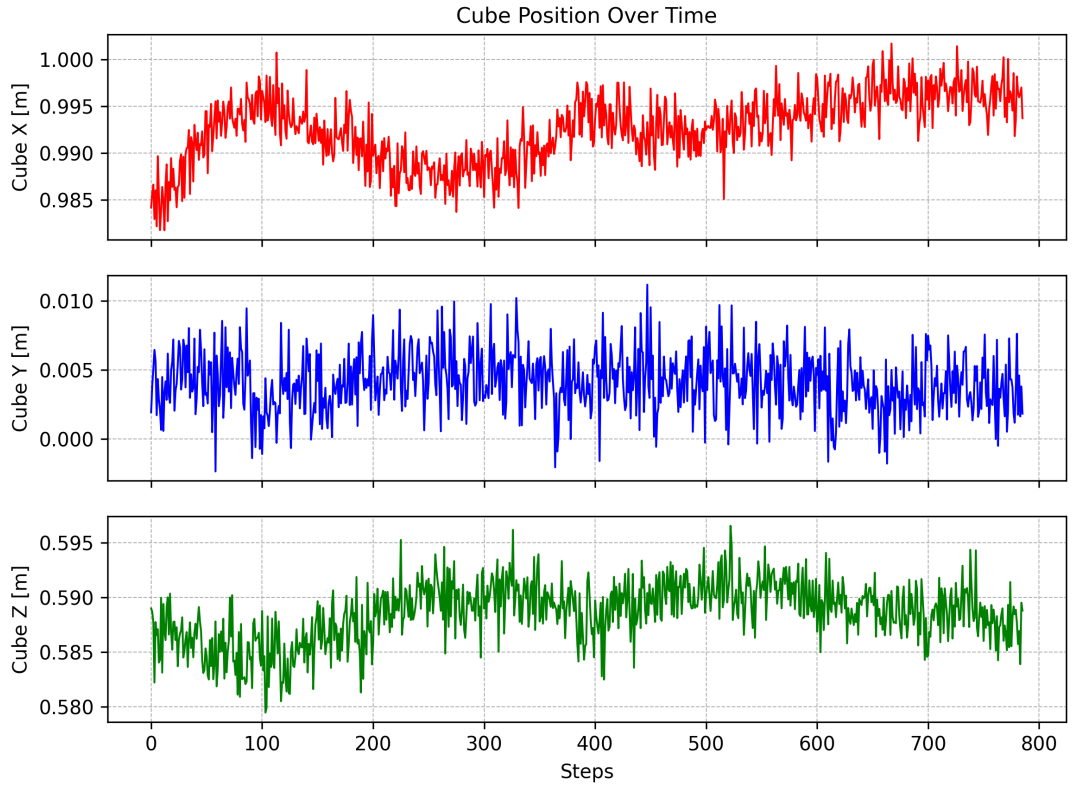

Figure 5.2: Results of the cube tracking algorithm (x, y and z coordinates in the world frame) while following the testing trajectory in the simulated IsaacLab environment using the image data from the virtual RGB-D camera.

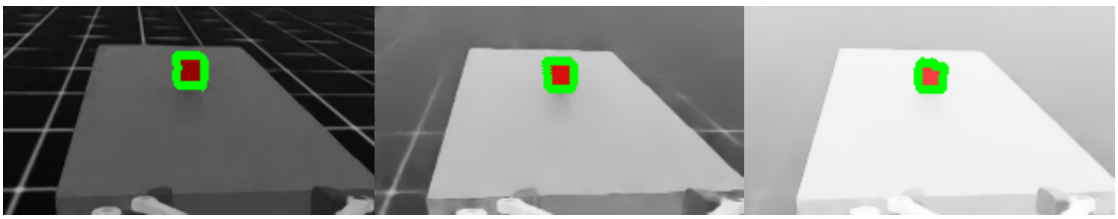

Figure 5.3: Cube tracking in the simulation environment under three different lighting conditions. The detected area is framed with a green contour. In all three situations, the tracking function returns a confident separation between the cube and the remaining environment.

Real World

To really put the algorithm to the test, the same trajectory is followed with the real robot using images streamed from the tool-mounted RealSense camera. The experiment is conducted in a visually cluttered laboratory with multiple objects of varying colors present in the background. The cube's position is measured at $[0.91, 0.0, 0.57]$. The first run is performed in a darkened room where the table is illuminated by artificial light. Under these conditions, the area detection broadly matches that of the simulated environment, as shown in Figure 5.4.

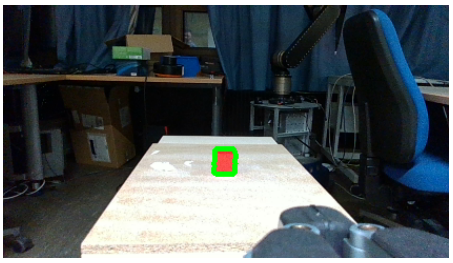

Figure 5.4: Cube tracking with a well illuminated table and a very distinct red cube. The algorithm is confidently detecting the entire cube.

Figure 5.5 shows a similar behavior in returned coordinate values compared to the simulated run, with the y-axis keeping steady values while x and z deviate more strongly as they are more affected by the camera movement. The overall range of the values is higher than in the simulated environment, with the x-coordinate values ranging from a little over 89 cm to slightly above 92 cm and the z-values having a lower bound at about 54.5 cm and maxing at approximately 59.5 cm.

Therefore, the range of the x-values increased by about 50% and the z-values by 100% compared to the simulated run. Although this may appear significant, it still indicates that the algorithm is capable of tracking a cube with a side length of 5 cm within a range of 4 cm, which remains acceptable for the intended application. Also, these values come with additional observations, such as the position of the cube's center on the image plane and the raw depth value from the camera. Together, this data should provide a sufficiently informative observation space for the algorithm to infer the cube's position with reasonable accuracy.

When it comes to more challenging lighting scenarios, the real setup struggles much more, especially with oblique incidence of light. To test this, three different less optimal

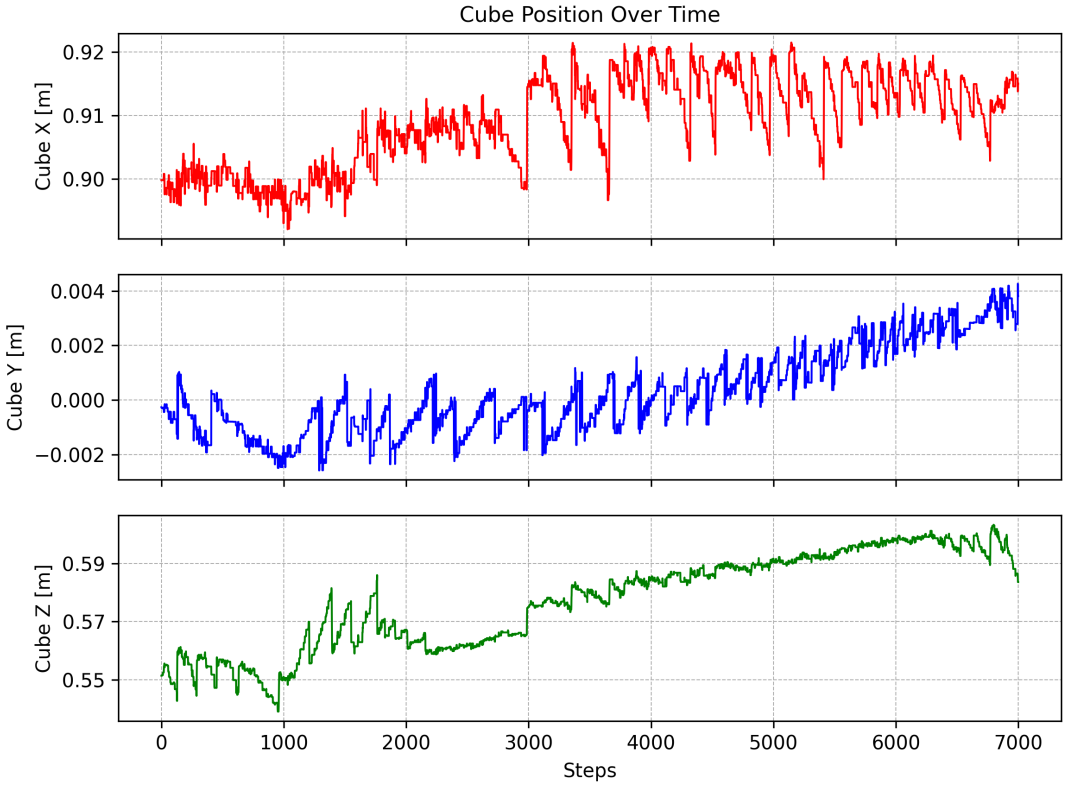

Figure 5.5: Results of the cube tracking algorithm (x, y and z coordinates in the world frame) while following the testing trajectory in the real laboratory environment using the image data from the RealSense camera with artificial illumination of the table.

lighting scenes were created that are captured through the RealSense camera and are shown with the tracked cube in Figure 5.6.

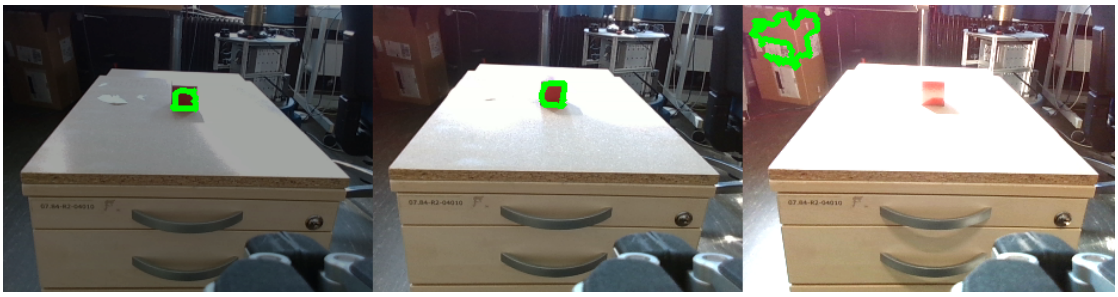

Figure 5.6: Cube tracking in three more complex lighting scenes. A dark scene with reduced brightness on the left, soft oblique light in the middle and very strong direct sunlight to the cube on the right.

The left image shows a dark scene with all windows covered by thin curtains that reduce the direct light exposure. To further darken the image, the brightness is reduced artificially by lowering the value (V) channel in the HSV color space, reducing the luminance of each pixel without altering the hue or saturation. This seems to be no greater problem for the tracking algorithm at first and produces only a slight reduction of the contour towards the center of the cube.

In the middle, more indirect light is added by uncovering surrounding windows, creating a soft oblique lighting. In this scene, the tracking contour almost fully surrounds the cube again.

The image on the right shows direct sunlight exposure that weakens the red color of the cube, while highlighting surrounding objects with tones of red. The green contour surrounds a completely different object in the background, entirely losing track of the cube's position. Although this issue could be addressed by adjusting the color thresholds or incorporating depth information to filter out background pixels, it is considered an edge case in the context of this work. Consequently, strong direct sunlight exposure to the scene is intentionally avoided.

Running the trajectory with the first two lighting scenarios and comparing the returned coordinate values as shown in Figure 5.8 indicates that the algorithm seems to handle them both quite well and the tracked positions in the two scenes don't deviate much from each other. However, in the dark scene the algorithm lost track of the cube for

some time. As it always returns the last known position, this effect can be seen at the straight line between roughly 2000 and 4000 steps.

Compared to the run with artificial lighting, the y and z-value especially show greater differences over the course of the trajectory. An inspection of the contours reveals that under angled lighting, the algorithm tends to track only parts of the cube, with the detected area shifting in different directions.

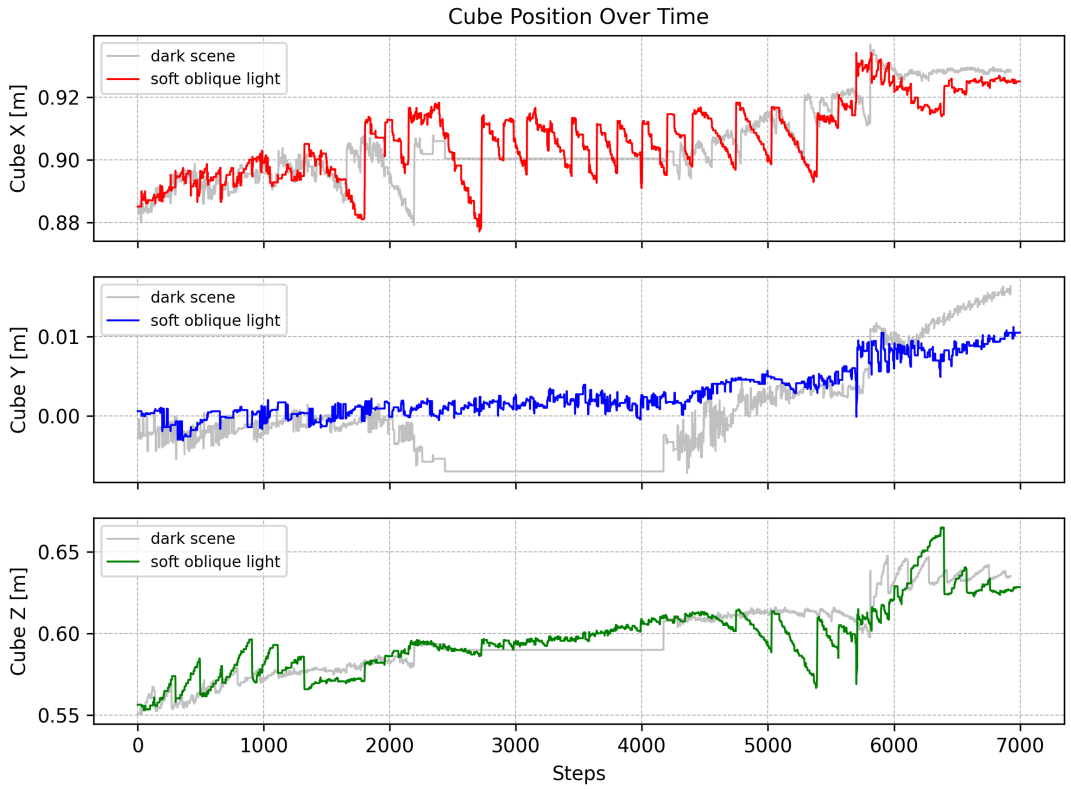

Figure 5.7: Results of the cube tracking algorithm while following the testing trajectory in the previously described soft oblique lighting scene shown in color. For comparison the tracked coordinates in the dark scene are shown in grey.

In conclusion, the cube tracking algorithm performs reliably under optimal conditions, particularly when the workspace is illuminated by artificial light. It also handles natural sunlight with soft, angled lighting without significant issues. However, under more extreme conditions, the tracking performance degrades, showing some limitations.

These issues could be addressed in future work, for instance, by integrating a tool-mounted light source to improve visibility in dark environments, optimizing the color thresholds, or by employing depth-based segmentation to better distinguish foreground objects from bright backgrounds.

5.2 Verifying PACT Framework Functions

Before the framework is fully deployed in practice, the core functions are tested in a more isolated way.

5.2.1 State Transfer

To validate the state transfer, the arm and the cube are moved into several distinct positions, after which the framework is initiated. Then the transferred simulated state is compared with the real counterpart.

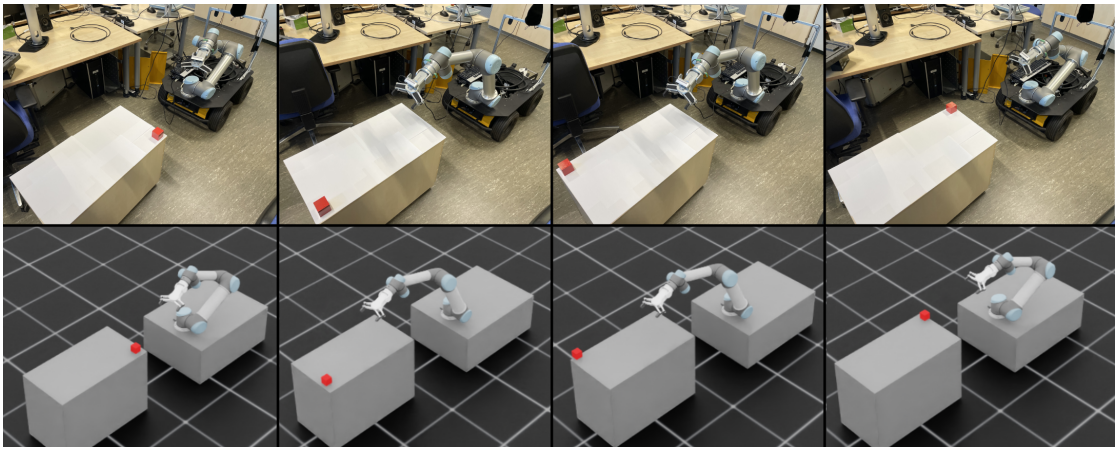

Figure 5.8: Testing positions of the cube and arm, initially arranged in the real world and then transferred to the simulation using PACT's state transfer functionality for comparison.

All states represent reality with high visual accuracy. The arm is replicated without any apparent deviations, while the cube shows small discrepancies as previously discussed. Due to these inaccuracies, the simulated cube is sometimes slightly set into the table's surface, causing unwanted behavior. This effect can be compensated by introducing a small positive offset to the z-axis, allowing the cube to fall into place. Since the table is

not detected by the system, its position depends on the accuracy of the measured values that are entered into the environment's configuration file.

5.2.2 Interrupt Detection

Testing the example interrupt functionality is a straightforward process. To verify that the system triggers torque interrupts correctly, the torque thresholds are reduced to a low value. On the real robot, this allows the interrupt to be triggered by manually applying slight pressure to the tool-end of the arm, while in the simulation, artificial commands are sent to cause the arm to collide with the table. To test the timeout condition, a random policy is used with the time limit set to just a few steps, ensuring that the interrupt fires safely before the policy can cause any damage to the real system.

Both interrupts are detected reliably and as the stepping frequency in both domains is set to 120 Hz, the detection happens almost instantly upon occurrence.

5.2.3 Validation Procedure

Once a validation run is initiated, PACT removes any randomization from the simulation environment. Then the current policy is executed on the transferred state. To verify this step before the retrained policy is available, the policy is replaced by a constant command value that lets the arm collide with the table. As tested previously, this triggers the torque interrupt and the framework inhibits the transfer of this policy to the real robot.

To also test a successful verification run, the policy is replaced by a function that only sends zeros as command values, and the grasping success is overwritten to always indicate a successful execution. Using this mocking setup, the verification is successful and the real robot starts to receive the zero commands.

5.3 Estimate of the Sim-to-Real Gap in the Present Setup

In order to gain a better understanding of the significance of the Sim-to-Real Gap in this setup, a first analysis of the domain differences is conducted in the following. Since the gap includes all mismatches between the real world and the simulation, it is not possible

to measure every aspect in detail. However, its core aspects will be evaluated to get a good understanding of how strongly the key elements diverge across the domains.

In subsection 5.1.2, the behavior of the cube tracking was already inspected. Based on the lighting situation, the size of the gap in this aspect can vary depending on how good the conditions are for the tracking algorithm. As the tracking has already been analyzed, this chapter now turns to the arm's behavior. The main describing factors of the arm are its position, set by the angles of its joints, and the torque applied to them. The velocity is also part of the observation space but less descriptive for the gap, as it is just a result of the position change over time.

To get a clearer picture of how the real UR5 and the Digital Twin behave, the real robot and the simulation environment are run in parallel using the implemented PACT architecture. A set of predefined action commands following the trajectory that was used to check the cube tracking robustness is sampled and sent to both systems at the same time. The joint angles and torques of both agents are returned and compared. The action commands move joint 1, 2, and 3 while the remaining joints receive 0 as command value in this trajectory. The results are shown in Figure 5.9

The first clear observation is that, due to the control algorithm operating with absolute joint angles under the hood, the steering behavior shows only small discrepancies between the simulation and the real robot. This is a valuable property in this setup, as it means the same control logic leads to consistent outcomes in both domains. However, a slight divergence appears over time. In the simulation, joints that should remain static show a small amount of drift, while the real robot stays completely stable. This results in an angle difference of approximately $-1.5\ ^\circ$ across all joints between the start and end of the movement. This offset occurs regardless of the implemented drift check. Still, this is a rather small deviation and has only a minor impact since the initial angles are always set by the real state and can only build up deviations during one episode. This suggests that the Sim-to-Real Gap in terms of joint-level control is small.

The torque values show greater differences between the simulation and the real robot. This is especially true for the first three, larger joints, which are located near the robot's base and carry most of the arm's weight with a long lever arm. As a result, they are particularly sensitive to inaccuracies in parameters such as mass distribution. Additionally, the torque values show a high degree of fluctuation, which is likely caused by the jittery movements produced by the previously discussed delays. The torques are therefore showing a larger gap.

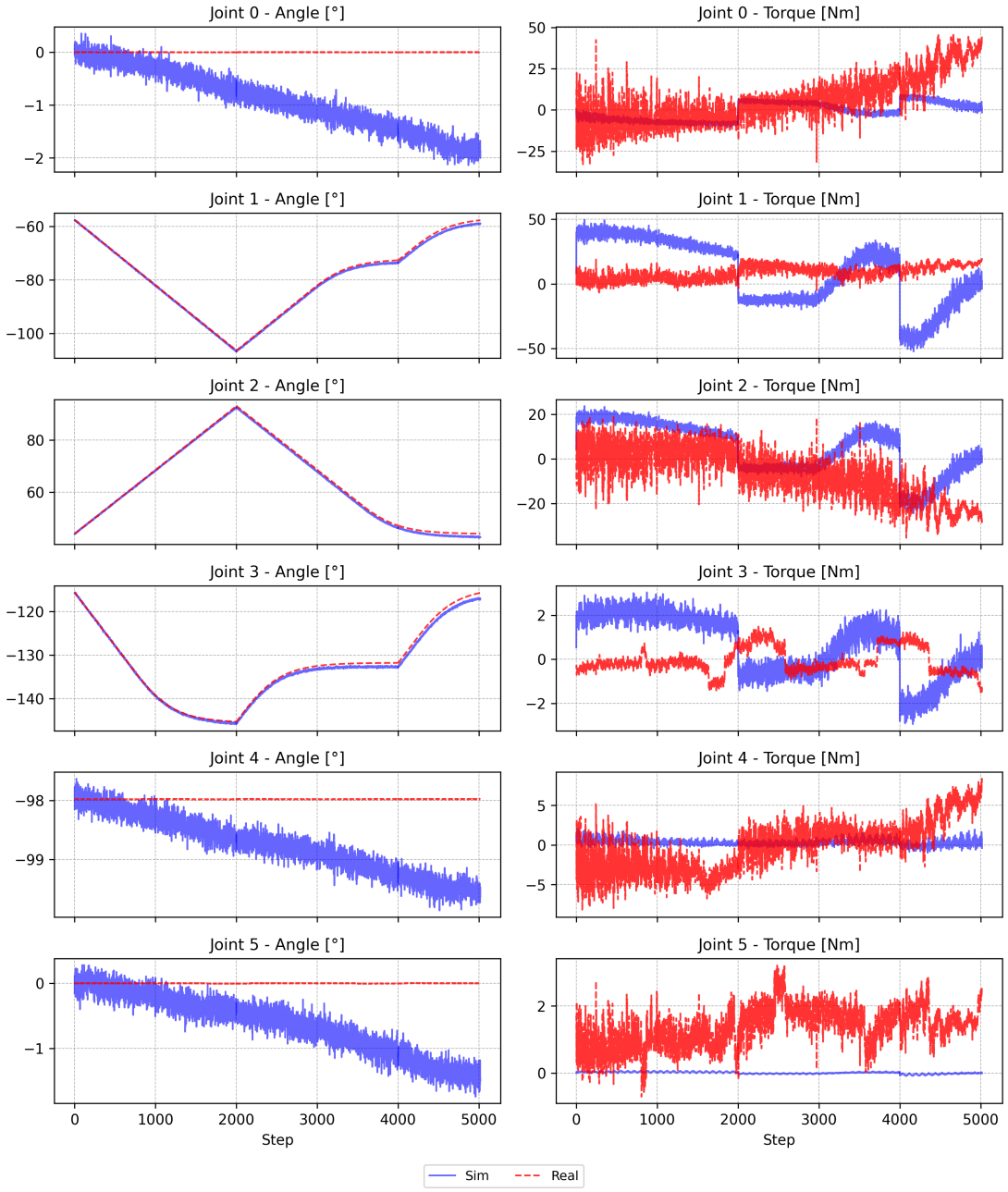

Figure 5.9: Comparison of the individual joint angles (in degrees) and measured torques (in Nm) of the UR5 robot in both the simulation and the real setup. A predefined sequence of action commands is sent simultaneously to both domains. The resulting values over time are visualized with dotted red lines for the real robot and solid blue lines for the Digital Twin.

5.4 RL Pretraining for the Grasping Task

Now that the robot control procedure and the cube position tracking are validated, a sufficiently capable policy has to be produced to serve as a starting point of the PACT workflow. This procedure is performed entirely in the IsaacLab environment using the Digital Twin. As stated previously, the curriculum consists of multiple reward-controlled curriculum levels. The results of each individual level are described below.

5.4.1 Keeping the Cube in Sight

After about 100 training episodes in the first curriculum step with 20 parallel agents, the algorithm is able to reliably keep the cube centered in the field of vision.

As the algorithm's actions are also subject to exploration encouraging noise during training, the agent learned to keep the cube centered, which secures its presence in the viewport even when the arm jitters or slightly changes its position.

In Figure 5.10, this behavior can be seen over multiple runs. The cube is locked into a specific position slightly above the center of the viewport. If initialized in a state where no cube is visible, the algorithm quickly adjusts the position of the camera on the endeffector as soon as the cube appears on the viewport.

5.4.2 Approaching the Cube

After learning how to keep the cube in sight, the next reward element of the curriculum is introduced. Now the distance from the cube to the camera is used to calculate the reward signal related to how close the agent moves its end-effector to the cube. Since this value can only be calculated when the cube is visible, this reward simply overlays the first, less sophisticated one.

The parameters of the exponential equation were empirically set to $\alpha = 0.02$ and $k = 5$, which led to a convergence towards stable and consistent behavior during training. With these values the reward increases starting at the initial position of about 1 meter, from almost zero, exponentially as shown in Figure 3.10 with a cutoff at 0.2 m.

The goal of this curriculum step is to maintain the behavior from the first curriculum and add an approach movement towards the cube. After another 500 steps with the

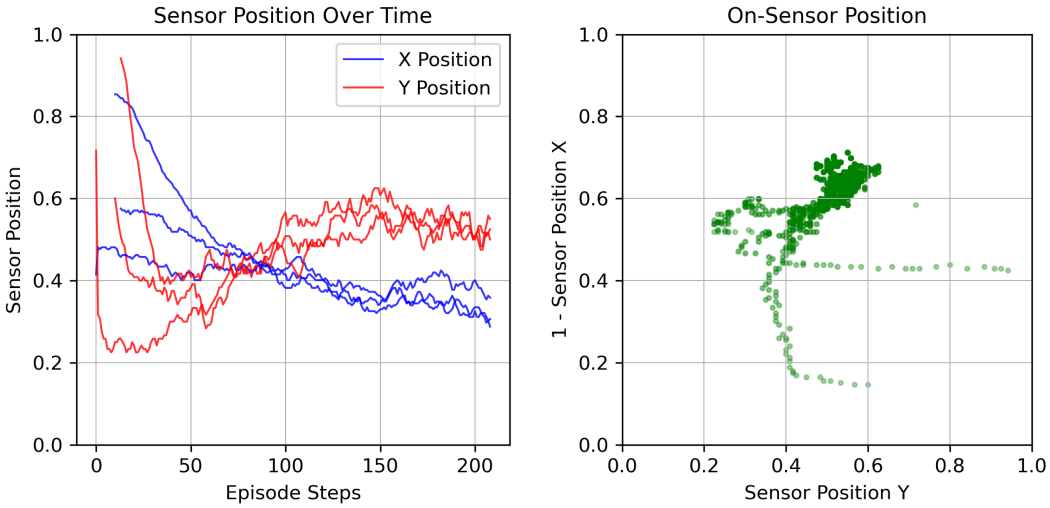

Figure 5.10: Normalized cube position on the camera sensor over multiple subsequent episodes: On the left both x and y coordinates are plotted over the episodes timesteps.
On the right the position of the cube on the camera image at each step is marked as a dot with increasing opacity over time showing where the cube's position would be on the observed image. As the position is calculated from the top left corner, x has to be flipped by calculating 1-x.

same training setup, the behavior developed as shown in Figure 5.11. The position of the cube on the sensor slightly shifted towards the left edge, but continued to stay within the viewport throughout the episode. The bottom plot clearly shows a steady approach towards the cube with a constant decrease in distance over the course of an episode. At the end, the distance corresponds approximately to the distance between the tip of the closed effector and the attachment point of the camera.

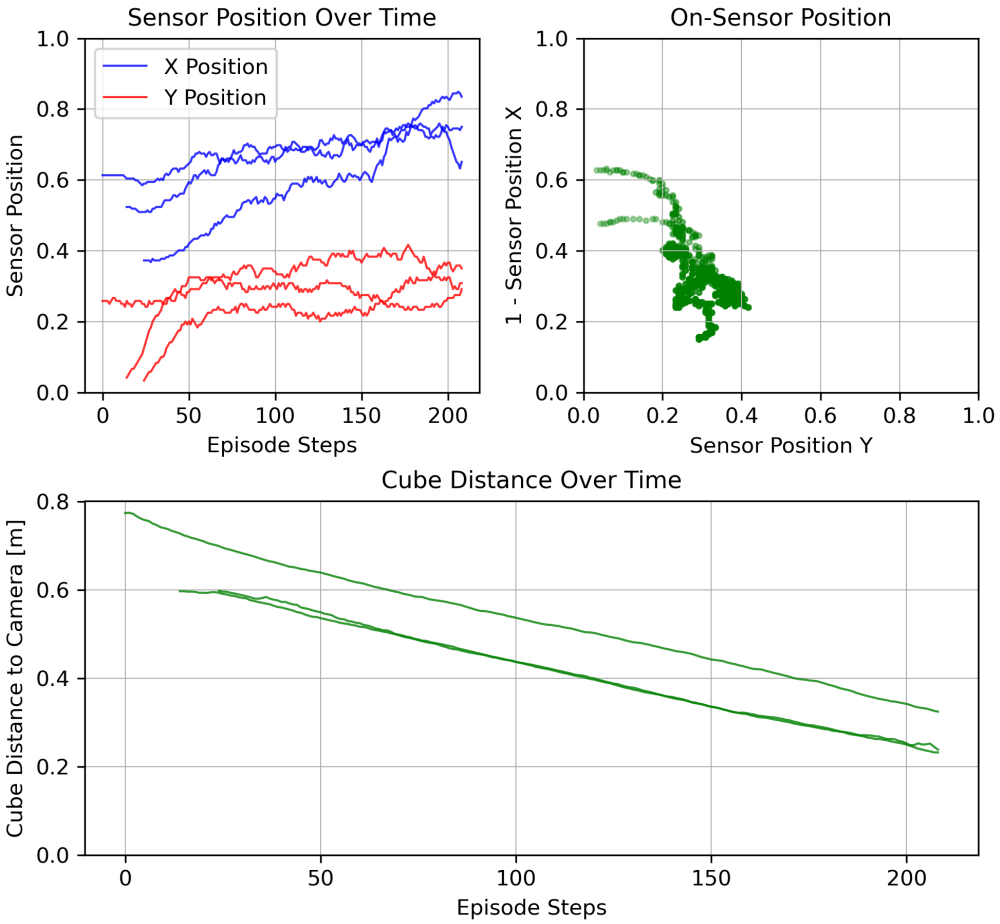

Figure 5.11: Normalized cube position on the camera sensor as shown in Figure 5.10 on the top half. The bottom graph shows the distance between the cube and the depth camera across the episode in multiple subsequent runs.

Although this seems to be a good behavior in theory, other sensory data reveal a major problem. If the torque at the joints of the arm is analyzed, it can be seen that critically high values occur during execution, peaking at way over 200 Nm. As shown in subsubsection 3.2.1, depending on the joints, the maximum torques of the UR5 range from 28

Nm to 150 Nm. Therefore, the current behavior will certainly break the hardware. This leads to the next consequential step in the curriculum.

5.4.3 Reducing Load on the Hardware

The excessive joint torques are mostly caused by abrupt or inefficient motions, with particularly high values often resulting from collisions with the table. This not only puts unnecessary load on the actuators but also indicates suboptimal policy behavior. By incorporating torque-related penalties into the reward function, the policy is guided toward discovering smoother, collision-free trajectories that are both more efficient and better aligned with the physical limitations of the hardware.

As soon as the next curriculum step and the corresponding penalties are introduced, the performance temporarily drops, and the algorithm starts to explore again. After about 500 training episodes in 20 parallel environments, the performance recovers and the behavior is relearned, but with significantly lower torques, demonstrating the algorithm's capability to re-optimize under stricter physical limitations. In Figure 5.12 the torques before and after penalty introduction are shown as mean values, with the penalties producing a significant reduction of more than tenfold in some areas.

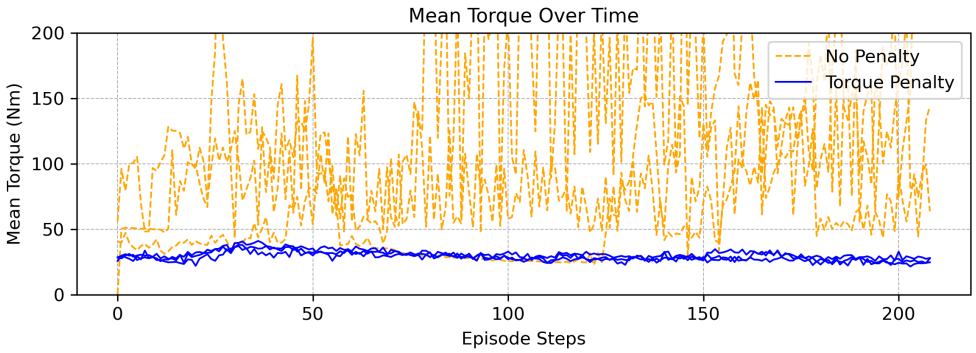

Figure 5.12: Mean torque over all joints of the articulation illustrating the overall level of of torque during the task execution. The torque curve of three subsequent runs before the introduction of the torque-related penalties is shown in yellow dashed lines. The blue lines show three runs with the policy which was then trained using the penalties.

This part of the curriculum is essential to ensure that, upon transfer to the real robot, the probability of damage to the system or its surroundings is reduced.

5.4.4 Grasping the Cube

The most challenging part in the curriculum is the grasping procedure. Building on the previous steps, the gripper is already steered in close proximity to the cube, and the agent explores interacting with it. Many movements do, however, result in undesirable collisions. There is a large number of dangerous trajectories and only very limited actions that lead to the goal behavior. In order to reach a sufficient performance, the agent has to learn for around 8000 episodes in 20 parallel environments with a training time of three full days. Tracking the grasp success over 50 runs starting from a typical resting position of the arm leads to 34 successful grasps, resulting in an average grasp success rate of about 70%. Figure 5.13 shows the full successful grasping behavior resulting from this CL level. The policy also developed emergent behaviors such as regrasping when the cube slips from the tools grip.

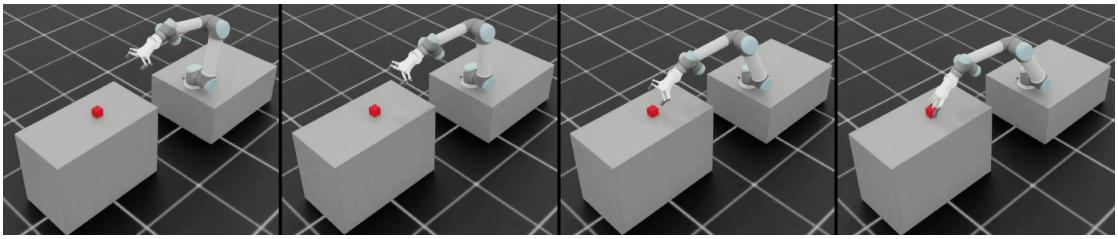

Figure 5.13: Learned grasping behavior in the last level of the curriculum.

Investigating the policy in several simulated runs shows that, although during training the initial states are randomized strongly around a typical resting position of the arm, the resulting policy still remains sensitive to initial states. Experiments with a variety of different starting positions result in a range in grasp success from slightly above the previously described 70% to even less than 50% depending on the initial position.

5.5 Deployment of the PACT Framework

5.5.1 Transferring the Policy to Real Hardware

After the initial extensive pretraining reached a reward threshold indicating a base level of performance, the first interaction within the workflow takes place. Following the described process, the framework captures the current state from the real robot and transfers it into the simulator. A validation step in the simulation follows, after which

(when successful) the first switch to the real domain is performed. This part makes it possible to directly compare the behavior across both domains and to experience the Sim-to-Real Gap firsthand. Figure 5.14 and Figure 5.16 show the successful validation run and the following first transfer to the real domain.

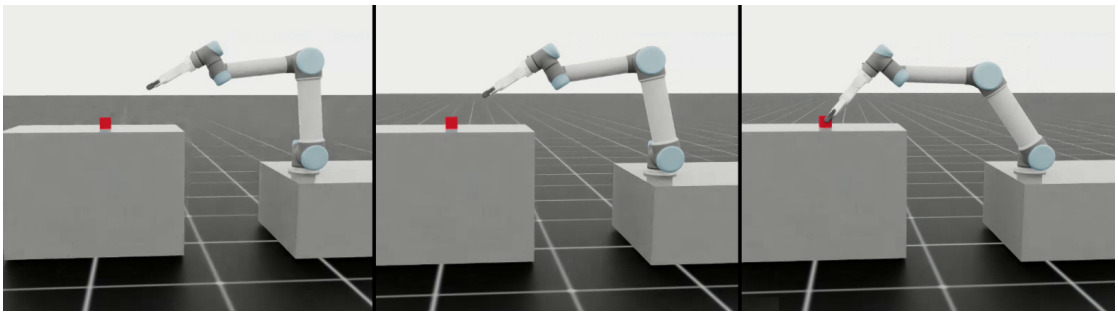

Figure 5.14: Validation run after reaching the training threshold with a successful grasping behavior starting from the current real world state.

Figure 5.15: First transfer of the policy to the real domain after a successful validation run in the simulation. The image on the right shows the tool stalling without initiating the grasp.

Visually apparent is that both trajectories are very similar in the beginning, and at first glance, there are no differences visible. The Sim-to-Real Gap becomes noticeable when the cube is very close to the camera and fine movements are required. In this situation, the state must closely match those seen in training for the policy to confidently initiate the grasp. During training, attempting to grasp at the wrong moment likely results in high torques and is penalized accordingly. This is exactly where the transferred policy struggles the most.

As soon as the tool reaches a grasping distance in the real domain, the tool position shifts slightly downwards and stalls. Only a very slight hovering movement is visible but no grasping is initiated within a reasonable time, triggering the timeout interrupt. This

interrupt results in a retraining procedure which will be described further in the next section.

This behavior is the typical Sim-to-Real Gap effect that causes a performance decline on domain transfer.

5.5.2 First Impressions on RL Retraining as Adaptation Procedure

Once the timeout interrupt is triggered, the PACT workflow initiates the adaptation procedure. The interrupt state in the simulation visually matches the real-world situation closely, again verifying the corresponding previous experiments.

In the current configuration, the adaptation procedure takes form as a continued RL training at the interrupt state with smaller randomization values. Upon reentering the training procedure, the policy's performance drops immediately during the first episodes and needs about 12 hours to recover to the previous performance. This seems to be linked to the previously described sensitivity to different initial states.

However, after recovery the algorithm is able to converge to a high mean reward and manages to repeatedly perform a successful grasp from that state.

Although the RL algorithm did recover, the entire retraining process by far exceeded any applicable time span for the robot to remain active during the adaptation procedure. This questions the suitability of interrupt-driven RL retraining as an adaptation process.

5.5.3 Reentering the System after Adaptation

Using the adapted policy to reenter into the workflow from the interrupt state reveals a significant improvement compared to the previous run. After the validation run in the simulation environment from the interrupt state succeeded, the policy takes over control of the real robot, picks up from the current state, and successfully performs the grasp.

Now, even when resetting the arm to a more distant resting position, the policy continues to approach the cube reliably and shows a significantly increased grasp success rate. This highlights that the retraining process not only solved the immediate issue but also improved the overall robustness of the policy. PACT provided a strong performance boost by applying focused retraining exactly at the point where the agent struggled.

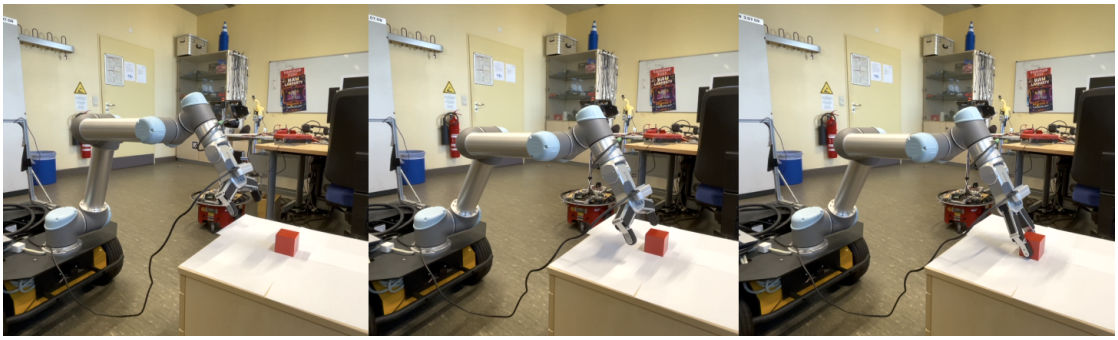

Figure 5.16: The adapted policy, retrained from the interrupt state now successfully continuing from the last state and grasping the cube.

PACT therefore provided a strong performance boost for the robotic agent by performing focused retraining on a situation that it struggled with.

5.5.4 Pushing the Limits of Adaptive Retraining

The first retraining process was time-consuming but successfully led to a significant improvement in the algorithm's real-world performance. However, the interrupt state handled in this case was less complex compared to more challenging scenarios, such as critical interrupts triggered by high-torque collisions. In order to push the limits of what RL retraining can do for adaptation to problematic states, a crash into the table is simulated by setting the torque interrupt threshold very low and manipulating the commands for the real-world run in a way that causes a collision with the table's surface. The interrupt is triggered correctly and the retraining is initiated. However, this state is quite difficult to recover from, as it severely limits what commands can be executed without damage. Any movement that results in a minor downward direction for the tool causes very high torques and is therefore penalized by the reward function.

The retraining from that state also causes a drop in performance, but unlike in the previous situation the policy never completely recovers. Even long after 12 hours of training, the agent got stuck in a local optimum of just elevating the arm and completely lost the goal-directed grasping behavior.

This demonstrates the limitations of RL retraining when used as an adaptation method for very complex situations.

6 Discussion and Interpretation of the Results

6.1 System Behavior: Steering and Cube Tracking

The steering logic not only enables precise movements, but is also well-suited to the reinforcement learning concept of sending commands step-by-step and evaluating the resulting new state. Since the commands are converted into angle deltas and then result in absolute joint angles which are applied using position controllers, the system is less sensitive to delays. As a result, applying the same sequence of commands in either the simulation or the real world reliably leads to the same joint configurations. The comparison between the two domains highlights this robustness, as only minor differences in joint angles accumulate over time.

The cube tracking algorithm calculates the position reliably within the constraints of the very simplified detection method. The discrepancies stay within the range of a few centimeters providing a reasonable accuracy for this setup. When it comes to robustness to different lighting scenes, the algorithm reaches its limitations at more complex oblique lighting or very bright direct sunlight exposure.

6.2 Evaluating the PACT Framework Design

The architecture developed in this work enables the simultaneous operation of a Digital Twin and a real-world robot by incrementally abstracting both domains to a simple, gym-compliant interface. The manager layer is able to switch between the two domains based on the defined logic. Thanks to the abstraction to a common interaction standard, the RL policy can connect to either domain interchangeably and switch domains in real time. As the architecture leverages the observation space to gain insights on the system

state, it reliably detects undesirable situations during execution in both domains and triggers interrupts, without the need to violate the gym-compliance. The system is able to transfer information from the real world to the twin and can therefore accurately recreate the current real-world state in the simulation environment.

The simplified cube detection calculates the cube's position with sufficient accuracy without relying on advanced object detection methods. However, it depends on a reasonably well-lit environment and assumes that the cube is the only red object present in the scene.

By leveraging NVIDIA IsaacSim and IsaacLab, the system benefits from a state-of-the-art simulator with GPU-accelerated performance and advanced domain randomization functionality.

The architecture developed in this work was designed to enable seamless interaction between a Digital Twin and a real-world robot, allowing both domains to operate under a unified, gym-compliant interface. This goal has largely been achieved, as the framework supports real-time switching between simulation and reality through a manager layer that handles domain transitions based on predefined logic. This flexibility ensures that the reinforcement learning policy can operate across both domains without modification.

A key strength of the architecture is its ability to detect undesirable system states through the observation space, triggering interrupts as needed without violating gym-compliance. This feature proved effective in both domains and contributed to the system's robustness. Additionally, the framework reliably transfers the current state from the real robot into the simulation, enabling accurate recreation of real-world conditions within the Digital Twin.

However, the architecture does face some limitations. The simplified cube detection method, while efficient and easy to implement, is sensitive to lighting conditions and assumes the cube is the only red object in the environment. This restricts its applicability in more complex or dynamic scenes. Incorporating more advanced detection methods or sensor fusion could address these limitations and improve robustness.

By leveraging NVIDIA Isaac Sim and IsaacLab, the architecture benefits from a powerful simulation environment with GPU-accelerated performance and domain randomization features. While this setup performs well for a single robotic arm, the GPU becomes a limiting factor when attempting to scale to multiple parallel environments with active RGB

and depth cameras. For more complex scenarios, higher-end hardware or optimization of the simulation workloads would be necessary.

6.3 Challenges and Insights from PACT Deployment

6.3.1 Ease of Deployment to the Real World

Thanks to the architecture of the PACT framework, deployment from the simulation environment to the real world works seamlessly. The layered abstraction within PACT allows the Digital Twin and the real robot to coexist smoothly, enabling control over both domains in parallel, sequentially, or independently with minimal effort. As a result, deploying the policy to the real robot requires no human intervention and is fully integrated into the standard workflow of PACT.

However, real-time control is sensitive to delays, which can cause stuttering in the robot's movements. Since the resource-intensive simulation engine continues running during real-world execution, steering commands are subject to occasional delays that affect the smoothness of the motion. While this effect can be minimized by reducing the overall speed of movement, it highlights an area for further improvement.

6.3.2 Effectiveness of the Pretraining Procedure

Due to parallelization and the fast simulation software, the pretraining is able to develop a goal-directed policy for the non-trivial continuous control of the robotic arm by simply manipulating the joint values within a training time of 3 days. Keeping in mind that this means the algorithm has to learn what is normally calculated as inverse kinematics from scratch and steer a robotic system through an unseen environment only using a few numerical values, this is still impressive. The hardware used to run the training just meets the minimum requirements for IsaacSim and therefore definitely is a strong limiting factor regarding training time.

The reward shaping curriculum learning strategy helped to effectively guide the policy towards the otherwise very sparse rewards. Training without this supporting strategy was also tested and resulted in hours of training without any progress. As the grasping procedure took by far the largest share of the training duration, it would also be feasible

to further divide the grasping into more curriculum steps. First experiments showed that reward elements like partial grasps can help in discovering the grasping behavior, but also tend to mislead the agent to new local optima.

Randomizing the simulation parameters and actions as well as observation values is a common technique to help increase robustness and was also implemented here. Nonetheless, the resulting policy leaves room for improvement regarding the robustness to initial states and the domain transfer to the real world, as described in more detail below.

6.3.3 Implications for the Robustness of the Pretrained Policy

The initially trained policy reliably steers the robotic arm by setting its joint angles, demonstrating robust approaching behavior across a variety of initial states. This robustness even carries over to the real-world domain, successfully transferring the approach control from simulation without major issues.

Grasping the cube, however, remains a greater challenge. In the simulation, the agent learns to grasp the cube after extensive pretraining, although it does not consistently achieve a success rate near 100% even after three days of continuous training. The grasp success varies depending on the initial position of the cube and arm, and therefore still lacks robustness regarding the initial states, despite strong randomization during training.

In the real domain, after a successful approach, the policy initially fails to start the grasping process. Here, the robustness from mere pretraining is not strong enough for a one-shot transfer to the real domain.

The first trained algorithm therefore shows basic robustness to the domain differences in core steering functionality, but when it comes to more precise movement in close proximity to the cube and the detection of the correct grasping position, the algorithm would need a stronger robustness to fully survive the domain transfer.

6.3.4 RL Retraining as Adaptation Procedure

Using RL as a method to adapt to critical situations like torque-related interrupts proved to be challenging in this setup. Although the workflow reliably detects interrupts and captures the relevant state, continuing RL training from that point did not lead to

recovery or improvement. The retraining process led the policy into a local optimum, where the agent adopted a safe but ineffective strategy to elevate the tool from the table without progressing toward the actual goal. Even after excessive training times, this local optimum could not be overcome.

However, this limitation does not disqualify RL as an adaptation strategy. Focused retraining was also successfully applied to the less complex situation, where the policy initially struggled to grasp the cube. After retraining, the policy was able to repeatedly perform the grasp in the real world, even when resetting the arm to more distant resting positions. This demonstrates that targeted RL retraining can improve performance, provided the complexity of the interrupt state remains manageable.

Nonetheless, the time needed for a full retraining in the current setup still exceeds what is practical for real-world applications. Thus, more stable approaches like planning algorithms or even a set of different pretrained policies that are selected and validated by PACT could offer a more reliable and efficient way to handle critical situations.

6.3.5 RL as the Source of Intelligence

Although RL might not be the ideal approach to help the agent adapt to critical situations during live execution, in the initial pretraining the policy was able to learn complex motor strategies from scratch, despite having access to only simplified observational information. Additionally supporting elements such as reward shaping and curriculum learning can be included in the learning process, making it easier for the algorithm to develop stable and effective strategies. The produced policy was robust enough to constantly perform the approaching behavior even after the one-shot transfer to the real world. The skill of grasping was initially lost in the Sim-to-Real Gap but was recovered by PACT using targeted retraining.

Despite some challenges, RL has demonstrated its potential as a powerful core component for enabling intelligent behavior on a real robotic system.

6.3.6 Evaluation of the Key Success Factors

With the framework fully implemented and practical experience gained from real-world application the key success factors can now be assessed.

Is the Digital Twin an accurate replication of the real robot?

The Digital Twin within IsaacSim closely resembles the real robot regarding the positional state. The joint measurements while executing the same commands on the real system and the twin in parallel showed only minor deviations in angle values. The torque values were less aligned but didn't reveal drastic differences.

Does the system capture real-world states with sufficient accuracy?

The joint positions in this setup can be directly obtained from the internal measurements of the robotic arm and therefore provide an accurate measurement of the real state. The cube position is captured using a custom algorithm and is less precise than the joint positions but still provides sufficient accuracy within a few centimeters.

Are interrupts detected reliably?

PACT's interrupt detection operates at a very high sampling rate, allowing it to quickly and reliably identify interrupt conditions such as high torque states. However, the conducted tests only covered selected example scenarios, and the detection accuracy may vary depending on the type and quality of the observational data used.

Is the system able to switch correctly between the domains?

PACT has proven to seamlessly switch back and forth between the real system and its Digital Twin. This capability not only works well within this setup but also provides a strong foundation for future developments and experiments.

Is the Digital Twin also capable to provide a sufficient pretraining ground?

Pretraining using PACT's IsaacLab integration provided a solid foundation for the RL policy. It was able to learn complex motor control strategies that allowed it to approach the cube and with additional targeted retraining, it was able to robustly bridge the Sim-to-Real Gap, achieving reliable grasping performance in the real domain.

Can dynamic adaptative RL retraining make the policy more capable?

Dynamic adaptation through reinforcement learning retraining showed mixed results in this setup. In less complex scenarios, targeted retraining significantly improved the policy's performance, increasing the ability to successfully grasp the cube, even from varied starting positions. However, in more challenging situations, like recovering from critical interrupt states caused by high-torque collisions, retraining struggled to overcome local optima. This suggests that while dynamic adaptation through RL retraining can boost performance in certain cases, it is not sufficient on its own for handling all critical scenarios. Also, more structured or guided adaptation methods may be needed to make the process more efficient, avoiding excessive training times that are impractical in real-world applications.

6.4 Strengths and Limitations of PACT

6.4.1 Strengths

PACT connects the real robot and its Digital Twin in a tightly integrated relationship, enabling both entities to operate in parallel or independently while maintaining full access to send commands and receive data from either domain. Switching between the two is seamless and requires no additional effort. This flexibility provides a strong foundation for a variety of use cases. The framework proved its usefulness early on over the course of this thesis, helping to analyze the Sim-to-Real Gap by executing a set of commands on both domains simultaneously while also reading and comparing the observations.

The framework also incorporates a GPU-accelerated pretraining pipeline leveraging IsaacLab, domain randomization, and curriculum learning for the development of robust RL policies.

PACT captures critical states and, upon detection in the real world, it transfers them to the Digital Twin for exploration and adaptation procedures. This allows targeting specific weaknesses in a safe environment. In the example grasping task this allowed the policy to improve its ability to grasp the cube in the real world, essentially bridging what remained of the Sim-to-Real Gap in this setup. Before returning to the real domain, the Digital Twin is used to validate the suitability of the adapted behavior, which reduces the risk of deploying harmful policies.

The framework therefore combines pretraining, deployment, and adaptation in a single workflow and was able to prove its applicability in a real-world experiment.

6.4.2 Limitations

Due to the computational load of running the simulation alongside the time-sensitive control method, the real robot currently has to operate at reduced speeds to prevent excessive jittering, which can lead to high torques on the arm. A possible solution would be to offloading the simulation to a separate, dedicated machine or increase the hardware performance of the current setup.

Although PACT successfully transferred basic motor strategies, more complex behaviors like grasping still required additional refinement after pretraining. The Sim-to-Real Gap remains a challenge and greater domain differences that cannot be covered by domain randomization are not taken into account by the current state of the framework.

RL as the main strategy for interrupt-driven adaptation is only of limited applicability, as the required training times are too long to be used in practice during live execution. Training times could be further reduced by more capable simulation hardware, but training RL algorithms is still relatively unstable and does not always produce the intended improvement.

6.5 Further Research Directions

To alleviate the issue of long retraining runs, PACT could also be used to select from a set of pretrained policies. For example, one could be an expert at grasping while another excels at recovering the arm from complex problematic states. The framework could then use its interrupt logic to initiate evaluation runs in the simulator to return with the best possible policy for every situation.

Since the task setup was reduced significantly, future implementations could experiment with more complex scenarios including multiple elements that are detected and recreated in the simulation.

As larger Sim-to-Real Gaps could become problematic in the current configuration, it would also be conceivable to include domain adaptation strategies into PACT. With

additional data covering more aspects of the environment, the adaptation could exceed the robot's state to include adjustments to environmental or simulation parameters, better aligning the Digital Twin with the real world. This way, the pitfall described when introducing the framework could be mitigated.

Building on that, the parallel execution could be used to continuously monitor the alignment of the simulation and the real world, and upon drift, modify simulation parameters to improve the accuracy of the simulation.

These directions offer potential pathways to further strengthen the adaptability and robustness of PACT, expanding its applicability to more complex tasks and environments.

7 Conclusion

7.1 Summary of Contributions

In this thesis, the PACT (Predictive Adaptive Collaborative Twin) framework was designed, implemented, and evaluated as an integrated system connecting a real robotic platform with its Digital Twin.

The framework allows seamless switching between the real robot and the simulation, providing a unified interface for training, validation, and adaptation workflows.

To demonstrate the framework's practical applicability, a simplified real-world robotic grasping task was created, together with a corresponding Digital Twin using NVIDIA IsaacSim. Core components of PACT were analyzed and validated individually. The Sim-to-Real Gap of the setup at hand was measured by executing synchronized action sequences and analyzing the deviations between the two domains.

PACT was finally successfully deployed by pretraining a reinforcement learning policy using the Digital Twin, supported by strategies for increased policy robustness and training efficiency, and finally transferring it to the real robot. Dynamic adaptation through targeted retraining at interrupt states was tested and improved the performance of the policy in less complex cases, while highlighting limitations for more challenging scenarios.

Through this work, the framework proved its capability to manage training, deployment, and adaptation, within a cohesive workflow. At the same time, challenges regarding computational requirements, Sim-to-Real transfer for complex behaviors, and the stability of live adaptation were identified, and future research directions were suggested.

7.2 Closing Remarks

The results illustrate the potential of this concept to increase robotic intelligence through the use of tightly integrated Digital Twins. However, the true capabilities of this approach are yet to be fully explored. This work therefore provides a strong foundation for future developments and offers a good starting point for even more adaptive and intelligent robotic systems that push the boundaries of what robotics can achieve in real-world applications.

Bibliography

[1] Chen, Ya-Ling ; Cai, Yan-Rou ; Cheng, Ming-Yang: Vision-based robotic object grasping—a deep reinforcement learning approach. In: *Machines* 11 (2023), Nr. 2, S. 275

[2] Dulac-Arnold, Gabriel ; Levine, Nir ; Mankowitz, Daniel J. ; Li, Jerry ; Paduraru, Cosmin ; Gowal, Sven ; Hester, Todd: Challenges of real-world reinforcement learning: definitions, benchmarks and analysis. In: *Machine Learning* 110 (2021), Nr. 9, S. 2419–2468

[3] Github, Isaac L.: *Franka Cabinet Direct Workflow Environment*. 2024. – URL https://github.com/isaac-sim/IsaacLab/blob/main/source/isaaclab_tasks/isaaclab_tasks/direct/franka_cabinet/agents/rsl_rl_ppo_cfg.py. – Accessed: March 6, 2025

[4] Grau, Antoni ; Indri, Marina ; Bello, Lucia L. ; Sauter, Thilo: Robots in industry: The past, present, and future of a growing collaboration with humans. In: *IEEE Industrial Electronics Magazine* 15 (2020), Nr. 1, S. 50–61

[5] Grieves, Michael ; Vickers, John: *Digital Twin: Mitigating Unpredictable, Undesirable Emergent Behavior in Complex Systems*. S. 85–113, Springer International Publishing Switzerland, 08 2017. – ISBN 978-3-319-38754-3

[6] Ibarz, Julian ; Tan, Jie ; Finn, Chelsea ; Kalakrishnan, Mrinal ; Pastor, Peter ; Levine, Sergey: How to train your robot with deep reinforcement learning: lessons we have learned. In: *The International Journal of Robotics Research* 40 (2021), Nr. 4-5, S. 698–721

[7] Intel RealSense: *Intel RealSense Software Development Kit*, 2025. – URL https://github.com/IntelRealSense/librealsense. – Accessed: March 8, 2025

[8] Intel RealSense: *ROS Wrapper for Intel RealSense Cameras*, 2025. – URL https://github.com/IntelRealSense/realsense-ros. – Accessed: March 8, 2025

[9] IntelRealsense: *Intel RealSense D435 Product Page*. 2025. – URL https://www.intelrealsense.com/depth-camera-d435/. – Accessed: Februar 20, 2025

[10] Kaelbling, Leslie P. ; Littman, Michael L. ; Moore, Andrew W.: Reinforcement learning: A survey. In: *Journal of artificial intelligence research* 4 (1996), S. 237–285

[11] Kiran, B R. ; Sobh, Ibrahim ; Talpaert, Victor ; Mannion, Patrick ; Al Sallab, Ahmad A. ; Yogamani, Senthil ; Pérez, Patrick: Deep reinforcement learning for autonomous driving: A survey. In: *IEEE Transactions on Intelligent Transportation Systems* 23 (2021), Nr. 6, S. 4909–4926

[12] Kreuzer, Tim ; Papapetrou, Panagiotis ; Zdravkovic, Jelena: Artificial intelligence in digital twins—A systematic literature review. In: *Data Knowledge Engineering* 151 (2024), 05, S. 102304

[13] Kurkcu, Anil ; Acar, Cihan ; Campolo, Domenico ; Tee, Keng P.: GloCAL: Glocalized Curriculum-Aided Learning of Multiple Tasks with Application to Robotic Grasping. In: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* IEEE (Veranst.), 2021, S. 7089–7096

[14] Makoviychuk, Viktor ; Wawrzyniak, Lukasz ; Guo, Yunrong ; Lu, Michelle ; Storey, Kier ; Macklin, Miles ; Hoeller, David ; Rudin, Nikita ; Allshire, Arthur ; Handa, Ankur u. a.: Isaac gym: High performance gpu-based physics simulation for robot learning. In: *arXiv preprint arXiv:2108.10470* (2021)

[15] Malik, Ali A. ; Brem, Alexander: Digital twins for collaborative robots: A case study in human-robot interaction. In: *Robotics and Computer-Integrated Manufacturing* (2020), 11

[16] Mazumder, Antar ; Sahed, Md F. ; Tasneem, Zinat ; Das, Prangon ; Badal, Faisal ; Ali, Md ; Ahamed, Md. H. ; Abhi, Sarafat ; Sarker, Subrata ; Das, Sajal ; Hasan, Md ; Islam, Manirul ; Robiul Islam, Md: Towards Next Generation Digital Twin in Robotics: Trends, Scopes, Challenges, and Future. In: *Heliyon* 9 (2023), 01

[17] Mazumder, Antar ; Sahed, Md F. ; Tasneem, Zinat ; Das, Prangon ; Badal, Faisal ; Ali, Md ; Ahamed, Md. H. ; Abhi, Sarafat ; Sarker, Subrata ; Das, Sajal ; Hasan, Md ; Islam, Manirul ; Robiul Islam, Md: Towards Next Generation Digital Twin in Robotics: Trends, Scopes, Challenges, and Future. In: *Heliyon* 9 (2023), 01

[18] Muratore, Fabio ; Ramos, Fabio ; Turk, Greg ; Yu, Wenhao ; Gienger, Michael ; Peters, Jan: Robot learning from randomized simulations: A review. In: *Frontiers in Robotics and AI* 9 (2022), S. 799893

[19] Narvekar, Sanmit ; Peng, Bei ; Leonetti, Matteo ; Sinapov, Jivko ; Taylor, Matthew E. ; Stone, Peter: Curriculum learning for reinforcement learning domains: A framework and survey. In: *Journal of Machine Learning Research* 21 (2020), Nr. 181, S. 1–50

[20] Nguyen, Hai ; La, Hung: Review of deep reinforcement learning for robot manipulation. In: *2019 Third IEEE international conference on robotic computing (IRC)* IEEE (Veranst.), 2019, S. 590–595

[21] Niu, Haoyi ; Qiu, Yiwen ; Li, Ming ; Zhou, Guyue ; Hu, Jianming ; Zhan, Xianyuan u. a.: When to trust your simulator: Dynamics-aware hybrid offline-and-online reinforcement learning. In: *Advances in Neural Information Processing Systems* 35 (2022), S. 36599–36612

[22] NVIDIA: *Isaac Lab Documentation: Articulation Asset*, 2025. – URL https://isaac-sim.github.io/IsaacLab/main/source/api/lab/isaaclab.assets.html#articulation. – Accessed: March 6, 2025

[23] NVIDIA: *Isaac Lab Documentation: Camera Sensors*, 2025. – URL https://isaac-sim.github.io/IsaacLab/main/source/overview/core-concepts/sensors/camera.html. – Accessed: March 6, 2025

[24] NVIDIA: *Isaac Lab Documentation: Isaac Lab Ecosystem*, 2025. – URL https://isaac-sim.github.io/IsaacLab/main/source/setup/ecosystem.html. – Accessed: March 6, 2025

[25] NVIDIA: *Nvidia PhysX*, 2025. – URL https://developer.nvidia.com/physx-sdk. – Accessed: March 3, 2025

[26] Onrobot: *Onrobot RG6 Product Page*. 2025. – URL https://onrobot.com/de/produkte/greifer-rg6. – Accessed: Februar 20, 2025

[27] Open Source Robotics Foundation: *ROS 2 Humble Hawksbill Release*. 2025. – URL https://docs.ros.org/en/humble/Releases.html. – Accessed: 2025-04-28

[28] Open Source Robotics Foundation: *ROS Wiki*. 2025. – URL https://wiki.ros.org/. – Accessed: 2025-04-28

[29] OpenCV: *Camera Calibration and 3D Reconstruction*, 2025. – URL https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html#projectpoints. – Accessed: March 6, 2025

[30] Patel, Shivansh ; Yin, Xinchen ; Huang, Wenlong ; Garg, Shubham ; Nayyeri, Hooshang ; Fei-Fei, Li ; Lazebnik, Svetlana ; Li, Yunzhu: A Real-to-Sim-to-Real Approach to Robotic Manipulation with VLM-Generated Iterative Keypoint Rewards. In: *arXiv preprint arXiv:2502.08643* (2025)

[31] Peng, Weikun ; Lv, Jun ; Zeng, Yuwei ; Chen, Haonan ; Zhao, Siheng ; Sun, Jichen ; Lu, Cewu ; Shao, Lin: TieBot: Learning to Knot a Tie from Visual Demonstration through a Real-to-Sim-to-Real Approach. In: *arXiv preprint arXiv:2407.03245* (2024)

[32] Perez, Javier A. ; Deligianni, Fani ; Ravi, Daniele ; Yang, Guang-Zhong: Artificial intelligence and robotics. In: *arXiv preprint arXiv:1803.10813* 147 (2018), S. 2–44

[33] Raviola, Andrea ; Guida, Roberto ; Bertolino, Antonio C. ; De Martin, Andrea ; Mauro, Stefano ; Sorli, Massimo: A comprehensive multibody model of a collaborative robot to support model-based health management. In: *Robotics* 12 (2023), Nr. 3, S. 71

[34] Rudin, Nikita ; Hoeller, David ; Reist, Philipp ; Hutter, Marco: Learning to Walk in Minutes Using Massively Parallel Deep Reinforcement Learning. In: *Proceedings of the 5th Conference on Robot Learning* Bd. 164, PMLR, 2022, S. 91–100. – URL https://proceedings.mlr.press/v164/rudin22a.html

[35] Salvato, Erica ; Fenu, Gianfranco ; Medvet, Eric ; Pellegrino, Felice A.: Crossing the reality gap: A survey on sim-to-real transferability of robot controllers in reinforcement learning. In: *IEEE Access* 9 (2021), S. 153171–153187

[36] Sekkat, Hiba ; Tigani, Smail ; Saadane, Rachid ; Chehri, Abdellah: Vision-Based Robotic Arm Control Algorithm Using Deep Reinforcement Learning for Autonomous Objects Grasping. In: *Applied Sciences* 11 (2021), Nr. 17. – URL https://www.mdpi.com/2076-3417/11/17/7917. – ISSN 2076-3417

[37] Silver, David ; Huang, Aja ; Maddison, Chris J. ; Guez, Arthur ; Sifre, Laurent ; Van Den Driessche, George ; Schrittwieser, Julian ; Antonoglou, Ioannis ; Panneershelvam, Veda ; Lanctot, Marc u. a.: Mastering the game of Go with deep neural networks and tree search. In: *nature* 529 (2016), Nr. 7587, S. 484–489

[38] Singh, Bharat ; Kumar, Rajesh ; Singh, Vinay P.: Reinforcement learning in robotic applications: a comprehensive survey. In: *Artificial Intelligence Review* 55 (2022), Nr. 2, S. 945–990

[39] Sun, Yuzhu ; Van, Mien ; McIlvanna, Stephen ; Nhat, Nguyen M. ; Olayemi, Kabirat ; Close, Jack ; McLoone, Seán: *Digital Twin-Driven Reinforcement Learning for Obstacle Avoidance in Robot Manipulators: A Self-Improving Online Training Framework*. 2024. – URL https://arxiv.org/abs/2403.13090

[40] Sutton, Richard S.: Reinforcement learning: An introduction. In: *A Bradford Book* (2018)

[41] Tang, Chen ; Abbatematteo, Ben ; Hu, Jiaheng ; Chandra, Rohan ; Martín-Martín, Roberto ; Stone, Peter: Deep reinforcement learning for robotics: A survey of real-world successes. In: *Annual Review of Control, Robotics, and Autonomous Systems* 8 (2024)

[42] Tobin, Josh ; Fong, Rachel ; Ray, Alex ; Schneider, Jonas ; Zaremba, Wojciech ; Abbeel, Pieter: Domain randomization for transferring deep neural networks from simulation to the real world. In: *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)* IEEE (Veranst.), 2017, S. 23–30

[43] Torne, Marcel ; Simeonov, Anthony ; Li, Zechu ; Chan, April ; Chen, Tao ; Gupta, Abhishek ; Agrawal, Pulkit: Reconciling reality through simulation: A real-to-sim-to-real approach for robust manipulation. In: *arXiv preprint arXiv:2403.03949* (2024)

[44] Towers, Mark ; Kwiatkowski, Ariel ; Terry, Jordan ; Balis, John U. ; De Cola, Gianluca ; Deleu, Tristan ; Goulao, Manuel ; Kallinteris, Andreas ; Krimmel, Markus ; KG, Arjun u. a.: Gymnasium: A standard interface for reinforcement learning environments. In: *arXiv preprint arXiv:2407.17032* (2024)

[45] Universal Robots: *Universal Robots Joint Torque Limits*. 2015. – URL https://www.universal-robots.com/articles/ur/robot-care-maintenance/max-joint-torques-cb3-and-e-series/. – Accessed: Februar 10, 2025

[46] Universal Robots: *Universal Robots ROS2 Driver*, 2025. – URL https://github.com/UniversalRobots/Universal_Robots_ROS2_Driver. – Accessed: March 8, 2025

[47] Universal Robots: *Universal Robots Website*. 2025. – URL https://www.universal-robots.com. – Accessed: Februar 10, 2025

[48] Universal Robots: *UR5e Safety Functions and Interfaces Manual*, 2025. – URL https://www.universal-robots.com/manuals/EN/HTML/SW5_19/Content/prod-usr-man/complianceUR5e/H_g5_sections/safetyFunctionsAndinterfaces/configurable_functions.htm. – Accessed: March 6, 2025

[49] Vinyals, Oriol ; Babuschkin, Igor ; Czarnecki, Wojciech M. ; Mathieu, Michaël ; Dudzik, Andrew ; Chung, Junyoung ; Choi, David H. ; Powell, Richard ; Ewalds, Timo ; Georgiev, Petko ; Oh, Junhyuk ; Horgan, Dan ; Kroiss, Manuel ; Danihelka, Ivo ; Huang, Aja ; Sifre, Laurent ; Cai, Trevor ; Agapiou, John P. ; Jaderberg, Max ; Vezhnevets, Alexander S. ; Leblond, Rémi ; Pohlen, Tobias ; Dalibard, Valentin ; Budden, David ; Sulsky, Yury ; Molloy, James ; Paine, Tom L. ; Gulcehre, Caglar ; Wang, Ziyu ; Pfaff, Tobias ; Wu, Yuhuai ; Ring, Roman ; Yogatama, Dani ; Wünsch, Dario ; McKinney, Katrina ; Smith, Oliver ; Schaul, Tom ; Lillicrap, Timothy ; Kavukcuoglu, Koray ; Hassabis, Demis ; Apps, Chris ; Silver, David: Grandmaster level in StarCraft II using multi-agent reinforcement learning. In: *Nature* 575 (2019), Nr. 7782, S. 350–354. – ISSN 0028-0836

[50] Weng, Lilian: *Domain Randomization for Sim2Real Transfer*. 2019. – URL https://lilianweng.github.io/posts/2019-05-05-domain-randomization/. – Accessed: 2025-03-05

[51] Wu, Philipp ; Escontrela, Alejandro ; Hafner, Danijar ; Abbeel, Pieter ; Goldberg, Ken: Daydreamer: World models for physical robot learning. In: *Conference on robot learning* PMLR (Veranst.), 2023, S. 2226–2240

[52] Wu, Yuxuan ; Pan, Lei ; Wu, Wenhua ; Wang, Guangming ; Miao, Yanzi ; Wang, Hesheng: RL-GSBridge: 3D Gaussian Splatting Based Real2Sim2Real Method for Robotic Manipulation Learning. In: *arXiv preprint arXiv:2409.20291* (2024)

[53] Wurman, Peter R. ; Barrett, Samuel ; Kawamoto, Kenta ; MacGlashan, James ; Subramanian, Kaushik ; Walsh, Thomas J. ; Capobianco, Roberto ; Devlic, Alisa ; Eckert, Franziska ; Fuchs, Florian u. a.: Outracing champion Gran Turismo drivers with deep reinforcement learning. In: *Nature* 602 (2022), Nr. 7896, S. 223–228

[54] Zeng, Andy ; Song, Shuran ; Welker, Stefan ; Lee, Johnny ; Rodriguez, Alberto ; Funkhouser, Thomas: Learning synergies between pushing and grasping with self-supervised deep reinforcement learning. In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* IEEE (Veranst.), 2018, S. 4238–4245

[55] Zhang, Tengteng ; Mo, Hongwei: Reinforcement learning for robot research: A comprehensive review and open issues. In: *International Journal of Advanced Robotic Systems* 18 (2021), Nr. 3, S. 17298814211007305

[56] Zheng, Qingchun ; Peng, Zhi ; Zhu, Peihao ; Zhao, Yangyang ; Zhai, Ran ; Ma, Wenpeng: An object recognition grasping approach using proximal policy optimization with yolov5. In: *IEEE Access* 11 (2023), S. 87330–87343

[57] Zhengxuez: *RG6 Gripper URDF Description*. 2025. – URL https://github.com/Zhengxuez/rg6_gripper_description. – Accessed: March 10, 2025

[58] Zhu, Shaoting ; Mou, Linzhan ; Li, Derun ; Ye, Baijun ; Huang, Runhan ; Zhao, Hang: VR-Robo: A Real-to-Sim-to-Real Framework for Visual Robot Navigation and Locomotion. In: *arXiv preprint arXiv:2502.01536* (2025)

A Appendix

A.1 Tools and aids used in this thesis

Table A.1: Tools and aids used

Tool	Purpose
LaTeX	Typesetting and layout tool used to create this document
Overleaf	Online LaTeX editor used for writing and managing this document
OpenAI ChatGPT	Assistance with translation from German to English and language refinement

Erklärung zur selbständigen Bearbeitung

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort	Datum	Unterschrift im Original