BERGISCHE UNIVERSITÄT WUPPERTAL

Modular Transfer Reinforcement Learning in Industrial Robotics

Modulares Transfer Reinforcement Learning für Industrieroboter

Der Fakultät für Elektrotechnik, Informationstechnik und Medientechnik der Bergischen Universität Wuppertal vorgelegte Dissertation zur Erlangung des akademischen Grades eines Doktors der Ingenieurwissenschaften

von

Christian Bitter (geb. Scheiderer)

Erstgutachter:	Prof. Dr.-Ing. Tobias Meisen
Zweitgutachter:	Prof. Dr.-Ing. Eike Permin

Tag der mündlichen Prüfung: 31.10.2025

Abstract

This dissertation makes significant contributions to the field of modular transfer reinforcement learning (TRL) in the context of industrial robotics. It systematically addresses the key issue of sample efficiency and transferability in the application of deep reinforcement learning (DRL) to industrial robotics. Given that monolithic end-to-end DRL approaches have high data requirements and lack interpretability and transferability, the dissertation proposes and validates a modular decision pipeline. This pipeline decomposes robot control into four stages that address different aspects of the decision-making process: perception, planning, execution and control. The contributions of this dissertation are organized according to these four stages.

The thesis first addresses the issue of applying the discretized DRL paradigm to industrial robot applications by devising an asynchronous DRL framework and continuous, smooth action spaces based on Bézier curves. This allows for real-time, jerk-free robot control, reducing mechanical stress and execution time. Further, the possibility to predict future vision-based states with generative models is explored. For the perception module, this thesis introduces a semi-supervised learning strategy which applies unsupervised autoencoder models to compress high-dimensional image states in combination with domain knowledge in the form of segmentation maps to extract latent representations usable by a DRL agent. The module is fully trained in simulation with domain randomization, requiring only a single annotated real-world image to achieve zero-shot transfer to the real-world.

Substantial contributions are further made regarding the exploitation of structure in robotic motion for TRL. Here, the thesis shows that hierarchical reinforcement learning (HRL) enables learning modular, task-agnostic behavioral policies at lower levels, which are shown to be transferable to new tasks. Additionally, the concept of assembly-by-disassembly is examined in the context of TRL. Based on the insight that learning a corresponding disassembly task is easier than the actual assembly task as the former is more constrained, the thesis proposes a strategy to first solve the disassembly task and afterward use the reversed trajectories to pretrain an assembly agent. The results demonstrate the strategy to successfully solve an assembly task on which a direct DRL approach fails. Finally, the thesis introduces a cross-robot imitation learning approach, where trajectories are mapped between robots of different morphologies using explicit forward/inverse kinematics and embodiment metrics. It is shown that the mapped demonstrations can be used for effective pretraining.

These contributions are robustly evaluated across three representative use cases: the academic wire-loop game, a vision-driven object picking scenario, and an industrial aircraft clip assembly task. Collectively, the results establish that modular TRL enables more sample-efficient, performant, and more maintainable automation in industrial robotics. The dissertation concludes with a critical appraisal, drawing lessons from the results towards future research directions for modular and transferable AI in industrial robotic applications.

Kurzfassung

Diese Dissertation trägt wesentlich zum Bereich des modularen Transfer Reinforcement Learning (TRL) im Kontext der Industrierobotik bei. Sie befasst sich systematisch mit der Problematik der Dateneffizienz und Transferierbarkeit von Deep Reinforcement Learning (DRL) in der Industrierobotik. Da monolithische end-to-end DRL-Ansätze hohe Datenanforderungen haben und es an Interpretierbarkeit und Übertragbarkeit mangelt, wird in der Dissertation eine modulare Entscheidungspipeline vorgeschlagen und validiert. Diese Pipeline unterteilt die Robotersteuerung in vier Phasen, die verschiedene Aspekte des Entscheidungsprozesses abdecken: Wahrnehmung, Planung, Ausführung und Steuerung. Die Beiträge dieser Dissertation sind nach diesen vier Phasen gegliedert.

Die Arbeit befasst sich zunächst mit der Anpassung von diskretem DRL für Robotikanwendungen, indem ein asynchroner DRL-Ansatz und kontinuierliche, glatte Aktionsräume mittels Bézier-Kurven entwickelt werden. Dies ermöglicht eine ruckfreie Robotersteuerung und reduziert die mechanische Belastung sowie die Ausführungszeit. Darüber hinaus wird die Vorhersage zukünftiger visueller Zustände mit generativen Modellen erforscht. Für das Wahrnehmungsmodul wird in dieser Arbeit eine semi-überwachte Lernstrategie eingeführt, die Autoencodermodelle verwendet, um hochdimensionale Bildzustände zu komprimieren, um kombiniert mit Domänenwissen in Form von Segmentierungskarten latente Repräsentationen zu extrahieren, welche von einem DRL-Agenten genutzt werden können. Das Modul wird vollständig in der Simulation mit Domain Randomization trainiert und benötigt nur ein einziges annotiertes reales Bild, um einen Zero-Shot-Transfer in die reale Welt zu erreichen.

Darüber hinaus werden wesentliche Beiträge zur Strukturausnutzung in Roboterbewegungen für TRL erbracht. Es wird gezeigt, dass hierarchisches Reinforcement Learning (HRL) modulare, aufgabenunabhängige Verhaltensstrategien auf unteren Ebenen erlernt, die sich auf neue Aufgaben übertragen lassen. Zusätzlich wird das Konzept der „Montage durch Demontage“ im Kontext von TRL beleuchtet. Basierend auf der Erkenntnis, dass das Erlernen einer Demontageaufgabe aufgrund einer stärkeren Eingrenzung meist einfacher ist als die eigentliche Montageaufgabe, wird in dieser Arbeit eine Strategie vorgeschlagen, bei der zunächst die Demontageaufgabe gelöst und anschließend die invertierten Trajektorien zum Vortraining eines Montageagenten verwendet werden. Die Ergebnisse zeigen, dass mit dieser Strategie eine Montageaufgabe erfolgreich gelöst wird, bei der ein direkter DRL-Ansatz versagt. Schließlich wird in dieser Arbeit ein roboterübergreifender Imitationsansatz vorgestellt, bei dem Trajektorien zwischen Robotern unterschiedlicher Morphologie unter Verwendung expliziter Vorwärts- und Rückwärtskinematiken und Embodiment-Metriken übertragen werden. Es wird gezeigt, dass die übertragenen Demonstrationen für ein effektives Vortraining genutzt werden können.

Diese Beiträge werden anhand von drei repräsentativen Fallbeispielen evaluiert: dem akademischen Wire-Loop-Spiel, einem visuell gesteuerten Objekt-Picking-Szenario und einer industriellen Montageaufgabe für Flugzeug-Clips. Insgesamt zeigen die Ergebnisse, dass modulares TRL eine effizientere, leistungsfähigere und besser wartbare Automatisierung in der Industrierobotik ermöglicht. Die Dissertation schließt mit einer kritischen Würdigung, die aus den Ergebnissen Schlüsse für zukünftige Forschungsansätze hinsichtlich modularer und transferierbarer KI in industriellen Robotikanwendungen zieht.

Acknowledgments

Although my name is on the cover of this dissertation, this work would not have been possible without the help and support of many people.

First and foremost, I am deeply grateful to my advisor Prof. Tobias Meisen for the continuous mentorship and support. I always appreciated the ability to follow my interests and carve my own path. Although this path included some detours and took longer than expected, your guidance and availability on and off-topic were invaluable to get to where I am today - both professionally and personally.

A special thanks go to my former colleagues and now friends: Jannik Peters for many things - from fun times on the road to valuable feedback on this thesis. Hasan Tercan for guidance as team leader, Constantin Waubert de Puiseau for letting me crash on your couch, Robert Maack for keeping me safe in foreign countries, and Richard Meyes for kicking this whole journey off. I am lucky to have worked with such a collaborative and supportive team, and cherish lots of good memories in and beyond the lab.

I would also like to thank the student assistants, collaborators, and project partners who have worked with me over the years. In particular Timo Thun, Tim Flegelskamp, Nik Dorndorf, Malte Mosbach, and Lucas Kiefer.

Without taking away from the importance of the above, I am extremely grateful to my family for having my back throughout this journey. My parents Renate and Thomas Scheiderer especially for childcare support, always available to help out in difficult times. Timon Vogler for hosting me when I needed a change of scenery - sorry for eating all your food and smearing your whiteboard. Oskar Bitter for just being you and showing me what really matters. Findus for being a loyal companion and for barking at students who are late to lectures.

Last but certainly not least, the biggest thank-you goes to my wife, Kristina. I know that my freedom to work on this thesis was not for free. Weekends and evenings spent writing meant time away from the family. Night shifts, stress, and resulting sickness were always felt and compensated by you the most. I am extremely grateful for your encouragement, patience, and the occasional necessary challenge to my resource management.

Publications

This dissertation is based in part on the following peer-reviewed publications, which were written in scope of research activities at the Institute of Information Management in Mechanical Engineering (IMA) of the RWTH Aachen University and the Institute of Technologies and Management of Digital Transformation (TMDT) of the University of Wuppertal. Note that publications by Christian Scheiderer and Christian Bitter are authored by the same individual, the author of this thesis.

Thesis-related publications

Conference publications

Meyes, R., Scheiderer, C., and Meisen, T. Continuous motion planning for industrial robots based on direct sensory input. *Procedia CIRP 72* (2018), 291–296

Scheiderer, C., Thun, T., and Meisen, T. Bézier Curve Based Continuous and Smooth Motion Planning for Self-Learning Industrial Robots. *Procedia Manufacturing 38* (Jan. 2019), 423–430

Scheiderer, C., Thun, T., Idzik, C., Posada-Moreno, A. F., Krämer, A., Lohmar, J., Hirt, G., and Meisen, T. Simulation-as-a-service for reinforcement learning applications by example of heavy plate rolling processes. *Procedia Manufacturing 51* (2020), 897–903

Scheiderer, C., Mosbach, M., Posada-Moreno, A. F., and Meisen, T. Transfer of Hierarchical Reinforcement Learning Structures for Robotic Manipulation Tasks. In *2020 International Conference on Computational Science and Computational Intelligence (CSCI)* (2020), IEEE, pp. 504–509

Scheiderer, C., Dorndorf, N., and Meisen, T. Effects of Domain Randomization on Simulation-to-Reality Transfer of Reinforcement Learning Policies for Industrial Robots. In *Advances in Artificial Intelligence and Applied Cognitive Computing*, H. R. Arabnia, K. Ferens, D. De La Fuente, E. B. Kozerenko, J. A. Olivas Varela, and F. G. Tinetti, Eds. Springer International Publishing, Cham, 2021, pp. 157–169

Maschler, B., Vietz, H., Tercan, H., Bitter, C., Meisen, T., and Weyrich, M. Insights and example use cases on industrial transfer learning. *Procedia CIRP 107* (2022), 511–516

Bitter, C., Peters, J., Tercan, H., and Meisen, T. Industrial Cross-Robot Transfer Learning. *Procedia CIRP 120* (Jan. 2023), 1297–1302

Magazine publications

Scheiderer, C., and Meisen, T. Auf eigenen Füßen: Machine Learning in der Industrie. *iX Special 2018 - Industrial Internet of Things* (2018), 48–51

Meyes, R., Scheiderer, C., Thiele, T., and Meisen, T. Selbstlernende adaptive Robotersteuerung: Kontinuierliche Bewegungsplanung für Industrieroboter auf Basis von Sensordaten. *Fabriksoftware* (2018), 42–44

Vietz, H., Maschler, B., Tercan, H., Bitter, C., Meisen, T., and Weyrich, M. Industrielles Transfer-Lernen: Von der Wissenschaft in die Praxis. *atp magazin 64*, 8 (2022), 86–93

Preprint publications

Bitter, C., Thun, T., and Meisen, T. Karolos: An Open-Source Reinforcement Learning Framework for Robot-Task Environments, Dec. 2022

Other publications

Conference publications

Bellgardt, M., Scheiderer, C., and Kuhlen, T. W. An Immersive Node-Link Visualization of Artificial Neural Networks for Machine Learning Experts. In *2020 IEEE International Conference on Artificial Intelligence and Virtual Reality (AIVR)* (Dec. 2020), pp. 33–36

Tercan, H., Bitter, C., Bodnar, T., Meisen, P., and Meisen, T. Evaluating a Session-based Recommender System using Prod2vec in a Commercial Application. In *Proceedings of the 23rd International Conference on Enterprise Information Systems* (2021), SCITEPRESS - Science and Technology Publications

Bitter, C., Tercan, H., Meisen, T., Bodnar, T., and Meisen, P. When to Message: Investigating User Response Prediction with Machine Learning for Advertisement Emails. In *2021 4th International Conference on Artificial Intelligence for Industries* (Piscataway, NJ, 2021), K. Li and J. Shih, Eds., IEEE, pp. 25–29

Peters, J., Waubert de Puiseau, C., Tercan, H., Gopikrishnan, A., Lucas de Carvalho, G. A., Bitter, C., and Meisen, T. Emergent language: A survey and taxonomy. *Autonomous Agents and Multi-Agent Systems 39*, 1 (Mar. 2025), 18

Contents

Abbreviations **xi**

List of Figures **xiii**

List of Tables **xxi**

1 Introduction **1**

1.1 Motivation 1

1.2 Research Questions 4

1.3 Use Cases 7

1.3.1 Use Case 1: Wire-Loop Game 7

1.3.2 Use Case 2: Object Picking 9

1.3.3 Use Case 3: Clip Assembly in Aircraft Manufacturing 11

1.4 Structure 14

2 Foundations **17**

2.1 Industrial Robotics 17

2.1.1 Kinematics Description 18

2.1.2 Parametric Path Description 19

2.1.3 Closed-Loop Robot Control 20

2.2 Deep Learning 21

2.2.1 Machine Learning Fundamentals 21

2.2.2 Artificial Neural Networks 22

2.2.3 Computer Vision 23

2.3 Deep Reinforcement Learning 25

2.3.1 Reinforcement Learning Fundamentals 25

2.3.2 Deep Reinforcement Learning Algorithms for Continuous State-Action Spaces 28

2.3.3 Hierarchical Reinforcement Learning 30
2.3.4 Environment Parallelization 31
2.4 Transfer Reinforcement Learning 32
2.4.1 Simulation-to-Reality Transfer 32
2.4.2 Policy Modularization 34
2.4.3 Imitation Learning 35

3 Related Work **37**
3.1 Real-Time Asynchronous Reinforcement Learning 38
3.2 Sim2Real Transfer with Domain Randomization 41
3.3 Robotic Movement Structure Exploitation 45
3.3.1 Cross-Task Transfer in Hierarchical Reinforcement Learning . 46
3.3.2 Assembly-by-Disassembly 48
3.4 Cross-Robot Transfer Reinforcement Learning 51
3.5 Research Gaps and Focus 55

4 Framework and Baselines **57**
4.1 Learning Framework 57
4.2 Use Case 1: Wire-Loop Game 61
4.2.1 Environment Design 61
4.2.1.1 Real-World Environment 61
4.2.1.2 Simulation Environment 64
4.2.2 Agent Design 67
4.2.3 Experimental Evaluation 68
4.2.3.1 Real-World Experiments 68
4.2.3.2 Sim2Real Experiments 70
4.3 Use Case 2: Object Picking 73
4.3.1 Environment Design 73
4.3.1.1 Fetch Benchmark Environments 73
4.3.1.2 Use-Case Specific Environment Interface 74
4.3.2 Experimental Evaluation 76
4.3.2.1 Analysis of Task Difficulty 76
4.3.2.2 Fetch Benchmark Experiments 77
4.4 Use Case 3: Clip Assembly 79
4.4.1 Environment Design 79
4.4.2 Experimental Evaluation 81
4.5 Mapping Research Questions to Use Cases 82

5 Smooth Continuous Robot Control 83

5.1 Smoothness-Constrained Action Space Design 84

5.2 Asynchronous Learning Framework 87

5.3 Dynamics Rollout Module Development and Validation 89

5.3.1 Dynamics Rollout Module Architecture 89

5.3.2 Dataset Collection and Training Results 91

5.4 Exploration of Design Choices 93

5.5 Asynchronous Wire-Loop Experiments 94

5.6 Summary 95

6 Sim2Real Transfer of Perception Modules 97

6.1 Pose Estimation Modules 98

6.1.1 Simulation Dataset 98

6.1.2 Experimental Evaluation 100

6.2 Image Compression Modules 103

6.2.1 Unsupervised Image Compression 103

6.2.2 Explorative Experiments on Agent Attention 107

6.2.3 Semi-Supervised Image Compression 109

6.2.4 Semi-Supervised Compression with Domain Knowledge 111

6.3 Summary 114

7 Hierarchical and Backward Planning 115

7.1 Hierarchical Policy Transfer 116

7.1.1 Hierarchical Actor-Critic Learning Framework 117

7.1.2 Fetch Environment Variations 118

7.1.3 Experimental Evaluation 120

7.1.3.1 Hierarchical Actor-Critic Experiments 120

7.1.3.2 Low-Level Policy Transfer Experiments 123

7.2 Assembly-by-Disassembly 128

7.2.1 Clip Disassembly Environment 128

7.2.2 Experimental Evaluation 129

7.3 Summary 130

8 Cross-Robot Execution Imitation 131

8.1 Robot-Task Environments 132

8.2 Cross-Robot Imitation Learning Framework 133

8.2.1 Cross-Robot Behavior Similarity 133

8.2.2 Cross-Robot Trajectory Mapping 134

8.2.3 Cross-Robot Imitation Learning 137

8.3 Experimental Evaluation 138

8.4 Summary 140

9 Critical Reflection and Outlook 143

9.1 Research Question 1 143

9.2 Research Question 2 144

9.3 Research Question 3 146

9.4 Research Question 4 148

9.5 Closing Remarks 149

Bibliography 151

Abbreviations

AI	Artificial Intelligence
ANN	Artificial Neural Network
BC	Behavioral Cloning
CAD	Computer-Aided Design
CNN	Convolutional Neural Network
CRIL	Cross-Robot Imitation Learning
CV	Computer Vision
DDPG	Deep Deterministic Policy Gradient
DH	Denavit-Hartenberg
DL	Deep Learning
DOF	Degrees of Freedom
DQN	Deep Q-Network
DR	Domain Randomization
DRL	Deep Reinforcement Learning
EC	Embodiment Correspondence
GAN	Generative Adversarial Network
GPU	Graphics Processing Unit
Grad-CAM	Gradient-weighted Class Activation Mapping
HAC	Hierarchical Actor-Critic
HER	Hindsight Experience Replay
HRL	Hierarchical Reinforcement Learning
IFR	International Federation of Robotics
IK	Inverse Kinematics
IL	Imitation Learning
IMA	Institute of Information Management in Mechanical Engineering
KL divergence	Kullback–Leibler divergence
MARWIL	Monotonic Advantage Re-Weighted Imitation Learning
MDP	Markov Decision Process

ML	Machine Learning
MLP	Multilayer Perceptron
MPC	Model Predictive Control
PCA	Principal Component Analysis
PER	Prioritized Experience Replay
POMDP	Partially Observable Markov Decision Process
PPO	Proximal Policy Optimization
ReLU	Rectified Linear Unit
RL	Reinforcement Learning
RNN	Recurrent Neural Network
RQ	Research Question
SAC	Soft Actor-Critic
sim2real	Simulation-to-Reality
SOTA	State-of-the-Art
TCP	Tool Center Point
TD	Temporal Difference
TE	Transition Efficiency
TL	Transfer Learning
TMDT	Institute of Technologies and Management of Digital Transformation
TRL	Transfer Reinforcement Learning
V-HACD	Volumetric Hierarchical Approximate Convex Decomposition
VR	Virtual Reality

List of Figures

1.1 Annual installations of industrial robots worldwide according to the International Federation of Robotics (IFR). Data aggregated from [120] and [121]. 2

1.2 Concept of a modular decision-making pipeline for industrial robotics consisting of perception, planning, execution and control stages. . . . 3

1.3 The four Research Questions (RQs) focus on the different stages of the modular decision-making process principle. The first RQ aims to enable suitable control for industrial robotic systems. The remaining three RQs focus on Transfer Reinforcement Learning (TRL) from simulations, other tasks, and other robots, respectively. 4

1.4 The wire-loop demonstrator requires the robot to guide a loop along a wire from one end to the other without making contact with the wire itself. 8

1.5 The wire loop game is played on a 2D game plane. The game is considered won when the loop reaches the goal zone. To reset the game, the loop is moved from the 2D game plane to the reset plane for collision-free movement back to the start pose or an intermediate checkpoint pose. 9

1.6 The object picking demonstrator consists of a Franka Emika Panda robot arm and a fixed camera for visual feedback. On the table, toy building blocks are placed for the robot to pick up. 10

1.7 Overview of the shell structure of an airplane. The shell consists of a metal skin, which is reinforced by vertical and horizontal struts. The struts are connected by clips. 12

1.8 The clip assembly demonstrator consists of an UR5 industrial robot, clips and 3D-printed installation sites. The goal of the assembly task is to insert the clips into the shell structure. 13

1.9 Schematic illustration of the structure of the thesis. Dots at the intersections indicate relevance of the use cases for the chapters and respective RQs. The experimental setup for RQ-4 reuses key elements from the use cases in a simplified scenario. 15

2.1 Bézier curves are parameterized by a set of control points. Multiple Bézier curves can be stitched together to form a continuous path by ensuring that the control points of the adjacent curves are collinear. . 20

2.2 In a closed-loop control system, a controller aims to minimize the error between a desired goal state and the system state measured by sensory equipment. 21

2.3 An illustrative example of an Artificial Neural Network (ANN) architecture. The three input features are processed by three consecutive hidden layers of 5 neurons each. The output layer consists of a single neuron, which is interpreted as the probability whether the input features describe a dog. 23

2.4 An illustrative example of a Convolutional Neural Network (CNN) architecture. The input image is processed by three consecutive convolutional layers of 2x2 kernels each. The output layer consists of a single neuron, which is interpreted as the probability whether the input image shows a dog. 24

2.5 Basic structure of a Reinforcement Learning (RL) setting. The agent interacts with the environment by selecting actions based on the current state and the policy. The environment transitions to a new state and issues a reward signal, which is used to optimize the policy. . . . 26

2.6 The Deep Deterministic Policy Gradient (DDPG) architecture consists of a deterministic actor and a state-action value critic. The actor chooses an action, which is judged by the critic. In training, the critic provides the actor with a gradient towards more favorable actions. . . 28

2.7 The Proximal Policy Optimization (PPO) architecture consists of a stochastic actor and a state-value critic. In training, the critic is used to compute the advantage of a chosen action, which serves as a gradient signal for the actor. 29

2.8 Hierarchical Reinforcement Learning (HRL) agents comprise multiple policy levels with different temporal abstraction. The higher levels are responsible for long-term planning and divide a task into subgoals, which are then achieved by the lower levels. The depicted Hierarchical Actor-Critic (HAC) architecture stacks DDPG agents. 30

2.9 Illustrative comparison of different parallelization strategies. Reset and step operations are represented as outlined and filled boxes, respectively. Interactions which belong to the same episode and thus are executed sequentially in the same environment are of the same color. 31

2.10 Overview of TRL approaches. Given a source policy, this policy can be deployed to the target domain. Depending on the transfer scenario, modification of the source policy may be required to adapt interfaces and/or facilitate learning. Given source demonstrations, a target policy can either be pretrained, or a pseudo-reward signal can be derived to guide the learning process. 33

2.11 The Domain Randomization (DR) approach deliberately randomizes a simulation to generate a distribution of training samples. With enough randomization, the agent will generalize to the real-world data distribution and overcome the Simulation-to-Reality (sim2real) gap. Adapted from [109]. 34

3.1 The literature survey process on the topic of real-time asynchronous RL for RQ-1. 17 relevant articles were identified which address latencies in real-world control settings. 38

3.2 Distribution of strategies for real-time asynchronous RL over the years. No clear trend regarding the emergence of a superior strategy can be inferred. 41

3.3 The literature survey process on the topic of DR for RQ-2. 17 relevant articles were identified which address advances to the basic DR approach. 42

3.4 Distribution of strategies on the topic of sim2real transfer with DR for RQ-2. Offline guided DR has drawn more attention than online active DR. Most recently, human-in-the-loop DR was proposed. . . . 45

3.5 The literature survey process on the topic of cross-task transfer in HRL for RQ-3. 7 relevant articles were identified which investigate transfer of HRL sub-policies. 46

3.6 Distribution of strategies for cross-task transfer in HRL over the years. While the literature is sparse, skill embeddings have drawn more attention than explicitly defined subgoal spaces. 48

3.7 The literature survey process on the topic of assembly-by-disassembly for RQ-3. 6 relevant articles were identified which use the solution of a corresponding disassembly task to solve an assembly task. 49

3.8 Distribution of the strategies for assembly-by-disassembly over the years. In particular the combination with Deep Reinforcement Learning (DRL) has not gained much attention. 51

3.9 Conceptual overview of cross-domain transfer learning. In order to transfer expert knowledge in the form of policies or demonstrations between two Markov Decision Process (MDP) environments, the domain gap must be bridged. 51

3.10 The literature survey process on the topic of cross-robot transfer reinforcement learning for RQ-4. 24 relevant articles were identified which propose cross-domain mapping strategies. 52

3.11 Distribution of strategies for cross-robot transfer reinforcement learning over the years. Using explicit mappings is the most data-efficient approach, but has been largely abandoned in favor of statistical methods and proxy tasks. 55

4.1 Software architecture of the DRL framework designed for modularity, environment parallelization, and distributed infrastructure. 59

4.2 Orchestrator benchmarking. The implementation effectively distributes experience collection across multiple processes. Adapted from [21]. . . 60

4.3 The states in the wire-loop use case are downsampled camera images of the loop and the wire. The downsampling factor was chosen to reduce the state space dimensionality while still retaining relevant image features. 62

4.4 The simulation setup for the wire-loop use case. The simulation is modeled after the real-world setup, featuring a UR5 robot, a loop tool, a camera, and a background plane for visual projection. 65

4.5 Comparison of real-world and simulation state images. To achieve more realistic state images, the same down-sampling and interpolation method is applied to the simulation images instead of directly capturing images at the target resolution. 65

4.6 Examples of the randomization strategies applied to the wire-loop use case. 66

4.7 Evaluation scenarios for the wire-loop use case. Taken from [114] . . . 69

4.8 DRL results for the real-world experiments in the wire-loop use case. The results show that the learning framework implementation is capable of training an agent to solve the wire-loop game. The agent progressively improves its performance, requiring less training with each new scenario. The agent is also able to successfully learn background noise in the fourth scenario. Adapted from [114]. 69

4.9 Experimental scenarios for the DR study to test sim2real transferability. 71

4.10 The benchmark environments FetchReach, FetchPush, and FetchPickAndPlace from the Fetch suite [134]. 74

4.11 Overview of the environments used for the object picking use case. With minor adaptations, a policy trained on the Fetch PickAndPlace environment can be used in the object picking use case. 75

4.12 Optimal control solutions for the Reach and PickAndPlace tasks. . . 76

4.13 Comparison of the training framework performance to Plappert et al. [134] for the Reach, Push, and PickAndPlace tasks. The training framework achieves better or comparable performance on the benchmark tasks. 78

4.14 In the clip assembly use case, a clip (blue) is inserted into a shell (green). 79

4.15 Example of the clip assembly environment. The shell is reduced to the vicinity of the installation location to improve simulation performance. 80

4.16 Mesh preprocessing required to improve collision detection in simulation for the clip assembly use case. 80

4.17 Illustrative example of suboptimal behavior induced by the dense reward function. 81

5.1 Inference of a Bézier curve from a target pose. 85

5.2 Bézier curve encoding strategies. 86

5.3 Sequence diagram of a standard RL framework. Due to the time-discrete nature, real-world latencies lead to non-continuous robot motion. 87

5.4 Sequence diagram of the proposed asynchronous RL framework. The next action is determined based on an intermediate state and provided to the robot in time to enable a continuous transition between actions. 88

5.5 The dynamics rollout module is based on the pix2pix architecture [77]. An encoder-decoder generator produces the next state image based on the current state and action. A patch discriminator attempts to distinguish between real and generated images to improve the quality of the generated images. 91

5.6 Examples of dynamics rollout model performance. Based on the observed state and action (not shown), the model predicts the resulting next state. Compared to the ground truth next states, the model is able to predict the wire shape with an impressively high quality for small action magnitudes. 92

5.7 Failure cases of the dynamics rollout model. The model is unable to predict the next state for large action magnitudes and potentially starts to hallucinate. 92

5.8 Training results of the Bézier agents. The Cartesian-3 agent was unable to converge to a solution, while the other agents outperform the baseline agent. Adapted from [158]. 94

6.1 Examples of the simulation dataset for the perception module of the object picking use case. 99

6.2 Bounding box predictions on the simulation test set. The perception module is able to accurately locate the object in the image. 100

6.3 Integration of the perception module into the real-world. To improve the performance, the model is fed with jittered image inputs. 101

6.4 Relative variation of perception module predictions for a video stream. The noise is compensated by a moving average filter over 15 frames. . 102

6.5 Concept of using autoencoders for process-relevant feature extraction. First, the autoencoder is trained using states as input and output to learn a compressed latent representation. Then, the decoder is discarded and replaced with an execution module which takes the learned latent representation as input. The combination of the encoder and execution module forms a DRL policy. 103

6.6 The reset routine of the real-world hardware setup leads to many similar states in the dataset. In many cases, the first state of a trajectory is virtually identical to the last state of the previous trajectory. . . . 104

6.7 Explorative dataset analysis to identify the optimal train/test split. . 105

6.8 Quantitative evaluation of the autoencoder. The input and reconstructed state images are shown for five randomly selected states. The reconstructed images are more blurry, however the wire and loop are still recognizable. Importantly, the shape of the wire is preserved, indicating that this process-relevant information is successfully encoded in the latent space. 105

6.9 DRL results with unsupervised autoencoder as perception module. The agent is able to learn the wire-loop game in just 450 episodes, significantly reducing the training effort compared to the end-to-end baseline agent. 106

6.10 Exemplary results of the segmentation autoencoder on simulated validation data. The model is able to reconstruct the segmentation masks for the validation data with high accuracy. 110

6.11 Exemplary results of the segmentation autoencoder on real-world test data. The segmentation is of poor quality, as the model is unable to capture the wire shape. 111

6.12 Concept of the enhanced approach. The real-world image is segmented into objects. For each object class, a color palette is generated from its respective segments. With this color palette, DR is constrained to generate simulation images with similar colors. 112

6.13 Exemplary results of the segmentation autoencoder on simulated validation data. The model is able to reconstruct the segmentation masks for the validation data with high accuracy. 113

6.14 Exemplary results of the segmentation autoencoder on real-world test data. The rough shape of the wire is captured using just a single real-world image. 113

7.1 Variations of the Fetch environments to investigate the transferability of the lower-level policies. Taken from [156]. 119

7.2 Performance of the HAC agents on the Reach task. The HAC agents perform significantly worse, as the added overhead of the hierarchical structure is not justified by the simplicity of the task. 120

7.3 Learning curve of the HAC agents on the Push task. The HAC agents are able to solve the task significantly faster than the standard DDPG baseline. 121

7.4 Exemplary solution trajectory of a HAC agent for the Push task. The agent successfully decomposes the task by using a subgoal halfway to the final goal. 122

7.5 An exemplary state of the Push task and corresponding heatmap of the subgoal Q-values. Yellow regions indicate high, violet regions low Q-values. 122

7.6 Learning curve of the HAC agents on the PickAndPlace task. The HAC agents are able to successfully learn the task, unlike the standard DDPG baseline. 123

7.7 Exemplary solution trajectory of a HAC agent for the PickAndPlace task. 123

7.8 Transfer results for the Push-Gate Task. No conclusive advantage of transferring pretrained lower-level policies can be observed. Adapted from [156]. 124

7.9 Transfer results for the Push-Gap Task. The transfer strategy appears to have a negative effect on the learning efficiency. Adapted from [156].124

7.10 Strategy of a HAC agent on the Push-Gap Task. The agent learned a risky strategy, skipping the gap in the tabletop. Taken from [156]. . 125

7.11 Q-value heatmap in the Push-Gap task. The highest Q-value is found over the gap, thereby incentivizing the lower level policy to attempt skipping the cube over the gap. Adapted from [156]. 125

7.12 Transfer results for the PickAndPlace-Wall task. The transferred agents clearly outperform the agents trained from scratch. Adapted from [156]. 126

7.13 Transfer results for the PickAndPlace-Table task. The transferred agents clearly outperform the agents trained from scratch. Adapted from [156]. 127

7.14 Visualization of a trajectory of a successfully transferred agent in the PickAndPlace-Wall task. 127

7.15 Reward hacking in the clip assembly environment. The agent is able to exploit the force constraint to move the clip out of the shell in a non-reversible manner. 128

7.16 Performance of the DDPG agents on the clip disassembly task. The agents are able to solve the task in 25,000 training episodes. 129

8.1 The modular robot-task environments allow for quick generation of different robot-task combinations to evaluate cross-domain transferability. Adapted from [21]. 132

8.2 The Embodiment Correspondence (EC) metric is the weighted sum of the distances between all key points of two robot arms. As key points, intermediate poses along the kinematic chains of the two robots are used to capture the general shape of the arms. Taken from [19]. . . . 134

8.3 The mapping strategy of the Cross-Robot Imitation Learning (CRIL) framework. For each state of the source trajectory, the Tool Center Point (TCP) pose is determined. Next, all possible joint configuration candidates are computed with Inverse Kinematics (IK). From all candidates, a graph is constructed, whose edge weights reflect the EC metric and the transition cost. The target trajectory with most similar behavior to the source trajectory is found by finding the shortest path through the graph. Taken from [19]. 135

8.4 Detailed view of the graph weight computation. The EC metric is computed from key points determined by Forward Kinematics (FK). The transition cost is computed from an approximation of the required action between two consecutive candidates. Taken from [19]. 136

8.5 Source agent learning curve for the Panda robot with stable convergence. 138

8.6 Example trajectory mapping demonstrating the mapping capabilities of the proposed CRIL framework. The general shape of the arm, i.e. the behavior, is preserved. Taken from [19]. 139

8.7 Pretraining performance of the Monotonic Advantage Re-Weighted Imitation Learning (MARWIL) agent on the target UR5 robot using the mapped target trajectory dataset. 139

8.8 Finetuning performance of the pretrained agent on the target UR5 robot. The transfer strategy clearly outperforms agents trained from scratch. Taken from [19]. 140

List of Tables

1.1 Comparison of use case characteristics with respect to relevance to the RQs. 7

3.1 Categorization of articles by strategy for real-time asynchronous RL. 39

3.2 Categorization of articles by strategy for DR. 42

3.3 Categorization of articles by interface between HRL policy layers. . . 46

3.4 Categorization of articles according to the application assembly-by-disassembly outside or within the context of DRL. 49

3.5 Categorization of articles by strategy for cross-domain TRL. 53

4.1 Training effort for different DR parameter settings. A positive correlation between the amount of randomization and the required training effort is observed. Further, a positive effect of randomization on transferability is visible. Notably, not all parameters must be randomized to achieve zero-shot transfer. 71

6.1 Gradient-weighted Class Activation Mapping (Grad-CAM) attention maps for different agents and states. State 1 is taken from simulation without DR, states 2 and 3 are taken from simulation with DR, and states 4-6 are real-world images. Agent 1 was trained without, agent 3 with partial, and agent 6 with full DR. Red regions indicate high and blue regions indicate low attention. Adapted from [154]. 108

7.1 Performance metrics for assembly task using Behavioral Cloning (BC) with varying number of demonstration trajectories. 130

1

Introduction

"In a properly automated and educated world, then, machines may prove to be the true humanizing influence. It may be that machines will do the work that makes life possible and that human beings will do all the other things that make life pleasant and worthwhile."
- Isaac Asimov

1.1 Motivation

The demand for robotic systems in the industrial environment is experiencing a significant upward trend in the present time. The International Federation of Robotics (IFR) has recorded an average annual increase in new robot installations of +11% in the years between 2014 and 2023 and expects a continued positive development in the coming years, as depicted in Figure 1.1 [120, 121]. A major reason for this is the continuing trend towards automation, which is driven on the one hand by steadily increasing demands on production processes in terms of speeds and cost efficiency [112]. On the other hand, continuous technological innovation leads to a reduction in acquisition costs as well as to improvements in the capabilities of industrial robots, making automation solutions increasingly attractive for new application areas as well [87, 107, 112].

A technological driver of robotic systems, which according to current forecasts will continue to gain relevance in the coming years, are Artificial Intelligence (AI) capabilities in the form of Machine Learning (ML) algorithms [107]. According to a position paper from leading voices in the field of AI, among others Yann LeCun (Chief AI Scientist at Meta) and David Silver (Principal Research Scientist at Google DeepMind), one subfield of ML promising significant potential towards achieving human-level machine intelligence necessary for the automation of increasingly complex tasks with industrial robots is Deep Reinforcement Learning (DRL) [110]. DRL combines advances in Deep Learning (DL) with the principles of Reinforcement Learning (RL). On the one hand, the field of DL comprises the study

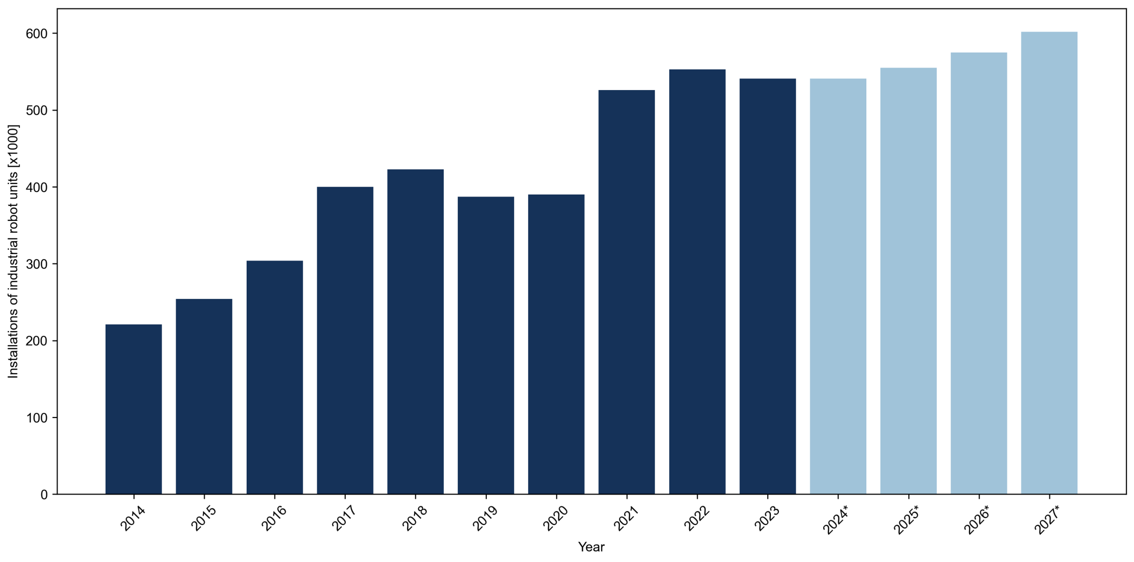


Figure 1.1 Annual installations of industrial robots worldwide according to the IFR. Data aggregated from [120] and [121].

and application of Artificial Neural Networks (ANNs) as powerful function approximators, which are able to learn complex patterns from data without the need for hand-crafted feature engineering [60, 90]. On the other hand, RL is a learning paradigm that allows an agent to autonomously learn optimal control strategies from experience gained through interactions with its environment [174]. The agent first observes the state of its environment as a basis for decision-making to select an action. The action is then executed, resulting in a change in the environment's state, which is again observed by the agent. In addition, the agent receives a reward that reflects the value of the actions for the given task. This information is used to condition the action-selection process, also referred to as the policy of an agent, towards choosing reward-maximizing actions, which equals the solution of the given control task.

The impressive capabilities of DRL have been demonstrated in a wide variety of decision-making and control applications, achieving and exceeding human-level capabilities in the board games of chess, Shogi and Go [164], the card game of poker [29], as well as the video games such as Dota [16] and StarCraft [194]. In robotics, RL has been utilized to learn complex dexterity tasks, such as solving a Rubik's cube with a human-like robot hand [3], teaching a dog-like quadruped robot to walk [132], or improving the autonomous-driving capabilities of cars [106].

DRL has the potential to revolutionize industrial automation by enabling more sophisticated decision-making in large and complex environments. However, despite its early successes, the application of DRL in industrial robotics is still limited. In their hype cycle for AI 2024, the leading research and advisory firm Gartner classifies ”Smart Robotics” as entering the ”trough of disillusionment”, expecting a period of 5–10 years before mainstream industrial adoption [78]. One major challenge for the application of DL in general and DRL in particular is the required amount of data for training a model. This phenomenon is known as the ”curse of dimensionality” [118], which describes the exponential effort and training data required with increasing dimensionality of the data space. In industrial robotics, this issue is amplified by the need for real-world interaction, which is not only time-consuming, but also bears the risk of damaging components or leading to unwanted collisions with objects in the robot's environment. To address the shortcomings of data-driven ML, the research area of Transfer Learning (TL) aims to reduce the amount of data required for training a model by reusing knowledge acquired in simulation or from related scenarios [178, 203].

With the vision of self-learning DRL agents deriving complex control policies from experience, many research efforts in the field of DRL have focussed on developing end-to-end learning pipelines [80, 81, 93, 168]. These approaches aim to delegate both the feature extraction from raw sensory data and the derivation of decision-making strategies to the agent, which promises to reduce the need for domain knowledge and manual engineering efforts. However, training DRL agents end-to-end does not come without drawbacks. Capturing the entire complexity of an industrial task within one policy means that arriving at a suitable control policy requires the maximum amount of training effort. Further, the black-box nature of an end-to-end DRL agent makes it difficult to verify and maintain agents in the long run. These drawbacks are particularly relevant for industrial applications, where both safe and efficient operation must be guaranteed. In contrast to the end-to-end approach, works such as Wang et al. [199] have highlighted the benefit of combining the self-learning capabilities of DRL agents with modular control pipeline architectures. Modularity is a core design principle of engineering that emphasizes the separation of concerns and the independence of components. This allows for faster and cheaper development of individual components, as well as better interpretability and maintainability through well-defined interfaces.

In this context, this thesis aims to contribute to the applicability of DRL in industrial robotics by leveraging the modularization of robotic decision processes into perception, planning, execution and control stages (cf. Figure 1.2), and investigating opportunities to efficiently develop individual policy modules using TL strategies. The contributions are guided by Research Questions (RQs) which are outlined in the next section. Afterward follows an introduction of three robotic use cases, in which the development and experiments of this thesis will be conducted. Finally, an overview of the thesis structure is provided.

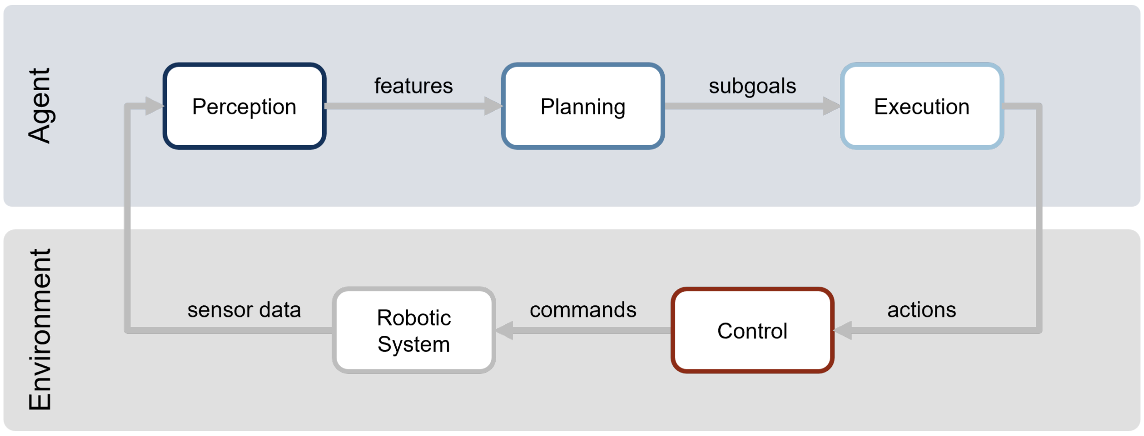


Figure 1.2 Concept of a modular decision-making pipeline for industrial robotics consisting of perception, planning, execution and control stages.

1.2 Research Questions

As depicted in Figure 1.3, the following RQs each focus on a different stage of the decision-making process chain: the control interface required to interact with the robotic system, the perception of the current environment state by extracting task-relevant information from raw sensory data, the planning of a task-specific high-level strategy, and the execution of robot-specific low-level behavior. The RQs are detailed in the following.

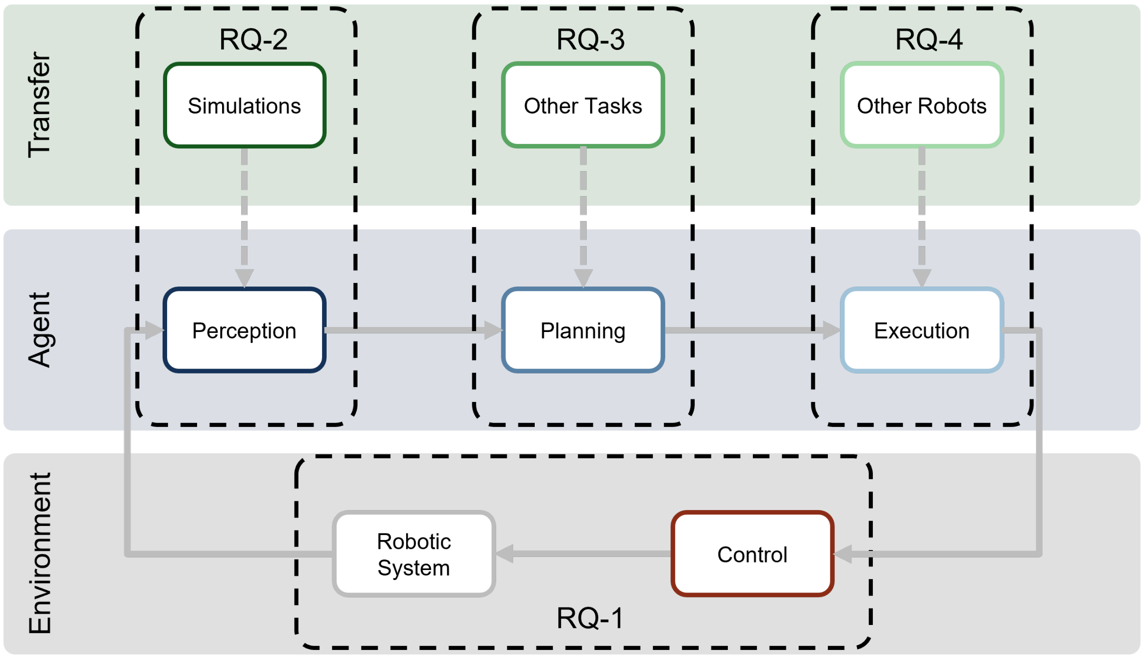


Figure 1.3 The four RQs focus on the different stages of the modular decision-making process principle. The first RQ aims to enable suitable control for industrial robotic systems. The remaining three RQs focus on Transfer Reinforcement Learning (TRL) from simulations, other tasks, and other robots, respectively.

RQ-1: How can smooth, continuous movement be achieved within the RL framework?

In general, robotic systems are controlled by setting motor voltages, which in turn result in joint torques that move the physical structure of the robot. While it is possible to use the joint torques as action space of a RL agent and training it end-to-end, many real-world applications benefit from using abstracted action spaces. Tasks where the robot is expected to follow a certain trajectory can be formulated using joint positions or end effector poses, which enables an agent to focus on the task on a higher level, rather than having to learn underlying physical relationships, which are now handled by a low-level controller. This not only reduces the required learning effort for one task, but also facilitates the transfer across domains by using domain-agnostic abstraction spaces, such as the robot-agnostic trajectory of a tool instead of the robot-specific joint torques.

However, using position-based action spaces comes at a cost, as the resulting trajectories are likely to exhibit jerky and discontinuous movements when used in the standard RL framework. This can be attributed to two reasons: On the one hand, the inherent time-discretization of RL requires each step to be completed before a new action is chosen. As a position controller is used to move the joints to a target position at which movement is stopped, the application of standard RL leads to undesired stop-and-go behavior. On the other hand, the independence of successive actions allows an agent to perform discontinuous trajectories with jerky movement. Such behavior in general is to be avoided, it puts high stress on mechanical components. Recognizing these shortcomings, the first RQ prompts for an adaptation of the standard RL framework to enable the application of RL together with position controllers.

RQ-2: How can perception modules for vision-based robotic tasks be developed in simulation, and how can their structure be designed when the relevant low-level features are not immediately obvious?

As stated above, the utilization of high-dimensional sensory data, such as camera images, is a major challenge for data-driven ML approaches due to the ”curse of dimensionality”. An intuitive solution to reduce the need for expensive real-world data is to use simulations. Simulations not only allow for faster, parallelized, and collision-free training runs, but also enable the incorporation of process variations that are difficult to be recreated in the real-world setup. In contrast to those advantages, an important issue with simulations is their inevitable deviation from reality due to coarser resolutions, time-discretization, or errors in a simulation. The first part of this RL focuses on overcoming this so-called Simulation-to-Reality (sim2real) gap and transfer pretrained perception modules from simulation to real-world applications.

While the features to be extracted from the sensory data are sometimes easily identifiable, such as poses of task-relevant objects, this is not always the case. On the contrary, it can be argued that especially scenarios where the relevant features are not immediately obvious are of particular interest for the application of DRL, calling for an ability to detect patterns which are hard to formalize by a human domain expert. In contrast, scenarios where it is possible to create a precise world model are more likely to be automated using traditional motion planning methods without the need for data-driven learning. Thus, the second part of this RQ demands an investigation into the development of perception modules to reliably extract latent feature spaces relevant for the subsequent decision-making pipeline.

RQ-3: How can decomposition of robotic movement be leveraged to modularize task-independent behavior policies, and how can reversibility of robotic movement be exploited to improve training efficiency?

In a broader context, decomposition and reversibility are general concepts applicable to a wide variety of planning tasks. On the one hand, decomposition describes the ability to break down a task into smaller subtasks, which can be solved individually. Given that the complexity of the subtasks is significantly lower than the complexity of the overall task, this strategy may significantly speed up the planning effort. Reversibility, on the other hand, describes the ability to follow a chain of decisions in both directions. This is particularly interesting for planning tasks where the optimal end state is known, since reasoning backwards from the end state may provide a more efficient path than reasoning forwards from the initial state. Both properties are frequently exploited in the robotics domain to efficiently find suitable movement trajectories.

Decomposition is at the core of modeling hierarchical task structures [69], and constructing hierarchical control schemes [104]. In context of DRL, Hierarchical Reinforcement Learning (HRL) approaches [34, 95, 126] have been proposed to improve the training efficiency of agents through the separation of an agent policy into different policy levels, each level receiving its objectives from the next higher level. The first part of this RQ calls for an investigation into how such hierarchical structures can be transferred across different tasks.

Likewise, reversibility is frequently used in robotics for backward planning trajectories in settings where finding a trajectory from goal to start is significantly easier than from start to goal. Example applications include pick-and-placing scenarios [103], and assembly-by-disassembly [182, 209]. The second part of this RQ focuses on scenarios where reversibility is only partially given, i.e. where some trajectories cannot be executed in reverse, for example due to non-rigid, deformable interactions between the robotic system and it's environment.

RQ-4: How can movement trajectories be transferred between different robot morphologies?

In the context of industrial robotic arms, it is possible for robots to possess similar capabilities in terms of their movement and general shape. However, when viewed from the perspective of a control agent, these robots may exhibit stark differences. The dissimilarity arises primarily in the representation of their state and action spaces, which can vary significantly due to differences in the underlying kinematic models, such as the number of joints or link lengths. This discrepancy poses a significant challenge in industrial applications where transitioning from an existing robotic legacy system to a more advanced technological system is not a straightforward task. This is especially amplified for DRL applications, where the data-extensive nature of training a new agent for a new robotic system poses a significant obstacle. To facilitate an independence of individual robotic models and promote a selection based on capabilities, rather than legacy, the fourth research question calls for methods which enable the transfer of existing control solutions across robots of different morphology.

1.3 Use Cases

The contributions of this thesis towards answering the above RQs are developed and evaluated using three robotic use cases, which will be referred to as **Wire-Loop**, **Object Picking**, and **Clip Assembly**. Within the shared context of controlling a medium-sized industrial robotic arm, the use cases provide distinct challenges and requirements regarding the different stages of the previously outlined decision-making process chain and are thus differently relevant to the RQs. The use cases are compared in Table 1.1 and are detailed in the following.

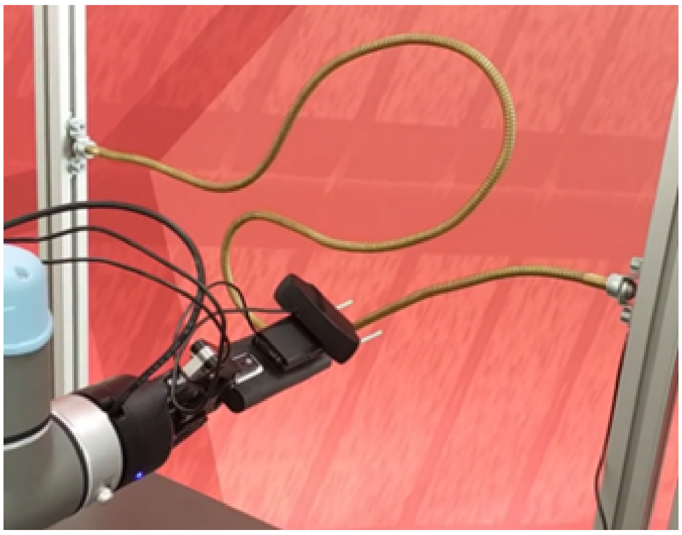
	Wire-Loop	**Object Picking**	**Clip Assembly**
Industrial Transferability	**Low**: academic; offers valuable insights, but transferability to industry unclear	**Medium**: academic scenario, but relates to industrial manipulation tasks	**High**: derived from real-world assembly process
Smooth, Continuous Control (RQ-1)	**High**: real-time adjustments to wire shape changes	**Low**: real-time disturbances not considered by experimental design	**Low**: real-time disturbances excluded in real-world assembly process
Visual Perception (RQ-2)	**High**: camera image input and important features not immediately obvious	**Medium**: camera image input; important features clear (object poses)	**Low**: no visual input required due to precise CAD simulation model
Planning Complexity (RQ-3)	**Low**: optimal trajectory only requires consideration of the local wire section	**Medium**: entire trajectory must be considered to avoid dead-ends	**High**: very limited set of correct trajectories, deviations lead to collisions
Cross-Robot Execution (RQ-4)	**Low**: limited collision-free workspace by design	**Medium**: robot-specific collisions possible	**High**: real-world assembly process uses a robot model with different morphology

Table 1.1 Comparison of use case characteristics with respect to relevance to the RQs.

1.3.1 Use Case 1: Wire-Loop Game

The first use case is based on the wire-loop game that requires the player, i.e. the robot, to guide a loop along a wire from one end to the other without making contact with the wire itself [111, 116]. While of academic nature, succeeding in the wire-loop game requires skills that are also required in industrial applications, such as an ability to process high-dimensional state spaces in the form of camera images, and an adaptability to dynamic changes in the environment, such as changes to the wire shape or the background behind the wire.

A hardware demonstrator depicted in Figure 1.4a serves as basis for investigating the real-world applicability of potential automation solutions. The demonstrator consists of an UR5 industrial robot and a metal wire of 150 cm length and 1 cm thickness, which is placed inside the working space of the robot. The wire is fixed on both ends, which constitute the start and goal of the game. In between the ends, the wire can be shaped arbitrarily to generate new scenarios and even introduce real-time process changes. As loop, the robot is equipped with a custom-made 3D-printed tool, which consists of two metal rods enclosing the wire. The tool also holds a camera which captures the wire section directly in front of the loop. The tool is depicted in Figure 1.4b. The game is constrained to a 2D-plane to rule out collisions between the wire and the robot arm, reduce the complexity for later automation solutions, as well as reduce the engineering effort required to ensure robust resetting routines.


(a) Overview of the wire-loop demonstrator showing the robot holding the custom-made tool and the entire wire setup. Taken from [158].

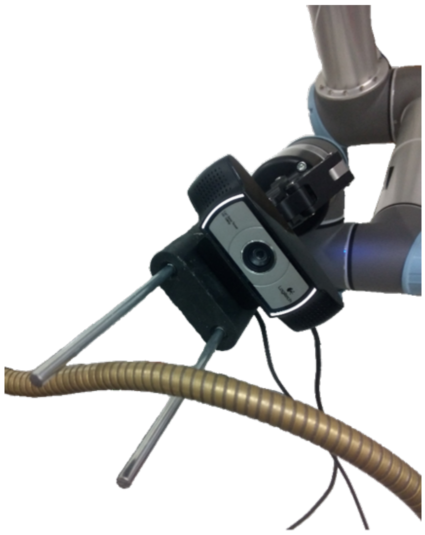

(b) The tool consists of two metal rods enclosing the wire and a camera for visual feedback. Taken from [114].

Figure 1.4 The wire-loop demonstrator requires the robot to guide a loop along a wire from one end to the other without making contact with the wire itself.

To automatically detect any contact between the loop and the wire, a simple electrical circuit is constructed. As both the wire and the loop are conductive, it is possible to measure any collision of the two by measuring the resistance between them. In the demonstrator, this is implemented by using an Arduino board and attaching the 3,3 voltage output to the wire and an analog input pin to the loop. With this setup, collision can be detected by monitoring the voltage at the input pin, which will be approximately 0V or 3.3V if the electrical circuit is open or closed, respectively. With the introduction of a threshold value, the continuous measurement can be converted into a boolean collision detection. Besides a collision event, which constitutes losing the game, the successful reach of the goal end of the wire must be detected. While the start can be expressed as an initial pose of the loop, the goal is specified as a zone in the game plane which is to be reached by the loop.

When collision is detected or the goal zone is reached, the robotic setup must be reset, i.e. the loop must be moved to the start of the wire or to a checkpoint position for the next game to commence. This is accomplished by defining two planes on which the loop is allowed to move within specified safety limits. In addition to the previously introduced game plane in which the wire is placed, a reset plane is defined. The reset plane is placed in parallel to the game plane at a distance where the loop cannot collide with the wire. When resetting the robot, the loop is moved perpendicular to the planes, from the game to the reset plane. On the reset plane, it is then moved freely over the desired reset position. Finally, the loop is again moved perpendicular onto the game plane. The reset procedure is visualized in Figure 1.5. To ensure the safety of the robotic setup, a safety stop must be initiated, if any collision is detected when moving on the reset plane, which indicates that the wire was bent outside the game plane and is potentially stuck to the loop. In addition, the collision detection must be closely monitored when re-engaging the loop into the game plane to avoid bending the wire outside the game plane or damaging the loop.

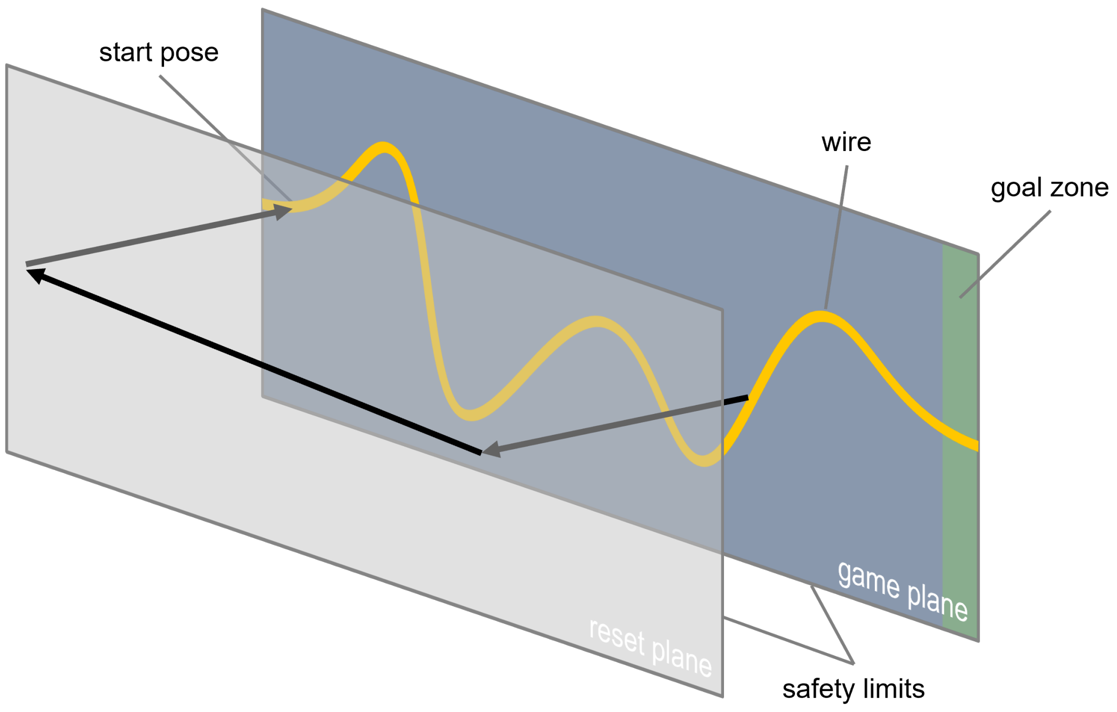


Figure 1.5 The wire loop game is played on a 2D game plane. The game is considered won when the loop reaches the goal zone. To reset the game, the loop is moved from the 2D game plane to the reset plane for collision-free movement back to the start pose or an intermediate checkpoint pose.

1.3.2 Use Case 2: Object Picking

Object picking constitutes a ubiquitous application in industrial robotics in scenarios such as palletizing [88], bin picking [23, 30], or warehousing [38, 97]. Object picking is typically accomplished through the deployment of a robotic manipulator, equipped with an end effector that is specifically designed to grasp and lift the object. The manipulator navigates towards the desired object location using motion planning algorithms, which ensure a safe and obstacle-free path. Upon arrival at the specified location, the end effector is engaged to seize the object using a suitable grasping or suction mechanism. Notably, object picking can be an arduous task for industrial robots as objects may present with varying attributes such as size, weight, texture, and shape, which can affect the efficiency and efficacy of the end effector. Additionally, the location and orientation of the object can also affect the robot's ability to pick up the object effectively. To address these challenges, industrial robots require advanced sensing and control systems, that enable the robot to adapt to the object's properties and pick it up with greater accuracy and efficiency.

The demonstrator for this second use case consists of a Franka Emika Panda industrial robot that is mounted on a table. Toy building blocks are placed on this table and within the workspace of the robot. As sensory equipment, a camera is installed on the table in a fixed location. As a tool, the robot is equipped with a gripper to enable the grasping of the building blocks, and a force-torque sensor to detect collisions with the environment. The hardware setup is depicted in 1.6.

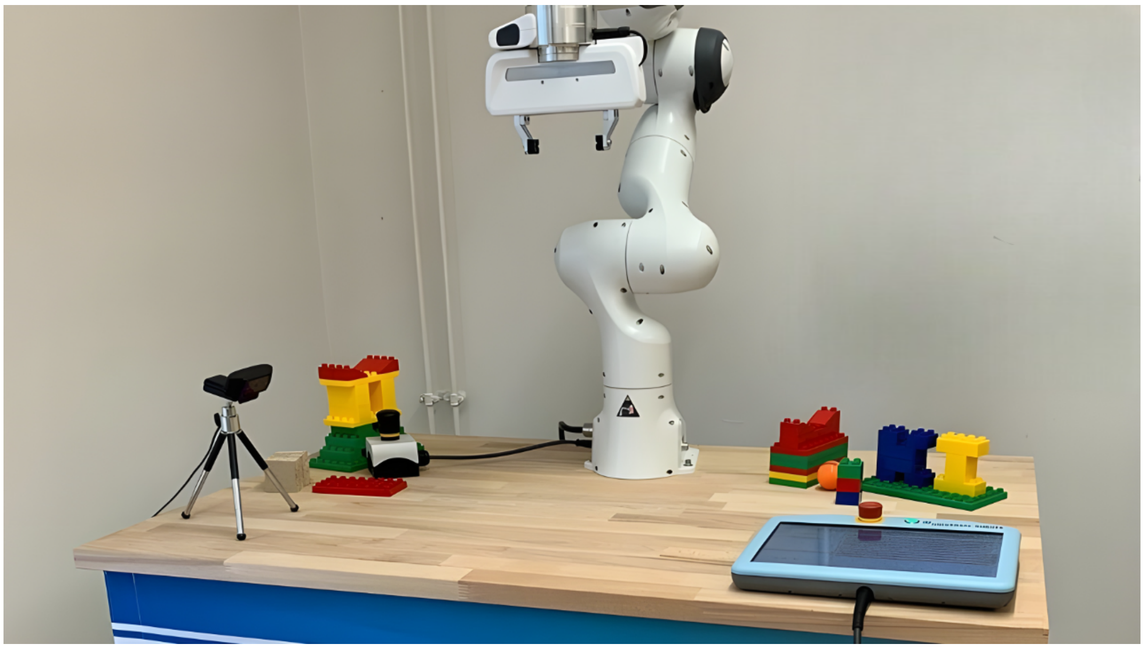


Figure 1.6 The object picking demonstrator consists of a Franka Emika Panda robot arm and a fixed camera for visual feedback. On the table, toy building blocks are placed for the robot to pick up.

A standard calibration procedure is used to calibrate the camera with respect to the robot base frame, which requires determining the robot-to-camera transformation matrix ${}_{r}T^{c}$. A calibration board is placed in front of the static camera, which contains a pattern of easily identifiable key points, such as chessboard corners. By identifying the key points in the image and using explicit knowledge about the key point configuration, e.g. the chessboard square dimensions, it is possible to compute the 3D coordinates of the key points in the camera frame ${}_{c}k_{i}$. Next, the robot Tool Center Point (TCP) is moved to each key point and the corresponding TCP position ${}_{r}p_{TCP,i}$ is recorded. Assuming that these two positions are equal, the robot-to-camera transformation matrix ${}_{r}T^{c}$ can be found by solving the linear system ${}_{r}T^{c} \cdot {}_{c}k_{i} = {}_{r}p_{TCP,i}$.

In order to successfully accomplish a pick-and-place task, a building block must be brought within a specified threshold distance to the target pose. For the purposes of this use case, the distance measure hereby only includes the positional distance and not orientation. Thus, determining if a handling task was successfully solved requires the localization of the building block. For pick-and-place tasks with one building block, it is possible to use the gripper's interface to determine if an object has been gripped, which can be derived from the remaining distance between the grippers fingers after a closing command. Afterward, the building block position can be estimated via the known gripper pose.

To ensure a safe autonomous operation of the robot, a bounding box which the gripper must not leave is defined. This bounding box safeguards against collisions of the gripper with the table to avoid damage to the demonstrator. Further, the gripper movement is restricted to four degrees of freedom, allowing translational movement in all three spatial dimensions, but restricting rotation to the axes perpendicular to the table surface. This design choice ensures that the gripper is always the lowest point of contact, effectively ruling out collisions between the robot and the table. The force-torque sensor is continuously monitored to induce protective stops when collisions with obstacles placed inside the bounding box occur.

1.3.3 Use Case 3: Clip Assembly in Aircraft Manufacturing

The third use case is provided by a real-world industrial assembly process in aircraft manufacturing, in particular the assembly of a shell section for the Airbus A320 family. A shell consists of four main components, depicted in Figure 1.7. A metal skin forms the outermost layer of the shell. To provide stability to the skin, vertical and horizontal struts are introduced. These struts are called formers and stringers, respectively. Finally, rectangular components are used to connect stingers and formers at their intersections, fusing them into a rigid structure. These components are referred to as clips, and their collision-free assembly into the shell is the objective of this application scenario. The assembly of clips is a task which is considered hard to automate and often times performed by manual labor, since each clip and corresponding installation location are unique and thus require a customized trajectory. In addition, the number of units produced in the aircraft industry is significantly lower than in other industries, e.g. automotive, reducing the economic efficiency of automation solutions. Addressing these challenges towards automation is the focus of this use case.

To automate the clip assembly process, the robot must first pick up a clip and transport it to its corresponding installation site. Second, the robot must find a collision-free path to insert the clip into the shell. The first subtask of object picking is already considered in the second use case. The second subtask of collision-free clip insertion is the focus of this third use case. To this end, a dedicated hardware demonstrator based of the real-world assembly line is utilized. This demonstrator consists of an UR5 industrial robot with a gripper mounted on a table. The robot is also fitted with a force-torque sensor to detect and mitigate collisions. Three different clips and their respective installation site are positioned within the workspace of the robot. The installation sites include the skin, stringers and formers, which are 3D-printed from the digital shell model and thus accurately represent the geometry of the real-world scenario. The clips are fitted with 3D-printed inlays that allow the robot to grasp them. The demonstrator is depicted in Figure 1.8.

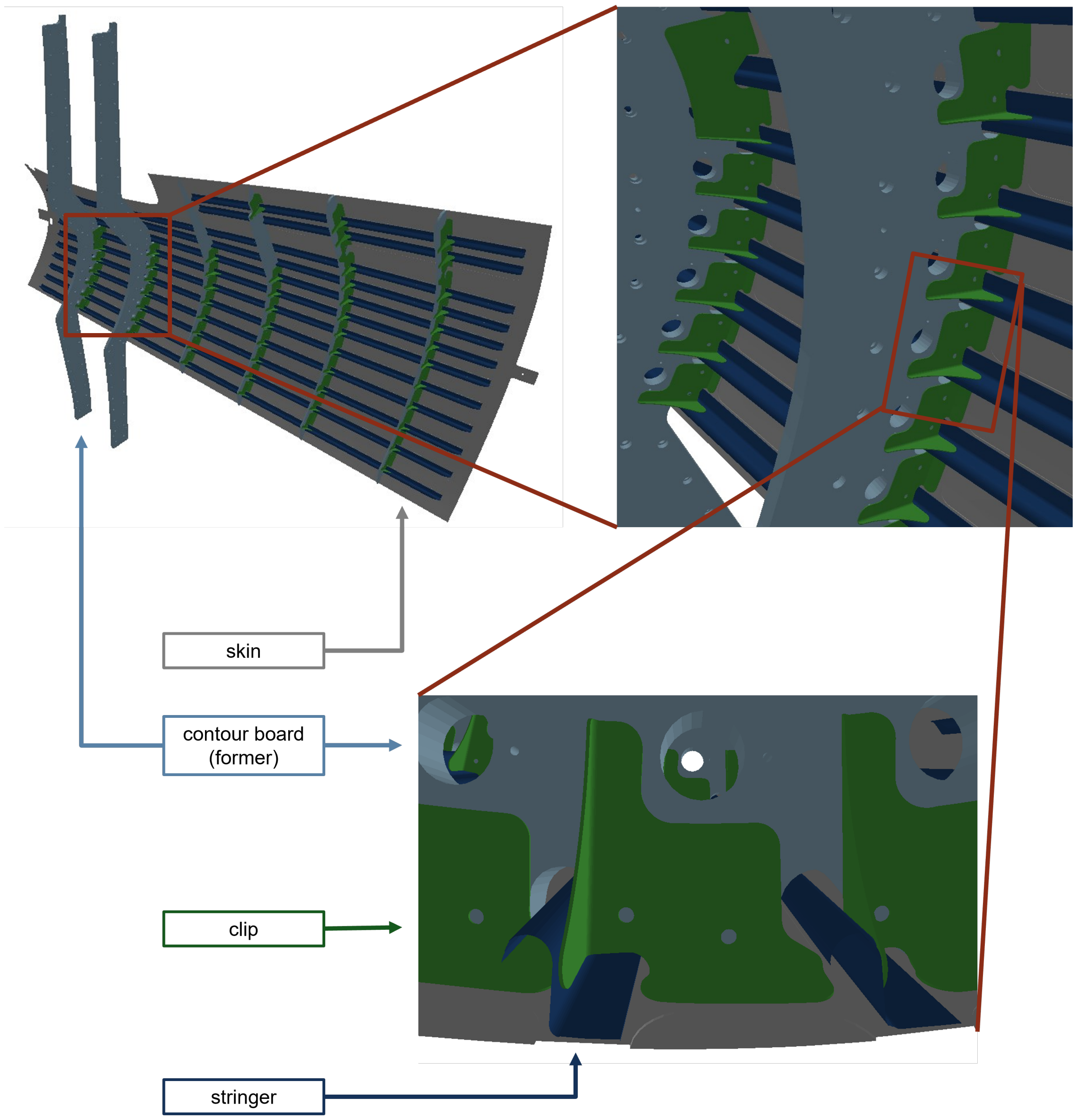


Figure 1.7 Overview of the shell structure of an airplane. The shell consists of a metal skin, which is reinforced by vertical and horizontal struts. The struts are connected by clips.

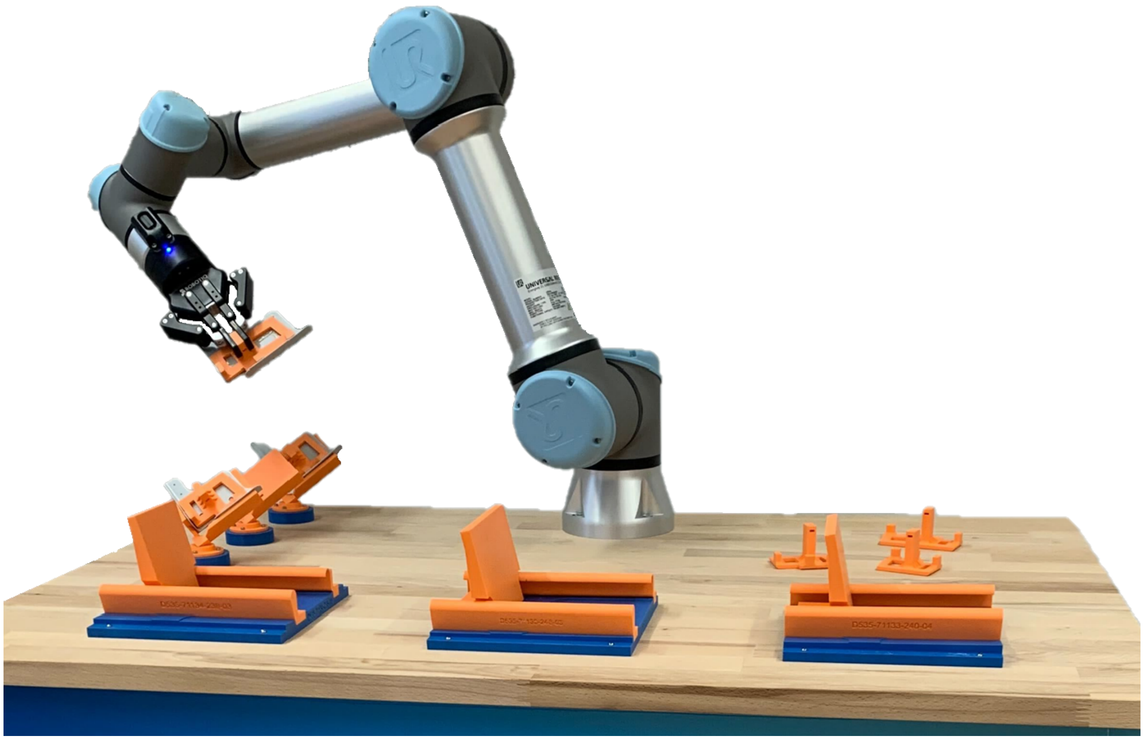


Figure 1.8 The clip assembly demonstrator consists of an UR5 industrial robot, clips and 3D-printed installation sites. The goal of the assembly task is to insert the clips into the shell structure.

The aim of clip insertion is the collision-free manipulation of a given clip into its target pose. Both position and orientation are essential to ensure that the clip is inserted correctly. To locate the pose of the clip at any given time, the clips are placed in fixtures which are specifically designed to accommodate the clip's geometry and constrain its movement. The fixtures provide a known and controlled position and orientation of the clips. Once a clip is picked up, its pose can be derived from the known gripper pose with a single static transformation. Thus, no additional sensory equipment is required to determine the location of the clips and the success of the assembly task.

The assembly procedure is split in four distinct phases: picking, insertion, removal, and placing. The picking is achieved based on the known location of the clip in its fixture, after which the clip is moved over the respective installation location. Here, the to-be-developed automation solution inserts the clip into the shell. At this stage, the demonstrator deviates from the real-world process. In the real-world process, the robot releases the clip and leaves it assembled in the aircraft shell. In the demonstrator however, the clip is not released, but disassembled to automatically reset the setup for repeated execution. This is realized by moving the entire assembly trajectory in reverse until the clip is returned to its fixture and the robot is back in its initial pose, ready to process the next clip. Throughout the entire movement, especially during the assembly and disassembly steps, the force-torque sensor of the robot is monitored to detect undesired collisions and avoid damage to the components of the demonstrator.

1.4 Structure

This section provides an overview of the structure of the thesis, which is schematically depicted in Figure 1.9. The subsequent chapter 2 introduces the relevant foundational concepts from the domains of industrial robotics, DL, Computer Vision (CV), DRL, and TL. Chapter 3 presents the State-of-the-Art (SOTA) relevant for the RQs, setting the stage for the original contributions of the thesis. Chapter 4 provides baselines for the three use cases using standard DRL and TL approaches, establishing performance expectations and identifying limitations that motivate the research directions taken by this thesis in the following four chapters.

Chapter 5 explores the concept of asynchronous RL as possibility to achieve continuous movement when using position control in combination with DRL. The proposed asynchronous RL framework allows an agent to determine the next action while the current action is still being executed, which enables a smooth transition between RL steps. In addition, parametric curves, in particular Bézier curves, are considered to constrain an agent's action space to continuous trajectories and avoid undesired jerk.

Chapter 6 investigates the development of perception modules and the possibility to pretrain them in simulation. On the one hand, supervised learning in the form of 6D pose estimation models are considered for the object picking task, which are shown to be transferable from simulation to reality using Domain Randomization (DR). On the other hand, unsupervised learning in the form of autoencoder models is applied in the wire-loop use case, where a low-dimensional representation of the relevant process features, i.e. the wire shape, is not self-evident. Through injecting domain knowledge in the form of segmentation maps, meaningful low-dimensional latent spaces are extracted.

Chapter 7 focuses on exploiting structure in robotic movement to improve learning efficiency, as well as for modularization. The decomposition of trajectories is used to modularize policies into hierarchical levels using HRL, where the higher levels learn overall strategies in the form of subgoal selection, and the lower levels learn tactical maneuvers to achieve the subgoals. In addition to improving learning efficiency, it will be shown that the modularized low-level policies are task-independent and thus reusable across different task instances. Further, the reversibility of trajectories is leveraged to achieve assembly-by-disassembly with DRL, by learning a policy for disassembly, and pretraining an assembly policy on reversed disassembly demonstrations.

Chapter 8 delves into the domain of cross-robot TL. Specifically, it focuses on possibilities to correspond two industrial robot arms of different morphology. A methodology is proposed which maps trajectory demonstrations from a source robot into trajectories of a target robot by using Forward Kinematics (FK) and Inverse Kinematics (IK) models, whereby the general shape of the robotic arm is retained. The mapped trajectories are then utilized to pretrain a DRL policy for the target domain in order to decrease the amount of new experience to be collected.

Finally, chapter 9 provides a comprehensive analysis and evaluation of the research conducted in the thesis and relates the results to the research questions. It reflects on the strengths, limitations, and potential implications of the findings. Additionally, future research avenues are outlined.

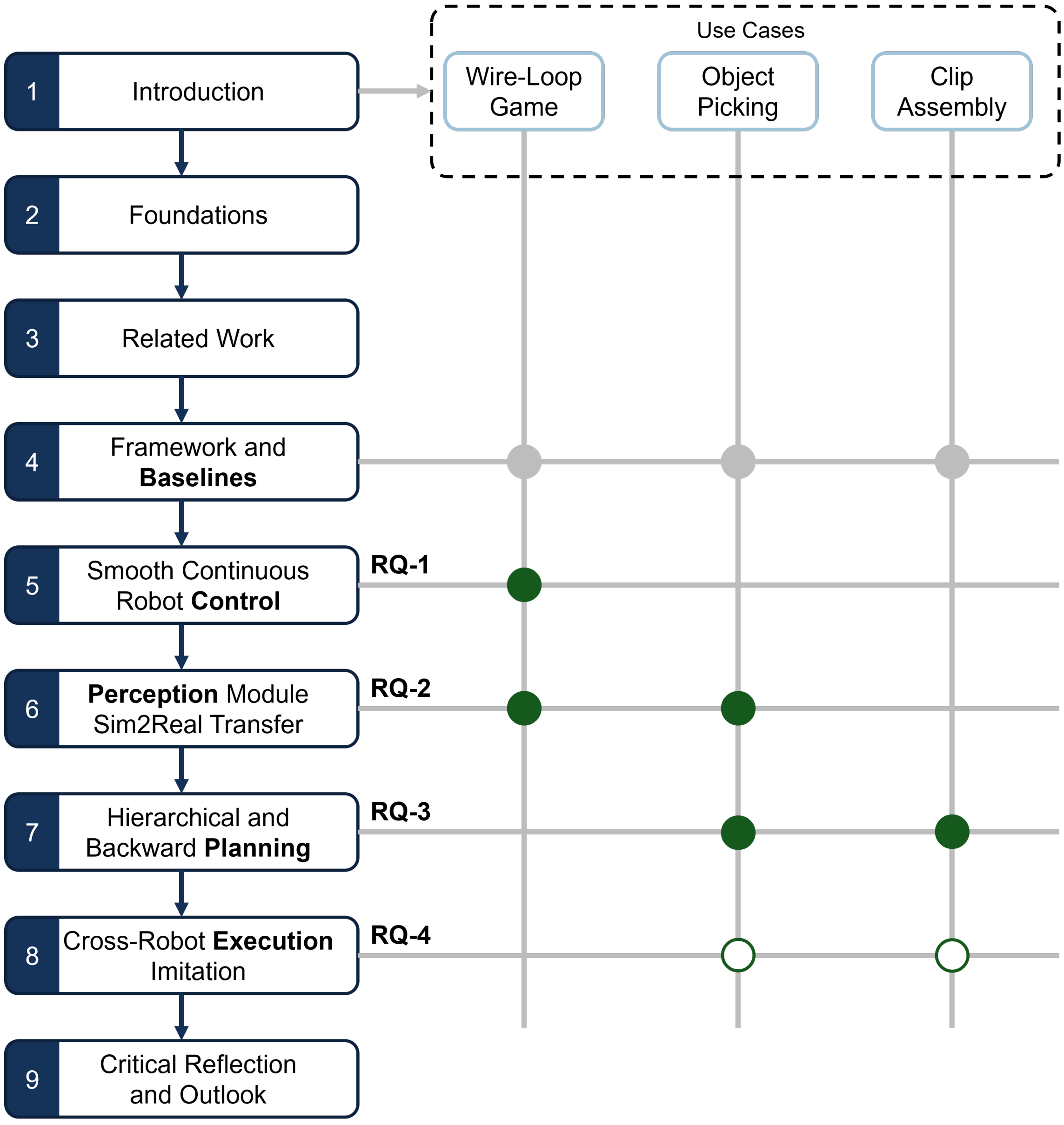

Use Cases
Wire-Loop Game
Object Picking
Clip Assembly
1 Introduction
2 Foundations
3 Related Work
4 Framework and Baselines
5 Smooth Continuous Robot Control
6 Perception Module Sim2Real Transfer
7 Hierarchical and Backward Planning
8 Cross-Robot Execution Imitation
9 Critical Reflection and Outlook
RQ-1
RQ-2
RQ-3
RQ-4

Figure 1.9 Schematic illustration of the structure of the thesis. Dots at the intersections indicate relevance of the use cases for the chapters and respective RQs. The experimental setup for RQ-4 reuses key elements from the use cases in a simplified scenario.

2 Foundations

"Early AI was mainly based on logic. You're trying to make computers that reason like people. The second route is from biology: You're trying to make computers that can perceive and act and adapt like animals."
- Geoffrey Hinton

This chapter provides the relevant background information on the research areas in whose intersection this dissertation is situated: industrial robotics, DRL and transfer learning. The chapter opens with an introduction of the application domain in the form of industrial robotics. The subsequent section introduces the field of DL, which offers the ability for data-driven modeling of complex processes instead of relying on domain-expertise driven and manual modeling. In this context, a particular focus is placed on CV as a key component of modern robotic systems to extract relevant information from visual sensory input. Next, the DRL paradigm is introduced as a control methodology that enables agents to autonomously learn and adapt through interactions with their surroundings. Finally, TRL is introduced as a means to address the challenge of data inefficiency when training DRL agents.

2.1 Industrial Robotics

In general, an industrial robot system combines the manipulation capabilities of an industrial robot, task-specific tools, sensory equipment, and a task-specific program to automate a given task in an industrial context. According to the ISO standard 8373:2021, an industrial robot is an "automatically controlled, reprogrammable multipurpose manipulator, programmable in three or more axes, which can be either fixed in place or fixed to a mobile platform for use in automation applications in an industrial environment" [76]. Mechanically, a manipulator constitutes an assembly of links, joints, and actuators that form the robot's core mechanical structure. Links are physically rigid elements, which are interconnected by joints. These joints, mechanical connectors, facilitate controlled motion between adjacent links. Most commonly, joints are either prismatic or rotary, constraining the relative motion to linear or rotary movement, respectively.

In order to modify the joint configuration and thus the shape of the manipulator, actuators such as electrical motors are employed. By providing an interface to the actuators, it is possible to move the manipulator in a desired fashion via task-specific software programs. In order to automate a given task, an industrial robot is usually combined with an end-effector. The end-effector is a tool required to accomplish the task, e.g. a gripper or a welding gun, which is attached to the multipurpose manipulator.

This thesis focuses on serial manipulators, which are the most common type of manipulator found in industrial applications with a 60% market share according to the IFR [121]. Serial manipulators are characterized by a single series of links and joints from the robot's base to the end-effector. Serial manipulators are often times an anthropomorphic structure, with a robotic arm used for positioning and a robotic wrist used for orientation of the end-effector. In contrast to serial manipulators, parallel manipulators exhibit multiple link-joint series from the base to the end-effector. Thus, the joint positions in a parallel manipulator are not independent of another. While parallel manipulators are mainly chosen for tasks requiring high precision at high speeds in limited workspaces, serial manipulators are preferred when a larger workspace is required.

In the following, the mathematical foundations relevant to describing a robot's kinematic structure and movement, as well as the fundamentals of robot control are introduced.

2.1.1 Kinematics Description

To describe and subsequently program the movement of an industrial robot, a mathematical formalization is required. To this end, the field of robot kinematics deals with the study of the geometric and spatial properties of the robot's motion without considering the actuator forces or torques involved. For two neighboring links in the kinematic structure of a robot, it is possible to formalize the spatial relationship between the coordinate systems of two adjacent links $n-1$ and n as a transformation ${}^{n-1}T_n(jp_n)$, parameterized by the connecting joint's position jp_n.

One popular method to describe such transformations is the Denavit-Hartenberg (DH) convention [41, 67, 130]. This convention enables the description of any transformation between two coordinate systems as a sequence of four elementary transformations, i.e. translations $Trans_{axis}(distance)$ along and rotations $Rot_{axis}(angle)$ around a Cartesian axis. Each elementary transformation is parameterized by a single parameter $distance$ or $angle$, resulting in four DH parameters $d_i, \theta_i, r_i, \alpha_i$ in total to describe an arbitrary coordinate system transformation

$$ {}^{i-1}T_i = Trans_{z_{i-1}}(d_i) \cdot Rot_{z_{i-1}}(\theta_i) \cdot Trans_{x_i}(r_i) \cdot Rot_{x_i}(\alpha_i) \tag{2.1} $$

It is important to note that the four DH parameters may be functions of the connecting joint's position, as it modifies the relative position of two links.

Given a parameterized model of the kinematic chain in form of sequential transformations, it is possible to analytically determine the pose, i.e. the position and orientation, of key elements in the kinematic chain for a specific configuration of joint positions. As transformations are chainable, the location of any coordinate system n along the kinematic chain with respect to the coordinate system of the robot's base equates to ${}^{0}T_n(jp_1, ..., jp_n) = {}^{0}T_1(p_1) \cdot ... \cdot {}^{n-1}T_n(jp_n)$. This procedure of determining a pose from the joint positions is called FK. One common application is the localization of the TCP ${}^{0}p_{TCP} = {}^{0}T_n(jp_1, ..., jp_n) \cdot {}^{n}p_{TCP}$, which is a process-relevant point defined in the end-effector's coordinate system, such as the point between the fingers of a gripper or the tip of a welding gun.

Conversely, the kinematic chain model can also be utilized for the opposite application: finding the joint positions which produce a desired pose, usually of the end-effector. This procedure is called IK. Unlike FK, performing IK results in ambiguous solutions, as most end-effector poses can be achieved with multiple different joint configurations. Thus, IK requires an additional selection strategy to resolve this ambiguity. Besides feasibility criteria, such as avoiding collisions and joint limits, selection strategies may include performance criteria, such as minimizing joint movement in an entire trajectory.

2.1.2 Parametric Path Description

To define and guide the motion of an industrial robot, paths represent the desired route that the robot should follow to reach a specific location. Trajectories delve deeper into the temporal aspect, defining how the robot should move along the path over time. The control procedures based on paths and trajectories are called continuous path control and trajectory control, respectively. In continuous path control, paths are communicated to the control program of a robot as a series of poses in planning space, i.e. the joint or end-effector pose space. Trajectory control additionally requires the specification of the velocity profile, with which the robot should follow the path [40].

Parametric curves are a common choice for expressing paths. Parametric curves are mathematical representations that describe the relationship between two or more variables in a systematic and continuous manner, allowing for the precise characterization of complex motion. A parametric curve function can be sampled at various locations to generate waypoints to be sent to the robot. One such family of parametric curves which will be utilized in this thesis are Bézier curves, which are a popular choice for designing paths in various industrial fields, such as CAD [52], as well as robotics [74, 165].

Bézier curves allow the parametrization of a function effectively using a set of control points [18, 50]. A Bézier curve of degree n is parameterized by $n+1$ control points $P_0, \ldots P_n$ and is defined as $B(j) = \sum_{i=0}^{n} b_{i,n}(j) \cdot P_i$, with the Bernstein basis polynomials $b_{i,n}(j) = \binom{n}{i}(1-j)^{n-1}j^i$. The curve parameter $j \in [0,1]$ allows the sampling of points continuously along the curve, whereby $B(0) = P_0$ and $B(1) = P_n$.

One desirable characteristic of Bézier curves is that the derivative in P_0 and P_n is tangent to $\overline{P_0 P_1}$ and $\overline{P_{n-1} P_n}$. This property allows multiple Bézier curves to be stitched together into a single continuous curve, enabling the modeling of complex paths with multiple simpler piecewise sections. The parametric continuity of a curve is thereby defined as its differentiability in all points. As Bézier curves are infinitely differentiable, also referred to as smooth or of class C^∞, the continuity of the overall curve depends on the transitions between Bézier curves. By ensuring that for two Bézier curves a and b of degree n, the curves touch at the end points $P_n^a = P_0^b$ and the control points $P_{n-1}^a, P_n^a, P_0^b, P_1^b$ are collinear, the resulting curve will be at least of continuity C^1. This is visualized in Figure 2.1. Further, it is always possible to find a reparametrization from the Bézier curve parameters j to the time parameter t which transforms a C^m continuous path into a C^m continuous trajectory [28].

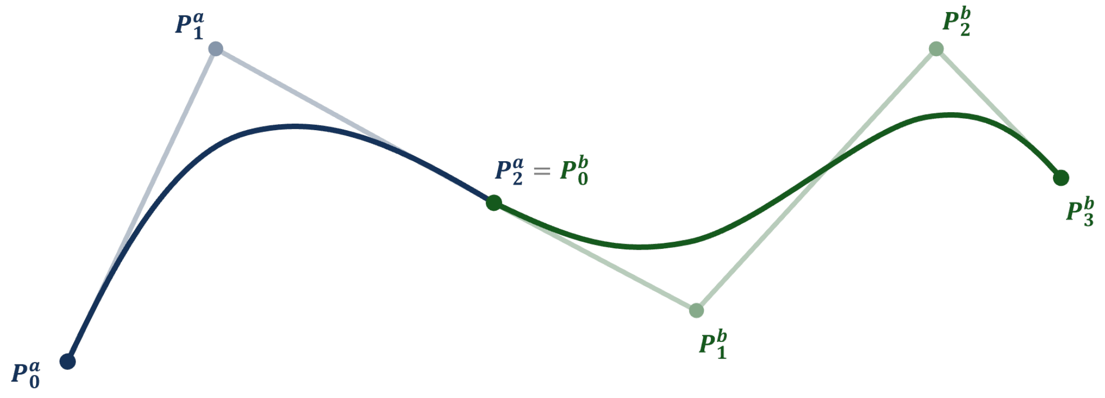


Figure 2.1 Bézier curves are parameterized by a set of control points. Multiple Bézier curves can be stitched together to form a continuous path by ensuring that the control points of the adjacent curves are collinear.

2.1.3 Closed-Loop Robot Control

The overarching goal of robot control is to orchestrate the robot's movements in such a way that it can effectively manipulate its environment as desired. This goal entails achieving a delicate balance between accuracy, speed, and safety in the robot's actions. Whether the objective is to assemble components, perform welding tasks, or handle delicate materials, the control system must generate suitable robot movements to ensure not only the successful completion of the task but also the preservation of the robot's integrity and the safety of its surroundings.

In general, an industrial robot system constitutes a closed-loop control system, depicted in 2.2. The task program thereby fills the role of the controller, which issues movement commands to the industrial robot and end-effector based on the system state measured via sensor equipment to achieve the task's objective (reference). As mentioned above, industrial robot manufacturers usually provide high-level interfaces, such as the ability to specify command poses which are translated by a robot-internal control program into required actuator voltages. This significantly simplifies the complexity of task programs, as planning can be performed in the joint space or the end-effector pose space in the form of continuous path or trajectory control.

Generating a suitable task program is a non-trivial task and requires careful consideration of the requirements of an industrial robot system. In the past, robots have mainly been employed in static and highly repetitive environments, in which the

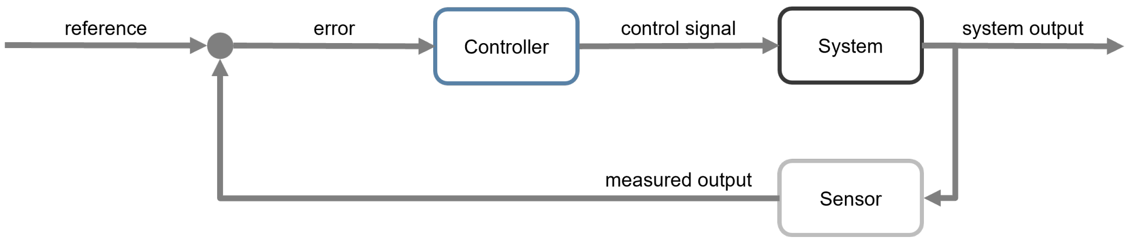


Figure 2.2 In a closed-loop control system, a controller aims to minimize the error between a desired goal state and the system state measured by sensory equipment.

motion of a robot can be manually programmed. However, keeping the process conditions static requires a high degree of engineering effort. Consequently, as industries evolve towards more dynamic and adaptable production environments, the need for more advanced control methods capable of handling increasingly complex process conditions becomes apparent. The subsequent sections introduce the key research domains foundational to the development of such advanced control methods.

2.2 Deep Learning

The field of DL describes the subfield of ML that focuses on the application of ANN to automatically learn and extract intricate features and patterns from data. The following provides an introduction into the field of ML, and subsequently highlights the key concepts behind ANN models. Further, the application domain of CV is introduced, with a particular focus on Convolutional Neural Network (CNN) architectures. CNNs as specialized ANN architecture for image data have established a dominance in the field of CV, achieving state-of-the-art performance in a wide range of tasks, including robot perception [138]. CNNs will be used throughout this thesis for any vision-related task.

2.2.1 Machine Learning Fundamentals

The field of machine learning concerns itself with methods and algorithms that enable computers to learn from data, identify patterns, and make predictions or decisions without being explicitly programmed for each specific task. Mathematically, a machine learning model can be described as a function $y = f(x, w)$ which projects an input x onto an output y in dependence of some parameters w. The general goal of machine learning is to first specify the structure of the function f, and subsequently find a parameter vector w with which the function will calculate an optimal output y from an input x.

The most significant capability of machine learning systems is known as generalization. Generalization describes the ability to determine a prediction function from training data, which not only fits the training data, but is also able to accurately handle inputs not included in the training set. This characteristic makes machine learning highly attractive for problems, where the input vector space is so vast that training with a small subset of this vector space is the only computationally feasible option.

The structure of the function f as well as the dimensions and meaning of the vectors x, w and y are dependent on the specific application. In general, the machine learning approaches can be categorized into three broad learning paradigms: supervised, unsupervised and reinforcement learning.

Methods belonging to the domain of supervised learning use labeled training data, meaning that, given a training sample x, the output an algorithm is expected to produce $\hat{y}$ is known. In the training process, the parameters w are adjusted in order to minimize the deviation, also referred to as the loss, between the model outputs and labels for all input-label pairs $(x, \hat{y})$ in the training data $\mathcal{L}_w = \sum_i |f(x_i, w) - \hat{y}_i|$. Supervised learning can further be split into classification and regression tasks. Classification tasks on the one hand require assigning discrete classes to a given input. The function to be found is interpreted as a decision boundary in the input space, and classifying an unknown sample becomes a matter of determining on which side of the boundary it lies. Regression tasks on the other hand are used when trying to approximate continuous labels.

In scenarios where labeled data is not accessible, unsupervised learning algorithms play a pivotal role in identifying intrinsic patterns and relationships within the data. These revealed patterns, including clusters, can subsequently be leveraged for applications like data compression [211] or anomaly detection [1]. However, distinct from supervised learning approaches, the absence of definitive ground truth labels makes it challenging to objectively assess the performance of unsupervised learning algorithms. As a result, these methods find more frequent use in data analysis contexts as opposed to prediction tasks.

As the third category of machine learning paradigms, RL is centered around scenarios in which an agent engages in iterative interactions with an environment. Given that RL is the central approach under investigation in this thesis for the automation of industrial robotic systems, a more comprehensive introduction to the paradigm is provided in section 2.3.

2.2.2 Artificial Neural Networks

ANNs have revolutionized the field of machine learning by serving as powerful tools for approximating complex functions. Inspired by the interconnected structure of neurons in the human brain, ANN are designed to capture patterns within data [147]. At the core of ANNs lies their layered architecture, consisting of input, hidden, and output layers. Figure 2.3 illustrates the architecture of an ANN. Each layer comprises interconnected nodes (neurons) and an activation function. The neurons process and transform the data and the activation function introduces nonlinearity to the network, allowing it to model complex relationships present in data. Mathematically, a layer describes the operation $y = \sigma(Wx)$, whereby W denotes the learned weight matrix and σ a nonlinear activation function[1]. This configuration is also referred to as a perceptron. By stacking multiple layers and using the output from a previous layer as input to the next layer, the resulting ANN structure, also called a Multilayer Perceptron (MLP), is able to model complex relationships between the input and output data.

[1]To simplify the notation, it is assumed that $x_0 = 1$ and thus the weight matrix entries $W_{i,0}$ constitute the bias terms of the layer.

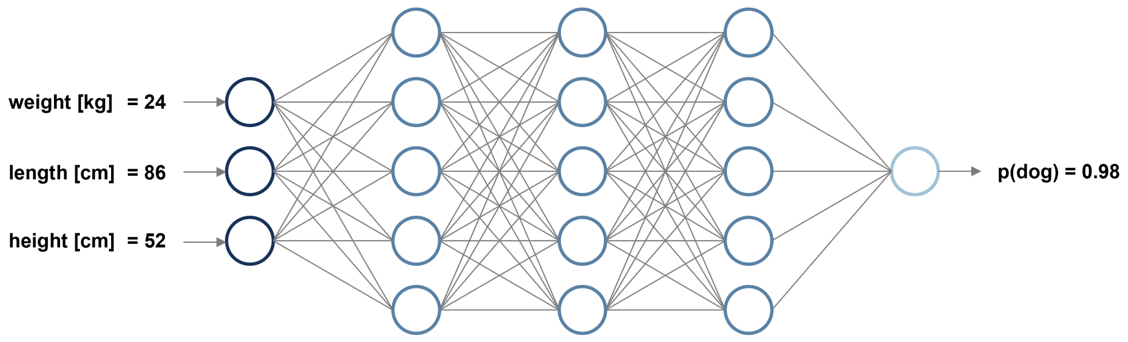


Figure 2.3 An illustrative example of an ANN architecture. The three input features are processed by three consecutive hidden layers of 5 neurons each. The output layer consists of a single neuron, which is interpreted as the probability whether the input features describe a dog.

ANNs are supervised learning models and as such are trained using labeled data. In particular, the weights of the ANN are adjusted using a technique called backpropagation [150]. Backpropagation exploits the hierarchical nature of ANN, where the output of a layer solely depends on its weight matrix and input, to efficiently calculate the gradient of a given error function with respect to the weights. Once the gradients are known, they can be used to adjust the weights towards minimizing the prediction error, i.e. improving the performance of the ANN.

While ANNs are typically trained with labeled data, they can also be used in unsupervised settings, for example as autoencoders [96]. An autoencoder consists of two ANN, an encoder and a decoder. The encoder compresses input data into a lower-dimensional latent representation $z = f(x)$ and the decoder reconstructs the original input from this compressed form $x = g(z)$. The key insight is that by forcing the network to pass information through a bottleneck of lower dimensionality, it must learn to preserve the most important features while discarding irrelevant details. This enables feature extraction and dimensionality reduction without requiring labels.

2.2.3 Computer Vision

CV is a field of artificial intelligence focused on enabling machines to interpret and understand visual information from the world [10, 172]. In general, the goal of a CV model is to extract meaningful features from visual sensory input, such as images and video. Key tasks include image classification, object detection, semantic segmentation, and pose estimation, all of which are essential for autonomous decision-making in robotic applications.

In recent years, DL has played a dominant role in the domain of CV with the ability to learn increasingly complex features from data, in contrast to early techniques relying on hand-crafted feature extraction and domain-expertise driven statistical analysis [10, 31, 128, 197]. However, the application of data-driven DL methods introduces a new challenge: the curse of dimensionality [118]. With respect to ANN, this phenomenon describes the exponential growth in the computational complexity of architecture required to handle increasing data dimensionality. This issue is particularly pronounced in CV, where the visual input data is typically high-dimensional, such as high-resolution images.

To address this challenge, specialized architectures have been developed to manage the exponential growth in computational demands. One of the most popular approaches is the CNN, which have emerged as a powerful solution for handling high-dimensional data like images [91, 213]. Unlike standard MLPs, CNNs are specifically designed to exploit the spatial structure and local correlations present in grid-like data. CNNs use convolutional layers to automatically learn features from small local regions of the input data, which is well-suited for capturing patterns like edges, textures, and shapes in images. By sharing weights across the entire image, CNNs dramatically reduce the number of parameters, making them more efficient and less prone to overfitting than fully connected MLPs.

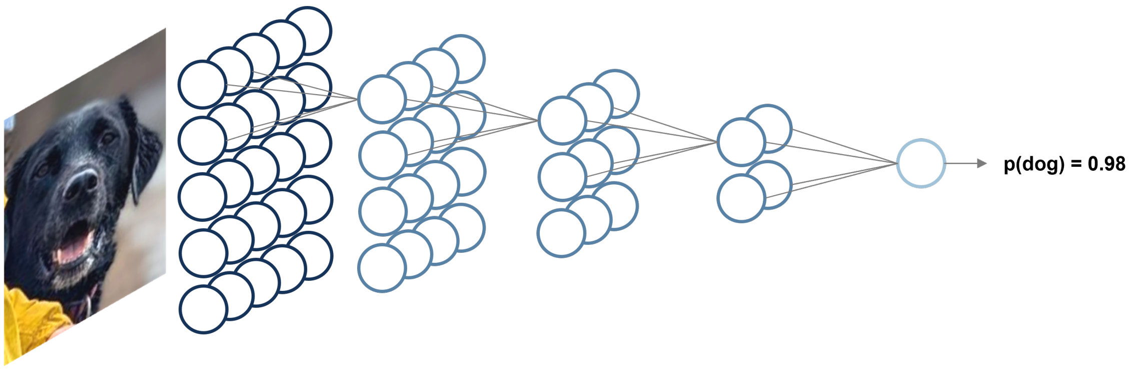


Figure 2.4 An illustrative example of a CNN architecture. The input image is processed by three consecutive convolutional layers of 2x2 kernels each. The output layer consists of a single neuron, which is interpreted as the probability whether the input image shows a dog.

One of the most popular[2] architectures in CV is the YOLO family of models, which are known for their high accuracy and real-time performance for object detection [143]. The original YOLO architecture constitutes a CNN designed for 2D object detection. In short, YOLO divides an input image into a grid of cells, and predicts class probabilities, bounding box coordinates, and confidence scores for each cell. Due to the simultaneous localization and classification in a single forward pass, YOLO is able to achieve real-time performance. Based on the success of YOLO in 2D object detection, the architecture was extended to 3D object detection by means of the YOLO6D model. The main difference to the original YOLO architecture is that YOLO6D predicts the bounding box corners in the 2D image plane, which are then projected into 3D space to determine the 3D pose of the object.

[2]The original YOLO paper was cited over 63,600 times as of July 2025.

2.3 Deep Reinforcement Learning

The following section first introduces the paradigm of RL, which enables an agent to learn a control task through interaction. Subsequently, two DRL algorithms for continuous state-action spaces, and thus particularly suited for industrial robotics applications, are outlined. In addition, HRL is introduced as a technique to decompose complex tasks into simpler sub-tasks. Finally, strategies for parallelizing environments to speed up the training process are discussed.

2.3.1 Reinforcement Learning Fundamentals

The RL paradigm describes scenarios, where an agent interacts with an environment to collect and learn from experience. A fundamental concept in RL is the Markov Decision Process (MDP), which provides a formal framework for modeling these interactions. At its core, a MDP represents a decision-making problem with four components: state space $\mathcal{S}$, action space $\mathcal{A}$, transition probability function $\mathcal{T}(s'|s,a)$, and reward function $\mathcal{R}(s,a,s')$. The state space encompasses all different environment states an agent may observe. The action space comprises all possible actions an agent can perform in the environment. The transition probability function describes the dynamics of the environment, i.e. the likelihood that a subsequent state s' will be reached when action a is performed in state s. Finally, the reward function calculates the desirability of taking an action in a given state, effectively encoding the underlying goal of the decision-making problem. A noteworthy characteristic of MDPs is the independence of individual transitions, i.e. each transition probability and reward solely depends on the current state and action, and not on any previous states, decisions or rewards. This characteristic is referred to as the Markov property and is crucial for the application of RL.

In virtually all real-world robotics applications, it is impossible to observe the complete state of the environment. Instead, sensors like cameras capture only partial observations of the underlying state. This scenario is formally described by a Partially Observable Markov Decision Process (POMDP), which extends the MDP framework with an observation space $\mathcal{O}$ and observation probability function $\Omega(o|s)$ that maps states to observations. However, for clarity and simplicity, the remainder of this thesis will use the standard MDP notation, treating observations as states, while acknowledging that partial observability is inherent in real robotic systems. As the implications of partial observability are not the focus of this thesis, it will be assumed that the observation space reflects all relevant information about the environment necessary to make an optimal decision.

The iterative interaction between an agent and its environment is depicted in Figure 2.5. At a given time step t, the agent first observes the current state of the environment $s_t \in \mathcal{S}$. Based on the state, the agent then chooses an action $a_t \in \mathcal{A}$. The decision-making process is referred to as the policy of the agent. The policy $\pi(a|s)$ is a function which describes the probability of choosing an action given a state. The execution of the action leads to a new state s_{t+1} according to the environment's underlying transition probability function. In addition, a scalar reward $r_t \in \mathcal{R} \subset \mathbb{R}$ is issued to the agent, which serves as learning signal to optimize the policy with a given RL algorithm.

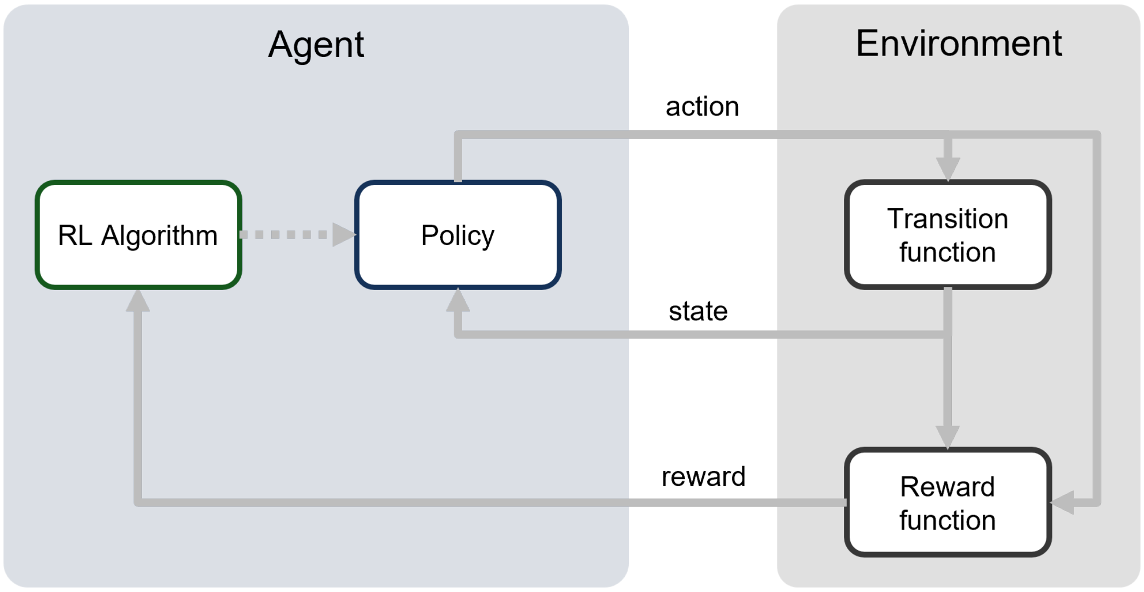


Figure 2.5 Basic structure of a RL setting. The agent interacts with the environment by selecting actions based on the current state and the policy. The environment transitions to a new state and issues a reward signal, which is used to optimize the policy.

The overall goal in a decision-making problem is the maximization of the expected discounted return $G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$. Here, the discount rate $\gamma \in [0, 1]$ determines to which degree potential future rewards are to be taken into consideration. Different values for γ will result in different optimal policies, as low discount rates favor immediate rewards, leading to more conservative policies. Conversely, high discount rates lead to greater risk-taking, as immediate rewards may be forfeited in hope of even greater future rewards.

With the expected discounted return as objective, the value of a state can be expressed by a state-value function $V_\pi(s) = \mathbb{E}_\pi[G_t|s_t = s]$. This function represents the expected discounted return when the environment is in a given state s and all subsequent actions are chosen from a policy π. Analogously, the value of an arbitrary action is expressed by a state-action-value function $Q_\pi(s, a) = \mathbb{E}_\pi[G_t|s_t = s, a_t = a]$, which describes the value of taking an action a in a state s and following the policy π thereafter. Due to the Markov property, the value functions can be expressed as dynamic programming equations $V_\pi(s) = \mathbb{E}_\pi[R_t + \gamma V_\pi(s_{t+1})|s_t = s]$ and $Q_\pi(s, a) = \mathbb{E}_\pi[R_t + \gamma V_\pi(s_{t+1})|s_t = s, a_t = a]$. These equations are also known as Bellman equations, and their recursive formulation enables the feasible computation without the need for calculating the infinite sums of all future rewards.

Formalized value functions allow for systematic policy evaluation and comparison. A policy π is considered better than another policy π' if $V_\pi(s) > V_{\pi'}(s)\ \forall\ s \in \mathcal{S}$. Thus, an optimal policy satisfies $V_{\pi^*}(s) \geq V_\pi(s)\ \forall\ s \in \mathcal{S}, \pi$. In RL, the goal lies in finding such an optimal policy by using the observed reward signals as feedback to continuously improve the decision-making capabilities.

When interacting with its environment, an agent faces the exploration-exploitation dilemma. At every decision, the agent on one hand has the option to exploit its current knowledge and choose the action with the highest value according to its policy. On the other hand, since it is possible that its current worldview is flawed and the policy may favor a suboptimal action, it could explore the effects of choosing a lower valued action, in the hope of discovering actions with actual higher values leading to better returns. While exploration is desired at the beginning of training to expose the agent to a wide variety of experience, exploitation is favored at later stages to take advantage of the agents' ability to avoid suboptimal situations.

In the pursuit of enhancing the training efficiency and effectiveness of DRL a multitude of strategies and techniques have emerged as powerful tools for achieving more rapid and effective learning. This section serves as an introduction to the key concepts that are commonly employed to elevate the capabilities of DRL in general, and for robotic applications in particular.

Uniform, Prioritized and Hindsight Experience Replay

Experience replay is a foundational technique in DRL that significantly enhances the learning process of RL algorithms by retaining and reusing previously collected experiences. It mitigates the challenge of efficiently utilizing experience data, ensuring that the agent is exposed to a diverse range of situations throughout training. Instead of using each experience only once, the experience data is collected in a buffer, from which it is sampled by the respective DRL algorithm and used to improve the policy.

The sampling also has the added benefit of training on more heterogeneous experience data, as otherwise the agent would condition too much to recent experiences. In its simplest form, a fixed-size buffer stores experiences, which are then uniformly sampled during training. This random sampling prevents the agent from conditioning to recent or frequent experiences, and instead maintains a balanced learning process.

Prioritized Experience Replay (PER) takes this concept a step further by assigning priorities to experiences based on their learning potential. Instead of sampling data from the pool of collected experience uniformly at every learning step, experiences associated with surprising or high-error states are sampled more frequently, emphasizing challenging situations. This intelligent prioritization accelerates the learning process by focusing on the most informative experiences.

In addition to deliberating selecting experiences with an expected high learning signal, it is also possible to modify experience to increase the value for learning. This concept is embodied by Hindsight Experience Replay (HER), which addresses the challenge of sparse rewards in goal-oriented tasks. It has proven to be particularly valuable in industrial robotics settings where achieving desired goals may be infrequent or difficult. HER works by exploiting explicit knowledge about the environment to alter goals retrospectively, effectively re-imagining unsuccessful attempts as successful ones. This technique allows the agent to learn from virtual successful trajectories, even though it is not capable of producing them itself.

2.3.2 Deep Reinforcement Learning Algorithms for Continuous State-Action Spaces

In the context of industrial robotics, the utilization of continuous state and action spaces is of paramount significance. Unlike discrete state spaces, which can be limiting in their modeling capabilities, continuous state spaces allow for a more accurate representation of the intricacies inherent in real-world scenarios. Similarly, continuous action spaces provide the granularity of control necessary to execute high-precision tasks within robotic systems. Due to their ability to model complex continuous relationships within data, ANNs have also prevailed in the domain of RL, whereby the application of ANNs in RL is referred to as DRL. A DRL architecture particularly suited to handle continuous action spaces is the actor-critic paradigm, which separates the policy (actor) from the value function (critic) [61]. This thesis employs two widely adopted actor-critic algorithms[3]: DDPG [102] and PPO [160], which are introduced in the following and visualized in Figures 2.6 and 2.7.

DDPG belongs to the class of off-policy algorithms. Off-policy describes the ability of an algorithm to utilize arbitrary experience data independent of the policy which collected said data. This flexibility enables the utilization an experience replay buffer, enhancing learning efficiency and data utilization. In DDPG, the actor consists of a deterministic policy $\pi(a|s) = \mu(s)$, while the critic models the state-action value $Q(s, a)$. Both modules are implemented as ANNs. As the policy is deterministic, exploration is achieved by superimposing the deterministic output of the policy with noise, thus $a_t = \mu(s_t) + \epsilon$. The magnitude of the noise ϵ is decayed throughout the training to achieve a transition from exploration to exploitation. The rate of decay is manually chosen and thus poses a weakness of DDPG. If not correctly tuned, the agent may either explore too much or too little. While the former causes training to progress more slowly than necessary, the latter can lead to suboptimal policies due to a lower variance in collected experience. With the experience collected throughout training, DDPG optimizes the critic in a supervised fashion by minimizing the state-action value Temporal Difference (TD) error $\mathcal{L}_{critic}^{DDPG} = Q(s, a) - [r + Q(s', \mu(s'))]$. As the actor should produce actions with a high state-action value, the policy is optimized by minimizing $\mathcal{L}_{actor}^{DDPG} = -Q(s, \mu(s))$.

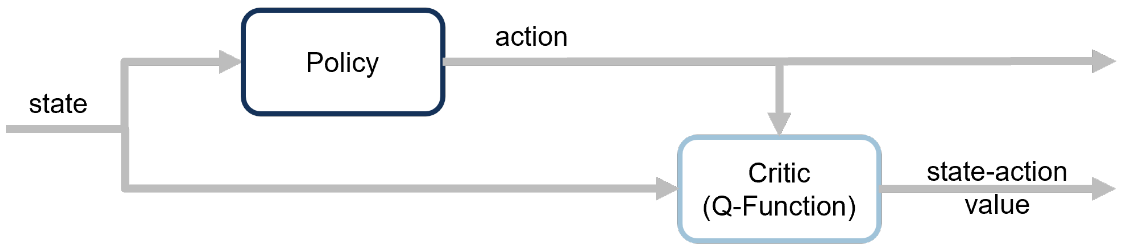


Figure 2.6 The DDPG architecture consists of a deterministic actor and a state-action value critic. The actor chooses an action, which is judged by the critic. In training, the critic provides the actor with a gradient towards more favorable actions.

[3]The original papers of Deep Deterministic Policy Gradient (DDPG) and Proximal Policy Optimization (PPO) were respectively cited over 19,900 and 27,700 times as of July 2025.

In contrast, PPO follows an on-policy approach, which means that it requires the training data to be collected using the current policy. As the policy is continuously adapted through training, this effectively means that PPO does not make use of an experience replay buffer. Despite this, PPO empirically exhibits more stable and predictable learning behavior, making it suitable for scenarios where safety and reliable convergence are paramount. Additionally, PPO's ease of implementation makes it a popular choice as DRL algorithm. The critic employed by PPO aims to model the state-value function $V(s)$, and is trained by minimizing the state-value TD error $\mathcal{L}^{PPO}_{critic} = V(s) - [r + V(s')]$. The actor used by PPO is a stochastic policy $\pi(a|s)$. Stochastic decision-making is achieved by using the policy ANN to predict the mean $\mu(s)$ and standard deviation $\sigma(s)$, which are subsequently used to sample from a Gaussian distribution to produce an action $a = \mathcal{N}(\mu(s), \sigma(s))$. An important feature of the Gaussian distribution is the ability to also compute the actual probability $\pi(a|s)$, with which an action was sampled. To optimize the policy, the critic is used to first compute the advantage of each experience $\hat{A}(s_t, a_t) = G_t - V(s_t) = \sum_{k=0}^{T-1} \gamma^k r_{t+k} + \gamma^T V(s_{t+T}) - V(s_t)$. The advantage is thereby estimated as the deviation of the discounted return from a collected experience trajectory and the state-value predicted by the critic. Given that the rewards used to compute the discounted return are closely tied to the policy's performance, it becomes evident that collected experience loses its relevance for training as soon as the policy undergoes changes. This characteristic underscores the on-policy nature of the PPO algorithm.

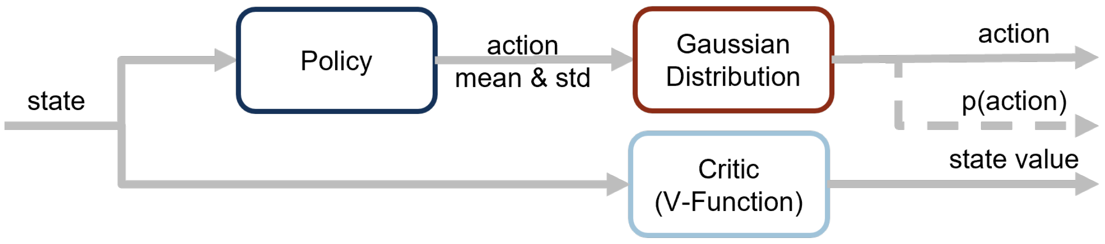


Figure 2.7 The PPO architecture consists of a stochastic actor and a state-value critic. In training, the critic is used to compute the advantage of a chosen action, which serves as a gradient signal for the actor.

The policy π is optimized by minimizing

$$\mathcal{L}^{PPO}_{actor} = \alpha log\pi(a|s) - min(\pi/\pi_{old}\hat{A}, clip(\pi/\pi_{old}, 1-\epsilon, 1+\epsilon)\hat{A}) \qquad (2.2)$$

The first term in this objective constitutes an entropy regularization, which effectively penalizes the policy for producing high-probability actions and thus incentivizes exploration. Additionally, the probability of actions with a positive advantage is increased, and conversely the probability of actions with negative advantage is decreased. To determine the direction of the policy adjustment, i.e. the policy gradient, the policy π_{old} with which the training data was collected is taken as reference. The policy gradient is clipped to limit the change in policy, which improves stabilization of the training process.

2.3.3 Hierarchical Reinforcement Learning

One major reason for poor sample efficiency of DRL algorithms is the implicit requirement of an agent to simultaneously learn short-term control and long-term planning to successfully solve a given task. This dual demand places a significant burden on the learning process, necessitating an extensive number of interactions with the environment to robustly attribute the outcome of action sequences to the respective short- or long-term strategy.

HRL approaches aim to address this issue by decomposing the decision-making process of an agent into different levels of temporal abstraction [45, 56, 57, 71, 175, 191]. Each layer constitutes a policy which are responsible for different stages in the decision-making process. The higher levels are used to divide a task into subgoals, which are to be achieved by the respective lower levels. Thus, by design the higher levels take on long-term planning responsibilities, and the lower levels are tasked with short-term situational control.

In this thesis, the HRL algorithm Hierarchical Actor-Critic (HAC) will be utilized [95]. As depicted in Figure 2.8, HAC stacks DDPG agents to achieve hierarchical policy structures. Algorithmically, HAC consists of a nested loop structure, where each loop tied to an agent layer handles a different level of temporal abstraction. The innermost loop interacts with the environment on the highest temporal frequency, receiving subgoals and passing feedback to the next outer loop. This structure continues until the outermost loop, the highest level, which is the level that receives the overall task goal. Importantly, from the perspective of an individual agent layer, the layers and environment below can be treated as abstracted high-level environment, where the state space includes the subgoals of the respective higher layer and the action space is the proposed subgoal. This allows for the application of the standard DDPG algorithm to each layer.

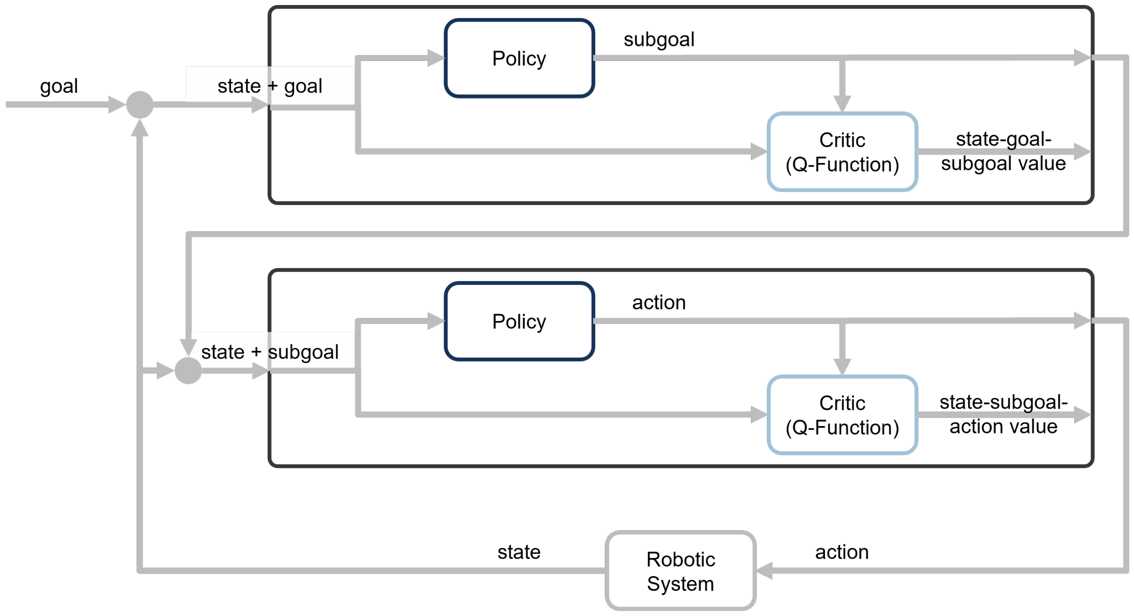


Figure 2.8 HRL agents comprise multiple policy levels with different temporal abstraction. The higher levels are responsible for long-term planning and divide a task into subgoals, which are then achieved by the lower levels. The depicted HAC architecture stacks DDPG agents.

2.3.4 Environment Parallelization

Training an agent on a given task requires training data in the form of experience collected through interaction with the environment. A common strategy to speed up the experience collection is the utilization of multiple environments in parallel, both in the form of simulations [89, 100, 204] or real-world industrial robotic systems [94]. In general, parallelization strategies can be divided into synchronous and asynchronous parallelization. A popular special case of synchronous parallelization is vectorized parallelization. The three different strategies are contrasted in Figure 2.9.

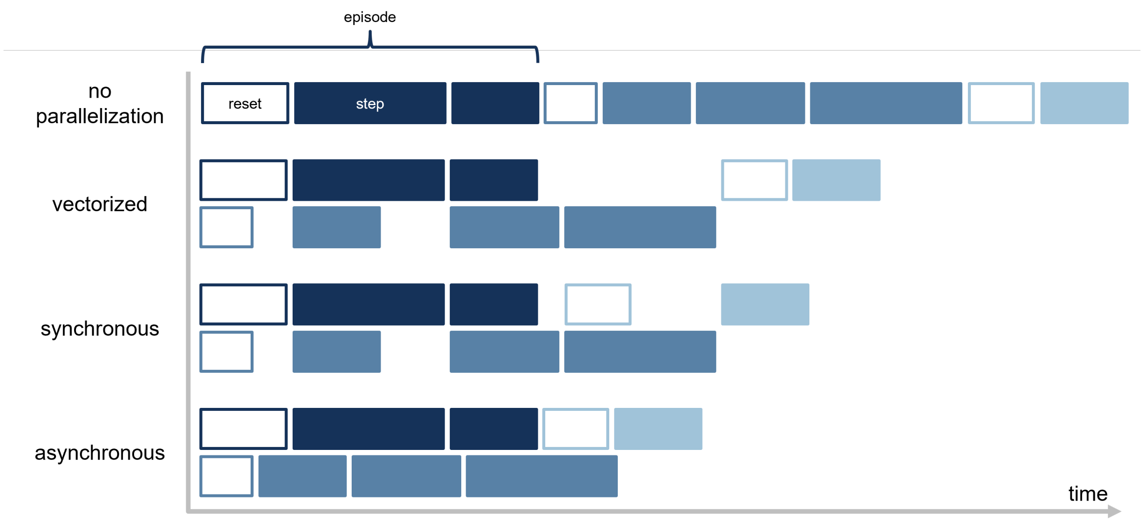


Figure 2.9 Illustrative comparison of different parallelization strategies. Reset and step operations are represented as outlined and filled boxes, respectively. Interactions which belong to the same episode and thus are executed sequentially in the same environment are of the same color.

With synchronous parallelization, the interactions with the environments are executed simultaneously. This approach offers a significant advantage in that each synchronized step per definition captures a result from every environment, streamlining the necessary overhead involved in matching the newly collected information to the respective trajectory. The downside however is, that each parallel execution takes as long as the slowest process.

A special instance of synchronous parallelization is vectorized parallelization, which is preferably employed when interactions with the environment can be computed via matrix operations. While easily implemented, all environments must carry out the same type of operation in parallel, i.e. reset or action execution. Thus, vectorized environments are not efficiently used when the episode lengths of the parallel environments vary.

In contrast, asynchronous parallelization allows each environment to progress independently, collecting experiences and resetting on an individual basis. This approach results in the fastest experience accumulation, as environments are not idly waiting for slower environments to complete their interactions. However, it introduces complexities related to data consistency and synchronization, necessitating a more sophisticated management overhead of the training process to ensure correct communication between the training loop and the environments.

2.4 Transfer Reinforcement Learning

TL describes the utilization of knowledge acquired in a source domain towards the reduction of training efforts in a target domain. How this is achieved through the application of specific transfer learning methods depends highly on the nature of the knowledge, as well as the differences between the source and target scenarios.

In the context of applying TL to RL, henceforth referred to as TRL, there are two types of source domain knowledge: expert policies and demonstrations. On the one hand, expert policies are capable of solving the source task to some degree, and thus inherently possess knowledge about the source scenario. An expert does not necessarily have to be a RL agent, but can be any type of controller deployable in the source scenario, even a human operator. However, in order to be transferable, an agent's policy must be explicitly accessible and queryable. On the other hand, demonstrations are recorded interactions of an expert with the source environment, and thus contain implicit knowledge about the solution of the source task.

One special case of TRL for industrial robotics is the utilization of a simulation as source domain to pretrain policies, which will be discussed first in the following. Subsequently, the two main categories of TRL will be outlined, which coincide with the two types of source domain knowledge: expert policy modification, and expert behavior imitation learning. Imitation learning is further divided into behavioral cloning and pseudo-reward imitation learning. While behavioral cloning uses state-action experience, pseudo-reward imitation learning relies on a pseudo-reward signal which captures similarity between expert and target agent behavior. The general approaches are visualized in Figure 2.10.

2.4.1 Simulation-to-Reality Transfer

Simulations offer an intuitive possibility for efficiently developing and pretraining control agents prior to deployment to a real-world process. This option is especially attractive for industrial applications, as experiences from suboptimal process states, such as products of lower quality or slower production cycles, or critical conditions, e.g. collisions or high wear and tear of components, are infeasible to obtain using real-world hardware. Especially for data-driven approaches like DRL algorithms, the ability to interact with virtual environments without consequences for the real-world production line significantly reduces both costs and risk in the industrial context.

One major challenge arising from the use of simulations for the pretraining of models is the fact that simulations do not model the real world perfectly. Instead, deviations exist due to various factors, from lower resolutions and discretizations required to make simulations computationally feasible to unknown parameters or inadequately modeled effects. The divergence between simulation and real world is referred to as the sim2real gap [79]. Different strategies exist for overcoming the sim2real gap, three of which will be introduced in the following. This thesis will employ the last strategy, DR. To better understand the advantages of this approach, two other popular strategies, system identification and domain adaptation, are presented as comparative approaches.

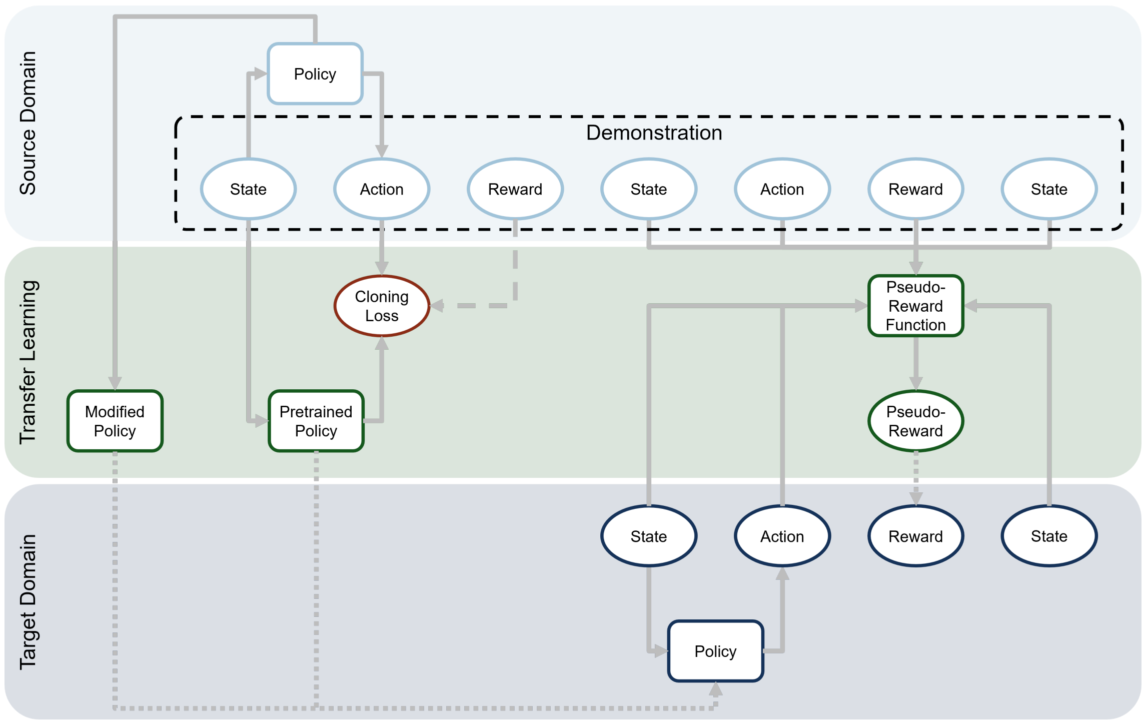


Figure 2.10 Overview of TRL approaches. Given a source policy, this policy can be deployed to the target domain. Depending on the transfer scenario, modification of the source policy may be required to adapt interfaces and/or facilitate learning. Given source demonstrations, a target policy can either be pretrained, or a pseudo-reward signal can be derived to guide the learning process.

System identification approaches aim to close the gap by improving the quality of the simulation and making the simulation as realistic as possible [33]. This approach however requires a deep understanding of the real-world process, including underlying physical principles. In addition, increasing the quality of a simulation often requires an increase in resolution of a simulation, which in turn increases computational efforts.

Domain adaptation is a different strategy which aims to augment the data produced by an imperfect simulation. The main objective lies in finding a mapping from the distribution of simulation data onto the distribution of real-world data. Once found, the mapping can be applied to enhance the realism of artificially generated training samples. Other approaches focus on mappings from both the simulation and real-world domain into a shared feature space on which the agent is trained. The development of suitable mappings however requires the availability of real-world data.

In contrast to the aforementioned approaches, DR neither requires perfect simulations nor real-world data to bridge the sim2real gap. Instead, the simulation is deliberately randomized by changing various aspects from visual components, such as colors and textures, to physical properties, such as friction parameters. Thus, the data generated by this procedure is not of one single simulation, but of a distribution of randomized simulations. As depicted in Figure 2.11, the intuition behind this approach is that an agent exposed to this data distribution will, given enough randomization, generalize to the real-world data distribution. Notably, it has been shown by various previous works that agents are able to generalize and find robust control policies applicable to the real world target task, despite the majority of artificially generated data not necessarily being realistic [8, 131, 185]. As the DR approach is the focus of this thesis, the related work will be examined in more detail in chapter 3.2.

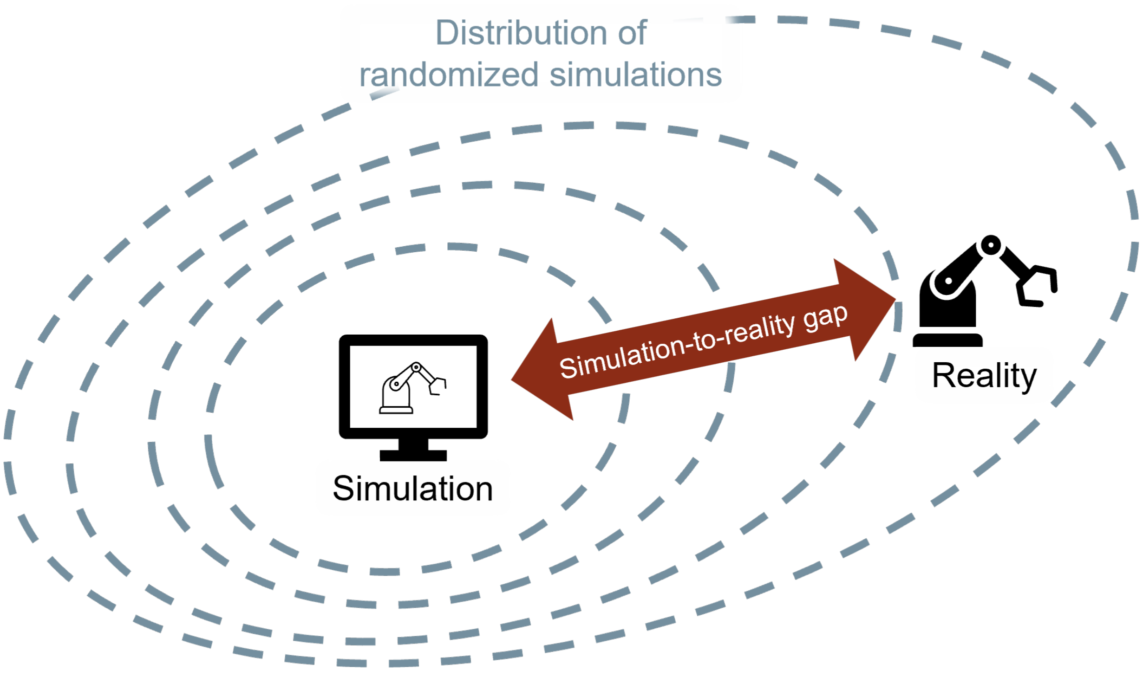

Distribution of randomized simulations
Simulation-to-reality gap
Simulation
Reality

Figure 2.11 The DR approach deliberately randomizes a simulation to generate a distribution of training samples. With enough randomization, the agent will generalize to the real-world data distribution and overcome the sim2real gap. Adapted from [109].

2.4.2 Policy Modularization

When a source policy is available, it is possible to construct the initial target policy using components of said source policy. In the simplest case, e.g. when the source policy was trained in an accurate simulation or with strategies such as DR, the policy may be taken directly without modifications. In other cases however, it is either advantageous or even required to modify the source policy. Advantageous modifications include the freezing or reinitialization of certain parts of a test ANN to respectively retain generalized and reset source-domain specific knowledge. A modification is required when the source and target scenarios differ with respect to their state and/or action space structure, necessitating the addition of translating structures, which enable the source policy to interface with the target environment.

To facilitate policy transfer, various previous works aim to explicitly modularize a source policy. Modularity is an important design principle in software engineering, which promotes the division of software systems into smaller, self-contained modules that can be developed, modified, and tested independently. This paradigm can be directly applied to the development of DRL control policies. When a policy can be explicitly split into domain-agnostic and domain-specific modules, it becomes possible to reuse the domain-agnostic components in the target domain, reducing the transfer efforts to the adaptation of the domain-specific components.

Müller et al. [122] apply the concept of policy modularization to separate an autonomous driving system into a perception module and a driving policy. This architecture allows the independent development of a domain-agnostic perception module using supervised learning, which is then used as feature extractor for the training of the driving policy in a simulation. The strategy enables the authors to directly transfer the resulting domain-agnostic driving policy from the simulation to the real world.

Devin et al. [43] divide policies for robot manipulators into task- and robot-specific modules. After training different task-robot module combinations, the authors demonstrate the ability to recombine and compose policies which successfully solve previously unseen robot-task scenarios.

Moreover, numerous previous works have demonstrated the possibility of modularizing task-independent behavioral skills into lower control policies, which can subsequently be reused in learning higher planning layers for new tasks [56, 71, 162, 191]. Here, the low-level task-agnostic policies are sometimes referred to as options [12, 53, 175].

One particular approach for policy modularization is the utilization of HRL. As HRL is investigated in this thesis, a closer examination of the related work will be provided in the next chapter.

2.4.3 Imitation Learning

When the domain knowledge is in the form of demonstrations generated by an expert, this data can be used to condition an agent towards the expert behavior. As the agent is expected to imitate the expert, TRL based on demonstrations is referred to as Imitation Learning (IL). As depicted in Figure 2.10, there are two main strategies to achieve IL. On the one hand the offline supervised pretraining of policies, also referred to as Behavioral Cloning (BC) [9, 148], and on the other hand the derivation of pseudo-reward functions which provide an agent with online reward signals that facilitate learning [11, 54, 215].

BC stands as a foundational strategy within the realm of IL, employing expert demonstrations as the source of knowledge transfer. By using this data to pretrain a student policy with supervised learning, BC aims to condition the student policy to mimic the behavior of the expert. For stochastic policies, this objective can be expressed as minimizing the Kullback–Leibler divergence (KL divergence) between the expert and the student policies $\mathcal{L}^{BC,\pi} = |log\pi(a|s) - log\pi_{expert}(a|s)|$. In the case of deterministic policies, the objective simplifies to the behavioral cloning loss $\mathcal{L}^{BC,\mu} = |\mu(s) - \mu_{expert}(s)|$. For both settings, the expert demonstrations consisting of the states s, actions a, and the action probabilities for stochastic experts $\pi(a|s)$ can be collected in an independent procedure and compiled into an offline dataset. While conceptually straightforward and effective, BC is not without limitations. As BC mimics the behavior expressed in the expert demonstrations, the procedure highly depends on the quality of the compiled demonstration data. Monotonic Advantage Re-Weighted Imitation Learning (MARWIL) takes the idea behind BC one step further by estimating the advantage of an action and re-weighting the available experience, putting a higher emphasis on high advantage actions [200].

This extension requires the additional collection of rewards during the generation of the demonstration dataset, either directly or indirectly in the form of advantage estimates. However, the supplementary information about the quality of expert decisions can significantly improve the data efficiency and success probability of a TL attempt.

In contrast to using demonstrations in an offline supervised fashion to mimic the behavior of an expert, a different strategy lies in deriving a pseudo-reward function that captures the similarity between the behavior of an expert, and the current behavior of an agent. This pseudo-reward is then combined with the reward obtained from the environment, and effectively guides the agent to imitate the expert behavior.

Both online and offline IL strategies have their respective advantages and disadvantages. If the available expert demonstrations are of high quality and directly translatable to the target domain, offline IL is an effective approach to pretrain a policy without the need for expensive online interaction with the target environment. However, in cases where the expert demonstrations are of low quality or are not directly mappable into target domain trajectories, online IL offer a more robust alternative.

3 Related Work

"If I have seen further, it is by standing on the shoulders of giants."
- Isaac Newton

This chapter introduces the related work relevant to the RQs investigated in this thesis. The following four sections discuss dedicated literature surveys conducted for each respective RQ. Each literature survey adheres to the Preferred Reporting Items for Systematic Reviews and Meta-Analyses (PRISMA) approach. First, three prominent academic literature databases were selected: IEEE Xplore[1], Scopus[2], and Web of Science[3]. These platforms were selected based on their reputation for comprehensive, peer-reviewed coverage and advanced search capabilities, ensuring access to high-quality and relevant literature. In those databases, the same keyword searches were conducted in January and again on July 5th 2025 to include more recent literature. The search terms were crafted by combining the primary search term *"reinforcement learning" AND "robot"*, which was used for all surveys, with a RQ-specific secondary search term, which will be provided in the respective section. The compiled collection of articles was then deduplicated to obtain a corpus of unique articles. This corpus was screened for relevance to the respective RQ, and the accepted articles were analyzed in detail. Additional relevant articles were identified in some instances through references and forward searches and were retroactively added to the corpus. Each literature survey discusses the identified related work and is followed by an overall assessment to put the findings in context of this thesis. It shall be stated that the assessments contain subjective interpretations and critiques by the author.

[1]`https://ieeexplore.ieee.org/`
[2]`https://scopus.com/`
[3]`https://webofscience.com/`

3.1 Real-Time Asynchronous Reinforcement Learning

As industrial robot systems are expected to interact with real-world environments, respective control agents must be capable of handling the realities of the physical world [49]. The standard RL framework is not inherently suited to be used for the application in scenarios with real-time control requirements due to the discretization of time into distinctly separate steps in the underlying MDP. Each time step begins with the observation of the environment and ends with feedback about the chosen action. When considering real-world tasks on a continuous timescale, this formalization leads to significant drawbacks. Most notably for industrial robotic applications, the assumption that an action has been completed before the next state is observed and evaluated does not consider latencies, such as the time required to take a camera image, communication overhead, inference time of the policy, or the processing of state, action and reward data. Towards the goal of real-time control with DRL as prompted by RQ-1, a literature survey was conducted as depicted in Figure 3.1.

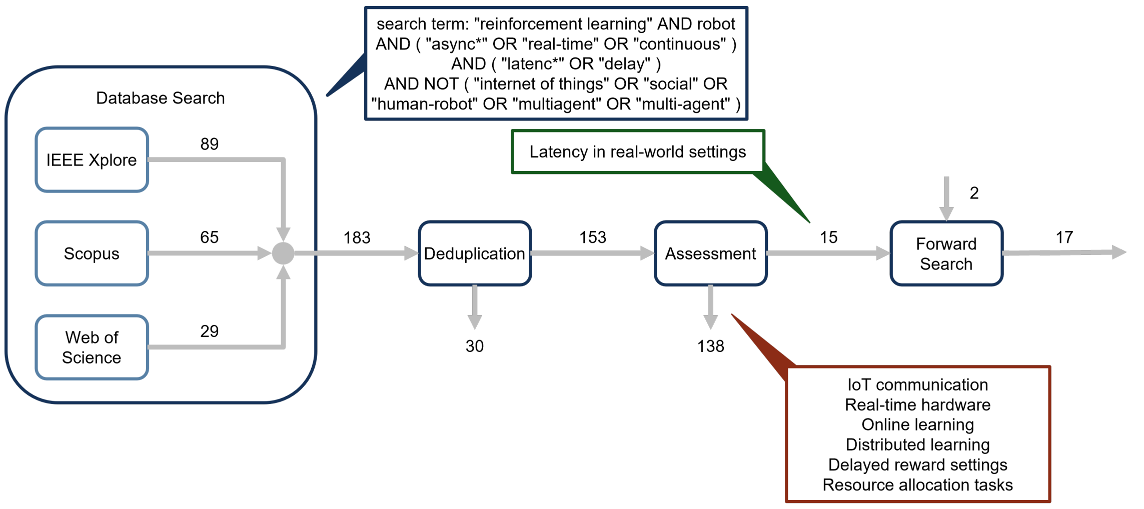

search term: "reinforcement learning" AND robot AND ("async*" OR "real-time" OR "continuous") AND ("latenc*" OR "delay") AND NOT ("internet of things" OR "social" OR "human-robot" OR "multiagent" OR "multi-agent")
Database Search
IEEE Xplore
89
Scopus
65
Web of Science
29
183
Deduplication
153
30
Assessment
15
138
Latency in real-world settings
2
Forward Search
17
IoT communication
Real-time hardware
Online learning
Distributed learning
Delayed reward settings
Resource allocation tasks

Figure 3.1 The literature survey process on the topic of real-time asynchronous RL for RQ-1. 17 relevant articles were identified which address latencies in real-world control settings.

First, an initial literature corpus was compiled from the above-mentioned databases with the following search term: *”reinforcement learning” AND robot AND (”async*” OR ”real-time” OR ”continuous”) AND (”latenc*” OR ”delay”) AND NOT (”internet of things” OR ”social” OR ”human-robot” OR ”multiagent” OR ”multi-agent”)*. The secondary search term aims to capture the challenge of real-time continuous control through asynchronous execution and decision-making, which is necessary due to latencies and delays in real-world robotic systems. A wildcard operator is used to capture both ”latency” and ”latencies” as acceptable. As the initial search term resulted in numerous irrelevant articles, the search term was further refined with a set of exclusion terms to filter out articles from unrelated fields. Articles discussing real-time systems in the context of *internet of things* focus primarily on conceptual, hardware- and network-level aspects of robotic system integration. A second set of articles was focused on social implications of robotics and requirements for real-time communication in human-robot interaction. Finally, articles discussing multi-agent systems were excluded due to their focus on fleet management, rather than real-time control of individual robots. The refined search term resulted in 183 articles, from which 17 articles emerged as relevant to RQ-1.

Upon analysis of the final corpus, the categorization provided in Table 3.1 is proposed, which differentiates three general strategies which will be referred to as: **Latency Observation**, **State-Action Observation**, and **Dynamics Rollout**. The strategies are discussed in the following.

Latency Observation	[159], [24], [153]
State-Action Observation	[83], [73], [65], [140], [139], [205], [25]
Dynamics Rollout	[198], [55], [32], [208], [42], [75], [201]

Table 3.1 Categorization of articles by strategy for real-time asynchronous RL.

Latency Observation strategies explicitly provide latency measurements to the agent as part of the observation. Early works by Schuitema et al. [159] provide the first example of taking latency into account through modifying the update rules of the RL algorithms Q-learning and SARSA. Given the restriction of constant delays, the authors successfully demonstrate the possibility of modelling latency into the RL framework.

To account for variable delays, the delay measurements can be provided to the agent as part of the observation. Sandha et al. [153] demonstrate that explicitly providing the delay leads to better results than simply randomizing the delay during training with the goal of achieving generalization. The authors use measurements of the policy inference time and environment sampling time to provide the agent with the delay information. Bohm et al. [24] utilize the observation age in form of milliseconds since the state was observed, also successfully demonstrating a positive effect of incorporating latency information on the learning performance.

State-Action Observation strategies provide the action history to enable implicit inference of the actual current state of the system. Katsikopoulos et al. [83] perform early, theoretical proofs of the possibility to transform delayed MDP into a standard MDP without delays by adding the action history to the state. These proofs are later confirmed by experimental evidence provided by other works, such as Haarnoja et al. [65] who feed the past five states and actions into the agent policy report improved performance on learning quadruped locomotion, although no further investigation into the nature of the delay and choice of history length is provided.

Adding the state-action history is a valid approach, however it is not without drawbacks attributable to the previously mentioned curse of dimensionality [118] (cf. section 2.2.3). Böhm et al. [25] address the issue by using a Recurrent Neural Network (RNN) to compress the increasing action history into a lower dimensional encoding. While this enables an agent to handle increasing dimensionality, Ramstedt et al. [139] show that the general approach of basing decision-making on the action history is problematic, as the action sequences shift throughout the learning process. This means that the experience data collected likely contains action sequences that will no longer be relevant as the policy changes. While Ramstedt et al. [139] are able to mitigate this issue by online updating the experience samples, the curse of dimensionality remains.

This issue with multistep state-action histories has been addressed by Ramstedt et al. [140] through a re-formalization of the RL framework to model real-time interactions as the simultaneous evolution of states (system dynamics) and actions (policy inference). This effectively reduces the dimensionality of the required history to a single state-action step, avoiding the curse of dimensionality. Ramstedt et al. [140] further highlight the ability to convert their formalization with variable latency into a constant delay MDP, whereby the latency variance translates into a stochasticity of the transition function. A similar approach is taken by Xiao et al. [205], who prove that knowledge of the last action and delay are sufficient to handle latencies by deriving a continuous-time formulation of the Bellman operator. The authors further argue that "it is often hard to accurately predict [latency] during inference on a complex robotic system" [205] and propose using the remaining action to be executed after state observation as suitable proxy feature.

Dynamics Rollout strategies aim to explicitly model the dynamics of the environment to predict the actual current state of the system, in contrast to providing the agent with information that allows for implicit inference of the actual current state of the system. Approaches mainly differ in the modelling approach, with earlier works using classical ML techniques such as locally weighted progression regression [198] and random forests [73], and newer approaches shifting to DL-based models from standard ANNs [42], over probabilistic ANNs [32, 55] to RNNs [208].

As the dynamics model is separated from the decision-making process, the approach of learning a dynamics model is not tied to the DRL paradigm. On the contrary, learning dynamics models is a common approach in automation and control concepts such as Model Predictive Control (MPC) [208]. The question when learning-based approaches like DRL are justified is not explicitly addressed in the literature. However, in the case of MPC one requirement is the ability to measure the distance from the predicted state to a desired goal. Scenarios where this is not given, e.g. due to a vision-based state, are potential candidates for DRL approaches.

Expectedly, the dynamics rollout models are only able to predict future states with a certain accuracy due to stochasticity in the system dynamics. Derman et al. [42] provide experimental evidence of this, showing that while their dynamics rollout model achieves better results than a state-action observation baseline strategy, the overall performance deteriorates with increasing delays as the prediction errors accumulate with increasing rollout horizon.

Most recently, in 2025, Wang et al. [201] propose a combination of the latency observation and dynamics rollout strategies. The authors use a delay estimation module to model the statistical distribution of latencies. This information is then used to select from a set of dynamics rollout models, which are pretrained under different latency conditions.

The distribution of appearances of the introduced strategic approaches within the survey corpus is depicted in Figure 3.2. Notably, there is no clear trend regarding the emergence of a superior strategy. This is likely due to the strategies exhibiting complementary strengths and weaknesses, making the suitability use-case dependent. In summary, the **Latency Observation** strategy is a simple approach to directly measure the underlying phenomenon of latencies in a low-dimensional representation. Different metrics can be applied to capture the latency, such as the observation age or the policy inference time. As these metrics model the same underlying phenomenon, there is no clear advantage in using one over the other. One drawback of the latency observation strategy is that it requires the ability to measure the latency, which may add additional overhead to the system design in form of monitoring functionalities. The **State-Action Observation** strategy is a more complex approach, as it requires the agent to maintain a history of states and actions. This information however is already available in the standard RL framework and thus easily implemented [75]. The related work has shown that the necessary history can be reduced to a single state-action step, avoiding the curse of dimensionality. The **Dynamics Rollout** requires explicit modeling of system dynamics, but in doing so effectively separates latency compensation from non-delayed decision-making.

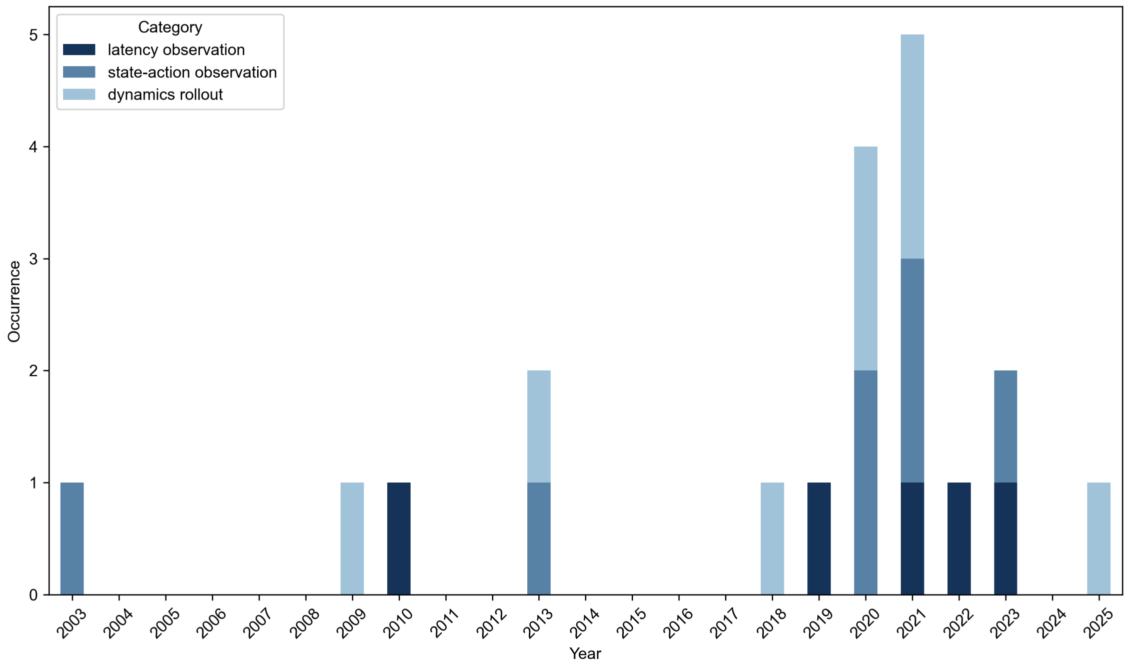


Figure 3.2 Distribution of strategies for real-time asynchronous RL over the years. No clear trend regarding the emergence of a superior strategy can be inferred.

3.2 Sim2Real Transfer with Domain Randomization

As introduced in chapter 2.4.1, the DR approach has emerged as a successful strategy for sim2real TRL. However, the success of the approach is highly dependent on the choice of parameters. When the set of varied simulation parameters is too small, the agent is less likely to generalize and transfer will be unsuccessful. On the other side, the set of parameters can be too large, encompassing parameters which are not relevant for the task at hand. While the second case is still likely to result in a successful transfer, it is not optimal, as the irrelevant parameter randomizations increase the complexity of the simulation and thus unnecessarily increase the training efforts spent on pretraining.

The following literature was identified with the search term *"reinforcement learning" AND robot AND "domain randomization" AND "transfer*" AND "sim*"*, using wildcards to capture different keyword variations, such as "transfer", "transferability", "sim-to-real", "sim2real", and "simulation". The resulting deduplicated corpus of 190 articles was screened for works that go beyond the pure application of DR and rather focus on improving the full randomization strategy. This resulted in a final corpus of 17 articles relevant to RQ-2. The full process is depicted in Figure 3.3.

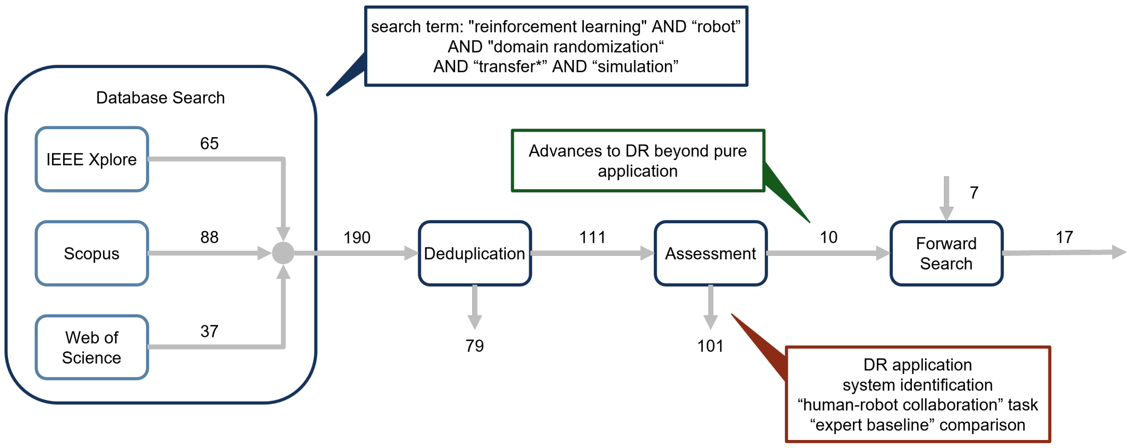

search term: "reinforcement learning" AND "robot" AND "domain randomization" AND "transfer*" AND "simulation"
Database Search
IEEE Xplore
65
Scopus
88
Web of Science
37
190
Deduplication
79
111
Assessment
101
10
Advances to DR beyond pure application
7
Forward Search
17
DR application
system identification
"human-robot collaboration" task
"expert baseline" comparison

Figure 3.3 The literature survey process on the topic of DR for RQ-2. 17 relevant articles were identified which address advances to the basic DR approach.

For the final corpus, the articles were analyzed and then categorized as depicted in Table 3.2 into four general approaches, which will be referred to as **Standard DR**, **Guided DR**, **Active DR**, and **Human-in-the-Loop DR**. The categories are discussed in the following.

Standard DR	[79], [151], [185], [131], [8], [125], [124], [82]
Guided DR	[72], [189], [92], [144], [183]
Active DR	[113], [135], [123]
Human-in-the-Loop DR	[101]

Table 3.2 Categorization of articles by strategy for DR.

Standard DR strategies all apply DR to achieve transfer to the real-world. While the approaches differ regarding the choice of parameters and potential randomization schemes, they all share that no external data is used to guide the training process. The research line of DR can be traced back to 1997 with the work of Jakobi [79], who introduces the "radical envelope-of-noise" hypothesis to address the shortcomings of simulations, i.e. the sim2real gap. The work marks the first successful sim2real transfer by randomization of simulation parameters for a simple ANN policy evolved with a genetic algorithm for a two-wheeled robot, as well as a gantry robot. The concept has since been applied to various applications, from drone flight tasks [151] to robotic grasping [185]. The latter work of Tobin et al. [185] marks an important milestone in research, as it coins the now popular term "DR", distinguishing it as a separate method from domain adaptation.

The naive approach to DR statistically increases the chances of transfer, but at the same time also increases the complexity of the simulation task as highlighted by various works [8, 131]. Peng et al. [131] point out that the choice of parameters to randomize highly influences the performance of the transfer, however they also observe that best results are achieved by randomizing as much as possible. Similar observations are made by Andrychowicz et al. [8], who further investigate the influence of the choice of parameters to randomize. The authors highlight the cost of randomizing as much as possible, as the learning task becomes more difficult, leading to longer training times. Specifically, the authors report training times of 1.5 hours without, versus 50 hours with DR. However, simultaneously the experimental results suggest that randomization of all parameters is required to achieve best transfer results.

To improve on this naive approach, Muratore et al. [124, 125] introduce a method to calculate an upper confidence bound on the optimality gap to estimate the transferability of a pretrained policy before deploying it to the real-world. After training on a set of simulation parameters, the policy is transferred and finetuned on multiple different sets of simulation parameters, which allows measuring the generalization of the policy. As the authors point out themselves, the validity of using this metric as measure of transferability is only given if the target domain is covered by the source domain distribution. Also, as the policy needs to be finetuned multiple times (using DRL) for each evaluation to achieve statistical significance, the overhead of the method is computationally expensive.

A different approach is proposed by Josifovski et al. [82], who propose a continual learning approach to DR, where randomization is continually added to the simulation environment. The authors show that this leads to both faster training and better transfer performance, as gradually increasing the complexity allows the agent to learn more robustly.

Guided DR strategies aim to improve on the ”blind” approach of randomizing in simulation and attempting to transfer to the real-world, by incorporating offline data from the target domain, such as observations or expert demonstrations.

Extending on the previously discussed works by Muratore et al. [123, 124], Leibovich et al. [92] propose a metric for estimating the sim2real transfer success of pretrained policies. The authors measure the similarity of the policy network activations for simulation and real-world data and provide experimental evidence, that the proposed metric correlates with the actual transferability.

Besides measuring the sim2real gap, other works propose to use target domain data to close the gap. Tsai et al. [189] use a single human demonstration to iteratively refine the randomization parameters towards better matching the target domain dynamics, focussing the parameter sampling from a uniform distribution to a distribution which promotes settings similar to the target domain. This approach is expanded by Tiboni et al. [183], who use multiple human expert demonstrations to estimate the probability of the demonstrations being generated by the simulation. The authors demonstrate superior performance compared to Tsai et al. [189].

Target domain data can also be used to derive curriculum learning schedules. Hermann et al. [72] use demonstrations to reset the simulation close to the desired goal state, i.e. states at the end of a demonstration trajectory. As the policy improves, the proposed algorithm increases the difficulty by increasing the variance of the randomization, resetting to states further away from the goal state, as well as resetting to random start states. The authors show that this approach significantly improves both the speed and overall performance of the pretrained policy. A different approach is proposed by Ren et al. [144], who train an autoencoder to compress high-dimensional simulation parameters (background images, meshes) into a latent space. By training a differentiable cost predictor model to estimate the policy performance, the authors are able to select points in the latent space which are expected to be more difficult for the policy. These latent space points are then decoded back into the original parameter space and used for the next training iteration.

Active DR strategies attempt to use online interactions with the target environment to improve DR. Using direct access to the target environment promises unbiased, direct feedback, in contrast to potential biases introduced by offline data.

In line with the previous attempts at measuring the sim2real gap, Possas et al. [135] model the posterior probability of domain parameters given observed trajectories. This model is then continuously evaluated on target domain trajectories collected with the current policy to sample the next set of domain parameters, selecting and training on simulations which are more similar to the target environment. Muratore et al. [123] propose to explicitly map domain parameters onto the overall goal, i.e. the policy performance on the target environment. The approach iteratively trains a policy on a set of domain parameters, before evaluating the policy in the target domain. By modelling this relationship, the authors are able to select the domain parameters which maximize the target domain performance. As the authors point out, the best choice of domain parameters does not necessarily have to match the target domain settings. However, the experiments show that if the target domain is covered by the distribution of domain parameters of the simulation, the parameters are likely to converge towards making the simulation as realistic as possible, closely relating the approach to system identification.

The sim2real gap can also be closed in an adversarial manner. Metha et al. [113] employ a discriminator to differentiate between simulation and target environment trajectories collected with the current policy. The score of the discriminator is taken as negative reward signal for a secondary adversarial agent which choses the next set of simulation parameters to train on. The intuition behind this approach is that the secondary agent is incentivized to generate simulations where the behavior of the primary agent differs. The authors show that their algorithm outperforms other methods regarding generalization to out-of-distribution target environments.

Human-in-the-Loop DR is an approach which actively involves human application experts to guide the DR process. This is in contrast to the previously discussed works, which aim to algorithmically improve and automate DR. Liebers et al. [101] point out that the process can be significantly boosted by involving human application experts. The authors identify different stages where such application experts may support: the generation and refinement of the simulation environment, the selection of suitable parameters for DR and their distributions, as well as validating agent performance. While the presented case study mainly focusses on the application of Virtual Reality (VR) to prepare and manipulate a simulation, the proposed workflow serves as valuable framework to be expanded on by future research.

Figure 3.4 depicts the distribution of appearances of the introduced approaches in the literature corpus over the years. Note that Jakobi et al. [79] from 1997 is omitted for better visibility. While no statistically significant trends can be inferred, a shift from intrinsic strategies to incorporating additional external resources is noticeable. Also, it is interesting to note that active DR approaches, while being considered by the research community, are limited to the years 2020 and 2021 and have since been neglected. This hints to the requirement of online target environment access being recognized as a severe limitation in the applicability of approaches.

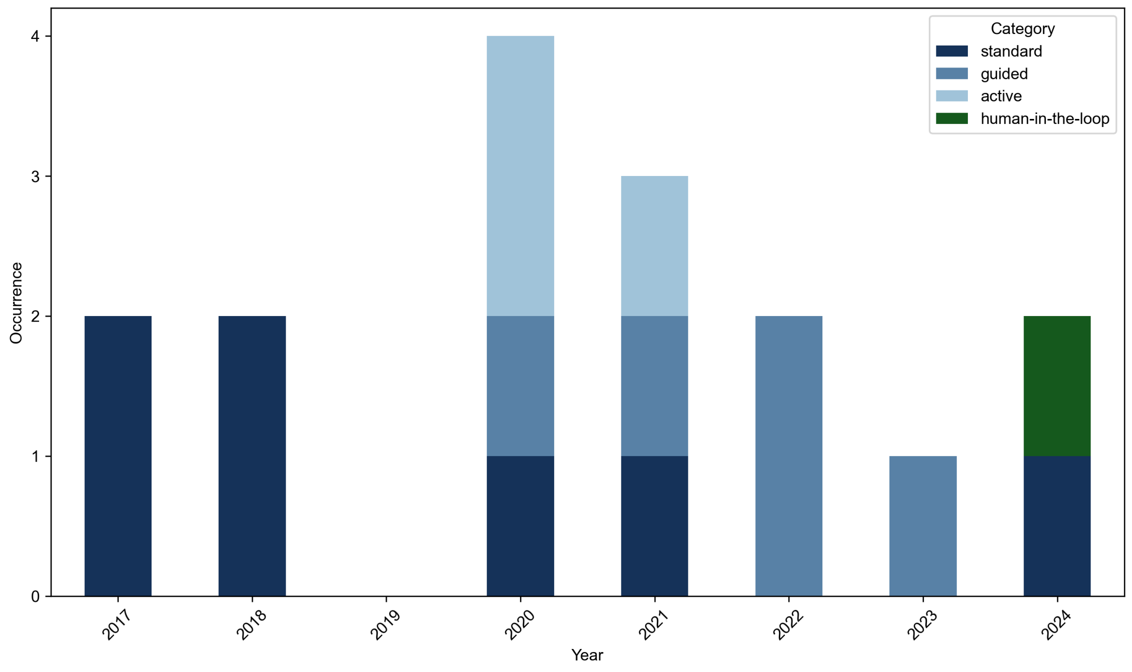


Figure 3.4 Distribution of strategies on the topic of sim2real transfer with DR for RQ-2. Offline guided DR has drawn more attention than online active DR. Most recently, human-in-the-loop DR was proposed.

3.3 Robotic Movement Structure Exploitation

As stated by Sutton and Barto [174] in their foundational work on RL, ”[p]ossibilities for exploiting structure in reinforcement learning and related planning problems have been studied by many researchers” [174]. Intuitively, position-based robotic movement trajectories exhibit two important characteristics: decomposition and reversibility.

Decomposition describes the fact that any trajectory that visits a set of waypoints can be regarded as a sequence of smaller sub-trajectories, each visiting a subset of the waypoints [175]. In terms of the MDP formalization, this is captured by the Markov property, which states that the next state depends only on the current state and action, making transitions independent of each other. **Reversibility** can be understood as the ability to invert a trajectory to achieve moving from a goal pose back to a start pose [58].

While the two concepts are related as strategies to exploit structure in robotic trajectories, they are fundamentally different in their approach and respective application. Therefore, two individual literature surveys were conducted, which will be reported in the following.

3.3.1 Cross-Task Transfer in Hierarchical Reinforcement Learning

As previously introduced, HRL modularizes policies into different levels of temporal abstraction, with the goal of reducing the complexity of the learning process [13]. Besides the advantages regarding training efficiency, the decoupling facilitates interpretability and promises transferability.

The following related work was identified using the search term *"reinforcement learning" AND "robot" AND "transfer*" AND "hierarch*"*, which was expanded with wildcards to capture different keyword variations, such as "transfer", "transferability", "hierarchy", and "hierarchical". As depicted in Figure 3.5, the resulting deduplicated corpus of 152 articles was filtered regarding relevance to the cross-task transfer of HRL sub-policies. Common reasons for rejection were no or sim2real transfer scenarios, the use of non-learned lower policies, e.g. manually designed motion primitives, different non-robotic application domains, multi-agent coordination scenarios, and works on task network graphs. Two potentially relevant articles needed to be discarded, one for which the full article was not obtainable [210], and the other for being in Turkish [59]. The selected 4 articles were further complemented with 3 additional relevant articles in a forward search, resulting in a final corpus of 7 articles relevant to the first part of RQ-3.

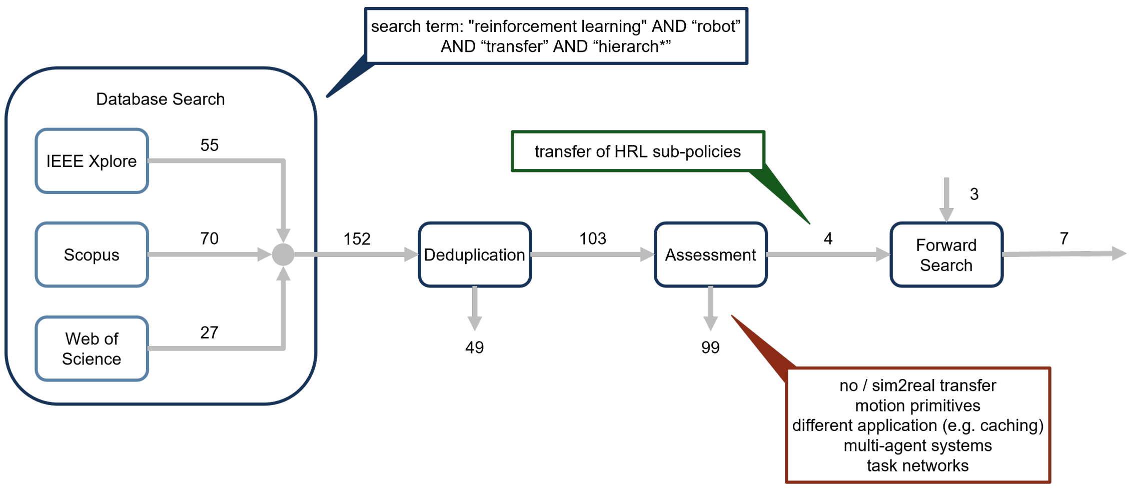


Figure 3.5 The literature survey process on the topic of cross-task transfer in HRL for RQ-3. 7 relevant articles were identified which investigate transfer of HRL sub-policies.

Approaches to HRL in general can be categorized into two groups according to the interface between the lower and higher level policies, summarized in Table 3.3. The two categories referred to as **Subgoal Spaces** and **Skill Embedding Spaces** are discussed in the following.

Subgoal Spaces	[17], [34]
Skill Embedding Spaces	[12], [163], [162], [152], [66]

Table 3.3 Categorization of articles by interface between HRL policy layers.

Subgoal Spaces are explicitly defined interfaces between HRL layers in the form of waypoints. Beyret et al. [17] show that for robot trajectories, using interpretable subgoal spaces offers explainability of the decision-making process in a HRL agent. Using target waypoints as subgoals, the authors are able to visualize and reason about the high-level strategy. Further, for any given state, the authors are able to produce interpretable heatmaps by querying the high-level state-action value function with hypothetical subgoals.

Christen et al. [34] ”argue that a key to transferability and generalization behavior in hierarchical RL lies in enforcing a separation of concerns across different layers” [34]. The authors separately train subgoal-conditioned controllers for manipulation and high-level planner modules for navigation, which are then combined. While the authors attempt transfer to different task variations, the experiments only test for zero-shot transferability, i.e. in case of failure the agents are not further refined to eventually solve the tasks. As less than 50% of these transfer attempts are successful, the claim of zero-shot transferability is questionable.

Skill Embedding Spaces are latent spaces representations which encode low level behavior and can thus be used as interfaces between HRL layers. Embeddings are learned from data and are thus relevant for scenarios where explicitly defining suitable subgoal spaces is not possible.

Baretto et al. [12] demonstrate the possibility to combine individual options, i.e. behaviors, through a weighting vector. While the authors rely on manually crafted options to be orthogonal to each other, the ability to express more complex behaviors through a finite set of base behaviors is a valuable insight.

To autonomously learn a skill embedding, Sharma et al. [163] propose the concept of unsupervised DRL. The authors provide a skill embedding to an agent, and simultaneously train an ANN to predict the next state from the current state and skill. The prediction accuracy is used as proxy reward for the policy, which thereby is incentivized to maximize the mutual information between the skill and resulting behavior. This results in a meaningful comprehensive skill embedding, which can be used by a high-level task-specific policy. In their follow-up work, Sharma et al. [162] expand this approach to an off-policy variant and demonstrate its effectiveness in a real-world robotic locomotion task.

Salter et al. [152] investigate the combination of HRL with KL divergence regularization in sequential task learning. The authors formulate an *expressivity-transferability trade-off*, which postulates that increasing the expressivity of learned skills enhances task-specific performance but inherently reduces their ability to transfer effectively across new tasks, while prioritizing transferability leads to more general skills at the expense of specialized task performance. The expressivity is hereby understood as an agent’s ability to distill knowledge, which is directly related to the number of policy input parameters, i.e. the state and goal. To reduce the parameters while retaining expressivity, the authors propose soft-attention mechanisms. In addition, a KL divergence regularization term is added to condition the current policy on policy instances from previous tasks to facilitate cross-task transferability.

Hao et al. [66] use offline demonstrations to inform the training of a skill embedding space between hierarchical layers. They use variational autoencoders to compress the demonstrations into a latent space, from which skill and action priors are derived. These priors are then used in subsequent HRL training by minimizing the KL divergence divergence between the priors and policy behavior. This allows for the incorporation of prior knowledge, while retraining flexibility in the policy to account for sub-optimalities in the demonstrations.

As seen from the distribution over the years in Figure 3.6, greater research focus both regarding volume and recency has been on learning generalizable skill embeddings. Skill embeddings provide the advantage of delegating the interface between hierarchical layers to the learning process, which in principle makes the approaches scalable to arbitrary application settings. However, reliably training transferable embeddings relies on statistical assumptions and is inherently difficult to verify due to a lack of interpretability of latent spaces. Therefore, the alternative approach of specifying subgoal spaces has its merits regarding interpretability and reduced learning effort due to provided structure.

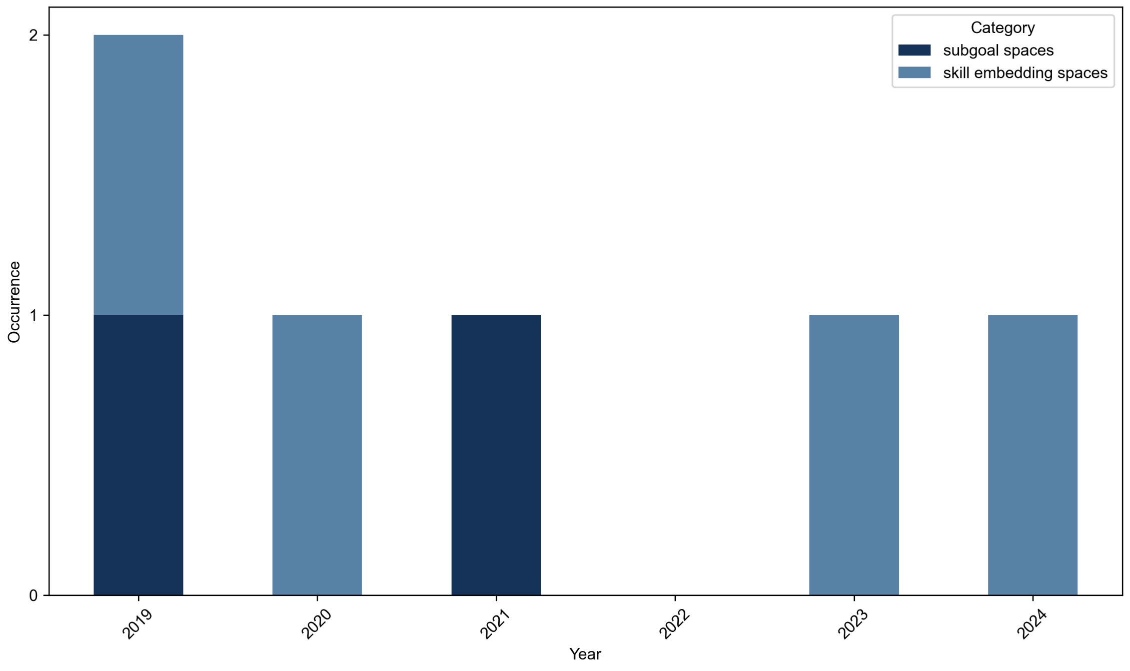


Figure 3.6 Distribution of strategies for cross-task transfer in HRL over the years. While the literature is sparse, skill embeddings have drawn more attention than explicitly defined subgoal spaces.

3.3.2 Assembly-by-Disassembly

The concept of assembly-by-disassembly is a well-known strategy in mechanical engineering with a long history of research [39]. The core motivation behind this concept is that the search space for finding an assembly trajectory may be much larger than for a corresponding disassembly trajectory. Intuitively, starting with a fully assembled object and removing parts one by one is often times much easier than figuring out the sequence and motions to assemble the object from scratch.

In this context, the following related work was identified using the search term *"reinforcement learning" AND "robot" AND "assembly" AND "disassembly"*. This resulted in a corpus of 21 articles, which was further filtered down to 4 articles relevant to the applicability of assembly-by-disassembly for DRL (cf. Figure 3.7). The majority of the articles were rejected due to only considering disassembly tasks, which were included in the query results as the mention of *disassembly* satisfies both keywords of the secondary search term *assembly* and *disassembly*. While this negates the original intention of the search term to identify works which explicitly focus on both assembly and disassembly settings, the search term was not adapted due to the small size of the raw corpus allowing for manual filtering. The selected 3 articles were further complemented with 3 additional articles in a forward search, which do not apply DRL, but were found relevant as capturing important aspects regarding robotic assembly-by-disassembly and thus to RQ-3. The resulting corpus is discussed in the following, separating the approaches into those which do not use DRL and those which do.

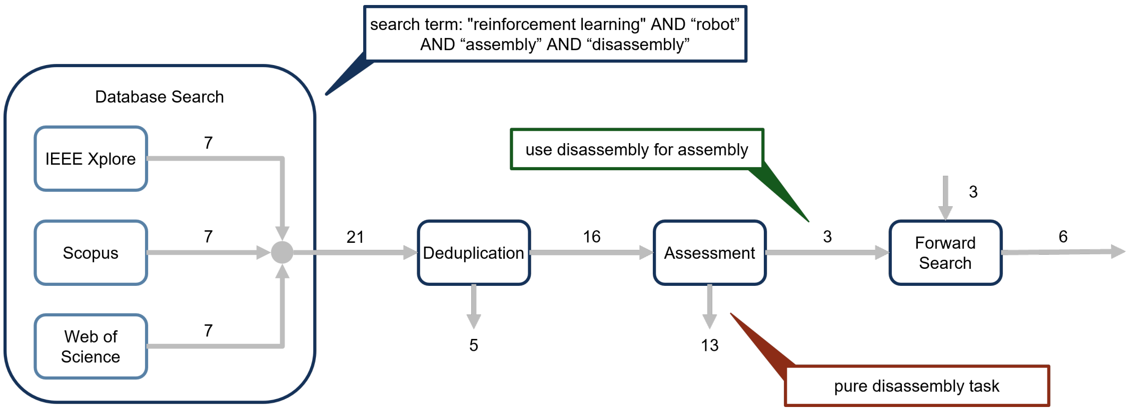


Figure 3.7 The literature survey process on the topic of assembly-by-disassembly for RQ-3. 6 relevant articles were identified which use the solution of a corresponding disassembly task to solve an assembly task.

no RL	[209], [182], [181]
with (D)RL	[167], [202], [166]

Table 3.4 Categorization of articles according to the application assembly-by-disassembly outside or within the context of DRL.

Assembly-by-Disassembly without RL

Zakka et al. [209] use disassembly as means for data collection. The authors describe an unsupervised disassembly process which utilizes an ANN to predict viable picking points for a robot arm. With the predicted picking points, they randomly pick up items and place them to the side. By reversing the recorded trajectories, the authors generate a dataset for assembling the kit, which is used to automatically reset the environment, as well as to learn the relation between items, their position in the kit, as well as the assembly sequence.

Tian et al. [182] utilize the CAD model of a multipart assembly. The authors use the assembled CAD model to successively remove parts using a hierarchical approach. In the outer loop, a breadth-first search algorithm attempts to find the sequence in which to remove parts from the assembly. The ability of removing an individual part is evaluated in an inner loop, where a breadth-first search algorithm over the action space. The identified trajectories are subsequently reversed and taken as-is for the assembly task.

In their follow-up work, Tian et al. [181] improve their approach by including feasibility checks. In particular, after each disassembly step, a physics simulation with the partially assembled construction is run to check if it remains stable. Thus, the authors are able to generate more realistic disassembly, and subsequently assembly, trajectories.

Assembly-by-Disassembly with (D)RL

Simonic et al. [167] decompose the disassembly task into a high-level graph representation and a low-level controller. The graph nodes represent waypoints from which multiple directions of motion are possible. The authors construct the graph dynamically, starting from the assembled state and applying random force actions to explore the state space and find new nodes. The trajectories between nodes are also saved as motion primitives, forming the edges in the graph. Once the graph is constructed, the authors are able to determine and invert the disassembly trajectory to generate a corresponding assembly trajectory. To account for potential errors during assembly, a retry mechanism performs partial disassembly in case the robot gets stuck. Notably, the authors only use RL in form of the SARSA algorithm to learn a high-level policy for finding the shortest path in the graph, i.e. the disassembly trajectory. The approach also assumes a finite set of actions to identify nodes, which is done by an inefficient brute-forcing of the action space.

Wantanabe et al. [202] apply the assembly-by-disassembly concept to determine the correct order of parts. Similar to Simonic et al. [167], the authors learn a disassembly sequence with DRL (Deep Q-Network (DQN)) which is subsequently inverted. While the authors motivate their work through a robotic assembly task, the contributions apply DRL to a simulated environment of stacked blocks, the actual assembly is performed using hardcoded poses for pick-and-place operations.

Simonic et al. [166] also revisit their previous work [167], highlighting the inefficiencies of the original approach to determine nodes through brute-forcing the action space. The authors instead propose a wall-following approach, where each path from a previous node is evaluated twice, respectively keeping contact with the left and right wall. A new node is added when these two trajectories diverge, indicating multiple possible directions of motion. The strategy of using RL for finding a disassembly trajectory and subsequent trajectory reversal to generate an assembly trajectory is identical to their previous work. Importantly, the authors also implicitly assume a 2-dimensional state space, which hints to a potential limitation of the approach regarding the applicability of the wall-following strategy to 3-dimensional spaces.

While assembly and disassembly tasks are heavily researched [141], the combination of assembly-by-disassembly and DRL is not yet widely explored. As evidenced by the literature survey, applications mainly focus on high-level sequence planning detached from the actual robotic manipulation. Also, the works that do use (D)RL employ simple algorithms such as DQN or SARSA, which assume discrete action spaces. One potential reason for this may lie in the direct competition with other approaches of knowledge transfer. Assembly-by-disassembly effectively requires successfully solving the disassembly task, which may be a challenging task in itself. Further, the approach relies on the reversibility of trajectories, which is not always guaranteed. In contrast, approaches which for example use human demonstrations may be more appealing to generate readily available trajectories to learn from. Nonetheless, as seen from the distribution over the years in Figure 3.8, assembly-by-disassembly has been a topic of interest since 2019, hinting to a potential future of more applications in the field.

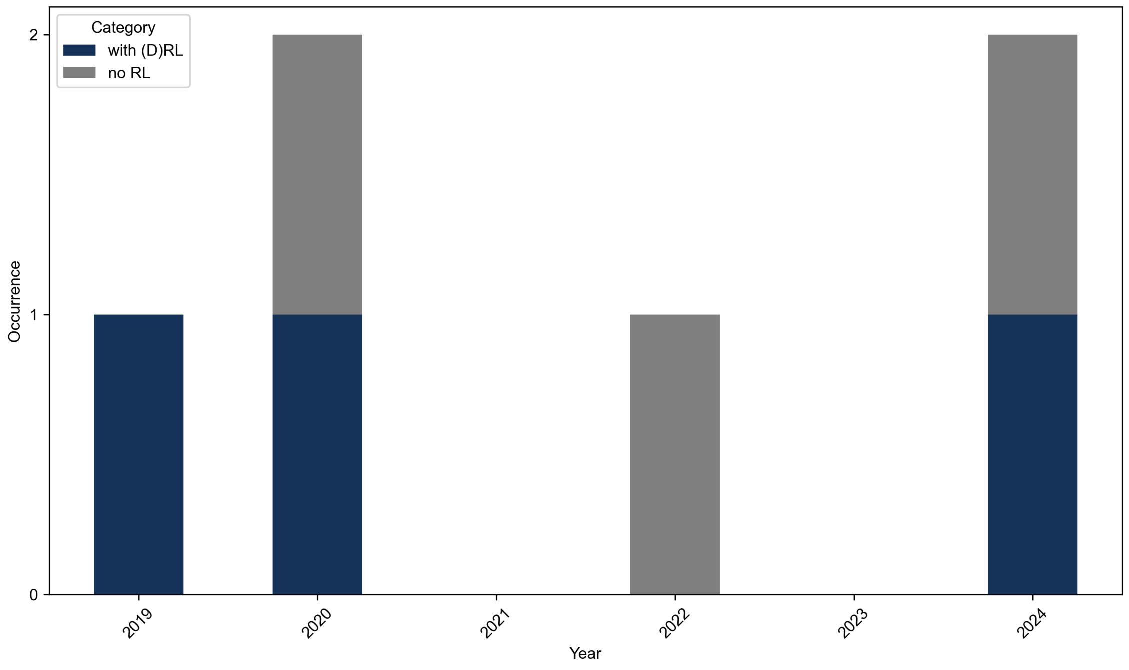


Figure 3.8 Distribution of the strategies for assembly-by-disassembly over the years. In particular the combination with DRL has not gained much attention.

3.4 Cross-Robot Transfer Reinforcement Learning

The general objective of TRL lies in the transfer of expert knowledge from the source domain to the target domain, which are both MDPs (cf. Figure 3.9). The differences between the source and target scenarios, i.e. the domain gap to be overcome, can occur in all MDP components, ranging from different distributions and dimensionalities of state and action spaces, to different dynamics and reward functions.

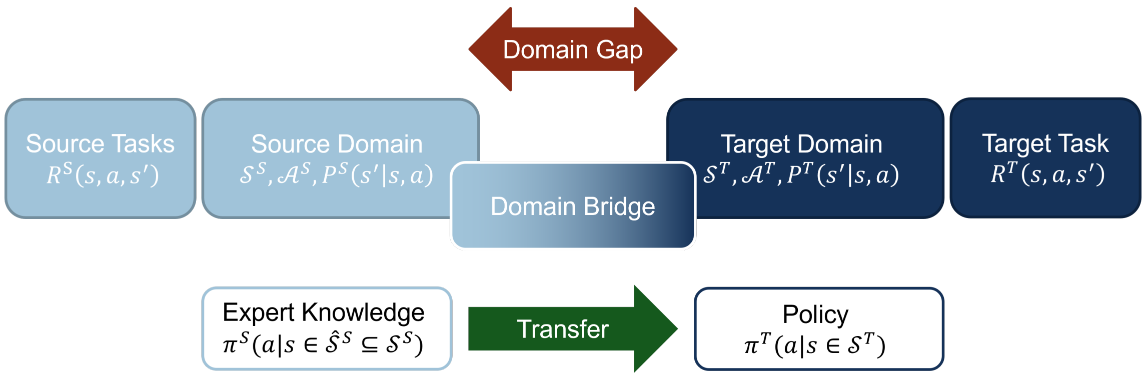

Domain Gap
Source Tasks $R^S(s,a,s')$
Source Domain $\mathcal{S}^S, \mathcal{A}^S, P^S(s'|s,a)$
Domain Bridge
Target Domain $\mathcal{S}^T, \mathcal{A}^T, P^T(s'|s,a)$
Target Task $R^T(s,a,s')$
Expert Knowledge $\pi^S(a|s \in \hat{\mathcal{S}}^S \subseteq \mathcal{S}^S)$
Transfer
Policy $\pi^T(a|s \in \mathcal{S}^T)$

Figure 3.9 Conceptual overview of cross-domain transfer learning. In order to transfer expert knowledge in the form of policies or demonstrations between two MDP environments, the domain gap must be bridged.

The following related work was identified using the secondary search term *"reinforcement learning" AND "robot" AND "transfer" AND ("cross*" OR "morpholog*" OR "embod*") AND NOT ("sim-to-real" OR "sim2real")*. The secondary search term aims to capture a focus on multiple robot models, which may be expressed by different terminology, such as "cross-robot", "cross-domain", different "morphologies", and different "embodiments". Wildcard operators are used to account for minor keyword variations. From this corpus, 8 articles were identified as relevant (cf. Figure 3.10). The corpus was further expanded with 16 articles in a forward search.

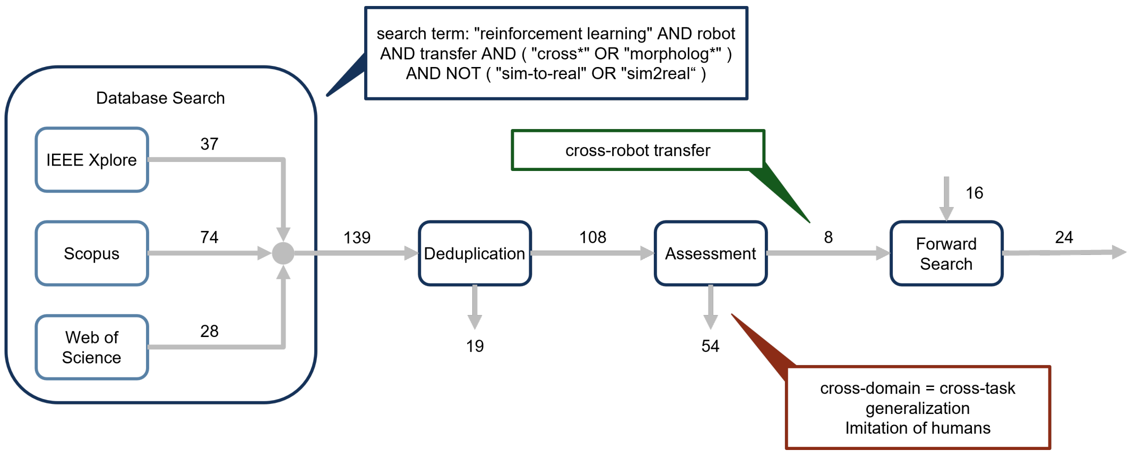

search term: "reinforcement learning" AND robot AND transfer AND ("cross*" OR "morpholog*") AND NOT ("sim-to-real" OR "sim2real")
Database Search
IEEE Xplore
37
Scopus
74
Web of Science
28
139
Deduplication
19
108
Assessment
54
cross-robot transfer
8
16
Forward Search
24
cross-domain = cross-task generalization
Imitation of humans

Figure 3.10 The literature survey process on the topic of cross-robot transfer reinforcement learning for RQ-4. 24 relevant articles were identified which propose cross-domain mapping strategies.

Acknowledging the vast imbalance between the number of articles identified in the forward search and the initial corpus, the validity of the search term must be reflected upon. A closer inspection of the forward search results revealed a high variability in the description of similar concepts. For example, "inspiration", "imitation" and "map" are used as synonyms for "transfer" [5, 11, 54, 84, 142, 196, 215]. This diversity in terminology suggests that the search strategy, while systematic, may not have fully captured the breadth of relevant literature. Expanding the search terms to include a wider range of related concepts could have yielded a more comprehensive initial corpus, albeit at the cost of significantly increasing the number of irrelevant articles. Nonetheless, the chosen approach provides a focused foundation for analysis without attempting to exhaustively cover every possible phrasing in the field.

The corpus was screened for articles explicitly considering transfer across different action spaces to focus on methods which are suitable for transfer across different robot morphologies. Upon review, four general mapping strategies were identified based on the type of assumptions made about the correspondence of the domains: **Hard-Coded**, **Finite Mapping Selection**, **Shared State Space**, **Correspondence Metrics**, and **Proxy Tasks**. The categorization of the articles is provided in Table 3.5, and the following sections discuss the approaches in detail.

Hard-Coded Mapping is arguably the most naive way to link two domains. This approach relies on the presence of explicitly formalized domain knowledge and is thusly not applicable for more complex scenarios. However, when possible, hard-coded mappings are by nature the most data-efficient approach, as no learning of mapping functions needs to take place. In 2005, Torrey et al. [187] demonstrate an

Hard-Coded	[187], [177]
Finite Mapping Selection	[105], [169], [176], [51]
Shared State Space	[11], [171], [215]
Correspondence Metrics	[5], [22], [4], [85], [84], [196], [54], [63], [214]
Proxy Tasks	[43], [64], [2], [142], [207], [14]

Table 3.5 Categorization of articles by strategy for cross-domain TRL.

early attempt at cross-domain transfer by manually designing a mapping function to translate state-action value functions between two low-dimensional robotic soccer tactics. Taylor et al. [177] also consider robotic soccer tactics and achieve transfer by providing mappings for states and actions from target to source domain, which allows them to reuse an existing state-action value function learned in the source domain.

Finite Mapping Selection strategies consider scenarios where the best mapping must be selected from a finite number of possible mapping functions. Like hard-coded mappings, these approaches rely on the presence of explicitly formalized domain knowledge and constitute early attempts at cross-domain transfer. The specific approaches measure the quality of each mapping function and select the best one. The specific approaches all share the same core idea of mapping entire transitions across domains, evaluating the next state from the mapped state and action, and finally comparing the determined next state to the mapped next state. The main difference lies in evaluation of the dynamics, which can either be done with a learned dynamics model [105, 176] or through online interaction with the target environment [51, 169].

Shared State Space strategies exploit an explicit link of domains via a shared state space, which in literature is predominantly used to derive pseudo rewards from source domain state trajectory demonstrations. Baram et al. [11] train a binary classification model to discriminate between source and target domain states, and use the classification output as pseudo-reward for the target domain agent. Srinivas et al. [171] project states into an embedding space and demonstrate that the distance between the embeddings of a state and a goal state can be used as a pseudo-reward across domains. Zolna et al. [215] expand on this approach and test different discriminator inputs to capture more features of a state trajectory, such as using two subsequent states to include the transition dynamics. Most notably, the best results are achieved when fading out the pseudo-reward over time. As the authors highlight, solely relying on the pseudo-reward may degrade the performance of the target domain agent. Especially in settings where the source and target dynamics differ, the discriminator is able to differentiate between the two domains with high accuracy, degrading the pseudo-reward signal.

Correspondence Metrics strategies aim to use assumptions of domain similarities to match trajectories. One approach is to use such similarity metrics as pseudo-rewards for imitation learning. Von Eschenbach et al. [196] explicitly link two domains by formalizing a similarity metric as the distance between key points along robot kinematic chains. Besides the pseudo-reward approach, the authors also train a ANN to map between the two domains by minimizing the distance metric. Instead of explicitly formalizing a similarity metric, statistical assumptions can be used to infer correspondences. Fickinger et al. [54] propose a Wasserstein-based distance metric between the state-action distributions of both domains to be used as pseudo-reward. This allows the authors to transfer behavior across domains where an explicit metric is not easily defined.

Under the assumption of statistical similarities, unsupervised learning and manifold alignment approaches can be used to project transitions from both domains into a shared embedding space. To achieve this, different methodologies were proposed, from restricted Boltzmann machines [5], over Principal Component Analysis (PCA) projections [22], to invertible manifold alignment [4]. The quality of mappings can further be improved by mapping the transitions back and forth and introducing cycle-consistency losses [85, 214], as well as adversarial discriminators that provide additional regularization [63, 84].

Proxy Tasks are simple tasks which are solved in both domains to generate trajectory datasets which can be assumed to be corresponding. Primarily, this matched data is used to improve on the previous unsupervised approaches through supervised learning, such as directly learning mapping functions [207]. Alternatively, the matched data can implicitly be used, as shown by Raychaudhuri et al. [142], who extend previously discussed unsupervised correspondence metric strategies by adding additional regularization losses based on the samples position in the respective proxy task trajectory to further align the embeddings.

Instead of mapping the trajectories back from the latent space, the embedding can be used directly for cross-robot transfer by deriving pseudo-rewards [64], or as input for different motion generation approaches [2]. As an example of the latter, Akbulut et al. [2] use the latent space as input to a decoder which generates motion primitives.

The ability to compress behavior into latent spaces has further been used to create skill embeddings with similar intent to the previously discussed HRL strategy of skill embedding spaces (cf. 3.3.1), the difference being that the following approaches do not apply the core HRL concept of temporal abstraction. Beaussant et al. [14] show that once a robot-agnostic latent space has been created, policies can directly be trained in the latent space, allowing for modular transfer across both robots and tasks. Devin et al. [43] demonstrate similar modularization of policies into task- and robot-specific modules by training on different task-robot combinations and using the respective module combinations as policies. This allows the authors to eventually arrive at a robot-task-independent interface and flexible recombination of the modules.

With regard to the distribution of appearance in the literature corpus depicted in Figure 3.11, hard-coding or selecting from finite sets of mappings were strategies of choice in the early years. With increasing complexity of application domains, the focus naturally shifted to more sophisticated approaches. While correspondence via shared state spaces were briefly explored in 2018 and 2019 in the context of different dynamics or shared visual state spaces, the special-case nature of the shared state space assumption resulted in a shift towards more generalized strategies using correspondence metrics or proxy tasks. Both approaches have complementary strengths and weaknesses. While for the use of proxy tasks online access to both domains poses a strong requirement, the availability of matched trajectories explicitly provides the required correspondence signals and is thusly the safer approach. In contrast, statistical approaches offer greater flexibility and scalability by inferring correspondences from distributional similarities between domains, sometimes without requiring explicit trajectory matches. This makes them particularly appealing when direct access to paired data is impractical or unavailable. However, these methods can be sensitive to domain shifts and rely on the correctness of statistical assumptions, potentially leading to suboptimal or unstable transfer performance. Ultimately, both strategies are expected to continue co-existing in the field and may be combined to achieve the best of both worlds, as demonstrated by Raychaudhuri et al. [142].

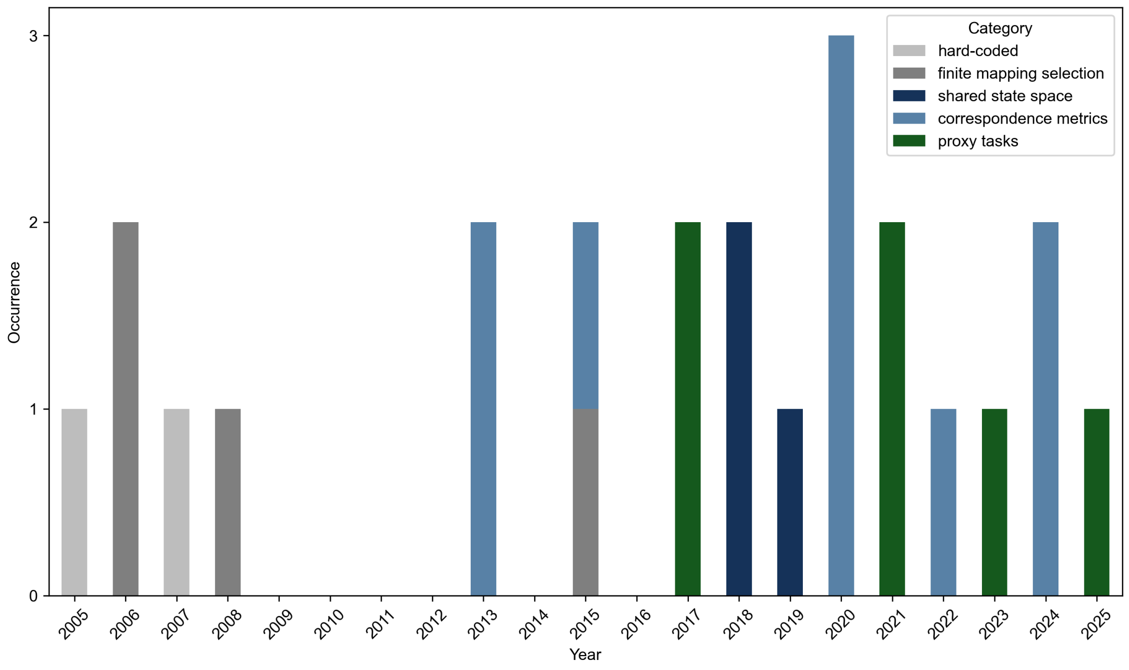


Figure 3.11 Distribution of strategies for cross-robot transfer reinforcement learning over the years. Using explicit mappings is the most data-efficient approach, but has been largely abandoned in favor of statistical methods and proxy tasks.

3.5 Research Gaps and Focus

The literature surveys provide the necessary context for addressing the RQs of this thesis in the upcoming chapters. For the demand of real-time control as demanded by RQ-1, literature provides a clear answer in the form of asynchronous DRL. In the spirit of modularization, the approach of dynamics models promises a clear separation between accounting for latencies and non-delayed decision-making. Motivated by recent advances in the field of CV regarding image generation, this thesis will focus on achieving DL-based dynamics rollout for vision-based state spaces.

With respect to RQ-2, the literature provides a well-established strategy for sim2real transfer in the form of DR. As discussed, numerous works have investigated the ability to measure the sim2real gap and improvements on the randomization parameter selection. Whereas previous works pretrain and transfer entire agents, this thesis will investigate how visual perception modules can be effectively be pretrained in isolation. A particular focus will be placed on scenarios where the interface between perception module and decision-making policy is not well-defined. To this end, this thesis will consider unsupervised learning approaches complemented with the injection of human domain knowledge to extract meaningful latent representations.

For the structure exploitation of robotic movement prompted by RQ-3, HRL and assembly-by-dissassemby are provided by literature as promising methodologies. For the former, it was found that explicit subgoal spaces have been established as suitable interfaces, however have been largely neglected in favor of the alternative skill embedding strategy. Arguing that explicit subgoal spaces are of high relevance for industrial applications due to their inherent interpretability promoting modularization, this thesis will revisit this strategy and provide further investigations into the cross-task transfer of low-level HRL policies. With respect to assembly-by-disassembly, the concept will be directly applied to position control agents for robotic systems, rather than high-level sequence planning as in previous works. To this end, the generated disassembly trajectories will be inverted and used for pretraining assembly policies with BC.

While the two strategies of statistical correspondence and proxy tasks have been identified as dominant in the research field on cross-robot TRL, this thesis will reconsider the seemingly discarded strategy of finite mapping selection for RQ-4. Instead of assuming no prior knowledge about the domains and inferring correspondences purely from data, this thesis acknowledges that in industrial robotics kinematics models are often available and may be used to link the domains and proposes the use of FK and IK models to translate trajectories between robot morphologies. To address the previously mentioned ambiguity of IK, a correspondence metric inspired by the previously identified work of von Eschenbach et al. [196] will be developed to select from the (finite) set of IK solutions.

4

Framework and Baselines

"Experience is the teacher of all things."
- Julius Caesar

As first step towards addressing the RQs of this thesis, baseline performances of standard DRL and TRL approaches in the three use cases must be established. In the following, a generalized DRL architecture is presented, which is applicable to all three use cases. Parts of the learning framework are based on the author's publications [21, 157]. Subsequently, for each use case, respective environment designs and experimental results are discussed. For the wire-loop use case, the experiments in the real world and in simulation are based on the author's publications [114] and [154], respectively.

4.1 Learning Framework

To train DRL agents, a suitable framework is required. While to-date a variety of sophisticated DRL frameworks exist [98, 99, 137], the framework landscape at the start of this thesis did not include a suitable framework for the needs of this thesis regarding stability, feature completeness, ease of use, and extensibility. To avoid risks associated with early-stage open-source projects, such as changing interfaces or project discontinuation, the decision was made to develop a custom DRL framework. Examples of frameworks which were identified as promising, but were since discontinued are OpenAI baselines [44] and rlpyt [173].

For the DRL framework, the following requirements were identified:

- **Modularity and Extensibility**: To support a wide variety of applications, first and foremost given by the different use cases, the framework must be modular. In particular to facilitate the extension by use-case specific features and adaptations warranted by the RQs to be addressed, the framework must be designed with general interfaces. The interfaces shall enable reusability of common functionalities across different use cases, while at the same time allowing for maximum flexibility in the implementation of use-case specific features and adaptations.
- **Environment Parallelization**: To reduce the training time of DRL agents, the framework must support environment parallelization for faster experience collection. While not applicable to the real-world instances of the use cases, this requirement is particularly important for any experiments and pretraining performed in simulation.
- **Distributed Infrastructure**: In most industrial settings, a robotic system is controlled by an on-site edge computer with limited computational resources. DRL however requires significant computational resources for ANN training, which is typically performed on a Graphics Processing Unit (GPU). While it is possible to integrate a GPU into the edge computer, this significantly increases the cost of the overall setup due higher hardware requirements regarding robustness in an industrial shop floor environment. To address this issue, the framework shall support a distributed infrastructure, in which execution on the edge computer can be decoupled from the training process on a remote GPU cluster.

The resulting framework architecture is depicted in Figure 4.1. In the following, different design decisions and implementation details are highlighted. At the root of the framework is the **Manager** process, which is responsible for coordinating the training loop, i.e. resetting environments, querying the agent for actions and relaying them to the environments, and compiling experience trajectories. In settings with distributed infrastructure, the **Manager** runs on the edge computer to minimize latencies to the environments and maximize data collection efficiency. To achieve the desired goal of decoupling the agent training from the robotic execution, the design choice was made to implement the **Agent** class with a *get_policy* function, which is used by the **Manager** to download the latest trained policy weights for local inference.

To satisfy the requirement for environment parallelization, the **Orchestrator** is implemented to manage the interaction between the training loop and the parallelized environments. An asynchronous parallelization strategy is chosen (cf. chapter 2.3.4) to maximize the interactions with the environments. To this end, the **Orchestrator** maintains a bidirectional communication channel to each process for sending execution commands and receiving resulting data. The communication follows a request-response message exchange pattern, where each command by the **Orchestrator** is answered by exactly one result message by the respective environment process. This ensures, that the processes act in a predicable, well-defined fashion. From an interface perspective, the **Orchestrator** provides functions to send and receive

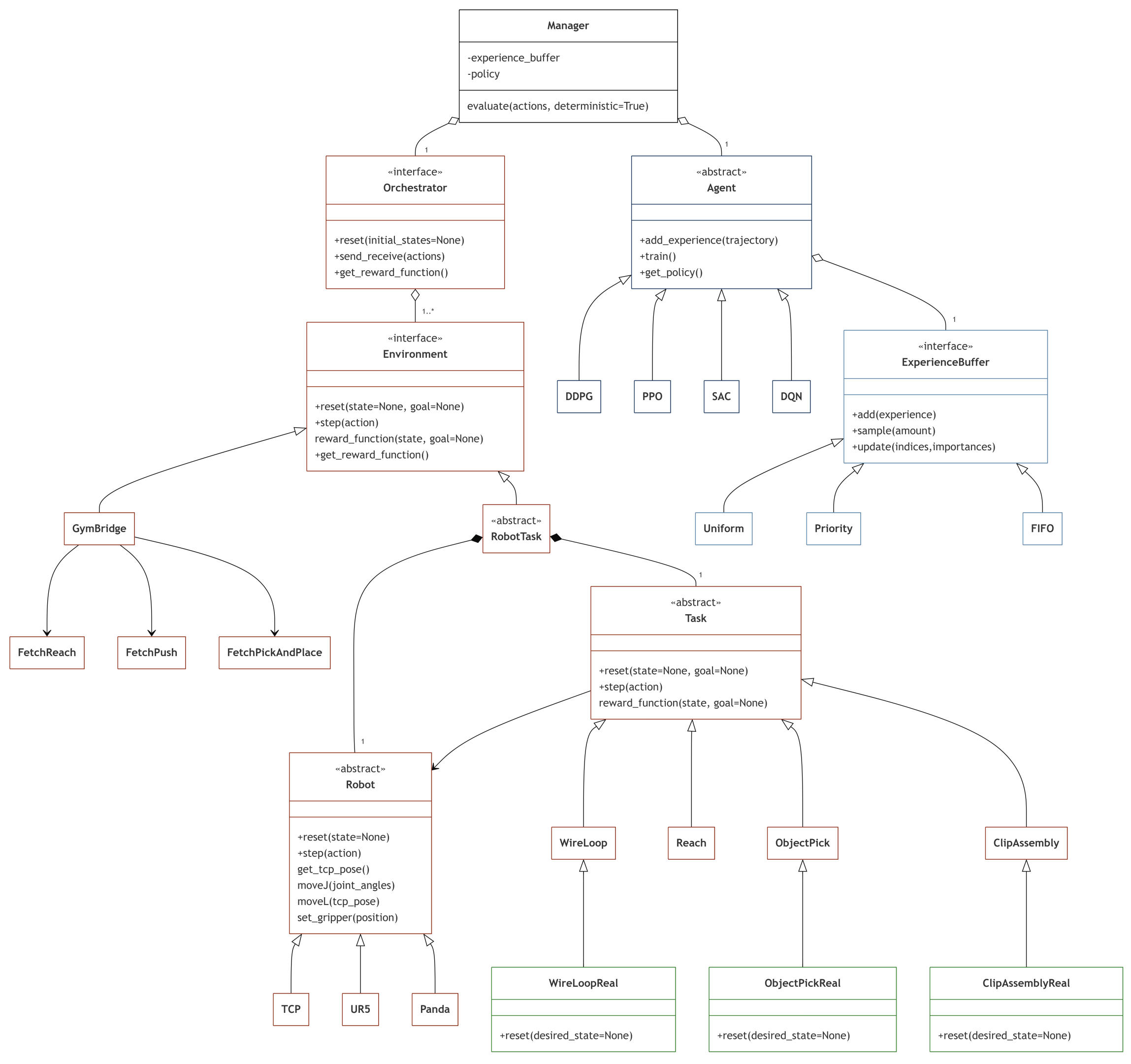


Figure 4.1 Software architecture of the DRL framework designed for modularity, environment parallelization, and distributed infrastructure.

information to and from the environments, which are usually executed sequentially and can thus be combined into a send_receive function. The messages send to and received from this function must be tagged with an environment ID, in order to send commands to the correct processes, as well as compile received data into valid experience trajectories. These IDs are also used by the **Manager** to maintain dedicated experience buffers for each environment.

The effectiveness of using environment parallelization for experience collection is depicted in Figure 4.2. In this benchmark experiment, different numbers of parallel environments are used to perform 1000 reset operations, while the wall clock time taken is measured. As expected, the durations significantly drop with increasing parallelization.

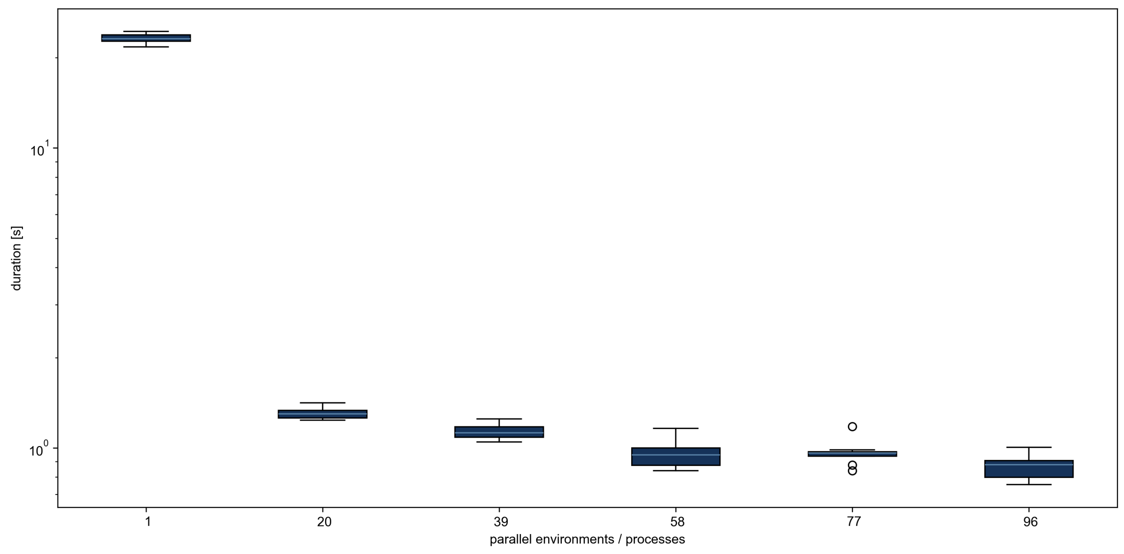


Figure 4.2 Orchestrator benchmarking. The implementation effectively distributes experience collection across multiple processes. Adapted from [21].

To enable the implementation of HER, three design choices were necessary. While not a direct requirement for the framework, supporting common optimization techniques such as HER can be understood as ensuring the extensibility of the framework. The first design choice is to support goal-based environments in the **Environment** interface, whereby standard non-goal-based environments are considered a special subset with $goal = None$. The second design choice is to buffer entire trajectories in the **Manager** to retain the temporal structure of the experience. This allows the **Agent** to apply post-processing techniques such as HER before splitting the trajectory into independent experience samples. The third design choice considers the need of HER to retroactively modify the reward of an experience sample, which requires access to the reward function of the environment. While it is possible to expose the reward function in the **Environment** interface, the resulting communication overhead of sending vast amounts of generated experience tuples from the **Agent** over the **Manager** to the **Environment** is prohibitive. Instead, the choice was made to serialize the reward function in the environment and synchronize it once with the agent at startup. This way, the environment always holds the single source of truth for the reward function to account for rapid prototyping, and communication is reduced to a single exchange.

Realizing environment-specific techniques is achieved via a generic *config* parameter in the reset functions of the environment hierarchy, which allows the **Manager** to pass additional configuration parameters, such as the randomization probability for DR.

The final major design choice is the modularization of environments into **Task** and **Robot** classes. On the one hand, the **Robot** instances are solely responsible for the robotic manipulator, translating actions into movement. The **Task** instances on the other hand contextualize the robotic manipulator in its surroundings by providing task-specific goals and reward functions and managing the sensory equipment. Simulated environments require the **Task** to supply surrounding elements, including manipulable objects and obstacles. For real-world scenarios, the **Task** might also be responsible for implementing necessary reset routines. The modularization allows for the reuse of robot-specific implementations across different scenarios, as well as facilitating the utilization of the framework for cross-domain TRL experiments by enabling fast composition of different task-robot combinations.

4.2 Use Case 1: Wire-Loop Game

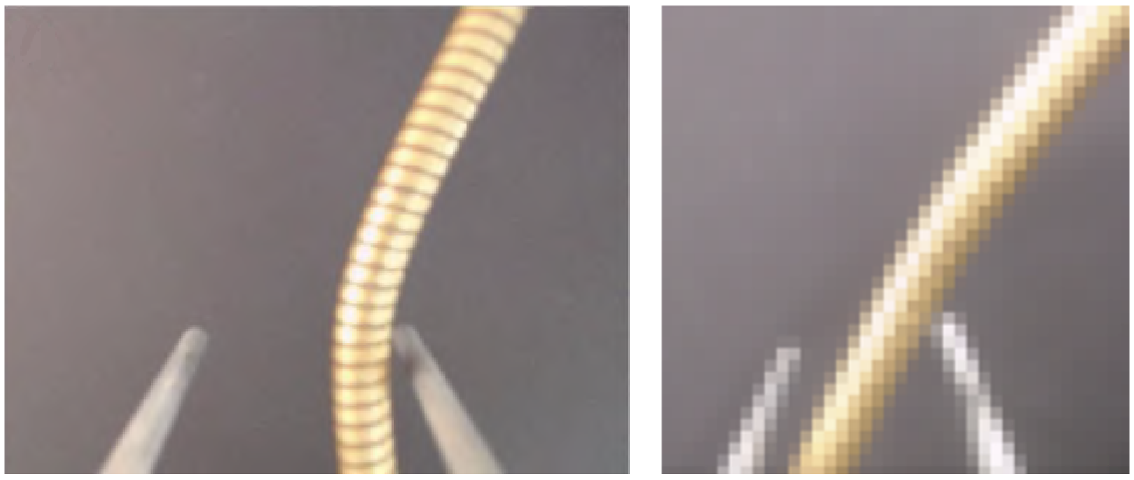
The wire-loop use case poses a custom scenario for which no suitable simulation benchmark environment or comparative studies are readily available. The automation of the use case with DRL is approached in two stages. First, the real-world setup is considered to establish a baseline performance to be used as a reference for any subsequent investigations into optimizations. Since the direct application of DRL to the real-world setup is expensive in terms of time and required manual supervision, the second stage focuses on the development of a corresponding simulation environment and the application of DR for sim2real TRL.

4.2.1 Environment Design

This section introduces the environments for the wire-loop use case. First, the real-world environment is discussed, followed by the simulation environment.

4.2.1.1 Real-World Environment

Given the hardware setup outlined in chapter 1.3.1, the first step is to define the state space. As the state must include all necessary information for an agent to make an informed decision, the state space must include all relevant information about the current game state. Intuitively, at a given position of the loop, only the next few centimeters of the wire are relevant, which are captured as 1920x1080x3 high-resolution images by the camera mounted on the loop. To reduce the dimensionality of the state space with the goal of improving learning efficiency, the images are downsampled to 40x40x3. This downsampling factor was manually chosen to reduce the state space dimensionality while still retaining the necessary features in the image, i.e. the wire and the loop (cf. Figure 4.3).



(a) Raw camera image. Taken from [114]. **(b)** Downsampled state. Taken from [154].

Figure 4.3 The states in the wire-loop use case are downsampled camera images of the loop and the wire. The downsampling factor was chosen to reduce the state space dimensionality while still retaining relevant image features.

As the state space captures a local view of the wire with no information about the global position of the loop, an action space is chosen which describes the motion of the loop relative to the current pose. Since the wire-loop game is constrained to a 2D-plane, the loop can be moved with three Degrees of Freedom (DOF), two translational and one rotational. This results in a three-dimensional action vector, which describes the forward and sideways motion as well as the rotation of the TCP relative to the current TCP frame. To ensure that the loop does not move out of the camera's field of view, a maximum translation of 3 cm in each translational DOF is chosen. For the rotational DOF, a maximum rotation of 90° is chosen, since any larger rotation would result in the loop rotating into the wire and losing the game in virtually all situations. Another restriction is taken by explicitly excluding backward motion, which is accomplished by constraining the first dimension of the action vector to be non-negative. This decision is made on the domain insight that regressing on the wire is neither required, nor beneficial for solving the game. In fact, any attempt to construct situations which can only be resolved by moving the loop backwards could always be resolved through a combination of sideways motion and rotation.

The final component required to formalize the environment is the reward function, which is defined as follows:

$$r_t = \begin{cases} -1 & \text{if collision or } t > t_{max} \\ 1 & \text{if goal or checkpoint reached} \\ \beta \cdot a_t^{forward} - (1-\beta) \cdot \frac{t - t_{cp}}{t_{max}} & \text{otherwise} \end{cases} \tag{4.1}$$

The derivation of the reward function is outlined in the following. First and foremost, the reward must reflect the objective of the wire-loop game, which is to guide the loop from the start to the goal without touching the wire. As a starting point, these game rules are translated into a sparse reward function, where the reward is +1 if the goal is reached, -1 if the wire is touched, and 0 otherwise. While this reward function holistically encodes the game rules, training an agent with this reward function is expected to be challenging due to the long horizon of the task. Considering the wire length of 150 cm and the maximum translation of 3 cm, approximately 50 steps are required to solve the game. Reaching the goal through exploration, i.e. choosing at least 50 random actions which do not lead to a collision is highly unlikely. Instead, the agent would only observe rewards of -1 and 0, and would likely deduce that the objective of the game is to avoid the wire. While this is partially correct, from the agent's perspective a valid strategy to maximize the reward would be to minimize the risk of collision by not moving at all. Therefore, additional incentives must be added to the reward function.

On the one hand, the agent can be incentivized to make progress by exploiting the domain knowledge that moving forward is beneficial for reaching the goal. This is achieved by rewarding the magnitude of the forward motion $a_t^{forward}$. On the other hand, the agent can be incentivized to progress along the wire by rewarding the distance covered. While the overall objective is to get closer to the goal, or vice versa farther away from the start, it is important to realize that constructing a reward based on those distances may lead to suboptimal behavior. Since the wire can be arbitrarily shaped, it is common that the loop will need to move backwards on the game plane, i.e. towards the start pose. If this were to be punished, the agent would be wrongly discouraged from the optimal solution. To solve this issue, checkpoints are introduced along the wire to measure progress.

As the unknown arbitrary wire shape makes it impossible to statically define the checkpoints on the game plane, they must be defined dynamically. To this end, the next checkpoint is considered reached if the loop has moved at least l_{cp} from the last checkpoint, where $l_{cp} = 5\ cm$ was chosen with the following rationale: The distance of 5 cm was chosen to be sufficiently small to ensure that an agent receives frequent positive feedback. At the same time, it is far enough to discourage reward hacking. Reward hacking is the term used to describe a situation where an agent learns to exploit the reward function design to maximize rewards through undesired behavior. In the case of the dynamically checkpoints, such reward hacking would occur if the agent would move backwards a distance of l_{cp}. While moving backwards is already restricted by the action space design, it is still possible for the loop to move backwards through a combination of sideways motion and rotation, which was referred to as tacking due to the similarity of a sailboard moving against the wind in a zigzag pattern. Without the restriction of the wire, the agent could rotate 90° and move sideways with 3 cm, hypothetically collecting the checkpoint reward with two steps. While this worst case behavior will not occur with the wire inside the loop, smaller backwards regress is still possible. It was estimated that choosing $l_{cp} = 5\ cm$ would require an agent to realistically perform more than 3 steps to successfully tack in backwards direction. In addition, these steps would need to be precisely executed and would be hard to learn, as the state does not include any information about the wire shape behind the loop. Thus, the risk of reward hacking is deemed negligible.

The concept of checkpoints allows introducing a timeout if the next checkpoint is not reached within a given amount of time steps t_{max}, which can be used to terminate an episode if the agent fails to make sufficient progress, which is also punished with a reward of -1. The timeout was chosen to be $t_{max} = 10$,which purposefully is not too restrictive in case a situation occurs where the agent needs to move precisely with smaller actions. To further incentivize the agent, different additional reward structures are possible, such as positively rewarding distance from the last checkpoint, or negatively punishing the time to reach the next checkpoint. In this case, the latter option was chosen as minimizing the time between checkpoints is always valid, while maximizing the distance from the last checkpoint may lead to unintended behavior analogous to the issues discussed above for using the distance from the start pose as reward.

With the above considerations, the final reward function is constructed (cf. Equation 4.1). In this reward function, $\beta \in [0, 1]$ is a tuning parameter to balance between the two reward components of maximizing forward motion and minimizing the time to reach the next checkpoint.

The checkpoints also allow an optimization in the reset procedure, as the loop can be reset to the last checkpoint position instead of the start pose. This allows the agent to continue learning on the wire section which it is currently struggling with, instead of repeatably having to traverse the beginning of the wire which it already knows how to navigate.

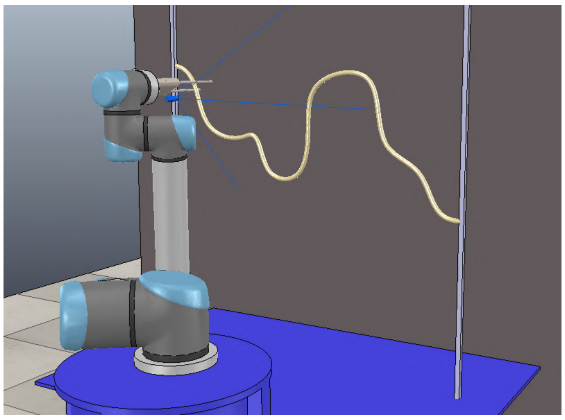
4.2.1.2 Simulation Environment

For constructing the simulation environment, the simulation software CoppeliaSim [145] is selected due to four main reasons. First, CoppeliaSim is specifically targeted at industrial robotics simulation and features a large library of pre-built robot models, including the UR5 robot model used in the wire-loop use case. Second, a distributed control architecture enables the simulation of multiple robots, which are controlled independently of each other, allowing for the parallelization of environments. Third, CoppeliaSim features a remote API for Python clients, which is vital to integrating the simulation environment with the existing Python-based learning framework. Finally, an educational license allows free and unrestricted use in the academic context of this thesis.

The simulation setup is depicted in Figure 4.4 and features the same UR5 robot model as the real-world scenario, a 3D model of the loop attached to the robot end-effector, and a camera mounted to the loop object. In front of the robot, a 3D model of a wire is placed, behind which a 2D background plane models the black cardboard sheet. With this setup, the resulting simulated camera image is displayed in Figure 4.4b.

To generate the state images, it is possible to set the simulation camera to take pictures directly in the target resolution of 40x40x3. However, as seen in Figure 4.5, these directly sampled images are very different from the real images. To achieve more realistic state images, the same down-sampling and interpolation method is applied to the simulation images as in the real world, which produces simulation states (cf. Figure 4.5) that are more similar to the real states (cf. Figure 4.5a).

In accordance with the methodology of DR, multiple parameter randomizations are implemented. First, the colors of the wire, the loop, and the background can be independently set to arbitrary RGB values. The textures of each item can also be independently varied, choosing from a set of predefined textures. As compiling the texture set required a manual processing step, the number of textures was limited to 20 to balance manual effort with required visual diversity. Additionally, the camera


(a) Overview of the simulation setup. Adapted from [154].

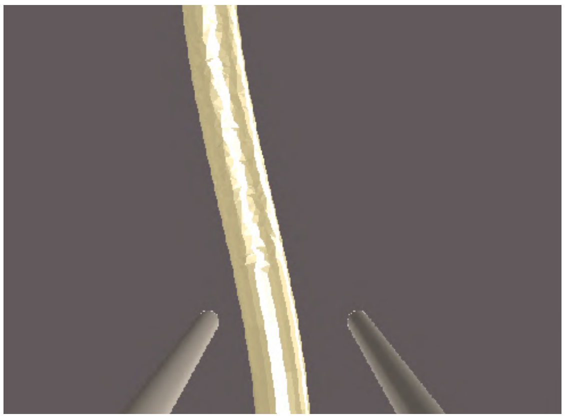
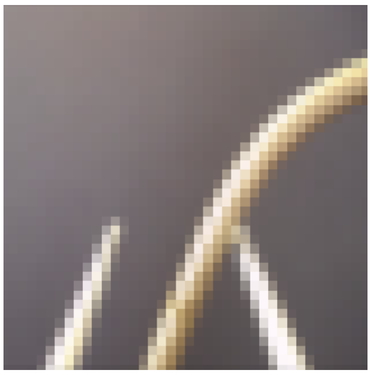
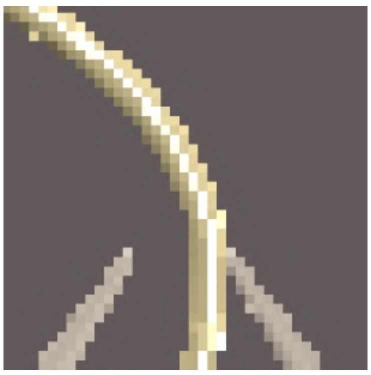

(b) Example of a simulated raw camera image (640x480x3).

Figure 4.4 The simulation setup for the wire-loop use case. The simulation is modeled after the real-world setup, featuring a UR5 robot, a loop tool, a camera, and a background plane for visual projection.


(a) Real-world image after down-sampling and interpolation.


(b) Simulation image with direct downsampling.

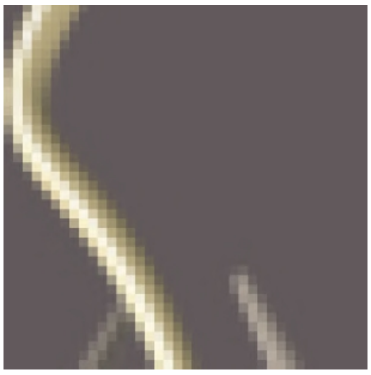
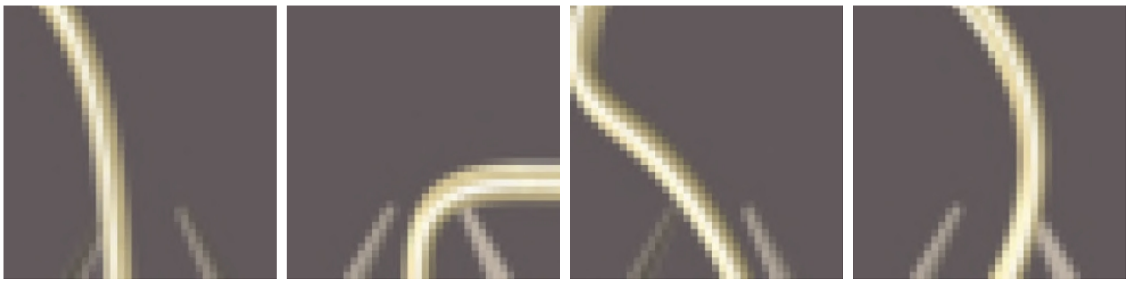
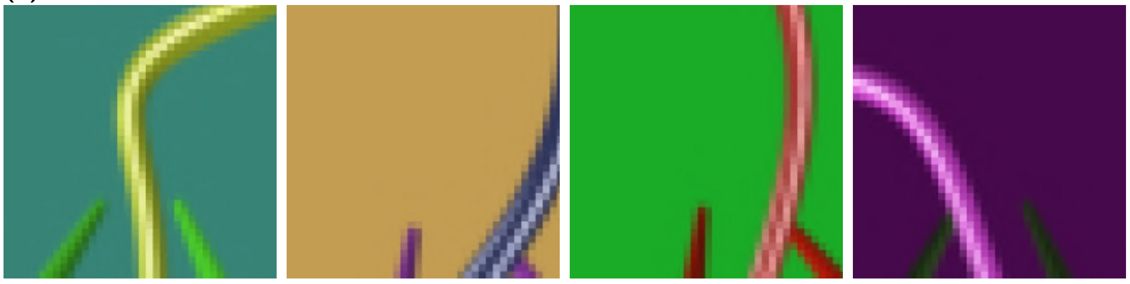
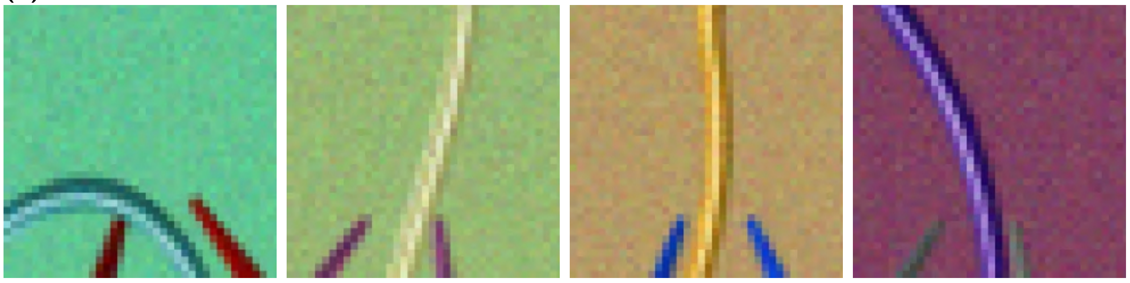

(c) Downsampled simulation image. Taken from [154].

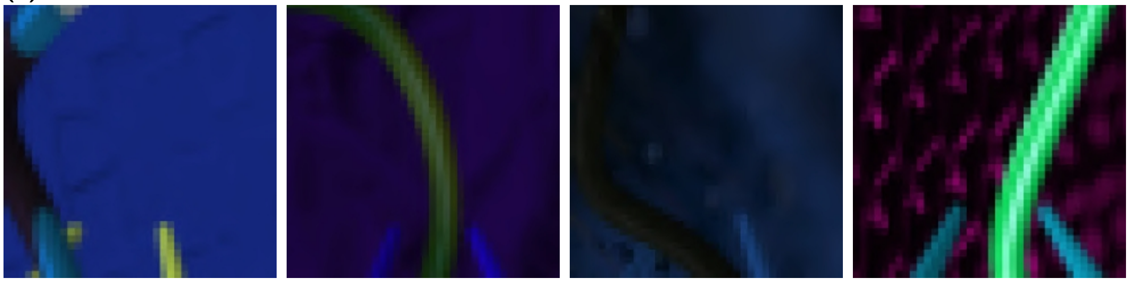
Figure 4.5 Comparison of real-world and simulation state images. To achieve more realistic state images, the same down-sampling and interpolation method is applied to the simulation images instead of directly capturing images at the target resolution.

can be randomly moved around the loop object, allowing for a displacement of up to 1.5 cm and a rotation of up to 5 degrees in each degree of freedom. Finally, random Gaussian noise can be optionally added to the state images. Other parameters, e.g. lighting conditions, were excluded due to lack of visible effect on the state images. Examples of the randomized states can be seen in Figure 4.6.


(a) No randomization.


(b) Randomization of colors.


(c) Randomization of colors and noise.


(d) Randomization of colors and textures.

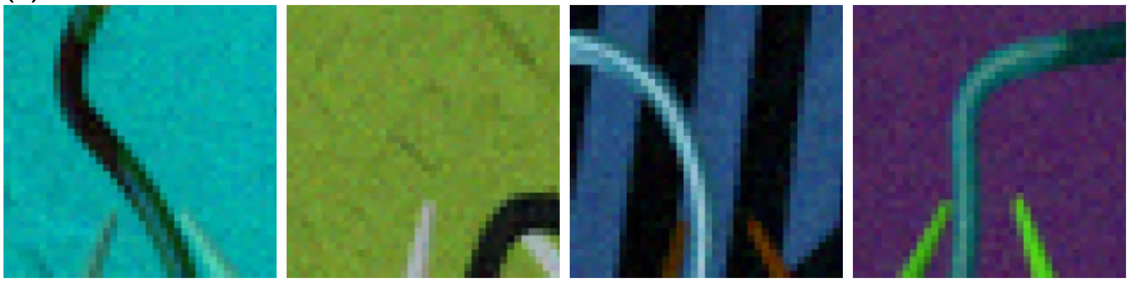

(e) Full randomization of colors, textures, and noise.

Figure 4.6 Examples of the randomization strategies applied to the wire-loop use case.

4.2.2 Agent Design

With the environment design and learning framework in place, the final component to be designed is the DRL agent. As the space of possible design choices is virtually infinite, a trial-and-error process was conducted, heavily relying on educated guesses based on best practices and reports from literature. In the process of arriving at the final design, the following considerations were explored:

1. **Agent architecture**: the final agent design was found through an iterative process, starting from an initial oversized architecture, which was then continuously reduced with observable performance improvements until arriving at the following final design. Given the image-based state space design, a CNN architecture was selected. The CNN architecture was chosen to be three layers of 30, 15, and 10 filters of size 5x5, 5x5, and 3x3, respectively. Each CNN layer is followed up with max-pooling of size 2x2. This architectural design of gradually reducing the feature dimensionality is standard practice in literature to extract a compact representation of the input image. For the actor, this CNN is followed by a MLP layer of size 200, which in turn is followed by three separate MLPs with each two layers of size 200 and 1. The three outputs are concatenated to form the action vector. The critic uses the same CNN model as the actor, feeding the CNN output into a MLP with three layers of size 200, and 1. The input of the second layer is concatenated with the three-dimensional action vector to allow the critic to represent the state-action value. In both actor and critic architecture, the fully connected layers use Rectified Linear Unit (ReLU) activation functions and a dropout rate of 0.2 is used to prevent overfitting.

2. **Algorithm + action space**: the initial choice of DRL algorithm was DQN, which uses discrete action spaces. This simplification led to faster training times, however the discretization also limited the precision of the agent, in addition to introducing a dependence on manual tuning of suitable actions. Therefore, the decision was made to switch to a continuous action space and DDPG as suitable learning algorithm.

3. **Reward function**: it was observed that selecting a tuning parameter $\beta = 1$ led to a performance decrease, indicating that explicitly rewarding the forward motion helps the agent to identify beneficial behavior faster.

4. **Dropout rate**: any experiments without dropout were unsuccessful, however no optimization of the dropout rate beyond the default value of 0.2 was conducted.

5. **State space**: in contrast to the sole down scaling of the camera image, different image preprocessing techniques were tested, such as grayscale conversion and cropping. The objective of these efforts was to reduce the dimensionality, as well as improve the signal-to-noise ratio of the state space. While the results showed as anticipated a clear benefit in the reduction of training time, a failure in certain scenarios was observed. One example is that when encountering bottlenecks where two wire sections are close to each other (cf. scenario 4 in the subsequent experimental study), cropping the image introduces critical blind spots, which make it impossible for the agent to precisely navigate. Thus, it was decided to provide the agent with as much raw information as possible and delegating the identification of relevant features to the learning process.

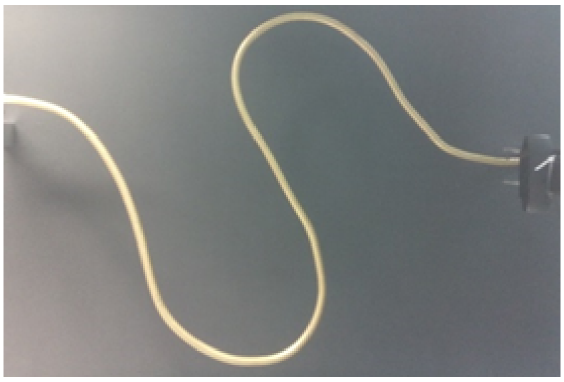
The lack of a more thorough exploration of different design choices is a clear limitation, mainly due to the real-world nature of the scenario. Not only is execution in the real world slower, but each interaction with the environment may lead to damage to the hardware setup. In the case of this use case, each collision between the loop and the wire and subsequent reset procedure may result in slight alterations of the wire shape, which accumulate over time. One frequent failure mode encountered was that the wire would get stuck to the loop during the reset procedure due to friction, which effectively pulled the wire out of the 2D game plane into the reset plane. This is catastrophic, as subsequently moving the loop to the start pose not only destroys the wire shape, but also bears the risk of damaging the hardware. Even though detecting this failure mode is straightforward via the existing collision detection to trigger an emergency stop, the failed experimental run and required human intervention to reset the hardware setup greatly diminished the ability for extensive hyperparameter tuning. For the following experimental study, these limitations have to be accepted. However, they will be revisited thereafter, where an experimental study will attempt to facilitate the training procedure through pretraining on simulations.

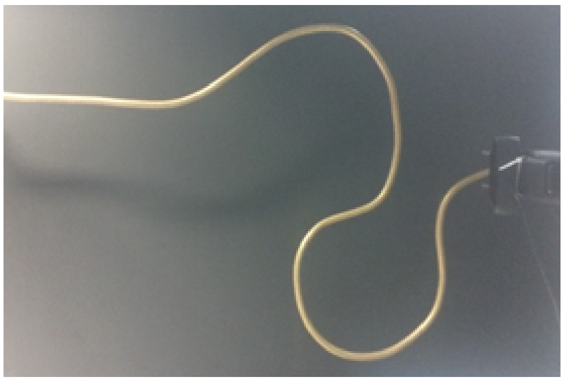
4.2.3 Experimental Evaluation

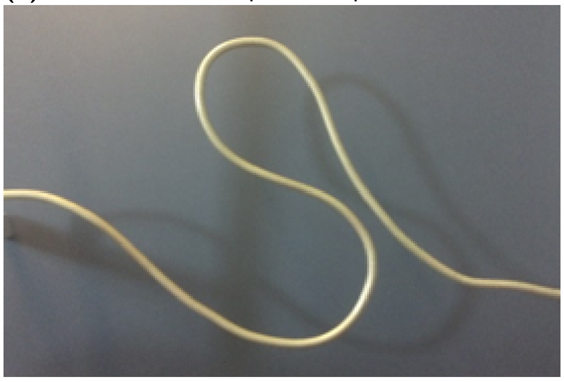
This section provides the experimental results for the wire-loop use case. First, the real-world experiments are discussed, followed by the simulation experiments.

4.2.3.1 Real-World Experiments

With the above training framework implementation in place, experiments were conducted to establish an initial performance benchmark. Further points of interest are the transferability of learned solutions to task variations, an agents' ability to adapt in situations where a higher precision in control is required, and the influence of visual background noise on the performance of an agent. To this end, DRL agents are trained on four scenarios depicted in Figure 4.7. Scenario 1 consists of a simple "s-shaped" wire and a uniform black background, which poses as the simplest scenario and proof-of-concept for learning the wire-loop game. Scenario 2 introduces a different, slightly more complex wire shape with sharper and more diverse turns. This scenario is used to assess an agent's ability to generalize to new wire shapes. Scenario 3 poses a wire shape even more complexity in the form of a bottleneck, which requires higher precision in control. Most notably, the wire cannot be kept in the middle of the loop to traverse the bottleneck, which is the risk-minimizing strategy on the first two scenarios. In addition, two wire sections are captured by the camera near the bottleneck, which are states that have not been encountered before. Thus, an agent needs to differentiate between the wire section it is currently traversing and needs to keep center of the loop, and the other wire section which poses an obstacle. Scenario 4 features a wire shape similar to scenario 1, however the black cardboard sheet is replaced with a projection screen, on which random color images are displayed to introduce visual background noise into the states. Thus, the fourth scenario is used to assess the influence of visual background noise on the performance of an agent.


(a) Scenario 1: A simple "s-shaped" wire.


(b) Scenario 2: A wire with more diverse turns.


(c) Scenario 3: A wire with a bottleneck.

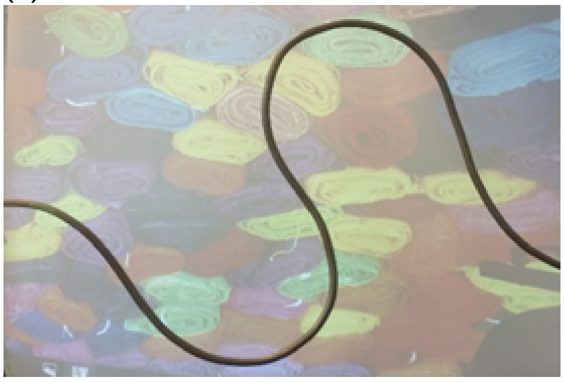

(d) Scenario 4: Added visual background noise.

Figure 4.7 Evaluation scenarios for the wire-loop use case. Taken from [114]

The training results are depicted in Figure 4.8. Note that training was continued on the first three scenarios, a new agent was trained from scratch on the fourth scenario to make it comparable to the results from the first scenario.

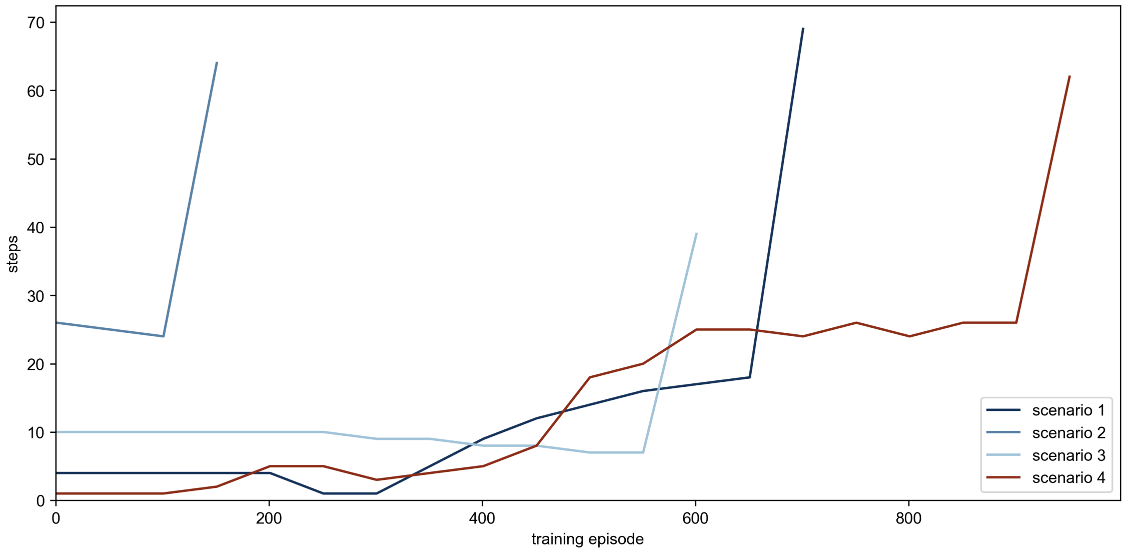


Figure 4.8 DRL results for the real-world experiments in the wire-loop use case. The results show that the learning framework implementation is capable of training an agent to solve the wire-loop game. The agent progressively improves its performance, requiring less training with each new scenario. The agent is also able to successfully learn background noise in the fourth scenario. Adapted from [114].

The results show that the learning framework implementation is capable of training an agent to solve the wire-loop game. On the first scenario, a solution is learned after 700 training episodes in 23 minutes, with a final testing performance of 69 steps. This performance will be used as baseline for subsequent experiments.

Continued training of the agent on the second scenario demonstrates a clear transferability of the solution acquired on the first scenario. Notably, the agent immediately manages to perform 26 steps without collision, presumably until it reaches a wire section with a novel shape. Overall, the agent is able to solve the second scenario after only 150 training episodes in 13 minutes, with a final testing performance of 64 steps. Therefore, it can be concluded that the agent learns a general strategy, rather than memorizing individual scenarios.

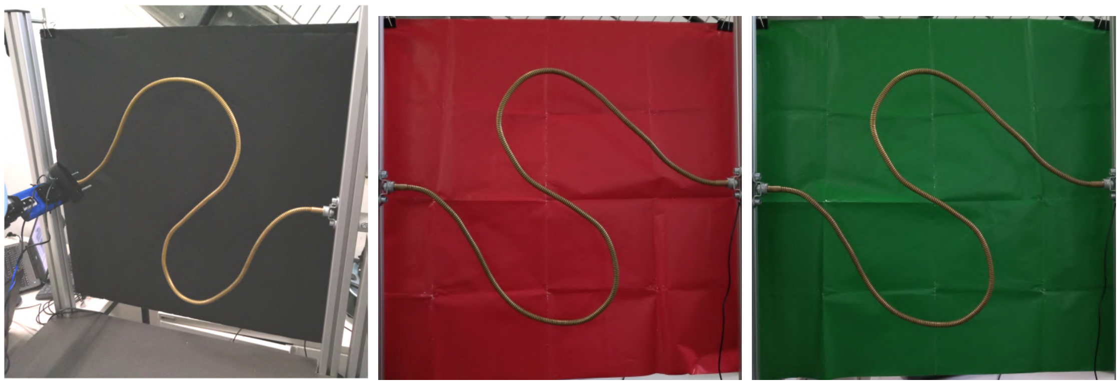
On the third scenario, the policy transfer is also visible, with the initial testing performance already at 10 steps, at which point the agent encounters the bottleneck. As expected, this situation poses a significant obstacle to the agent, which reflects in the remaining learning curve being stuck at the 10-step mark, until the bottleneck is overcome, and the agent solves the scenario with 39 steps after 35 minutes. Thus, the agent is able to adapt to a higher precision in control when needed. Remarkably, this solution constitutes a significant improvement over the first scenario, with a 56.5% reduction in steps. The reason for this can be seen in the learning curve, which continuously decreases down to 7 steps. Presumably this is not due to a negative learning effect, i.e. the agent failing earlier on the wire, but rather inversely indicates a performance increase, as the agent learns to traverse the wire up to the bottleneck faster. Once the bottleneck is overcome, the agent is able to find an overall faster solution. This insight is further evidence that the agent learns a general strategy, in this case by learning optimizations transferable to later unseen parts of the wire.

The added visual complexity in the fourth scenario expectedly results in a longer training procedure compared to the first scenario, with a total of 950 training episodes in 61 minutes. Again, a shorter solution was found (62 steps), hinting to the agent constructively using the increased training time to continuously optimize its strategy. The significant increase in training effort clearly highlights the importance of reducing variance in the environment, such as visual background noise.

In conclusion, the initial results show that the learning framework implementation is capable of training an agent to solve the wire-loop game. With these initial results, a solid foundation for further investigations and optimizations is established. As highlighted above, the biggest limitation of the current setup is its real-world nature, which renders experiments time-consuming and potentially destructive. To address this limitation, the subsequent investigation will focus on the use of simulations to facilitate the training procedure.

4.2.3.2 Sim2Real Experiments

With the objective of improving the application of DRL in the wire-loop use case, the following investigation aims to confirm the hypothesis that the pretraining will have some positive effect on reducing the real-world training effort. DR will be applied to improve the generalization of pretrained agents to the real-world setup. As adding additional randomizations is expected to increase the training effort in simulation, the influence of different randomization parameters is investigated. In the following, agents are pretrained in simulation with different randomization settings. The pretrained agents are then transferred to the real-world scenario depicted in Figure 4.9a, which is analogous to the previously discussed baseline scenario, with a black cardboard background behind the wire.



(a) Standard black background scenario. Taken from [154]. **(b)** Transfer scenario with red background variation. **(c)** Transfer scenario with green background variation.

Figure 4.9 Experimental scenarios for the DR study to test sim2real transferability.

The pretraining and transfer results in form of required training episodes are reported in table 4.1. Three experiments are performed for each setting, out of which the highest value is reported to provide an upper bound on the required training effort.

ID	Randomizations				Training Episodes	
	Color	Texture	Noise	Camera	Simulation (Pretraining)	Real-World (Transfer)
baseline					0	650
1					450	750
2	✓	✓	✓		900	600
3	✓			✓	1850	450
4	✓		✓	✓	2200	400
5	✓	✓		✓	2600	**0**
6	✓	✓	✓	✓	3500	**0**

Table 4.1 Training effort for different DR parameter settings. A positive correlation between the amount of randomization and the required training effort is observed. Further, a positive effect of randomization on transferability is visible. Notably, not all parameters must be randomized to achieve zero-shot transfer.

As a first observation, no significant effect can be attributed to pretraining in the basic simulation setup with no randomizations (setting 1), with the best and worst run requiring 100 training episodes less and more than the baseline, respectively. Thus, the basic simulation setup without randomizations does not provide any benefit in terms of transferability. It was however clearly possible to fully close the sim2real gap and achieve zero-shot transfer through extensive randomization, as seen in settings 5 and 6. To further validate this result, the agents were additionally tested on background variations by using red and green colored sheets to change the background color (cf. Figures 4.9b and 4.9c). As expected, the agents were directly transferable to these additional scenarios, presumably since the color randomizations required the agents to become invariant to the background color.

Regarding the effect of different randomization parameters, the results show a clear difference in transferability results between the different parameter sets. In general and as expected, adding more randomizations increases the complexity of the simulation source domain, and thus increases the required pretraining effort. Interestingly, a clear negative correlation between amount of randomization and transfer success can be identified. This indicates that more complex simulation environments increases the generalization of an agent.

Investigating the individual randomization parameters, it can be concluded that the application of Gaussian noise does not have any effect on the transferability (setting 3 vs. 4 and 5 vs. 6). Adding texture randomizations however has a clear benefit (setting 3 vs. 5). Also, variations in camera location significantly improve the transferability (setting 2 vs. 6). Contrasting texture and camera randomizations, it can further be seen that the latter have the highest impact (setting 2 vs. 4).

These findings are plausible, as the real-world states always exhibit some sort of color-texture variation, but this variation is unlikely follows a Gaussian distribution. Also, in simulation the camera pose was only approximated and not finely calibrated, thus the positive impact of camera location variation to account for this known domain gap is comprehensible.

In conclusion, the application of DR successfully eliminated the sim2real gap and enabled zero-shot transfer. While the best results were achieved through maximum randomization, it was also possible to achieve the same transferability without noise randomization, which decreased the training effort spent in simulation. Although this insight is valuable as it points to an opportunity to reduce implementation and training efforts, the question of how to find the optimal subset of randomization settings before implementation remains open. Therefore, for industrial applications where real-world training is to be minimized at highest priority, the strategy of randomizing as much as possible and accepting potential expendable training efforts in simulation is recommended.

4.3 Use Case 2: Object Picking

Object picking, of which the second use-case is an instance of, is a ubiquitous task in robotics. With the rising popularity of DRL, a number of benchmark environments have been developed to standardize the development process and provide a common ground for comparing different methods.

For this reason, the following investigation starts with a first attempt to learn the object picking task using a publically available benchmark suite. In addition, as the selected benchmark environment does not exactly match the use case setup, the required modifications will be highlighted.

4.3.1 Environment Design

This section first introduces the selected benchmark environments. Subsequently, the benchmark interfaces are related to the use-case specific hardware setup, and the required modifications are discussed.

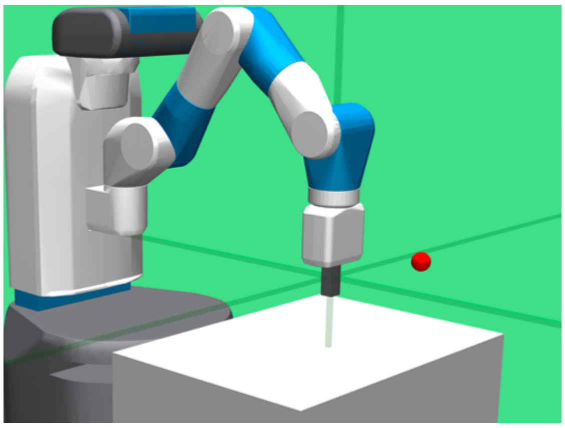
4.3.1.1 Fetch Benchmark Environments

One prominent benchmark suite is Farama Gymnasium [188], previously known as OpenAI Gym [27]. The suite provides a standardized environment interface, as well as a collection of benchmark environments. Given the popularity[1] of the Gymnasium benchmark suite, the environments can be considered stable, eliminating potential sources of error arising from custom-built environment implementations. When starting with this work, the Fetch suite was introduced by Plappert et al. [134] as benchmark focussed on DRL for robotic manipulation tasks. The authors argue that these environments pose challenging robotic manipulation tasks for DRL methods, and provide a thorough evaluation of different optimization techniques and hyperparameters. The benchmark was cited over 670 times as of July 2025, which serves as a strong indicator of its adoption by the DRL research community.

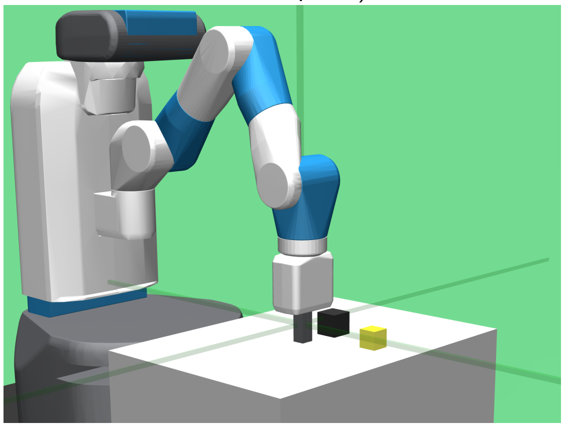
The Fetch benchmark suite consists of a Fetch robot arm with a two-fingered gripper simulated with the MuJoCo physics engine [186]. Four tasks are provided: Reach, Push, Slide, and PickAndPlace. The Reach task is the simplest one, which requires a control agent to reach goal poses with the robot's TCP. The other three tasks require a cube to be moved to a goal pose by respectively pushing, sliding, or pick-and-placing it with the robot's gripper.

The PickAndPlace task is the most similar to the use case, as it also requires the robot to pick up an object and bring it to a goal pose. However, as the analysis of Plappert et al. [134] shows, it is also the most difficult one to learn. Therefore, Reach as the simplest task, and Push as an intermediary will also be considered to provide a more comprehensive evaluation. The selected tasks are depicted in Figure 4.10.

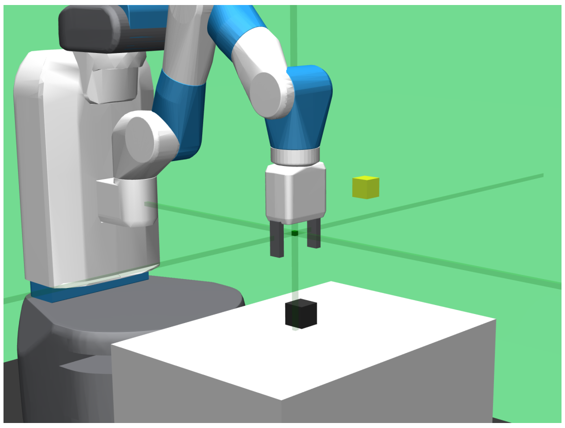
[1]The repositories of the Gym and Gymnasium have respectively 36,200 and 9,600 stars on GitHub as of July 2025.



(a) Reach: The task is to reach a goal pose (red sphere) with the TCP.



(b) Push: The task is to push a cube (black box) to a goal pose (yellow box). Taken from [156].



(c) PickAndPlace: The task is to lift a cube (black box) to a goal pose (yellow box). Taken from [156].

Figure 4.10 The benchmark environments FetchReach, FetchPush, and FetchPickAndPlace from the Fetch suite [134].

The continuous state spaces of the environments consist of a 10-dimensional vector encoding the robot state, a 15-dimensional vector for the cube state (not included in Reach), and a 3-dimensional vector describing the TCP or cube goal pose. The continuous action spaces consist of four dimensions: three dimensions for the end-effector position displacement, and one additional value controlling the opening and closing of the gripper. Notably, this formalization constraints the gripper to a fixed orientation, which significantly simplifies the tasks. In accordance with the findings of Plappert et al. [134], a sparse reward function is applied:

$$r_t = \begin{cases} 0 & \text{if } \left\| p_{goal} - p_{cube/TCP} \right\| < 5\ cm \\ -1 & \text{otherwise} \end{cases} \tag{4.2}$$

4.3.1.2 Use-Case Specific Environment Interface

Given the use-case specific hardware setup introduced in chapter 1.3.2, two major environment design decisions need to be made to align the benchmark environments with the real-world scenario. The resulting environment concept is depicted in Figure 4.11.

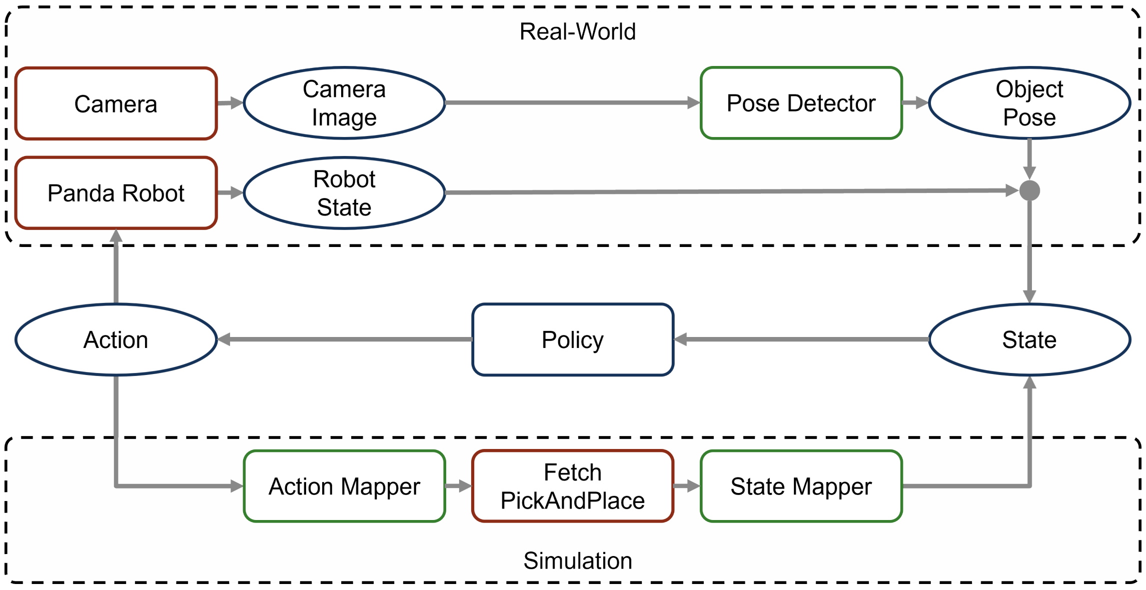


Figure 4.11 Overview of the environments used for the object picking use case. With minor adaptations, a policy trained on the Fetch PickAndPlace environment can be used in the object picking use case.

First, the state space of the benchmark environments are specific the utilized Fetch robot arm. To account for this, the state space of the benchmark environments, in particular the PickAndPlace environment to be used later, must be mapped into the use-case specific state space. To achieve this, the following state mappings are applied:

1. All state variables related to the Fetch robot base coordinate system are mapped into the Panda base coordinate system, which is located at the midpoint of the long edge of the table.
2. All velocity variables are removed, since they are not easily obtainable in the real-world setup.
3. The gripper states are mapped to a binary open/closed state value.

For the action space, the relative movement is already defined as relative TCP displacement, which can be considered robot-agnostic. Therefore, the only adjustment required is to use a binary open/closed gripper action value instead of the continuous one provided by the benchmark environments.

4.3.2 Experimental Evaluation

This section presents the experimental results on applying the learning framework to the benchmark environments, with the goal of establishing its performance in comparison to the benchmark results provided by Plappert et al. [134].

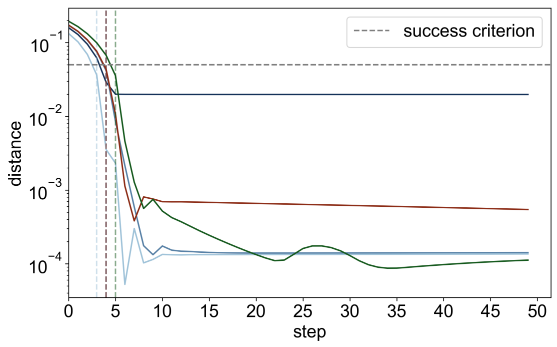
4.3.2.1 Analysis of Task Difficulty

In order to gain a better understanding of the task difficulty, a preliminary investigation is conducted to establish an upper bound on the performance that can be expected from an optimal control strategy. To this end, rule-based controllers are implemented for the Reach and PickAndPlace tasks. The Push task is not considered, as modelling the physical dynamics of the block sliding on the table was deemed beyond the scope of this investigation.

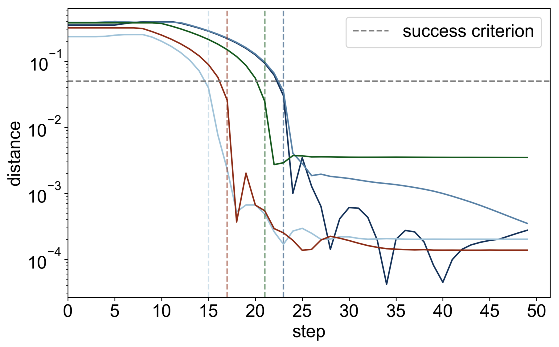
For the Reach task, the control strategy is straightforward. The controller computes the vector pointing from the current TCP position to the goal position, whereby the positions are provided as part of the state. This vector is then normalized with the maximum achievable velocity to yield the action. The maximum achievable velocity of 3.2 cm/step is determined empirically by analyzing the traversed TCP distances when providing maximum action values to the environment.

The PickAndPlace task can be solved by first moving the TCP to the object's position, then closing the gripper to grasp the object, and finally moving the TCP to the goal position. For the last stage, better results are achieved by using the object position instead of the TCP position to account for grasping offsets.

The resulting control strategies are depicted in Figure 4.12 for five different random environment initializations. The Fetch task as most primitive control task can be solved in less than 5 steps, assuming the predefined success threshold of 5 cm, by moving the TCP in a straight line to the goal position. The PickAndPlace task requires more steps, taking more than 15 steps to complete. Thus, in anticipation of the following DRL experiments, it can be concluded that the PickAndPlace task is expected to be significantly more difficult to learn than the Reach task.



(a) The Fetch Reach task is solvable in less than 5 steps.



(b) The Fetch PickAndPlace task requires more than 15 steps to complete.

Figure 4.12 Optimal control solutions for the Reach and PickAndPlace tasks.

4.3.2.2 Fetch Benchmark Experiments

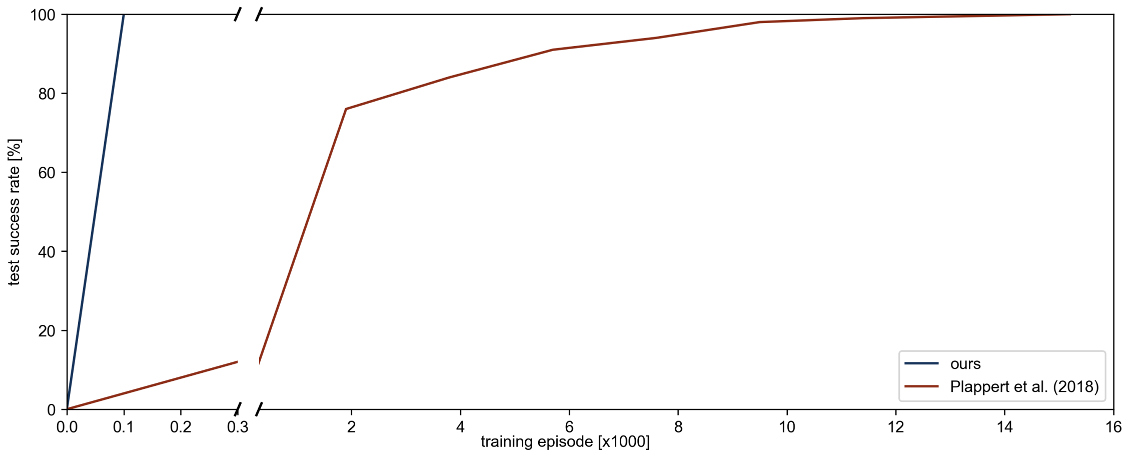
With the training framework in place, DDPG agents were trained on the three selected tasks. The hyperparameters were chosen identical to the ones used by Plappert et al. [134]. The only exception is the application of HER. The authors specify a probability of 80% for replacing the goal with the last achieved goal of a trajectory (strategy `final`). This basic application can be significantly improved by using different sampling strategies, as already proposed by Andrychowicz et al. [7], the authors of HER. In this work, the best-performing `future` strategy of Andrychowicz et al. [7] is applied, which samples a random goal that was achieved later in the trajectory. This allows for a more aggressive application of HER, where each experience tuple can be replayed with multiple different virtual goals. Therefore, the HER implementation is parametrized by the amount of virtual samples per real experience samples, which is set to 4. This achieves the same distribution of 80% virtual samples in the experience replay buffer.

Each agent is trained five times with different random seeds to account for the stochastic nature of the entire training process. During training the agents are tested every 1000 episodes, except for the Reach task, where the test interval is set to 100 episodes due to the simplicity of the task. As performance metric, the success rate of the agent over 10 test rollouts is computed. Training is performed until either a perfect score of 100% test performance is achieved, or a threshold of 25,000 episodes is reached. The episode threshold was chosen to keep the wall clock training time below 8 hours as training was performed on a shared GPU server with limited availability. The results for the individual tasks are depicted in Figure 4.13. The figures show the performance of the previously outlined training framework (denoted as 'ours'), as well as the benchmark results taken from Plappert et al. [134] for comparison.

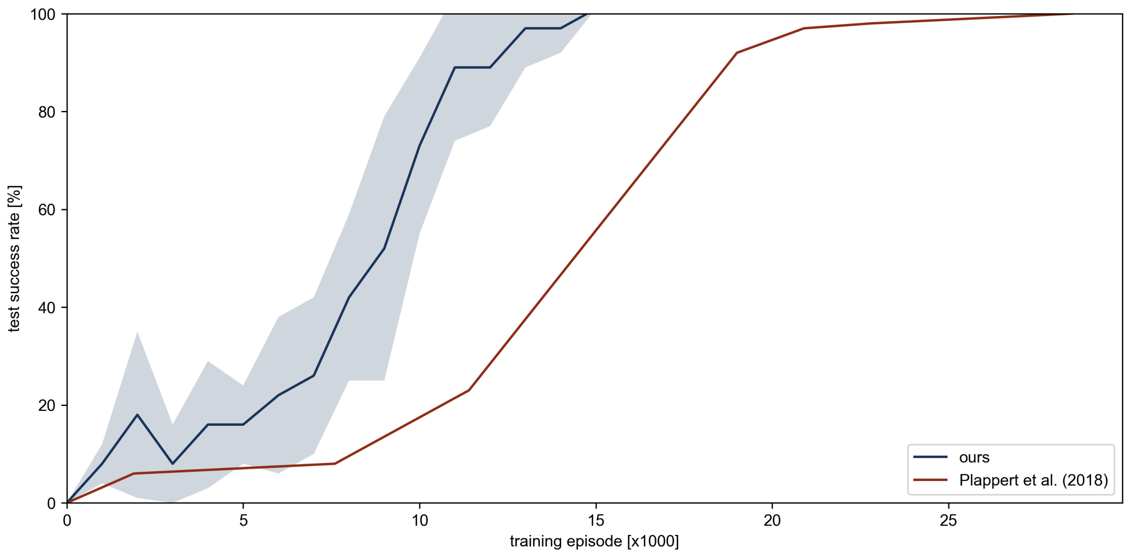
The results show that the training framework implementation achieves better performance than the implementation of Plappert et al. [134] on the Reach and Push tasks. As the only meaningful difference between the two implementations lies in the application of HER, this suggests that the improved performance is primarily due to the `future` sampling strategy.

For the most relevant task with respect to the second use case, the PickAndPlace task, neither implementation was able to fully solve the task. While the performance initially rises faster compared to Plappert et al., the incline decreases earlier and the two performance curves meet at 25,000 episodes, incidentally coinciding with the maximum episode threshold. Thus, no definitive conclusion as to which implementation is better can be drawn. However, the experimental results were deemed sufficient to draw the following conclusions.

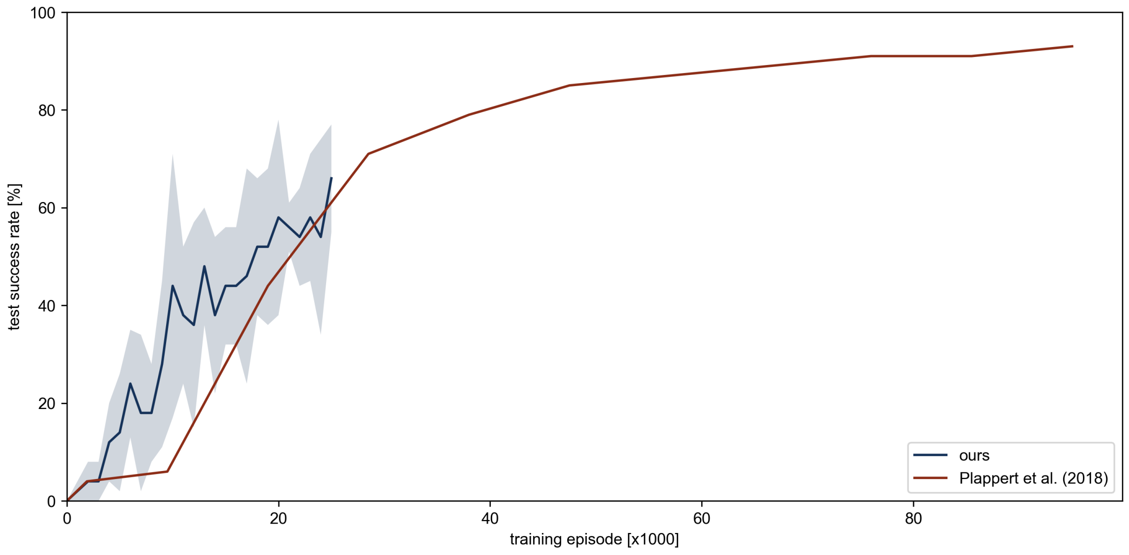
The application of HER with the `future` sampling strategy achieves better results than the basic application of HER with the `final` sampling strategy. Also, the training framework implementation is able to learn benchmark tasks with a satisfactory performance and can thus be used as a starting point for further development. Third, as Plappert et al. [134] were not able to achieve a 100% success rate on the PickAndPlace task, and the performance to the own implementation is comparable, better results with the own implementation are not expected. Additionally, relating the 95,000 episodes used by Plappert et al. [134] to the capabilities of the own



(a) The training framework achieves better performance on the Fetch Reach task.



(b) The training framework achieves better performance on the Fetch Push task.



(c) The training framework achieves comparable performance on the Fetch PickAndPlace task. The task was not sufficiently solved by either implementations.

Figure 4.13 Comparison of the training framework performance to Plappert et al. [134] for the Reach, Push, and PickAndPlace tasks. The training framework achieves better or comparable performance on the benchmark tasks.

hardware setup of roughly 8 hours per 25,000 episodes puts the training time of a single experiment at approximately 30 hours. While this time frame may be acceptable for a single training of a final policy, it is infeasible for any form of iterative development given the limitations of the shared GPU server. Thus, the initial DRL setup is not suitable to reliably develop control policies for the object picking use case in a reasonable time frame.

4.4 Use Case 3: Clip Assembly

The clip assembly use case is derived from a real-world aircraft manufacturing process, requiring the collision-free insertion by finding a precise robot trajectory. To avoid inevitable damage to the real-world demonstrator through repeated collisions throughout a DRL training process, it was decided to first use a CAD model of the real-world assembly line as basis for developing a custom simulation environment. Due to the high quality and documentation standards of the aircraft industry, the digital model is of high accuracy. In addition, the same model was used to fabricate the 3D-printed parts of the demonstrator (cf. chapter 1.3.3). Thus, any results obtained in the simulation are expected to be directly transferable to demonstrator, and later the real-world assembly line. In the following, the simulation environment is introduced. Subsequently, an initial attempt to learn the task with DRL is discussed.

4.4.1 Environment Design

Given the industrial nature of the use case, the simulation environment is designed to reflect the real-world assembly task as closely as possible. The task is to insert a clip into a shell, starting from an initial pose approximately 5 cm away from the installation location (cf. Figure 4.14).

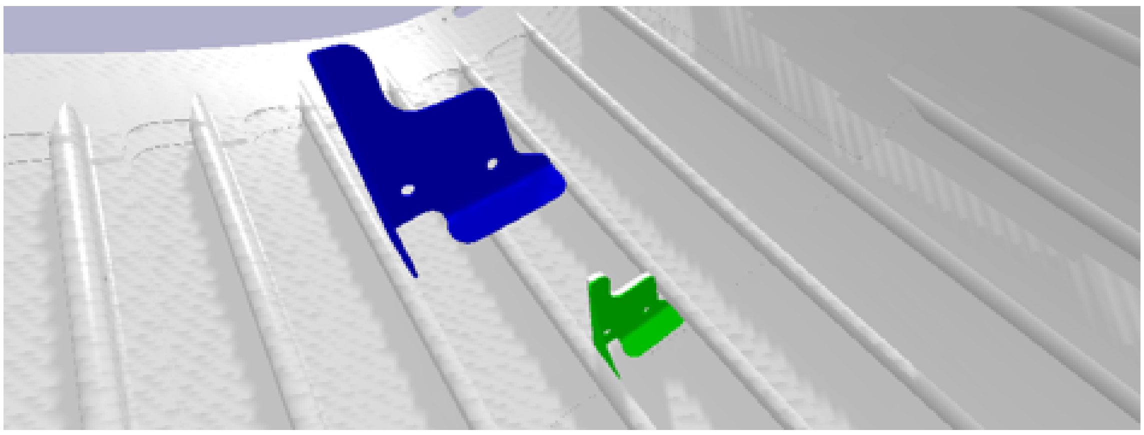


Figure 4.14 In the clip assembly use case, a clip (blue) is inserted into a shell (green).

A reduction to the relevant parts of the process is necessary to ensure that simulation is performed as fast as possible. To this end, clip-specific environments are created in Pybullet, whereby for a given clip only the meshes in a vicinity of the installation location are considered. This is achieved by clipping the meshes with a sphere of a given radius, centered at the installation location. Examples of the resulting environments are depicted in Figure 4.15.

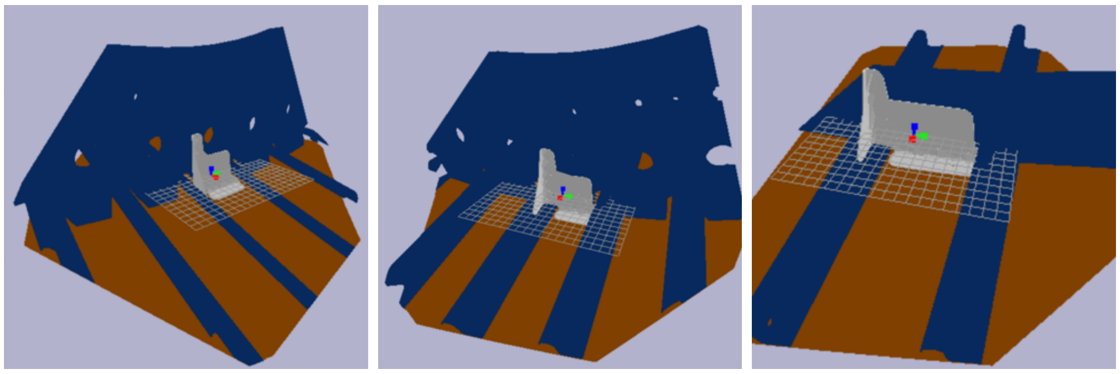
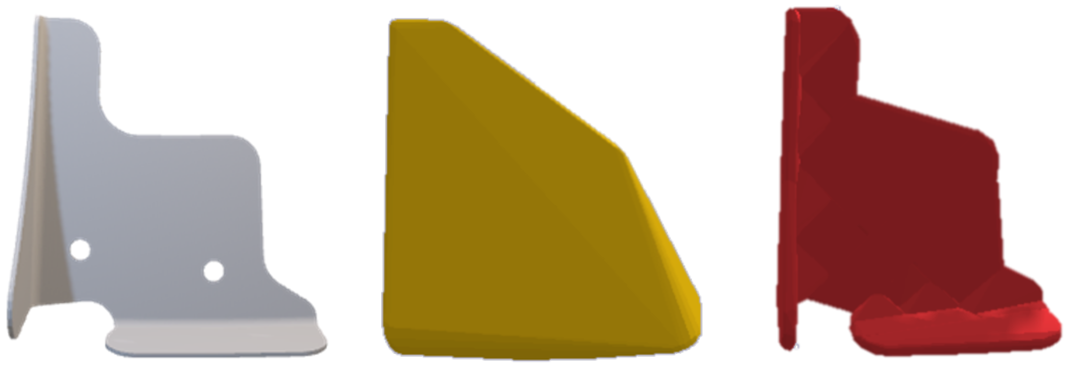


Figure 4.15 Example of the clip assembly environment. The shell is reduced to the vicinity of the installation location to improve simulation performance.

To ensure that the collisions in simulation are realistic, it is also necessary to adjust the collision shapes. In particular, the concave nature of the meshes is not suitable for simulation, as they are approximated by a convex hull. As can be seen in Figure 4.16, this convex approximation makes the collision-free insertion of the clip impossible. To address this issue, the meshes must be decomposed into a set of convex shapes, which is achieved using the Volumetric Hierarchical Approximate Convex Decomposition (V-HACD) algorithm [108].



(a) The original mesh of the clip contains concave parts, which are not suitable for collision detection.

(b) The convex hull approximation of the original mesh is too coarse and leads to collisions in simulation.

(c) V-HACD provides a set of convex shapes, which are suitable for collision detection and precise enough for assembly.

Figure 4.16 Mesh preprocessing required to improve collision detection in simulation for the clip assembly use case.

Translating the clip insertion task into an environment is straightforward. To describe the state, the position and orientation of the clip is considered, which are provided relative to the goal pose. The 6D action space describes relative movement of the clip. At reset, the clip is placed at a distance of approximately 5 cm to the goal pose, as per specification of the real-world assembly task.

To simulate clip movement, a constraint is applied to the clip, which pulls the clip towards the next waypoint with a specified force. This approach was also found to be an effective way to detect collisions by checking if the waypoint was reached within a certain amount of simulation steps.

To communicate the task of assembly, the following reward function is used

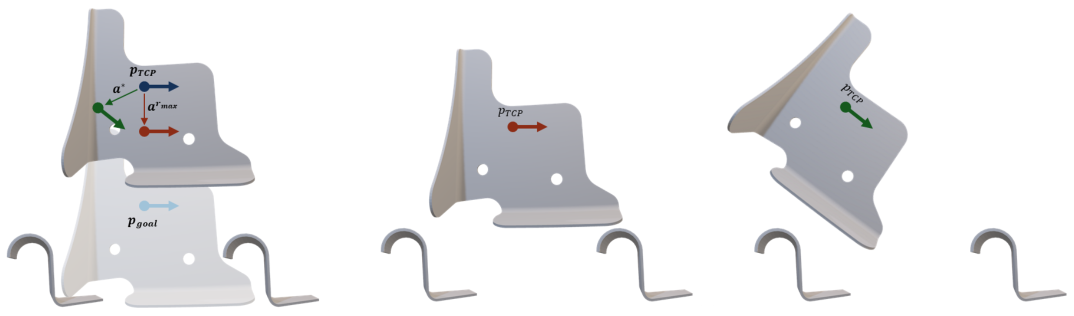
$$r_t = \begin{cases} -1 & \text{if collision} \\ 1 & \text{if } \|p_{goal} - p_{TCP}\| < (1mm, 1^\circ) \\ 0.1 * (\exp(-10d) - 1) & \text{otherwise} \end{cases} \tag{4.3}$$

which rewards successful assembly with 1 and otherwise issues small rewards with a gradient towards the goal pose. Collisions with the shell are penalized heavily and lead to a termination of the episode.

4.4.2 Experimental Evaluation

To train agents on the defined simulation environment, the same DRL framework used for the wire-loop simulation can be reused. This clip assembly use case also benefits from the environment parallelization strategy. Notably, HER is not used, since each clip scenario only consists of a single goal pose, and thus the resulting environments are not goal-conditioned. With this training setup, the first training runs were found to be unsuccessful. The agents did not converge to the desired behavior, and instead defaulted to strategies that kept the clip far away from the shell.

These negative results can be attributed to two main issues with the given setup. First, the fact that finding a suitable assembly trajectory is a task which requires very high precision. As finding such a trajectory through exploration is highly unlikely, the agents learned that avoiding collisions by keeping a safety distance to minimize the risk of punishment due to collision. Second, the dense reward function from equation 4.3 was found to be a potential source of error. While the distance to the goal provides as a rough indicator of where to move the clip and leads to initial progress towards the shell, the vector towards the goal, and thus the maximum reward, does not coincide with the optimal assembly trajectory. Figure 4.17 illustrates this issue. Thus, the misalignment of the reward function and the optimal assembly trajectory likely leads to suboptimal learning behavior.



(a) Initial pose, goal pose, and actions. **(b)** The reward-maximizing action leads to a suboptimal pose. **(c)** The optimal action performs a rotation necessary for insertion.

Figure 4.17 Illustrative example of suboptimal behavior induced by the dense reward function.

While the suboptimal reward function is a design flaw that can be remedied by using a sparse reward function, the inherent difficulty of high precision assembly for DRL due to the high potential for failure is a fundamental challenge. The latter will be the focus of investigation in this use case.

4.5 Mapping Research Questions to Use Cases

In this chapter, a generalized DRL was introduced and applied to three representative industrial robotics use cases. In addition, standard TRL strategies in the form of DR were employed. The achieved results will provide necessary baselines for the contributions of this thesis. Revisiting the general suitability of the use cases for different aspects of the modular decision-making pipeline outlined in chapter 1.3, the following value of the use cases towards addressing the RQs can be substantiated.

The flaws regarding the time-discrete nature of the standard RL framework for real-time **control** addressed by RQ-1 are particularly pronounced in the wire-loop use case where continuous, smooth trajectories are desired to both solve the game as fast as possible, as well as mimic dynamic human game play. These goals are direct proxies for industrially relevant parameters of reduced process time and reduced wear and tear on mechanical components. Therefore, the wire-loop use case is a suitable candidate to investigate RQ-1.

For the generation and sim2real transfer of **perception** modules as prompted by RQ-2, both the wire-loop and object picking use cases are suitable due to their use of visual state spaces. In the object picking use case the required feature space in terms of object pose is well-defined. In the wire-loop use case, the relevant features in the images, i.e. the wire and the loop, are easily identifiable. However, a suitable encoding of these features is not self-evident. Therefore, the main focus towards answering RQ-2 will be on the wire-loop use case, while the object picking use case will still be considered to investigate sim2real transfer of perception modules.

To address the improved **planning** through exploitation of robotic movement structure in RQ-3, the object picking use case is a suitable candidate to investigate the cross-task transfer of HRL, as the overall task can be conceptually decomposed as consecutively bringing the object to a series of subgoals. The clip assembly use case, by definition as assembly scenario and due to its high precision requirements, serves as best candidate for the integration of assembly-by-disassembly with DRL.

The final RQ-4, cross-robot **execution** of trajectories, will not be addressed in a specific application-driven use case. Instead, a simplified proof-of-concept task will be considered and experiments will be restricted to simulation. The experimental setup will however benefit from all use cases, in particular regarding the reuse of robot simulation models.

5 Smooth Continuous Robot Control

"Robotics are beginning to cross that line from absolutely primitive motion to motion that resembles animal or human behaviour."
- Jeffrey Jacob Abrams

Smooth and continuous robot trajectories are desired both from a process and a maintenance perspective in a wide range of industrial applications. Smoothness describes the lack of sudden directional changes, while continuity describes the lack of abrupt changes in velocity. In general, sharp directional changes in a trajectory require the robot to decelerate, which increases the overall process time. From a general maintenance perspective, frequent de- and acceleration of the robot can put significant stress on mechanical components and thus decrease their lifespans. Thus, realizing smooth, and continuous trajectories is a highly desirable property in industrial robotic applications. In this context, the discrete-time nature of standard RL frameworks does not account for real-world effects such as latencies of camera equipment or policy inference. In particular with the use of position-based control, this leads to the robot stopping at each state until the next action is provided. Further, since a policy selects actions independent of the previous actions, the resulting trajectories may not be smooth. This chapter addresses these shortcomings towards answering **RQ-1: How can smooth, continuous movement be achieved within the RL framework?**

The following sections investigate the realization of smooth and continuous trajectories within the DRL framework, using the wire-loop use case as experimental basis. First, an inherently smooth action space based on Bézier curves is proposed to explicitly enforce smooth trajectories. Next, an asynchronous DRL framework is introduced to enable continuous movement between individual actions. The asynchronous DRL framework and Bézier curve action space were previously published by the author [158]. The framework is further complemented with a dynamics roll-out model to anticipate future states and preserve the Markovian property of the environment.

5.1 Smoothness-Constrained Action Space Design

Revisiting the action space design of the wire-loop use case defined in chapter 4.2, an action is interpreted as relative transformation to the next TCP on the 2D game plane. The translation and rotation DOF are hereby handled independently, allowing for relative loop movements such as moving to one side, and simultaneous rotating to another. This design allows for maximum flexibility, since a control strategy, such as a DRL agent, can perform arbitrary movements.

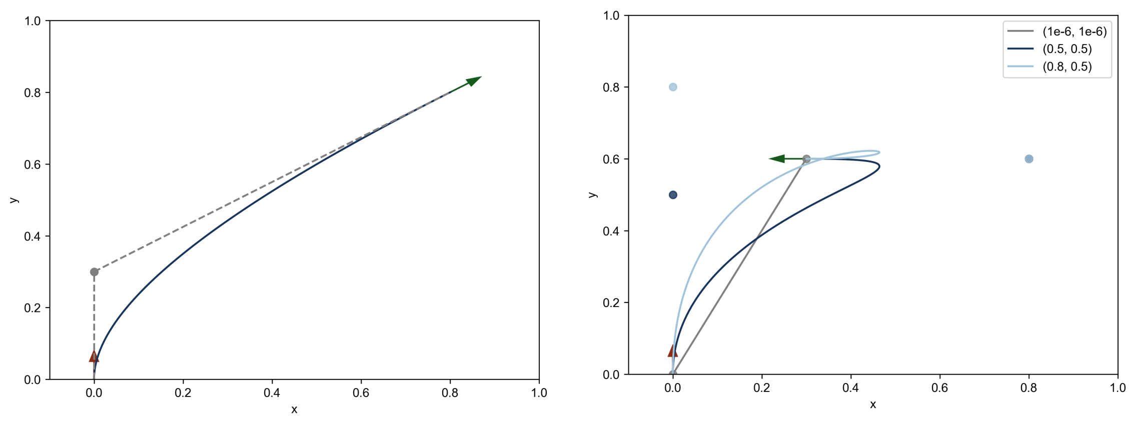
From the perspective of smoothness however, the transitions between two independently chosen actions exhibit a discontinuous velocity profile. Since the desired velocity between two states is directly derived from the action $v_t = s_{t+1} - s_t = \alpha a_t$, it is evident that switching from one action to another in any given state results in an abrupt change in the desired velocity vector.

In contrast to using linear segments to connect a set of waypoints, Bézier curves were introduced in chapter 2.1.2 as popular choice in fields such as computer graphics and robotics to generate inherently smooth paths. In particular, the ability to combine two Bézier curve segments to form a continuous path of class C^1 by constraining the control points, as well as the ability to reparametrize a Bézier curve path into a trajectory of same continuity class, makes them a suitable choice for the wire-loop use case.

Considering that the Bézier curve derived from the previous action $P_0^{t-1}, ..., P_n^{t-1}$ is fixed, the state s_t will be at position P_n^{t-1}. Thus, the first control point P_0^t is constrained to $P_0^t = P_n^{t-1}$, with the orientation given by the tangent $\vec{T}_n^{t-1} = \overrightarrow{P_{n-1}^{t-1} P_n^{t-1}}$. To ensure C^1 continuity, the second control point is constrained to $P_1^t = P_0^t + k \cdot \vec{T}_n^{t-1}$, where k is a positive number. Thus, the first control point has one degree of freedom. All other subsequent control points do not influence the C^1 continuity of the trajectory, and can thus be arbitrarily placed. Therefore, a Bézier curve of degree n can be parametrized with $2n - 1$ parameters.

Regarding the integration of Bézier curves into the environment design, it is possible to use the same action space as before and use Bézier curves to interpolate between the waypoints. As depicted in Figure 5.1a, in situations where the translation and rotation go in the same direction, a Bézier curve of degree 2 can be formed by using the waypoints as start and end control points, and inferring the intermediate control point as intersection of the waypoint tangents. This approach however does not work in situations where the translation and rotation go in different directions, as the tangents will not intersect. Such cases require a Bézier curve of degree 3, whereby the placement of the two intermediate control points is not self-evident. As seen in Figure 5.1b, the inherently arbitrary control point placement heavily influences the resulting trajectory, in the worst case leading to self-intersecting trajectories. Although this action space formulation would be a valid approach, and a DRL can be expected to implicitly learn to avoid unfavorable trajectories, the approach followed by this thesis is to interpret the action space as a direct placement of the control points.

To this end, this thesis investigates two different strategies for encoding the control points of a Bézier curve as actions. The two encoding strategies are depicted in Figure 5.2.



(a) A Bézier curve of degree 2 can be inferred from a target pose if the tangents of the end points intersect.

(b) Some target poses require Bézier curves of degree 3 or higher. In this case, the placement of intermediate control points is not self-evident.

Figure 5.1 Inference of a Bézier curve from a target pose.

The first encoding strategy constructs the control points of a Bézier curve using Cartesian coordinates (cf. Figure 5.2a). By defining the action vector as $a_t^{cartesian,n} = [a_t^{x,n-1}, a_t^{y,n}]$, where

$$a_t^{x,n-1} \in \{x \in \mathbb{R}^{n-1} \mid -x_{lim} \leq x_i \leq x_{lim}\}$$

$$a_t^{y,n} \in \{y \in \mathbb{R}^{n} \mid 0 < y_i \leq y_{lim}\}$$

encode the x and y coordinates of the control points in the TCP coordinate frame, respectively. The TCP coordinate frame is defined at the first control point P_0^t with the y-axis pointing up in the direction of movement, i.e. in $\vec{T}_n^{t-1}$ direction.

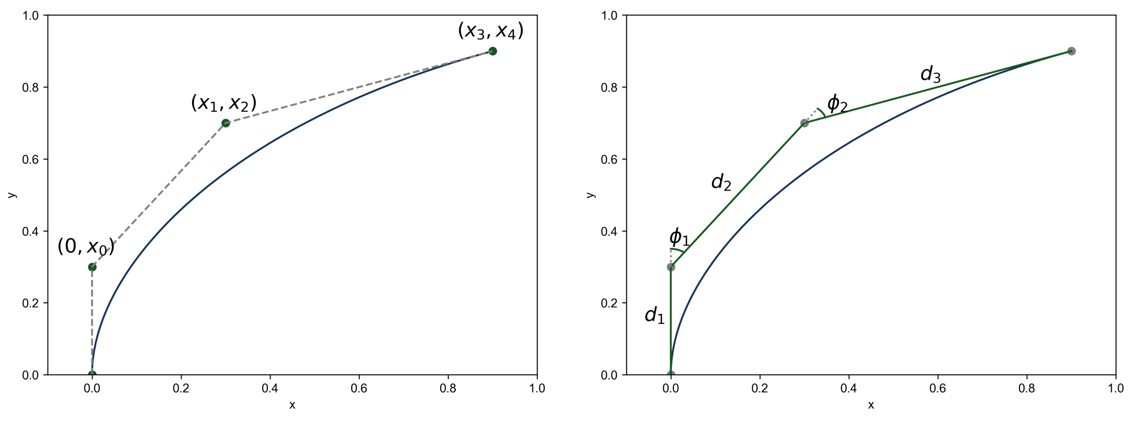
The second encoding strategy uses polar coordinates to construct the control points (cf. Figure 5.2b). Here, the action vector is defined as $a_t^{polar,n} = [a_t^{d,n}, a_t^{\phi,n-1}]$, where

$$a_t^{d,n} \in \{d \in \mathbb{R}^{n} \mid 0 < d_i \leq d_{lim}\}$$

$$a_t^{\phi,n-1} \in \{\phi \in \mathbb{R}^{n-1} \mid -\phi_{lim} \leq \phi_i \leq \phi_{lim}\}$$

encode the distances and angles between the control points. In contrast to the Cartesian encoding, the control points are not placed independently, but are scaffolded onto the previous two control points.

After the control points are determined with either encoding strategy, it is necessary to sample waypoint poses from the resulting Bézier curve which can then be sent to and traversed by a position controller. The waypoints should thereby not be sampled arbitrarily: If on the one hand too few waypoints are sampled, the resulting movement may deviate significantly from the desired trajectory. Sampling on the other hand too many waypoints may exceed the precision granularity achievable by the robot. Thus, the position controller must skip waypoints which are already considered reached. In the best case, this only means avoidable computational and communication overhead. However, with an increasing number of waypoints it is likely that the movement of the robot will suffer if the position controller encounters a lot of similar waypoints and skipping them takes longer than reaching them.



(a) Cartesian encoding. Control points are placed independently in the TCP coordinate frame.

(b) Polar encoding. Control points are placed sequentially based on the previous two control points.

Figure 5.2 Bézier curve encoding strategies.

To achieve an optimal sampling of waypoints, the implementation of this thesis employs a simple waypoint refinement algorithm, whose pseudocode is provided in Algorithm 5.1. Given a Bézier curve $B(t)$, the algorithm first initializes the end points $B(0)$ and $B(1)$ as waypoints. For each transition between successive waypoints, the algorithm checks if the transition satisfies specified constraints. Constraints may include thresholds for the maximum distance or rotation between waypoint poses, or more sophisticated conditions such as the deviation between the desired Bézier curve segment trajectory and the approximated linear movement between the waypoints. The algorithm recursively increases the waypoint resolution where a more fine-grained movement is required to follow the Bézier curve.

```
Input: low and high sampling parameters $t_{low}$ and $t_{high}$, Bézier curve $B(t)$
Output: Optimal sampling parameters $t_{low}, ..., t_{high}$
 1: function REFINE_WAYPOINTS($t_{low}, t_{high}, B(t)$)
 2:     $W_{low} \leftarrow B(t_{low})$
 3:     $W_{high} \leftarrow B(t_{high})$
 4:     if transition $W_{low} \rightarrow W_{high}$ satisfies constraints then
          return $[t_{low}, t_{high}]$
 5:     else
 6:         $t_{mid} \leftarrow (t_{low} + t_{high})/2$
 7:         $lower \leftarrow refine\_waypoints(t_{low}, t_{mid}, B(t))$
 8:         $higher \leftarrow refine\_waypoints(t_{mid}, t_{high}, B(t))$
          return $lower[: -1] \cup higher$          ▷ Note: Remove duplicate $t_{mid}$
 9:     end if
10: end function
```

Algorithm 5.1 Bézier curve sampling algorithm.

5.2 Asynchronous Learning Framework

One apparent challenge for continuous control with RL is the time-discrete nature of standard RL frameworks. This is especially true for position-based control, where each interaction step ends with the robot coming to a complete halt as the individual actions are completed. Such behavior is highly unfavorable as it results in non-fluent trajectories. In addition, standard RL does not account for real-world effects such as latencies of camera equipment or policy inference. As illustrated in Figure 5.3, the lack of consideration of those latencies in the wire-loop use case leads to time intervals between separate steps where the robot does not move until the next action is provided.

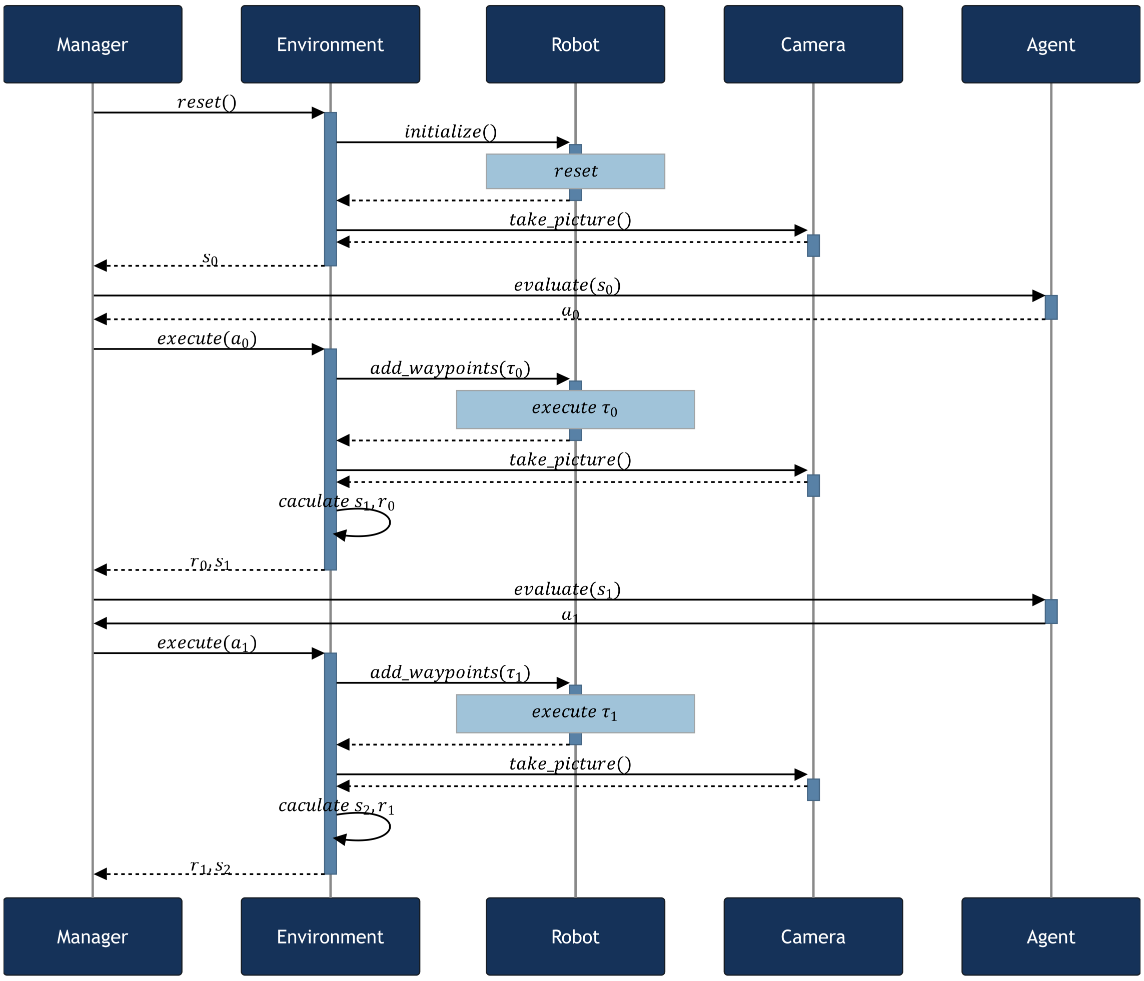


Figure 5.3 Sequence diagram of a standard RL framework. Due to the time-discrete nature, real-world latencies lead to non-continuous robot motion.

One strategy to achieve continuous robot motion inside the time-discrete RL framework is to use an action space based on a derivative of the position-based trajectory, such as the velocity or acceleration. With this option, the robot is able to continue moving while the next action is determined. However, it is not trivial how a terminating event, e.g. a collision, during this time interval is to be incorporated into the synchronous communication framework. If on the one hand this negative event is reflected in the next reward, it is wrongly associated with an action that was not executed. On the other hand, associating the event to the correct action requires a process for switching out the previous reward already communicated by the environment. In addition, using position or angle derivatives as control parameter makes the resulting trajectory dependent on the length of a given time interval. With latencies being unpredictable by nature, the time interval in between actions may vary, leading to noisy states and non-deterministic behavior. This effect can be mitigated by minimizing latencies by proper system design, specifically using real-time operating systems, high-performance hardware, and low-latency communication protocols.

To enable the use of DRL with position-based control, a reformulation of the standard DRL framework is necessary to enable parallel decision-making and execution. To this end, this thesis proposes an asynchronous execution framework depicted in Figure 5.4, which explicitly considers the implications of real-world latencies. The framework is outlined in the following.

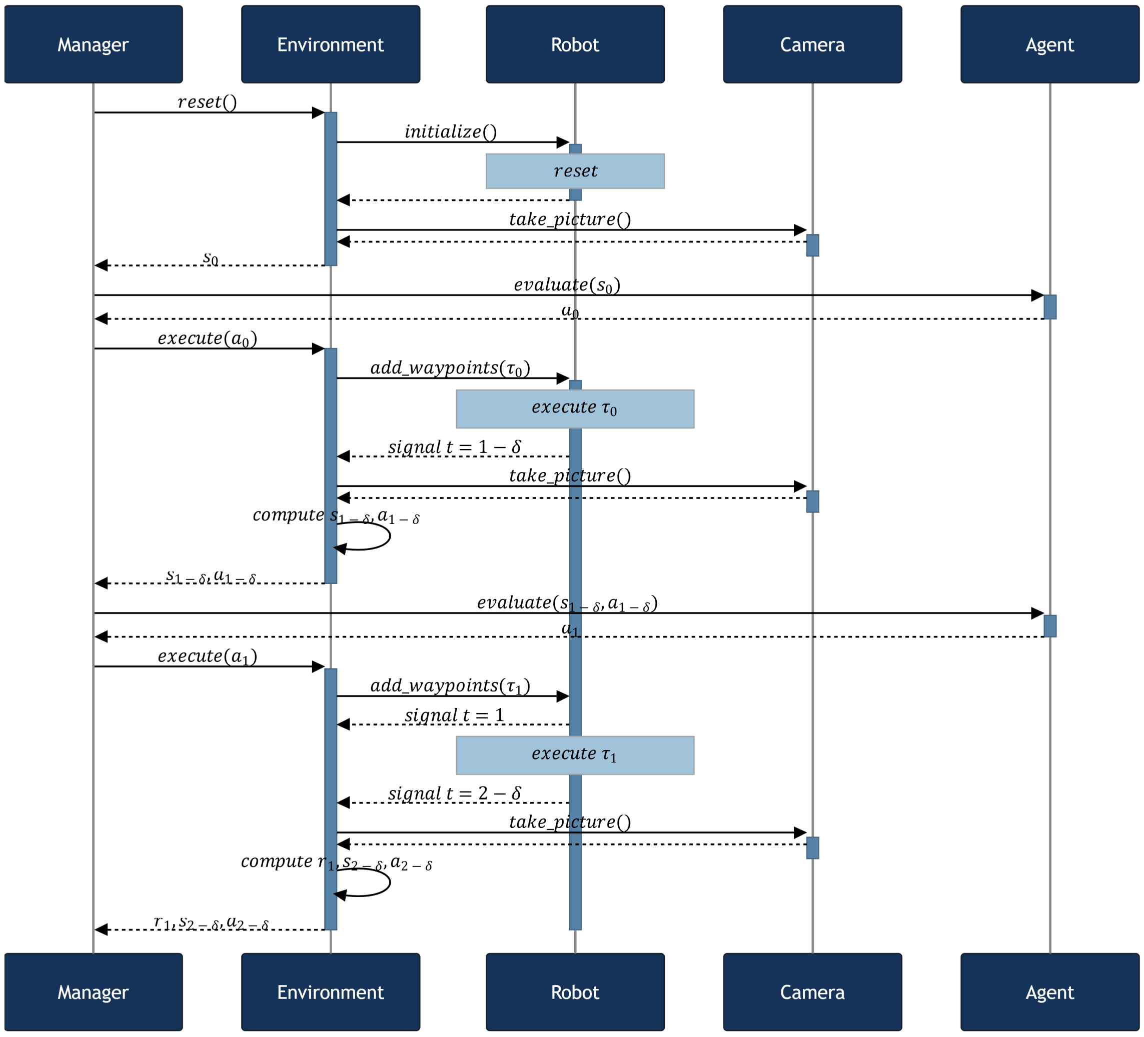


Figure 5.4 Sequence diagram of the proposed asynchronous RL framework. The next action is determined based on an intermediate state and provided to the robot in time to enable a continuous transition between actions.

In general, the execution of the first, the last, and the intermediate actions in a trajectory require separate treatment. The execution of the first action in an episode a_0 results in the transition to the next state s_1, which is required to determine the next action a_1. However, as the first action is not completed at this stage, neither the next state s_1 nor the reward r_0 is available. Instead, an intermediate state

$s_{1-\delta}$ is observed. Here, $\delta \in [0,1]$ is the time interval before the next state, at which the intermediate state is captured. To preserve the Markovian property of the environment, the remaining action $a_{1-\delta}$ must be observed as well. As outlined in chapter 3.1, previous work has proposed to either provide additional features to the policy, such as the latency or action to be executed, or to use a dynamics rollout model to predict the future state. In this framework, an agent is provided with the intermediate state and remaining action $(s_{t-\delta}, a_{t-\delta})$. This enables both state-action based policies $\pi(a_t|s_{t-\delta}, a_{t-\delta})$, and using explicit dynamics rollout models $\hat{s}_1 = p(s_{1-\delta}, a_{1-\delta})$.

For any subsequent execution of actions except the last step in an episode of length T, a_k for $0 < k < T$, the 'step' function returns the intermediate state $s_{k+1-\delta}$ and $a_{k+1-\delta}$ required for policy inference, as well as the now available true state s_k and reward r_{k-1} associated with the previous action. At this point the experience tuple $((s_{k-1-\delta}, a_{k-1-\delta}), a_{k-1}, r_{k-1}, s_k)$ is complete. Note that for the state the intermediate state-action pair is used, since it was provided to the policy for inference. The next state however is the true state s_k, since it is the state resulting from the action execution.

The last action execution a_T depends on when it becomes clear that the state s_{T+1} is terminal. If on the one hand it is known that the action a_T is the last one, e.g. due to the maximum number of episode steps being reached, the last reward r_T can be determined after the final action is completed. Thus, the last execution provides the next state s_{T+1}, the previous reward r_{T-1}, and the last reward r_T. If on the other hand, the episode is terminated due to an unforeseen event, e.g. a collision, the action a_T is treated like any previous action, resulting in the next state s_{T+1} and the previous reward r_{T-1}. With the attempted execution of the subsequent action a_{T+1}, it will become clear that the episode was already terminated. Thus, the action a_{T+1} is not executed, and the function only provides the reward r_T to complete the last experience tuple.

5.3 Dynamics Rollout Module Development and Validation

In the following, the latter approach is chosen for further investigation. Albeit the generation of high-dimensional images is challenging, the approach decouples the synchronous policy model from the dynamics rollout model, which can be developed independently. In contrast, adding latency features or actions to the state space produces policies which are only applicable to the specific latency range, or action space definition.

5.3.1 Dynamics Rollout Module Architecture

To achieve dynamics rollout of high-dimensional states, such as the camera images of the wire-loop use case, this thesis investigates the applicability of conditional Generative Adversarial Networks (GANs), which have been shown to generate high-quality, realistic images [26]. In general, a dynamics rollout model constitutes the generator of a GAN, which aims to model the environments transition probability function $\mathcal{T}(s'|s,a)$. The generator is trained on a dataset of transitions (s,a,s') sampled from the environment. The utilization of (s,a) as input makes the GAN conditional, as the generator should not only produce realistic images, but also account for the dynamics resulting from the previous state-action pair. To enhance the quality of the generated images, a discriminator $d(s)$ is trained to distinguish between real and generated images.

In a preliminary experimental study, different GAN architectures were evaluated, identifying the pix2pix architecture proposed by Isola et al. [77] as best suited. The architecture is based on U-Net, which is an autoencoder with skip connections between the encoder and decoder layers to better preserve high-dimensional image features [146]. Also, the pix2pix framework employs a patch discriminator, which focuses on local image patches instead of the entire image. The final architecture is depicted in Figure 5.5, for which the following design decisions and adjustments were made:

- **State Dimensionality:** As the pix2pix architecture was designed for images of size 256x256, it was decided to use a higher image resolution of 480x360x3 in the wire-loop use case, which is the resolution used to capture the images prior to downsampling to 40x40x3.
- **Action Preprocessing:** With the insight that the desired dynamics rollout constitutes an 2D image translation task, the action fed into the rollout model is reduced to the 3D vector of translation and rotation, analogous to the action space used in the previous baseline setup. In addition to reducing the action to the relevant information, this also makes the rollout model applicable across different action spaces, such as the Bézier curve action spaces, for which determining the required reduction is straightforward.
- **State-Action Encoding:** To condition the generator on both state and action, the original pix2pix architecture must be modified. As the state is high-dimensional and the action is low-dimensional, the action is run through a separate encoder to produce a latent vector of the same size as the pix2pix autoencoder bottleneck latent vector of size 512. This allows for a concatenation of the latent vectors. The resulting joint vector of size 1024 must be reduced back to the original bottleneck size of 512 in order to feed it into the decoder, which is achieved by a single MLP layer.
- **Multi-Patch Discriminator:** The original pix2pix architecture utilizes a patch discriminator, which focuses on local image patches instead of the entire image. During the experimental process, it was found that using three discriminators instead of one significantly improves the quality of the generated images. The results reported below use discriminators for patch sizes 4x3, 8x6, and 16x12. The patch sizes account for the 4:3 aspect ratio of the state images.

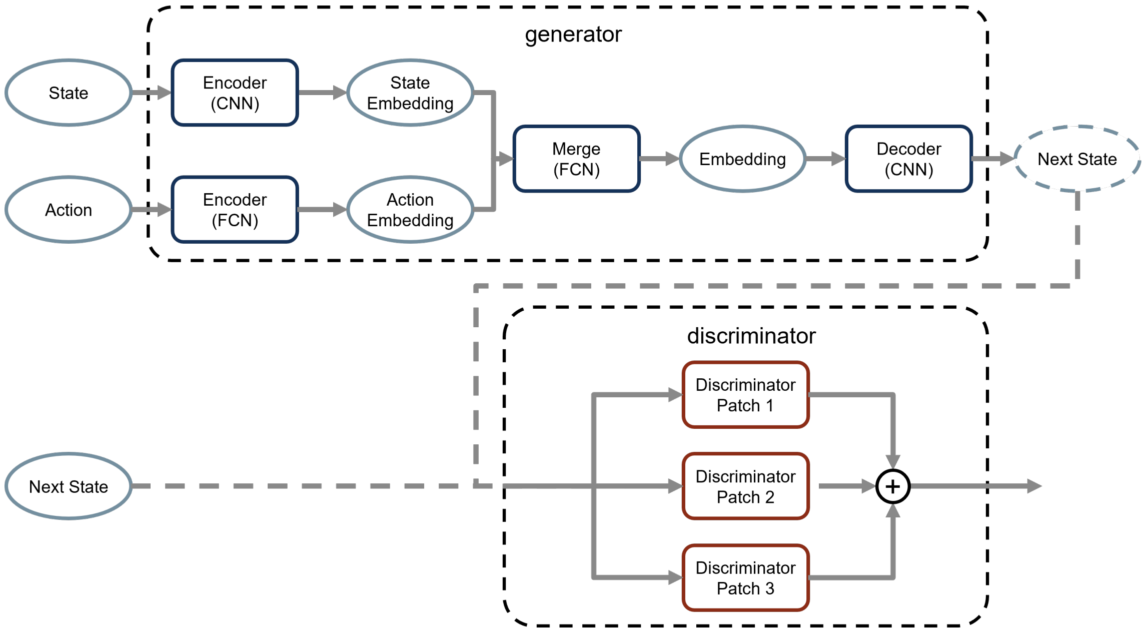


Figure 5.5 The dynamics rollout module is based on the pix2pix architecture [77]. An encoder-decoder generator produces the next state image based on the current state and action. A patch discriminator attempts to distinguish between real and generated images to improve the quality of the generated images.

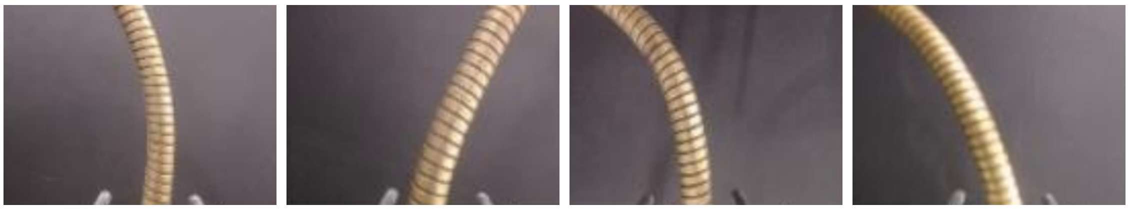
5.3.2 Dataset Collection and Training Results

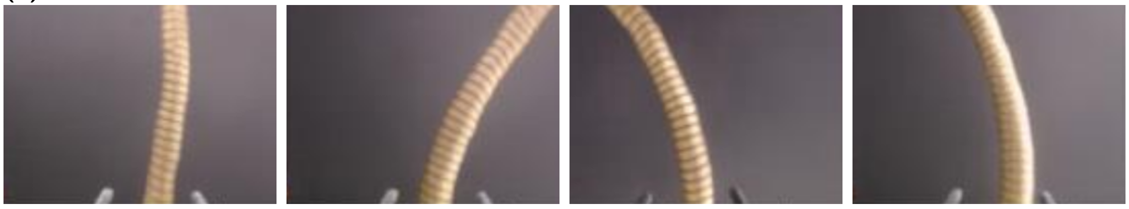
To collect a suitable dataset to train the dynamics rollout model, a manual data collection process was conducted. First, a lightweight user interface was developed which allows for manually specifying actions to be executed on the robot. This allows for generating trajectories which solve a given wire shape. To increase the amount of data collected per manually specified trajectory, two strategies were employed. On the one hand, each trajectory that solves a given wire shape in one direction can be inverted to produce a trajectory which solves the same wire shape in the opposite direction. This effectively doubles the amount of data collected. On the other hand, each action can be split into multiple smaller actions, which follow the same trajectory. This not only increases the amount of data, but also allows the human to teach in trajectories at maximum action magnitude, and automatically enrich the dataset with smaller action variations. It was decided to divide each action in 2 and 3 smaller actions, which increases the data amount by a factor of 6.

These two strategies were combined, resulting in a total increase of the dataset size by a factor of 12. A dataset of approximately 10,000 transitions was collected and used for training the dynamics rollout model.

In general, the dynamics rollout model generates promising results, as depicted in Figure 5.6. Despite the relatively low number of image samples considering the higher image resolution, the trained model is able to correctly predict the next state in the majority of test cases. Both the wire shape compared to the ground truth, and details such as the fork tips and the wire stripe pattern are reproduced with an impressively high quality.

As to be expected, the dynamics rollout fails for large action magnitudes, as can be seen from the examples provided in Figure 5.7. In such cases, the camera shifts to


(a) State


(b) Predicted next state

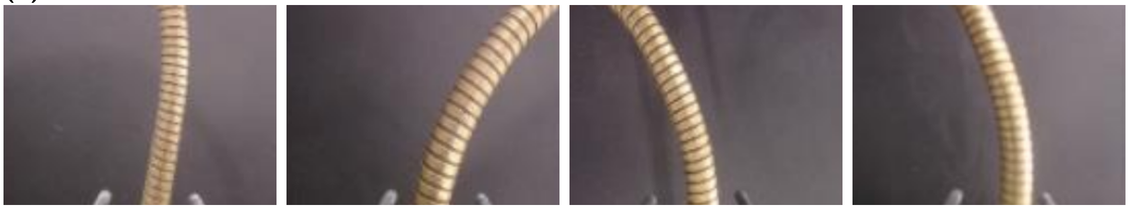

(c) Ground truth next state

Figure 5.6 Examples of dynamics rollout model performance. Based on the observed state and action (not shown), the model predicts the resulting next state. Compared to the ground truth next states, the model is able to predict the wire shape with an impressively high quality for small action magnitudes.

a completely different section of the wire, whose shape cannot be inferred from the previous state. Understandably, the dynamics rollout model either generates only a partial wire shape, or hallucinates multiple potential wire sections.

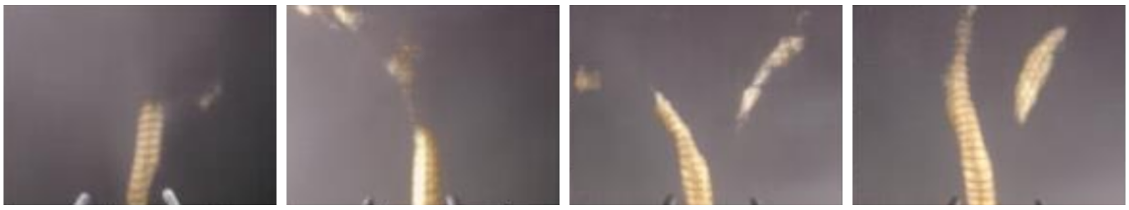


Figure 5.7 Failure cases of the dynamics rollout model. The model is unable to predict the next state for large action magnitudes and potentially starts to hallucinate.

Self-evidently a dynamics rollout model can only predict what it can infer from the current state. This insight can most likely also be applied to the case when providing latency features or actions to the policy, as the policy is expected to implicitly perform a dynamics rollout. While further empirical validation is needed to confirm this hypothesis, preliminary analysis suggests that asynchronous DRL implementations require the state space to encompass at least two subsequent movements. This constraint inherently restricts the maximum achievable action magnitude of asynchronous DRL agents to approximately half that of their synchronous counterparts.

5.4 Exploration of Design Choices

With the previously outlined asynchronous execution framework, the design of the Bézier curve action space, and the dynamics rollout model, all components for realizing asynchronous DRL in the wire-loop use case are in place. However, some design choices remain to be made.

First and foremost, the integration of the dynamics rollout model into the asynchronous execution framework must be reflected on. Given the previous results of the dynamics rollout model, the capability of producing high-quality future state predictions is promising. However, it was also shown that for large action magnitudes, the dynamics rollout model is unable to predict the next state, as the required information cannot be inferred from the current state. Addressing this limitation is already considered in the proposed asynchronous execution framework, and can be achieved by using an intermediate state closer to the next state. In the above framework formulation, this means that δ should be chosen as small as possible, while being large enough to account for the latency required to determine the next action for achieving seamless transitions. Alternatively, the action space must be limited, as mentioned previously.

On reducing the δ value in the wire-loop use case, it was experimentally found that the latency of taking a picture, evaluating the policy, determining the new waypoints, and sending them to the robot is below 50ms. With a maximum velocity of $v_{max} = 50mm/s$, which was chosen to minimize the risk of hardware damage, this translates to a distance between the intermediate state and the next state of $2.5mm$. During this exploration, it was also found that with this distance, the intermediate state is similar enough to the next state to prompt the hypothesis, that in the given setup dynamics rollout is not required. While this breaks the Markovian property of the environment, it is a reasonable assumption for the practical application in the given use case setup. This hypothesis is supported by the work of Derman et al. [42], who show that action delay MDPs can be converted into standard Markovian MDPs with stochastic transitions. In the case of the wire-loop use case, the expectation is that the added noise corresponding to the small delay is negligible enough to be compensated by the agent. In addition to the major design choice of omitting the dynamics rollout model, the following design choices were made:

- **Action space constraints:** To retain comparability with the baseline experiments, the Bézier action spaces are limited to produce curves of approximately $l_{max} = 4.5\ cm$, which is chosen based on the maximum possible displacement of $l_{max,baseline} = \sqrt{2 \cdot (3\ cm)^2} = 4.24\ cm$ in the baseline setup.

- **Reward function:** Since the new Bézier action space no longer contains the forward motion component $a_t^{forward}$ previously used in the reward function, the term was removed. This equates setting the tuning parameter β to 1. While similar reward function components were tested for the Bézier action space, e.g. by including the Bézier curve length to incentivize longer actions, no significant impact was observed. While this changes the reward function, the forward motion component was shown to be beneficial for the learning process in the baseline scenario (cf. chapter 4.2.2). Therefore, omitting it in the following experiments is not expected to provide any advantage to the

Bézier-based agent, but may actually make learning more challenging. As a result, any improvements observed over the baseline can be attributed to the new approach itself, rather than to changes in the reward function.

- **Waypoint spacing:** Asynchronous decision-making is achieved by sampling each Bézier curve with a maximum distance constraint of $5mm$ between waypoints, which results in an effective spacing between $2.5mm$ and $5mm$, and triggering the capture of an intermediate state once the second to last waypoint is reached.

5.5 Asynchronous Wire-Loop Experiments

Given the previously outlined design choices, the performance of four different agents is evaluated. Two agents, Cartesian-2 and Cartesian-3, use the Cartesian encoding, while the other two agents, Polar-2 and Polar-3, are based on the polar encoding. Every encoding strategy is used with both quadratic and cubic Bézier curves, which is indicated by the respective suffix number. The results are depicted in Figure 5.8.

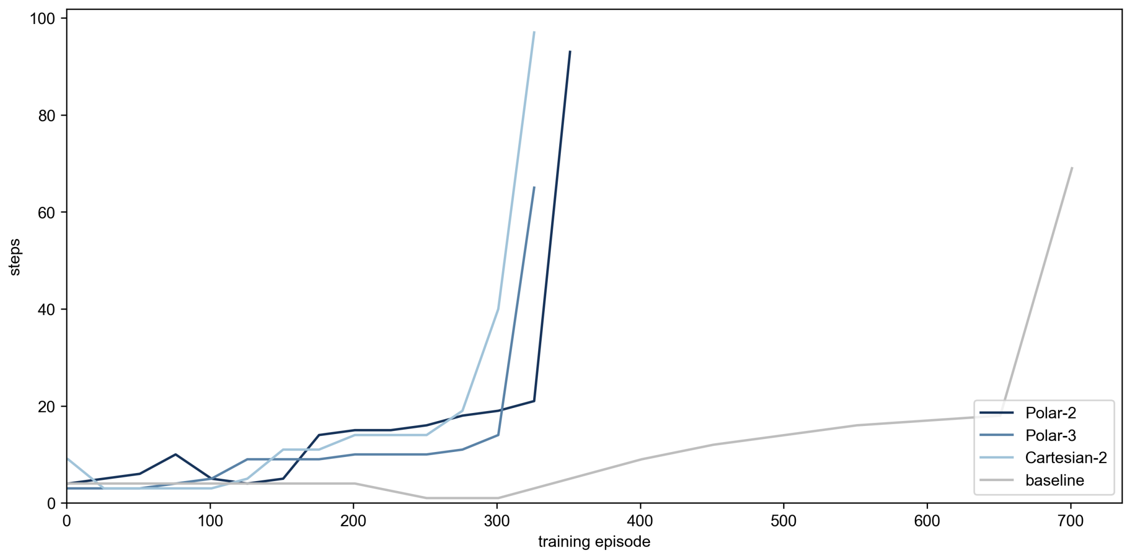


Figure 5.8 Training results of the Bézier agents. The Cartesian-3 agent was unable to converge to a solution, while the other agents outperform the baseline agent. Adapted from [158].

In general, the results demonstrate a clear improvement over the previous baseline agent regarding the required learning effort, reducing the number of episodes by a factor of 2. The Polar-3 agent achieved the overall best performance, reaching a solution with the least amount of steps (65 steps) in the shortest amount of time (325 episodes).

The performance curve of the Cartesian-3 agent is omitted, as this agent was unable to converge to a solution. This may indicate a lack of scalability of the Cartesian encoding method. As the placement of control points with the Cartesian encoding is arbitrary, the potential for unfeasible trajectories increases with the number of control points to be placed. Further, the ANN-based policy was initialized so that the initial actions are close to the center of the action space. In the case of Cartesian encoding, this means that all control points are initially placed close to the center of the allowed area. This starting position is very unstable, as it is possible for the control points to be arranged in such a way that the Bézier curve is self-intersecting. The results therefore suggest that the Polar encoding with its higher constraint on the placement of control points is more stable, and thus more suitable for the wire-loop use case.

To quantify the effectiveness of the asynchronous execution framework, the wall clock time is measured for the baseline agent and the Polar-3 agent. While the former requires approximately two minutes to reach the end of the wire, the latter achieves the same task in only 30 seconds. This significant difference highlights the benefit of performing asynchronous decision-making to improve the overall efficiency of DRL agents in industrial robotics applications.

5.6 Summary

This chapter primarily focused on the first RQ, which prompted an adaptation of the standard RL framework to enable smooth, continuous robot movement. An asynchronous DRL framework was proposed and validated for the wire-loop use case. The main concept of the framework is to introduce intermediate states which are observed before the current action execution is completed, which allows an agent to preemptively determine the next action and achieve continuous transitions between individual actions. The continuous transitions are complemented by a constrained actions space, whereby parametrized Bézier curves are used to enforce smoothness between individual trajectory segments. In the experimental validation, it was shown that the constrained action space has the additional benefit of reducing the required learning effort by a factor of 2. Further, the faster execution through asynchronous decision-making reduced the execution time by a factor of 4.

Based on the previous related work proposing the use of dynamics rollout models to anticipate the next state and restore the Markovian property of asynchronous control, the possibility to generate such a dynamics rollout model for high-dimensional image-based state spaces was demonstrated using a GAN architecture. Besides the proof-of-concept validation of the rollout approach, the experimental results also showed the boundaries of the asynchronous framework. In particular, asynchronous decision-making relies on the ability of an agent to infer the next state from the intermediate state, which degrades with an increasing distance between the intermediate and next states.

The final experimental validation however revealed that the added overhead of developing a dynamics rollout model is not required to achieve smooth, continuous robot movement. Instead, the asynchronous execution framework can in practical application be simplified by using intermediate states which are far enough from the next state to account for the latency of the decision-making process, but near enough to be considered similar. While this does not discount the approach of using a dynamics rollout model as a general solution, it rather validates the final approach of introducing intermediate states.

6 Sim2Real Transfer of Perception Modules

"It's not what you look at that matters, it's what you see."
- Henry David Thoreau

As industrial robotic applications grow more complex, so does the need for high-dimensional sensor data to fully capture the state of real-world environments. This is a challenge for the application of DRL, as according to the curse of dimensionality [118], the required training effort grows exponentially with the state space dimensionality. Fortunately, providing an agent with raw image data to deduce process-relevant information in an end-to-end manner is not the only option. Separating the feature extraction from the decision-making policy promises several advantages: First, it is possible to employ techniques and existing models from the computer vision domain [37, 216]. Second, vision- and decision-making aspects can be considered in isolation, and this separation of concerns should speed up the overall development process. Finally, an explicit interface between vision and decision-making with human-interpretable features allows for better transparency and explainability of the agent's decision-making process, which is not given with end-to-end policies [36, 117]. The approach of modularization however does not come without drawbacks and risks. Designing the interface between modules is a manual intervention, which relies on a thorough understanding of the process at hand. Any error in feature extraction may withhold vital information from the policy and prohibit it from properly learning the task. In contrast, end-to-end policies are in principle able to recover through continued learning and adaptation to new situations. This chapter addresses these challenges towards answering **RQ-2: How can perception modules for vision-based robotic tasks be developed in simulation, and how can their structure be designed when the relevant low-level features are not immediately obvious?**

For the object picking use case, the DRL approach struggled to learn the task in the Fetch environment (cf. chapter 4.3.2.2), which already presumes the object pose to be known. Instead of making the scenario even harder to learn by using raw camera images, it was deemed more promising to explicitly extract the object pose from the image by means of CV. In this chapter, existing YOLO models for 6D object pose estimation are employed to extract object poses from camera images. In particular, simulation-to-reality transfer is considered to address the challenge of acquiring a real-world dataset with 6D pose annotations.

For the wire-loop game use case, the goal is to develop a perception module which is able to extract the process-relevant information from the camera images, which is the wire and the loop. A three-step investigation is conducted to determine a suitable feature representation. First, unsupervised autoencoders are used to compress the image data into a low-dimensional embedding. Second, the attention of trained DRL agents is visualized to validate the hypothesis that an agent focusses on the same features as a human. The attention visualizations in the wire-loop use case were previously published by the author [154]. Finally, the insights of attention visualization are incorporated into the image compression process.

6.1 Pose Estimation Modules

Image-based object detection is a major research area in the computer vision domain. Numerous methods have been proposed to address this problem, ranging from traditional hand-engineered feature detectors to modern DL approaches (cf. chapter 2.2.3). For this work, the YOLO6D model is selected for object pose estimation in the pick-and-place use case [179]. Specifically, the implementation provided by Viviers et al. [195] is taken as foundation, which translates the original YOLO6D approach by Tekin et al. [179] based on YOLOv2 onto the updated, more performant YOLOv5 architecture. Viviers et al. [195] provide an extensive comparison of the YOLO6D model with other SOTA methods and demonstrate both highest accuracy and fastest inference time on a public dataset, which are both important metrics for real-world robotic applications.

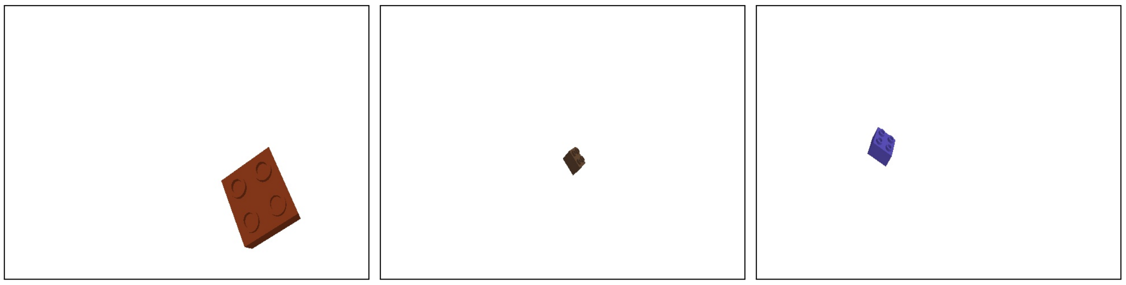
To train a YOLO6D model, a dataset with 3D annotations is required. This poses a major challenge, since 3D annotations are not easily obtainable for real-world images. To further complicate the matter, the YOLO6D approach enhances the variance of the training data through extensive data augmentation, most notably by using a binary segmentation mask to isolate the object of interest and replace the background with a random image. Self-evidently, the segmentation masks must also be generated in addition to the 3D annotations. Instead of constructing a suitable hardware setup to capture 3D annotations, e.g. by using markers whose 3D pose can be identified in an image in combination with manually measuring the objects pose to such markers, the following investigation focuses on the utilization of simulation data to pretrain a YOLO6D model for real-world object pose estimation.

6.1.1 Simulation Dataset

Constructing a simulation dataset requires a suitable simulation environment. In this case, the physics simulation engine Pybullet [35] is selected due to its Python

API, as well as its explicit focus on sim-to-real transfer for robotics. The simulation setup necessary for this dataset features just two elements: A mesh model of the relevant object, and a camera. Albeit the requirement of an object mesh model poses a limitation to the approach, it can be argued that in industrial settings digital models of relevant objects are often times available, or can be acquired e.g. through manual modeling, 3D scanning [47], or DL-based reconstruction techniques from 2D images [62].

To create a diverse dataset, camera images and corresponding segmentation masks of the object, a toy building brick, are taken from different viewpoints, distances, and orientations. Examples are depicted in Figure 6.1. Since only the relative pose between object and camera is of interest, it was decided to place the object at origin and move the camera around the object. In accordance with the concept of DR, the color of the toy brick is randomized with every image. As described above, variation of the background will be performed as a data augmentation step by the selected YOLO6D training framework [195].


(a) The toy brick object is captured from different viewpoints and distances. In addition, the color of the object is randomized with every image.

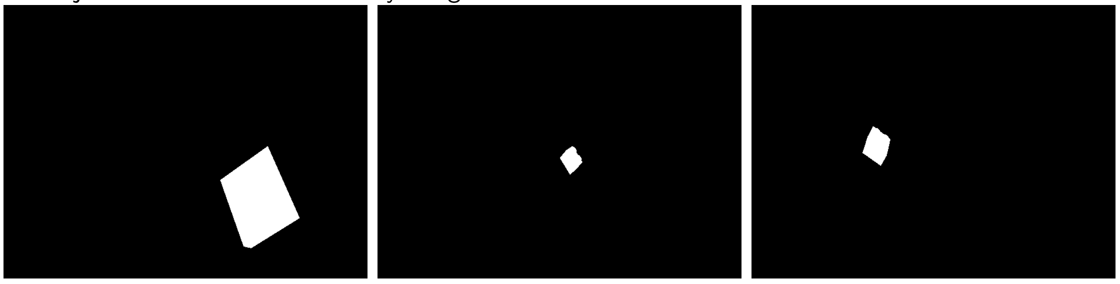

(b) The corresponding segmentation masks of the toy brick object are recorded to later replace the background with random images.

Figure 6.1 Examples of the simulation dataset for the perception module of the object picking use case.

Simulating the camera requires two 4x4 matrices: The view matrix ${}_{camera}T^{world}$ describes the transformation from world to the camera coordinate system, and will be randomized to capture the object from different viewpoints. The projection matrix ${}_{image}P^{camera}$ contains the camera's intrinsic parameters, such as focal length and optical center, and provides the projection of 3D points into the 2D image plane. The projection matrix is constant, as the camera's intrinsic parameters are constant. Determining the camera's intrinsic parameters is an ubiquitously used technique in the computer vision domain [212], and is thus not further elaborated on.

To record a data point, the object color and camera pose are randomized. Next, an image and a corresponding segmentation mask are captured using the Pybullet API. Finally, the object key points ${}_{world}K = [x, y, z, 1]$ defined in homogeneous coordinates are projected into the 2D image plane to obtain the final ground truth annotation using a simple matrix operation

$$_{image}\hat{K} = {}_{image}P^{camera} \cdot {}_{camera}T^{world} \cdot {}_{world}K \tag{6.1}$$

Note that the projection must further be normalized by the fourth component of $_{image}\hat{K}$ to obtain the final projection

$$K = \frac{_{image}\hat{K}}{\hat{w}} = [\frac{\hat{u}}{\hat{w}}, \frac{\hat{v}}{\hat{w}}, \frac{\hat{d}}{\hat{w}}, \frac{\hat{w}}{\hat{w}}] = [u, v, d, 1] \tag{6.2}$$

For YOLO6D, only the 2D coordinates (u, v) are used and stored as ground truth labels, and the depth d is discarded.

6.1.2 Experimental Evaluation

With the outlined data generation process, a dataset of 10,000 images is created, ensuring sufficient data by using a 10 times larger dataset than Viviers et al. [195]. The dataset is split into a training, validation, and test set with a ratio of 80/10/10.

A YOLO6D model is trained on the training set for 500 episodes using the standard parameters provided by Viviers et al. [195]. The trained model achieves a mean 2D corner error of 0.93 pixels, a mean translation error of 9.3 mm, and a mean orientation error of 2.23 degrees. While the results do not surpass the current SOTA, Yang et al. [206] report winning the 2024 Robotic Sim2Real Challenge with pose estimation precision within 1 cm and 2 degrees. For this object picking use case, the accuracy is expected to be sufficient for reliable object picking. The visual inspection of the bounding boxes in Figure 6.2 confirm the model's ability to locate the bricks in the image with a high precision.

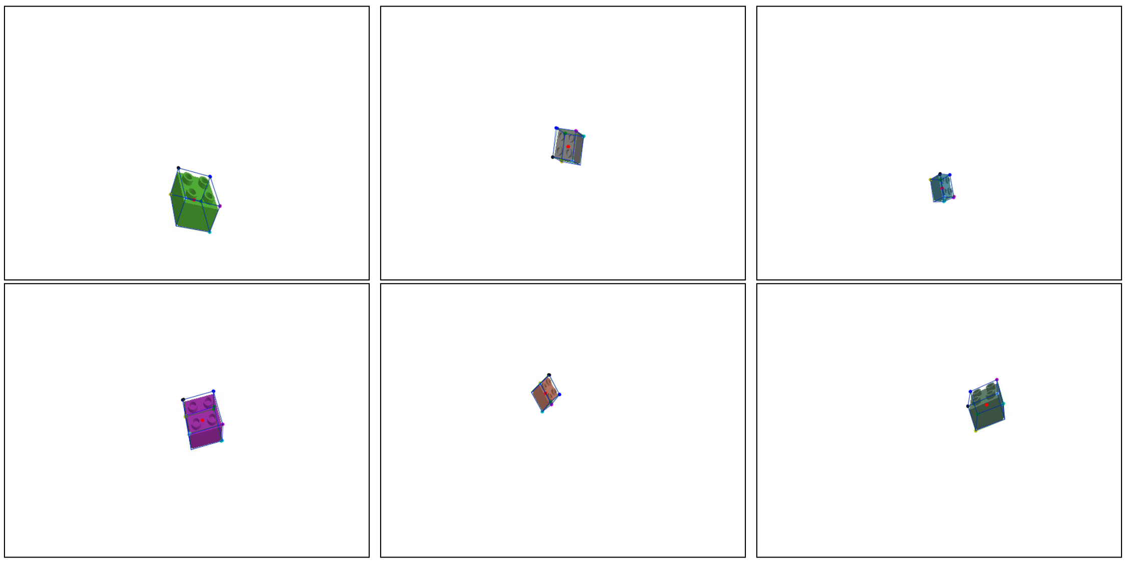
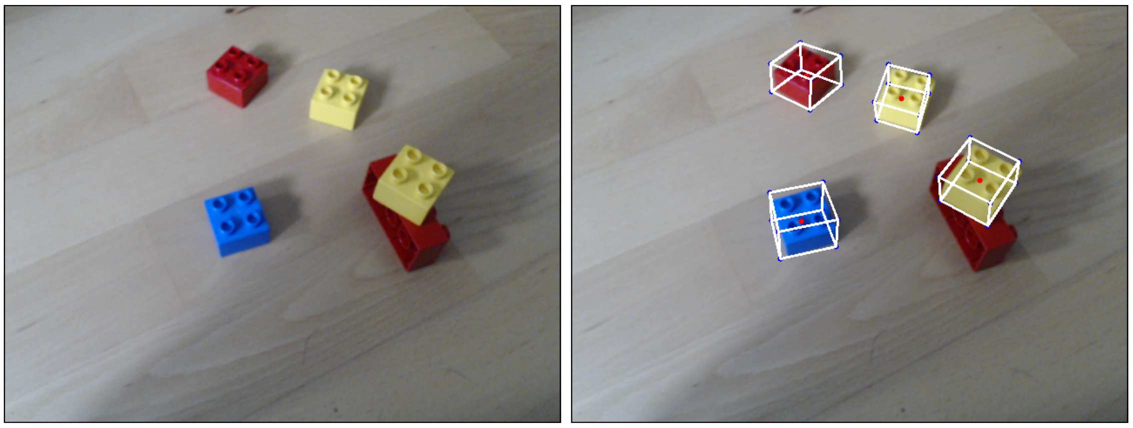


Figure 6.2 Bounding box predictions on the simulation test set. The perception module is able to accurately locate the object in the image.

With the YOLO6D model trained, the next step is to transfer it to the real-world and integrate it into a pick-and-place pipeline. Deploying the perception module requires transforming the YOLO6D model predictions from the camera frame into the robot base frame, which is possible due to the camera calibration procedure already present in the demonstrator (cf. chapter 1.3.2). The camera calibration procedure is used to transform the YOLO6D model predictions from the camera frame into the robot base frame.

It was found that directly feeding the camera images into the model was insufficient, with the perception module only sporadically locating a brick. Instead of retraining the model with additional randomization, randomizing the real-world camera images proved a suitable approach. By adding a color jitter, which adds random changes to brightness, contrast, saturation, and hue, the performance of the perception module was significantly improved. Best results were achieved by simultaneously feeding the raw image and multiple jittered versions to the YOLO6D model and averaging the predictions. To balance the decreased throughput of the perception module caused by the increased model inference time, eight jittered versions are used. Figure 6.3 contrasts the performance before and after the color jitter modification. With this setup, an evaluation is conducted, in which object picking is attempted 100 times at the predicted poses based on a single camera frame. This evaluation yielded a success rate of 64%.



(a) Predictions for raw image input. No 2x2 toy brick object is detected.

(b) Predictions for jittered image input. All 2x2 toy brick objects are detected.

Figure 6.3 Integration of the perception module into the real-world. To improve the performance, the model is fed with jittered image inputs.

To understand the prediction behavior of the perception module in the real-world, a stationary brick is placed on the table and the perception module is evaluated over 100 camera frames. As can be seen in Figure 6.4, the perception module predictions exhibit significant noise on a real-world video stream, especially in the z-axis denoting the depth estimation. This noise explains the observed low picking success rate. To counteract this issue, a moving average filter is applied over 15 frames, which smoothes the perception module predictions. The filter size of 15 frames was determined empirically to provide sufficient smoothing while not introducing too much latency.

With this adjustment of smoothing the predictions over time, the pose estimations stabilizes. The evaluation experiment is conducted again, which yielded a success

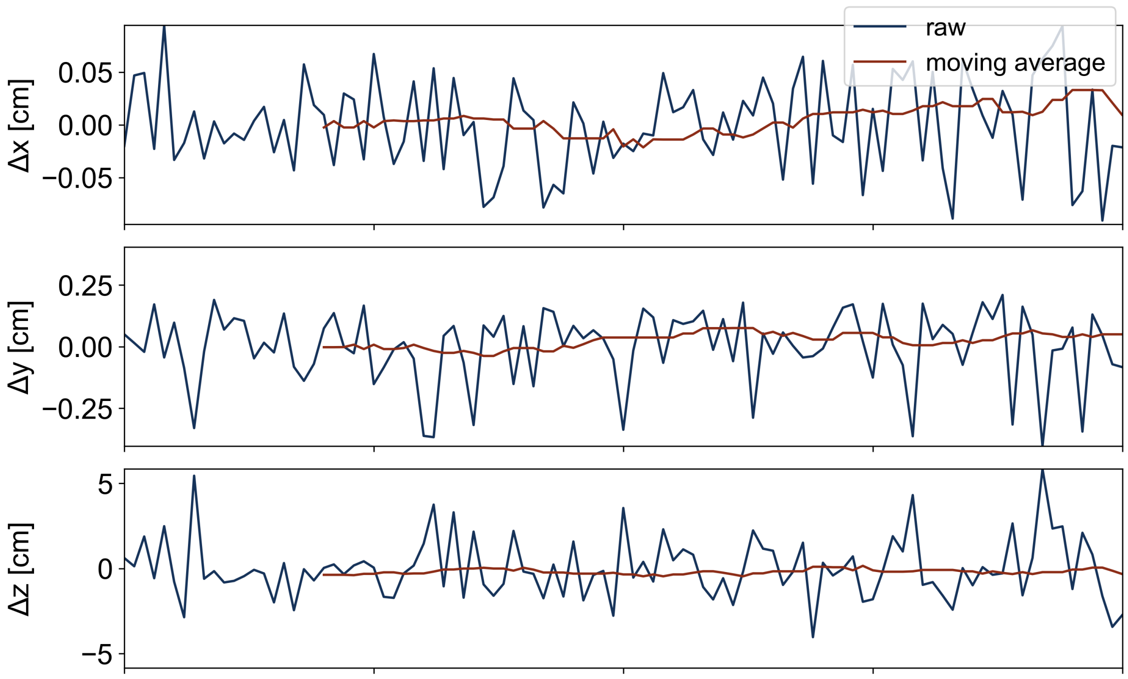


Figure 6.4 Relative variation of perception module predictions for a video stream. The noise is compensated by a moving average filter over 15 frames.

rate of 96%. The positive results of the object picking pipeline validate the modular approach of separating perception from control, as the perception component can be developed and tested independently before integration with the final DRL policy. Further, the remaining development effort for the DRL policy can be limited to the environment simplification already present in the Fetch environment of presuming knowledge about the object pose, which is a major advantage.

6.2 Image Compression Modules

In contrast to the pick-and-place use case, where the process-relevant information in form of the object pose is easily identifiable, the same cannot be claimed for the wire-loop use case. While the scenario is well understood and the relevant components of wire and loop can be quickly pointed out in a state image, the question as to how these features can be effectively extracted is not as straightforward. To approach this problem, the following investigation will take a three-step approach. First, unsupervised DL is considered for compressing state images into latent space representations using a real-world dataset. To remove the dependency on real-world data, a semi-supervised DL approach is taken to compress state images into latent space representations using a simulation-based dataset collected with DR. Third, the possibility to inject human domain knowledge into the semi-supervised model to enhance sim2real transferability is investigated.

6.2.1 Unsupervised Image Compression

Before attempting to explicitly inject domain knowledge into a perception module, it is first investigated whether process-relevant features can be extracted through unsupervised image compression using autoencoders (cf. section 2.2.2). For the wire-loop task, an autoencoder can potentially learn to encode the essential geometric relationships between wire and loop into a compact latent representation, while filtering out nuisance factors like lighting variations or background details. This compressed representation could then serve as a more efficient and robust state space for the reinforcement learning policy compared to raw images. The approach is outlined in Figure 6.5.

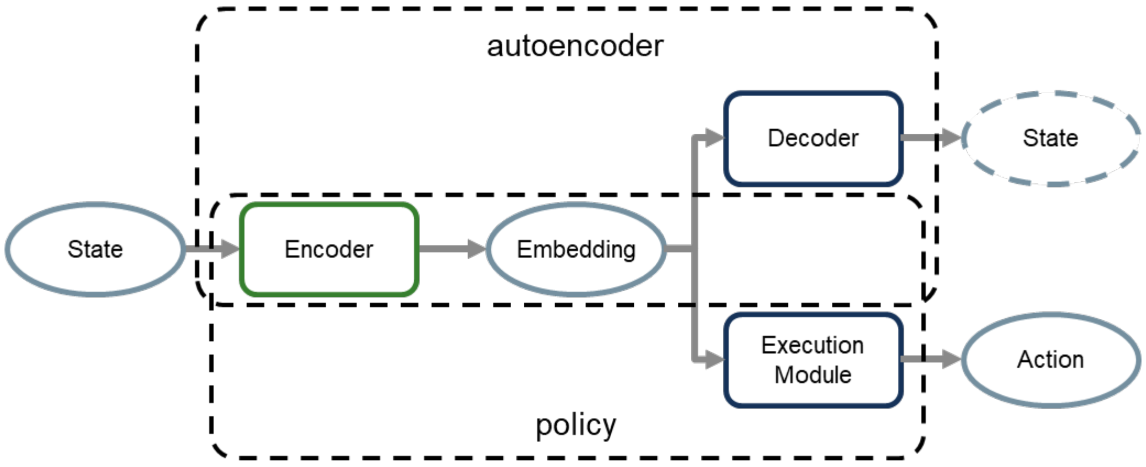


Figure 6.5 Concept of using autoencoders for process-relevant feature extraction. First, the autoencoder is trained using states as input and output to learn a compressed latent representation. Then, the decoder is discarded and replaced with an execution module which takes the learned latent representation as input. The combination of the encoder and execution module forms a DRL policy.

To test this hypothesis, a proof-of-concept investigation is conducted using a convolutional autoencoder architecture and a dataset of real-world images collected from the wire-loop setup. The convolutional autoencoder is chosen due to its ability to learn spatial hierarchies of features, as well as its structural similarity to the already employed CNN section of the DRL agent. The encoder consists of three layers of 32, 16, and 8 filters of size 3x3 and stride 2. Each layer is followed by a batch normalization layer and a LeakyReLU activation function. This final encoder design is an iteration on the original policy CNN architecture with the following changes. First, the number of filters is switched to powers of two, mainly to follow popular conventions aimed at improving training efficiency due to binary memory access of computer systems. This change only slightly affects the number of trainable parameters, and is expected to have a negligible impact. Second, the LeakyReLU activation function is used instead of ReLU to improve training stability. This assumption is evidenced by comparative studies in literature [119, 136]. Third, the MaxPooling layers are replaced by adding stride to the convolutional layers to simplify the architecture. This change is mainly motivated by the findings of Springenberg et al. [170], who show that this replacement is possible without affecting performance.

The encoder effectively compresses a 40x40x3 image into a 4x4x8 embedding, which constitutes a compression down to 2.6% of the original image size. The decoder mirrors this structure to reconstruct the input image, using transposed convolutional layers to upsample the latent representation back to the original image dimensions. The network is trained using a basic mean squared error reconstruction loss between the original and reconstructed image.

For the training dataset, experience data logged from previous experiments is used. The raw dataset contains 9420 experience tuples from 2391 trajectories. With the data being the result of DRL training runs, it is highly unbalanced. This is due to the fact that the reset routine employed on the real-world hardware setup moves the loop back to the previous state when the wire is touched. This means that in many cases, the first state of a trajectory is virtually identical to the last state of the previous trajectory. Examples are provided in Figure 6.6.

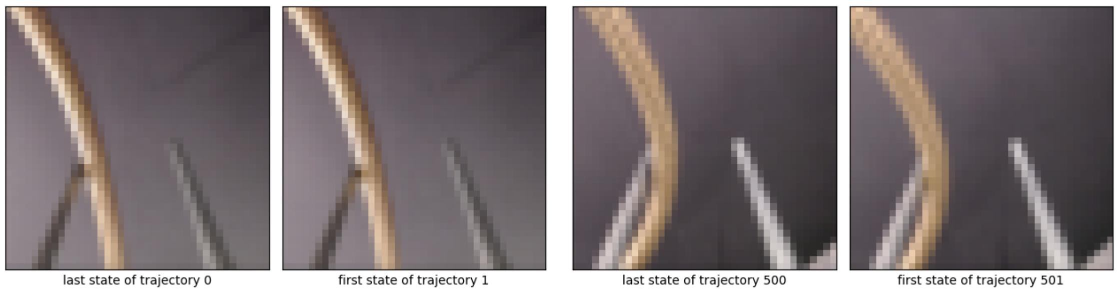
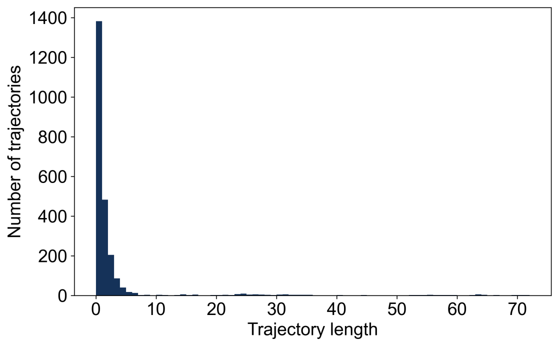

last state of trajectory 0
first state of trajectory 1
last state of trajectory 500
first state of trajectory 501

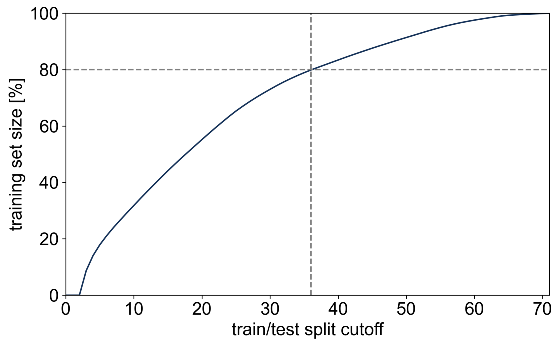
Figure 6.6 The reset routine of the real-world hardware setup leads to many similar states in the dataset. In many cases, the first state of a trajectory is virtually identical to the last state of the previous trajectory.

The over-representation of reset states is significantly amplified by the fact that in the beginning of the training, the agent touches the wire immediately after reset, thereby visiting the same state over and over again. This is reflected in the trajectory length distribution depicted in Figure 6.7a, showing that 1382 (57.8%) of the trajectories are 1 step long, which translates to 14.6% of the total experience tuples.

The major issue with this unbalanced dataset is that any random split of the data into training and testing set will certainly result in the test set containing a lot of



(a) Distribution of trajectory lengths.

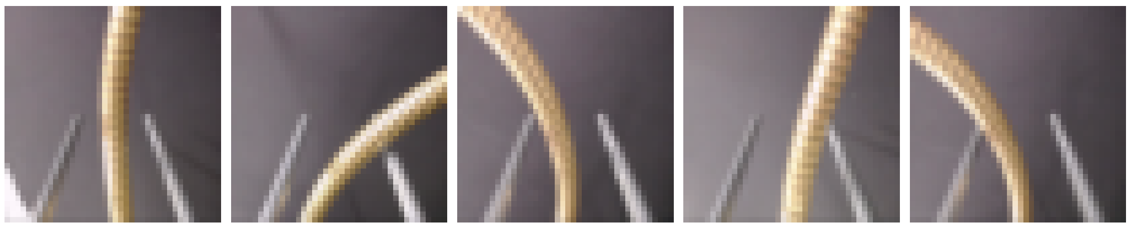


(b) Train/test split cutoff selection.

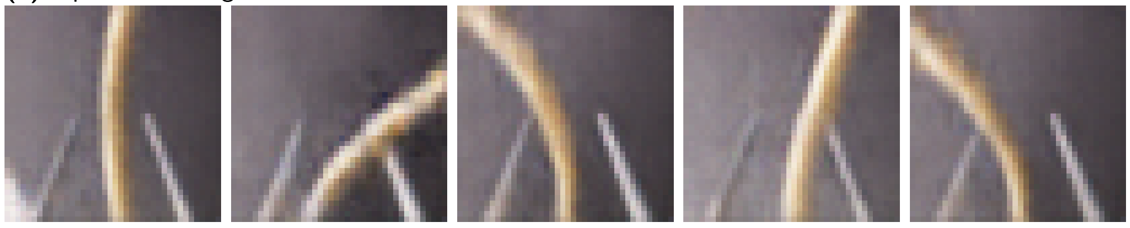
Figure 6.7 Explorative dataset analysis to identify the optimal train/test split.

states which the model is exposed to during training. This bears the risk that the model overfits to this specific data distribution, and this issue not reflecting in the test performance. To counteract this, the last experience tuple of each trajectory is removed, reducing the dataset to size 7030. Further, the train/test split is conducted by selecting all experiences which are at the end of long trajectories, with the assumption that these states constitute wire sections which have not been frequently visited during training. A time step of 32 is used as cutoff threshold to achieve an 80/20 train/test split (cf. Figure 6.7b).

The previously described autoencoder is trained on the state images. A qualitative evaluation is provided in Figure 6.8, showing the input and reconstructed state images. The results clearly show that the autoencoder is able to reconstruct the input state images with high accuracy. As to be expected, the reconstructed images are more blurry, however the wire and loop are still recognizable. Importantly, the shape of the wire is preserved, indicating that process-relevant information is successfully encoded in the latent space.



(a) Input state images.



(b) Reconstructed state images.

Figure 6.8 Quantitative evaluation of the autoencoder. The input and reconstructed state images are shown for five randomly selected states. The reconstructed images are more blurry, however the wire and loop are still recognizable. Importantly, the shape of the wire is preserved, indicating that this process-relevant information is successfully encoded in the latent space.

The final step of this investigation is to confirm that the compressed state space can be utilized by a DRL policy. For this, a policy is created by combining the pretrained perception module, i.e. the encoder, with a freshly initialized MLP analogous to the baseline DRL agent. The policy is then trained on the real-world hardware setup using the existing learning framework. The learning curve is depicted in Figure 6.9.

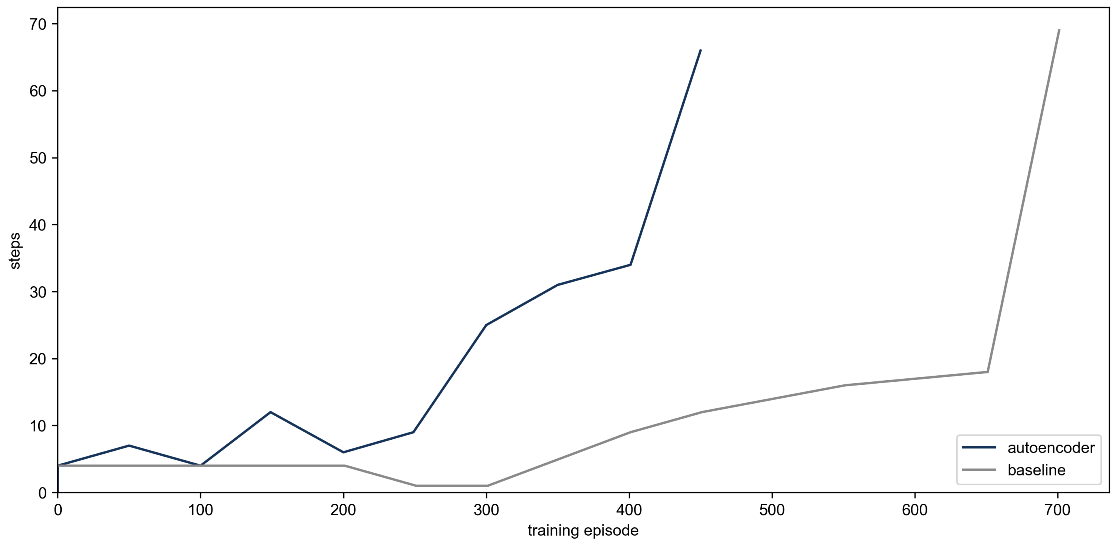


Figure 6.9 DRL results with unsupervised autoencoder as perception module. The agent is able to learn the wire-loop game in just 450 episodes, significantly reducing the training effort compared to the end-to-end baseline agent.

The pretrained perception module enables the agent to learn the wire-loop game in just 450 episodes, significantly reducing the training effort compared to the end-to-end baseline agent. Thus, the initial hypothesis that unsupervised image compression can be used to extract process-relevant features from state images is confirmed.

With this result, this first proof-of-concept investigation is concluded. It is highly likely that the performance gains could be further improved by finding more optimal architecture designs, such as narrower bottlenecks to compress the state space even further, or potentially by freezing the encoder weights and only updating the MLP weights during training. These optimizations were not considered in this study. Instead, the focus of investigation is shifted on addressing two limitations of the current approach, which are the dependence on a dataset of real-world images and the lack of domain knowledge incorporation. The second limitation can be understood in the unsupervised nature of the autoencoder, which attempts to reconstruct the input image regardless of process-relevance. This can partially be seen in Figure 6.8, where in the first example lower left corner a white patch resulting from the edge of the cardboard background, which is clearly process-irrelevant, is reconstructed. Such a lack of process-understanding can always be expected from unsupervised autoencoders, as the reconstruction loss does not differentiate according to process-relevance. This prompts the question of how to inject process knowledge into the perception module, which is the focus of the following investigations.

6.2.2 Explorative Experiments on Agent Attention

To develop methods suitable for injecting process knowledge into the perception module, it is first investigated how vision-based DRL agents perceive the wire-loop task. The goal of this study is to understand what features are considered most relevant for the agent's decision-making as foundation for developing perception modules that retain this information. While it is self-evident that process-relevant features must be incorporated into an agent's decision-making, the actual way in which states are perceived is not. From the computer vision domain, several methods have been developed to make CNNs more interpretable. One popular method is Gradient-weighted Class Activation Mapping (Grad-CAM), which visualizes the regions of an image which most contribute to a CNN's prediction [161]. The results from literature suggest that the models focus on human-interpretable features for supervised classification tasks. To confirm these findings for DRL, the Grad-CAM method is applied to the wire-loop task.

In its original form, Grad-CAM is geared towards classification tasks, i.e. CNNs where the output is interpreted as a one-hot class prediction. This is also reflected in the name "Gradient-weighted Class Activation Mapping" explicitly containing "Class". To make the method applicable to policy CNN predicting continuous actions, the technique needs to be adapted slightly. This can be achieved with two minor modifications. First, classical Grad-CAM restricts the heatmaps to features which have a positive influence on the inspected class output, i.e. increases its value. For regressive outputs however, only the magnitude and not its direction is relevant. Thus, the activation map should highlight areas which change, both increase or decrease, an action output. This is achieved by removing the ReLU activation function in the original Grad-CAM formulation. Second, instead of focussing on each action output individually, the activation maps of all action dimensions are combined.

For this investigation, trained agents are taken from the previous end-to-end sim2real transfer experiments (cf. chapter 4.2.3.2). Agent 1 was trained without DR, agent 3 was trained with partial DR, and agent 6 was trained with full DR. These agents are chosen as they exhibit different transferability, with agent 1 being the least, and agent 6 being the most transferable. Using different state images as input, attention maps are generated with the adjusted Grad-CAM method. The first state is taken from simulation without DR, states 2 and 3 are taken from simulation with DR, and states 4-6 are real-world images. The resulting activation maps are depicted in Figure 6.1, showing the attention of the agents in red and uninfluential image regions in blue. The agent number corresponds to the respective randomization setting ID in the sim2real transfer experiments.

The results show that Grad-CAM attention maps can be used for understanding the decision-making process of vision-based agents. In situations where the agents perform well, the focus predominantly coincides with human-interpretable process-relevant features, i.e. the wire. This is for example visible for all three agents on state 1 included in all the DR settings state distributions. Conversely, in states where agents are expected to perform poorly, such as agent 1 in all but the first state, the focus lies on different image sections, which are not relevant for the task. The most precise attention is shown for agent 6 even for unseen real-world states, which coincides with it's zero-shot transferability. This qualitative correlation between

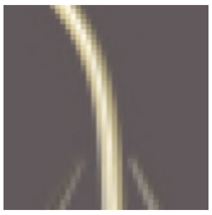
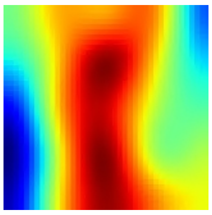
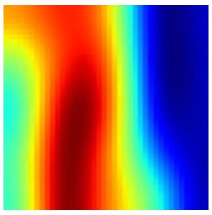
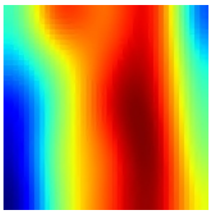
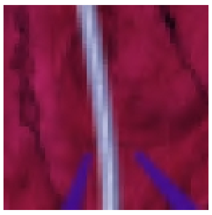
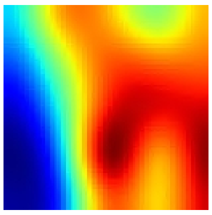
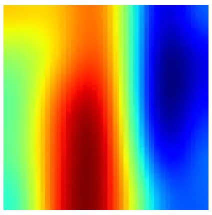
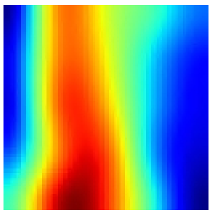
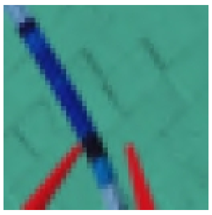
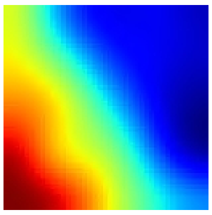
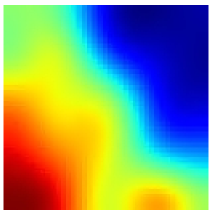
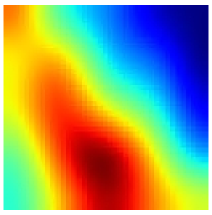
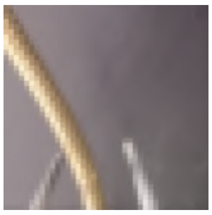
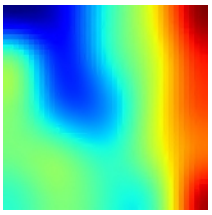
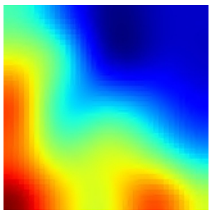
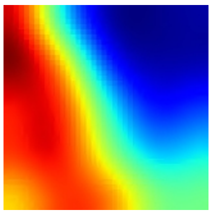
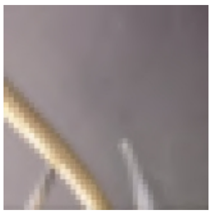
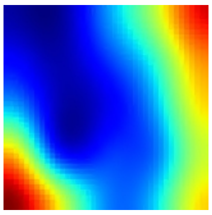
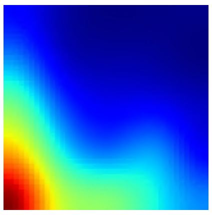
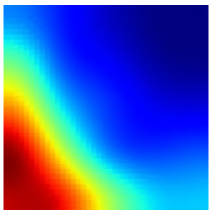
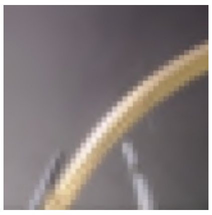
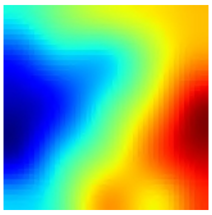
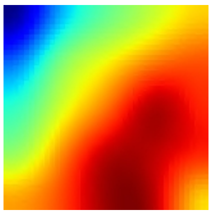
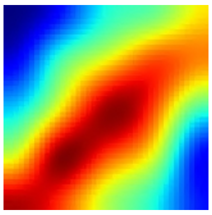
State ID	State	Agent 1	Agent 3	Agent 6
1				
2				
3				
4				
5				
6				

Table 6.1 Grad-CAM attention maps for different agents and states. State 1 is taken from simulation without DR, states 2 and 3 are taken from simulation with DR, and states 4-6 are real-world images. Agent 1 was trained without, agent 3 with partial, and agent 6 with full DR. Red regions indicate high and blue regions indicate low attention. Adapted from [154].

agent performance and attention thus renders the methodology a promising tool for estimating the transferability of pretrained agents. In particular since the method only requires state images, which can even be obtained before implementing the full real-world environment. The experimental evidence that an agents focus coincides with human process understanding provides the necessary foundation for attempts at explicitly injecting domain knowledge into the learning process.

6.2.3 Semi-Supervised Image Compression

Based on the insight that DRL agents focus on the entirety of the wire, this investigation will develop a compression strategy which extracts process-relevant sections from the raw state image. In the CV domain, this approach is called semantic segmentation mapping, where the goal is to assign a class label to every pixel in an image, e.g. in this use case all pixels belonging to the wire. A model which is capable of generating segmentation masks effectively learns to compress the raw state input into a binary format masking process-relevant elements.

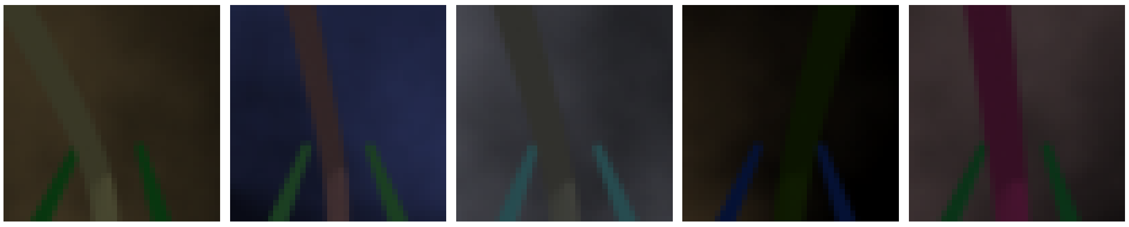
Given the previous successful utilization of autoencoders for unsupervised image compression, an analogous encoder-decoder architecture is chosen for the segmentation mapping approach to further distill information through a latent space bottleneck. Therefore, the proposed approach is a semi-supervised image compression approach, as the model is trained with supervised learning using explicit labels (segmentation masks), but the desired output is the compressed latent space representation.

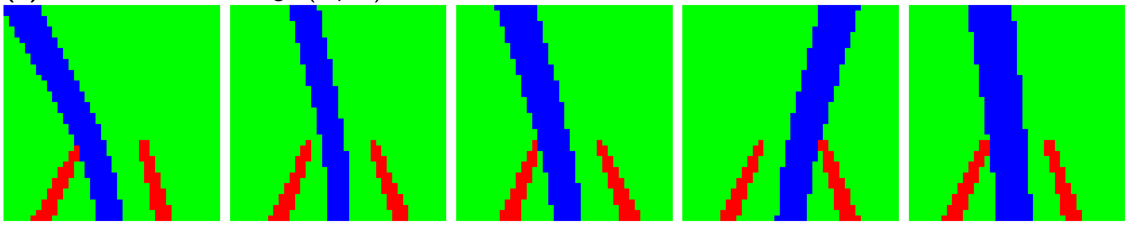
As highlighted above, a limitation of the previous approach to unsupervised image compression is the need for real-world data. This requirement is amplified in this semi-supervised approach, as for each state image a corresponding segmentation mask must be generated. Given the previous success of simulation-to-real-world transfer in the wire-loop use case, the natural expectation is that this limitation can be overcome by pretraining in simulation, where not only arbitrary amounts of data can be generated at high variance with DR, but also ground truth segmentation masks are readily available. Since agents trained with full DR were previously found to be zero-shot transferable, this study investigates the hypothesis that DR is a suitable approach for pretraining a semi-supervised segmentation model.

While it would have been possible to reuse the wire-loop game simulation from chapter 4.2.1.2, the requirements in this investigation are fundamentally different. First and foremost, this data collection effort does not require a complete modelling of the wire-loop game including physical collisions. Additionally, moving the loop to valid positions on the wire, i.e. where the wire is in the loop without collision, requires significant computational overhead.

For those reasons, a dedicated simulation setup is implemented using the Pybullet physics engine. The main difference to the wire-loop game simulation from chapter 4.2.1.2 is that the loop is kept fixed, and the wire is described as a quadratic Bézier curve, where the first two control points are carefully placed close to the loop, such that the wire always goes through the loop without collision. The third control point is then randomized to generate different wire shapes. By varying the control points, the thickness of the wire, the distance to the loop, as well as analogous to chapter 4.2.1.2 colors, textures, and camera positions, a dataset of high variance can be generated quickly (cf. Figure 6.10a).

The autoencoder architecture is kept identical to the previous unsupervised approach, with the only difference being that the mean squared error loss is replaced with a cross-entropy loss, as the decoder now predicts the class probability of each pixel. As the previous unsupervised approach achieved good results with 7030 real-world images, a dataset containing 10,000 simulated state images and their corresponding segmentation masks is generated, depicted in Figure 6.10a and 6.10b. The data is split with an 80/20 ratio into training and validation data and the model is trained for 1000 episodes. Figure 6.10c shows the final model's ability to reconstruct the segmentation masks for the validation data containing simulation images (cf. Figure 6.10a). In general, the model is able to reconstruct the segmentation masks with high accuracy.


(a) Simulated state image (input).


(b) Segmentation mask (label).

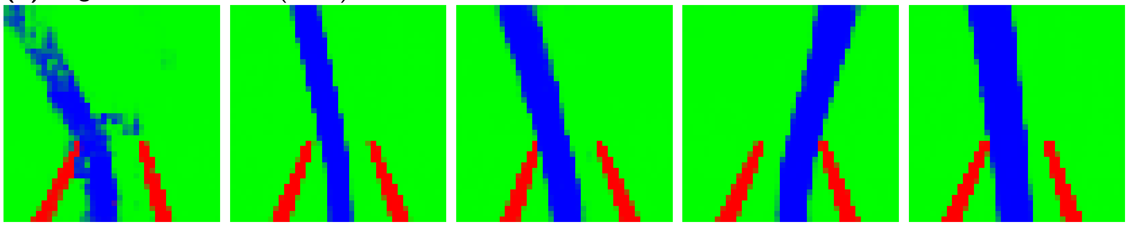
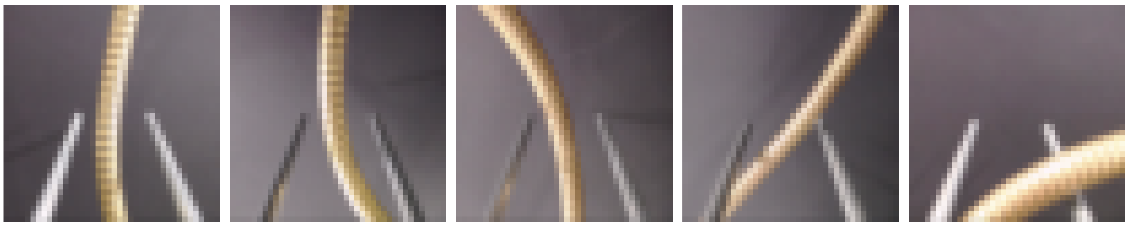

(c) Reconstructed segmentation mask for simulated validation data (prediction).

Figure 6.10 Exemplary results of the segmentation autoencoder on simulated validation data. The model is able to reconstruct the segmentation masks for the validation data with high accuracy.

On transfer to real-world images however, the model performs very poorly. Figure 6.11 shows the model's performance on real-world test data. The model completely fails to capture the wire shape, and only learned very coarsely that the loop rods are expected at the bottom of an image. Given the previous success of DR, these findings are surprising at first. As the model already performs well on the simulation validation set, increasing the training dataset size is not expected to significantly improve the sim2real transferability.

This negative outcome leaves the following conclusion: On the one hand, unsupervised image compression is able to distill process-relevant information, but requires


(a) Real-world state image (input).

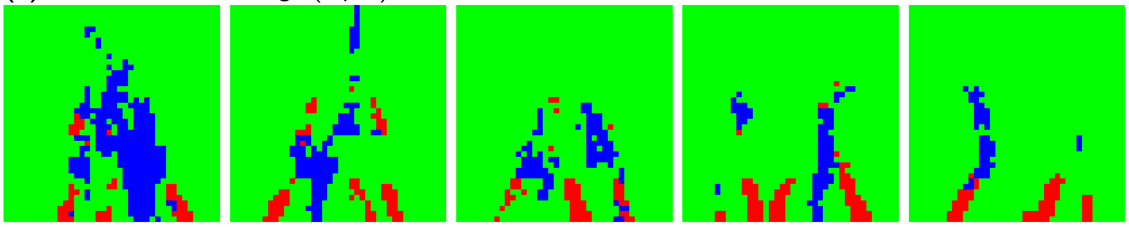

(b) Reconstructed segmentation mask for real-world test data (prediction).

Figure 6.11 Exemplary results of the segmentation autoencoder on real-world test data. The segmentation is of poor quality, as the model is unable to capture the wire shape.

the availability of real-world images. On the other hand, using a purely simulation-based dataset generated with DR does not lead to generalizable compressions. These two approaches can be considered as different extremes on a spectrum of using pure real-world data vs. pure simulation data. This leads to the hypothesis, that combining the two approaches by including some real-world data may lead to better results, which will be investigated in the next section.

6.2.4 Semi-Supervised Compression with Domain Knowledge

In DL applications, the manual annotation of real-world data presents a significant hurdle due to the high cost and considerable time investment required for expert-driven labeling, especially when precise segmentation masks are needed. Furthermore, reliance on large quantities of annotated real-world images can impede rapid prototyping and limit the scalability of CV pipelines. Thus, it is crucial to devise strategies that minimize dependency on real-world data collection and annotation while still achieving robust model performance. This motivates the investigation into leveraging domain knowledge to constrain simulation parameters, aiming to enhance generalization and decrease the real-world labeling burden without compromising model accuracy. This approach is in line with the argumentation by Liebers et al. [101], who propose the integration of human domain knowledge into the DR process. Taking the argument for minimal real-world data to the extreme, the following investigation aims to produce a generalizable segmentation model with a single real-world image.

With the restriction of using just a single real-world image to keep the dependency on real-world data minimal, data driven approaches for finding mappings between simulation and real-world images are not feasible. Instead, the following approach is based on the realization that DR aims to produce a high variance of inputs with the assumption that generalization over a wide distribution includes the real-world data. This is especially powerful in settings where the real-world data distribution is unknown. In case of the wire-loop use case however, it is possible to quantify some aspects of the real-world data distribution, such as the wire, loop, and background colors. While it is hard to fully simulate, the hypothesis stands that constraining DR to colors similar to the real-world not only simplifies the task by constraining the source domain distribution, but also allows for a model to incorporate color information.

The approach is depicted in Figure 6.12. First, a segmentation mask is generated from the real-world image. For each object, the pixels are stored and used as a color palette for the simulation images. To generate a randomized simulation image, a segmentation mask is taken from simulation and each object class is filled with a randomized variation of its corresponding color palette. As before, a dataset of 10,000 simulated images is generated and used to train the semi-supervised segmentation model for 1000 episodes. The results are depicted in Figure 6.13. As with the previous approach, the model is able to reconstruct the segmentation masks for the validation data with high accuracy.

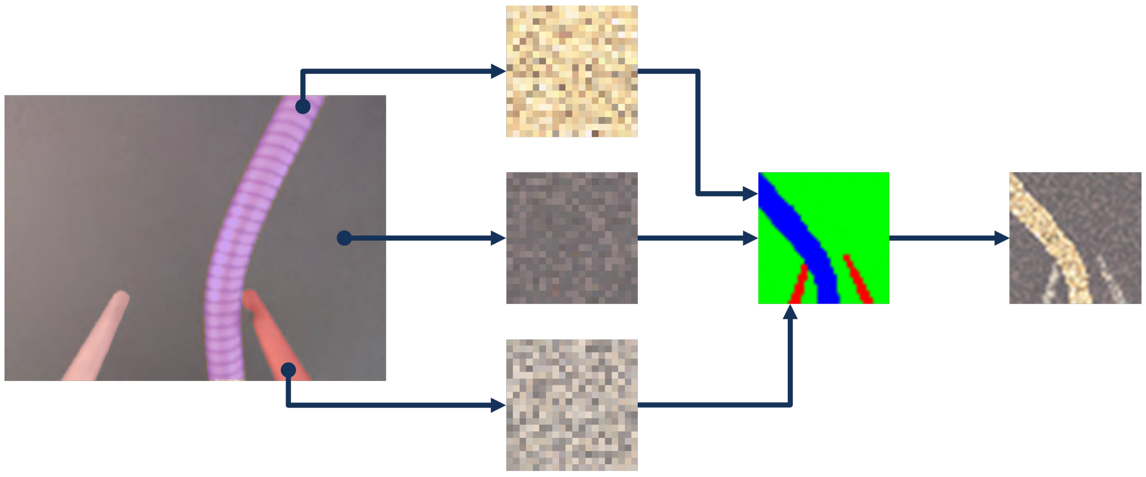
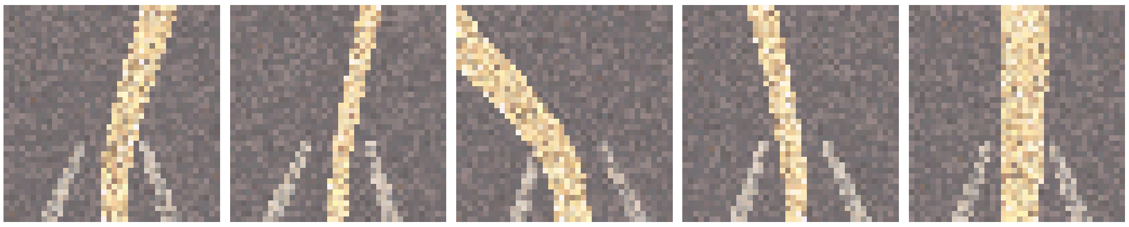
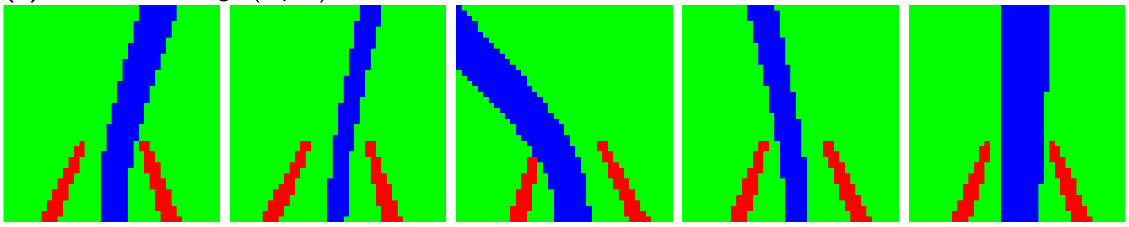


Figure 6.12 Concept of the enhanced approach. The real-world image is segmented into objects. For each object class, a color palette is generated from its respective segments. With this color palette, DR is constrained to generate simulation images with similar colors.

To evaluate the transferability of the model, real-world images are again used as test data. As depicted in Figure 6.14, the model's performance on real-world test data is much more accurate. Although not perfect, the general shape of the wire is always captured. The loop on the other hand is not properly segmented to a satisfactory extent. As seen with the agent attention visualizations however, the wire is by far the most process-relevant information, which is plausible since the loop is stationary in the camera view. Thus, the likelihood that this final image compression captures a latent feature representation which contains all process-relevant information is still expected to be high.


(a) Simulation image (input).


(b) Segmentation mask (label).

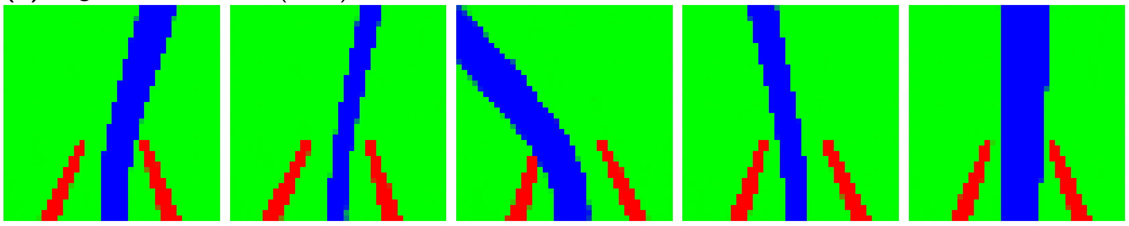
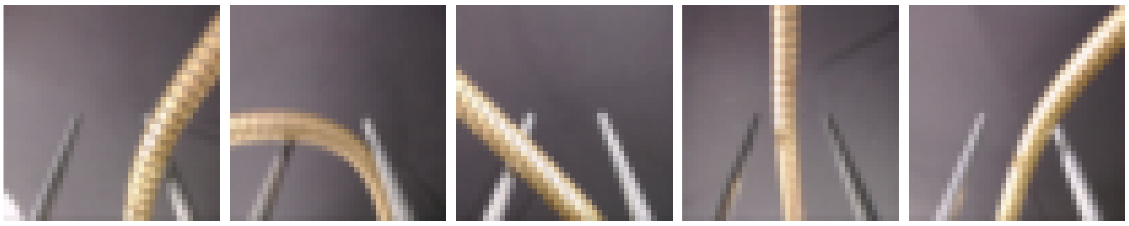

(c) Reconstructed segmentation mask (prediction).

Figure 6.13 Exemplary results of the segmentation autoencoder on simulated validation data. The model is able to reconstruct the segmentation masks for the validation data with high accuracy.


(a) Real-world image (input).

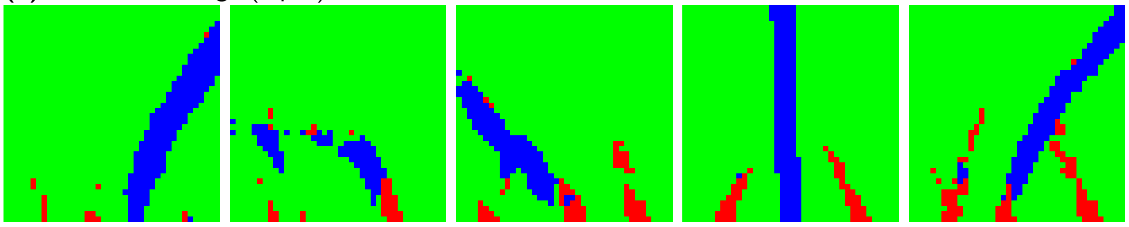

(b) Segmentation mask (prediction).

Figure 6.14 Exemplary results of the segmentation autoencoder on real-world test data. The rough shape of the wire is captured using just a single real-world image.

6.3 Summary

In this chapter, the second RQ was addressed, which prompted the utilization of simulations for the development of perception modules.

For the object picking use case, a YOLO6D pose estimation module was developed to extract the position and orientation of process-relevant objects from camera images. The sim2real transferability of the module was achieved through the use of DR in training, combined with an augmentation strategy on the real-world images and an averaging of the predictions over multiple frames to achieve sufficient performance.

For the wire-loop use case, a semi-supervised image compression approach was developed to distill process-relevant information from real-world images into a latent space. In a first step, an unsupervised autoencoder was trained on real-world images, using the trained encoder as a feature extractor for a subsequently trained policy. While this approach reduced the DRL training effort, the need for real-world data was identified as a limitation. In a second step, it was attempted to instead use simulation data, exploiting the availability of ground truth segmentation masks to inject domain knowledge into the encoder-decoder training. However, the transferability of the model to real-world images was deemed insufficient. Therefore, in a third step, a single real-world image was used to reduce the color randomizations to more realistic values, which was found to significantly improve the transferability of the model.

7

Hierarchical and Backward Planning

"Strategy without tactics is the slowest route to victory. Tactics without strategy is the noise before defeat."
- Sun Tzu

Many real-world industrial robotic applications require long-term planning capabilities to navigate complex tasks that involve multiple stages, uncertainties, and interactions with dynamic environments. Traditional DRL methods, while effective for simple problems, often struggle to handle the complexity of such tasks due to issues like exploration inefficiency and credit assignment. Both issues relate to the fact that an agent may need to perform many actions before a reward signal is received. On the one hand exploration inefficiency describes the challenge of finding positive rewards through random exploration, the chance of which diminishes with the number of actions that need to be executed correctly in sequence. Credit assignment, on the other hand, describes the challenge of determining which actions are responsible for the received reward. Evidently, bad actions despite which a positive outcome is achieved should not be rewarded, and good actions in an unsuccessful sequence should not be penalized. To address these challenges, this chapter explores the exploitation of structures in robotic movement towards answering **RQ-3: How can decomposition of robotic movement be leveraged to modularize task-independent behavior policies, and how can reversibility of robotic movement be exploited to improve training efficiency?**

The first part of RQ-2 is approached through the application of HRL in the object picking use case. Instead of trying to solve a long-horizon task in its entirety, an intuitive approach is to split the task into a set of subtasks which are shorter and thus easier to learn. In the context of DRL, this approach is known as HRL, which was introduced in chapter 3.3.1. In summary, HRL follows a "divide-and-conquer" approach by splitting the decision-making process of a DRL agent into a hierarchy of policies, where higher-level policies select subgoals and lower-level policies select actions to achieve these subgoals. While at first glance this hierarchical structure introduces additional complexity, it promises faster learning. On the one hand, the lower-level policies gain faster reward feedback through shorter subtask horizons. On the other hand, the higher-level policies learn the overall task through the selection of fewer subgoals. In other terms, the lower-level policies learn tactics to navigate the environment without the need to reason about the overall task, while the higher-level policies learn strategies to solve the overall task on an abstracted level without the need to consider step-by-step decision-making. The subdivision into task-specific strategy and task-agnostic tactics also promises potential for transferability of the lower-level policies to other tasks. The investigation into the cross-task transferability of hierarchical policies was published by the author [156].

The second part of RQ-2 is examined through the integration of a backward planning strategy to the clip assembly use case. Backward planning is based on the idea that some tasks can be solved more easily by starting from the final goal and working backwards to the initial state. In the context of assembly, this approach is also referred to as assembly-by-disassembly. Since it is arguably a lot easier to remove a part from an assembly than to put it together, backward planning promises to be a more efficient approach for finding suitable assembly trajectories by first finding disassembly trajectories, and subsequently reversing them to form assembly trajectories.

7.1 Hierarchical Policy Transfer

As seen from the previous baseline experiments in chapter 4.3.2.2, the pick-and-place task poses a challenging problem for standard DRL agents. Albeit considered a simple task from an industrial perspective, learning success depends on the agent finding a sequence of more than 15 actions through exploration before a positive reward is received. In addition, the application of HER arguably does not guide the agent towards learning how to pick up the object, as a correspondence between actions and achieved goal, i.e. the object's position, can only be established after the object has been grasped.

The following investigation aims to apply HRL to the pick-and-place task to overcome the shortcomings of standard DRL by explicitly facilitating the decomposition of the task into more manageable subtasks. This promise of faster learning must be confirmed first. Assuming that this assumption will hold, the hierarchical decision-making structure of HRL further prompts the hypothesis that the lower-level policies may be presumed task-agnostic and thus should be transferable to other tasks where the same tactics are applicable.

To achieve this objective, the remaining chapter outlines the extension of the learning framework with HRL, as well as the adaptation of the Fetch environments to create more complex scenario variations. The subsequent evaluation first validates HRL as suitable approach for solving the object picking use case. Further, the transferability of lower-level policies trained on the standard Fetch environments to the more complex scenario variations is investigated.

7.1.1 Hierarchical Actor-Critic Learning Framework

To apply HRL to the object picking use case requires an extension of the previously developed learning framework. First, a suitable HRL algorithm must be chosen. HAC was identified as most suitable candidate due to its intuitive design of using DDPG agents as hierarchical layers (cf. chapter 2.3.3). This allows the standard DDPG architecture to be interpreted as a single-layer HAC agent, allowing for a direct comparison of the HRL approach with the previous baseline results and an isolated evaluation of the benefit of additional hierarchical layers by keeping any DDPG-related hyperparameters constant. From an implementation perspective, the choice of HAC has the additional benefit of enabling the reuse of the existing DDPG implementation, which significantly reduces development overhead.

In their paper introducing HAC, the authors describe the algorithm as a structure of nested loops, where each loop represents a layer of the hierarchy [95]. In the innermost loop, the actual interaction with the environment takes place, while the outer loops are used to infer the subgoals. While this description is intuitive, it only considers a single environment, or environments which are always at the same time step. Given the asynchronous parallelization of environments, this design was deemed unsuitable for the current framework.

Instead, the HAC training routine follows the following three stages, which constitute a single interaction with the environment:

1. **Top-Down Decision-Making**: Given the current time step t, the state s_t and the goal $g = g_t^n$, the agent iterates through its layers $\pi^n, ..., \pi^0$, starting with the highest-level layer π^n. For each layer, it is checked whether the layer must update its prediction at the current time step. If so, the layer policy $\pi^i(g_t^{i-1}|s, g^i)$ is executed with the current state and the subgoal provided by the layer above. The so far produced data is accumulated in an experience replay buffer $D^i = ((s_t, g^i), g_t^{i-1})$. Importantly, if a layer is determined to update, all respective lower layers are also updated.

2. **Environment Interaction**: The last layer's action prediction is passed to the environment. The reward r_t, next state s_{t+1}, and achieved goal $\hat{g}_{t+1}$ are observed.

3. **Bottom-Up Feedback**: Iterating through the layers in reverse order, starting with the lowest-level layer π^0, the reward r_t^i is determined, which will be discussed in detail below as the hierarchical dependence requires a more elaborate reward design. In addition, it is determined whether a respective layer has completed a trajectory, i.e. whether the subgoal g_t^i has been achieved, a terminal environment state has been reached, or the trajectory has reached its maximum length. Once a trajectory is completed, it is moved to the experience replay buffer of the respective layer.

The main advantage of this design is that it can be run for an arbitrary subset of parallelized environments at arbitrary different time steps. It is thereby compatible with the asynchronous environment parallelization through the orchestrator of the framework (cf. chapter 4.1).

Determining the reward r_t^i for each hierarchical layer is inherently challenging because the effectiveness of a higher layer depends on the ability of all subordinate layers to accomplish their assigned subgoals. Even if a particular layer selects optimal subgoals, there is no guarantee that the lower layers will successfully achieve them. As a result, the reward assigned to a specific layer cannot be solely based on the agent's overall performance. To address the issue of hierarchical dependency, HAC differentiates between two modes: exploration and testing.

Analogous to standard DRL, exploration mode is used to explore the environment by adding exploration noise to the policy predictions. As a result, the performance of the lower layers cannot be assumed to consider the provided subgoals. To avoid punishing upper layers for suboptimal behavior of the lower layers, the provided subgoals are replaced in hindsight with the actual achieved subgoals by the lower layers. This corrects the trajectories such that they reflect the actual experience and do not corrupt the learning process.

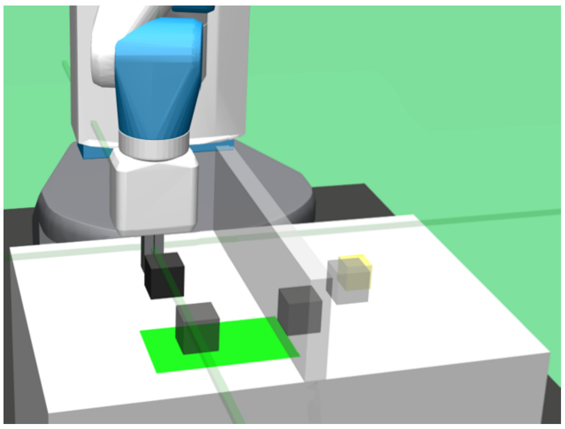
In testing mode, no noise is added to the policy predictions. This can be interpreted as the layers attempting to achieve the provided subgoals. Here, a layer is rewarded for the feasibility of the subgoals it provides. If its lower layers fail to achieve the subgoal, the layer is punished. This way, the layers are incentivized to provide achievable subgoals.

7.1.2 Fetch Environment Variations

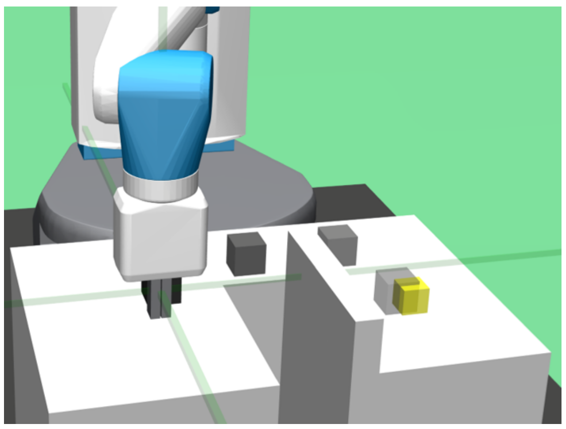
To investigate the transferability of the lower-level policies, the Fetch benchmark environments Push and PickAndPlace were extended with additional obstacles or intermediate objectives to generate more complex scenario variations. The resulting transfer environments are depicted in Figure 7.1 and constitute the following modifications:

- **Push-Gate**: A barrier divides the tabletop, blocking any path from the initial cube position and final goal pose. To solve the task, the agent must first move the cube onto the green area to remove the barrier, before proceeding toward the goal.
- **Push-Gap**: A gap in the tabletop obstructs the direct path to the goal. The agent must navigate the cube around the gap to reach the goal. A wall is placed on the far side of the gap to prevent the agent from attempting to skip the cube across the gap.
- **PickAndPlace-Wall**: A fixed wall is placed in the middle of the tabletop, which requires lifting the cube to reach the goal. The goals are constrained to the table surface on the opposite side of the wall from the initial cube position.
- **PickAndPlace-Table**: An elevated table structure is placed on the tabletop, onto which the cube must be lifted.

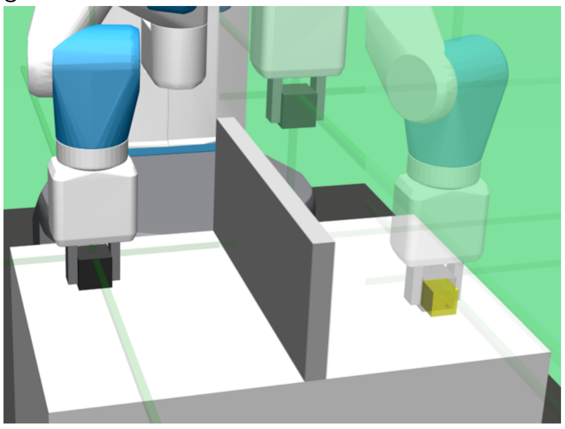
Each of these variations demands solutions that diverge from direct, shortest-path strategies, presenting challenges that go beyond the optimal strategies of the base environments.



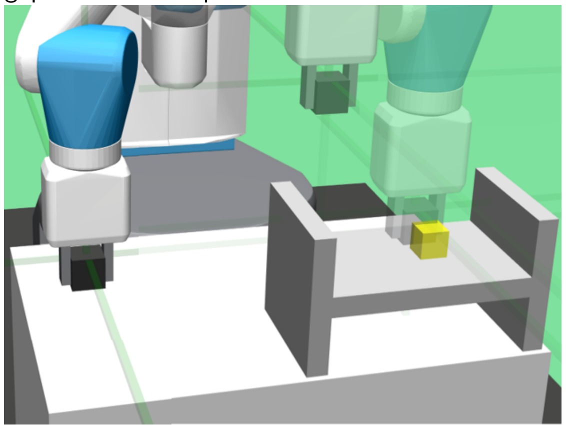
(a) Push-Gate: The cube must be moved onto the green area to remove the barrier.



(b) Push-Gap: The cube must be moved around a gap in the tabletop.



(c) Pick-and-Place-Wall: The cube must be lifted over a wall.



(d) Pick-and-Place-Table: The cube must be lifted onto a table.

Figure 7.1 Variations of the Fetch environments to investigate the transferability of the lower-level policies. Taken from [156].

7.1.3 Experimental Evaluation

In the following experimental evaluation, HAC agents are trained on the Fetch environments to validate the hypothesis of improved learning performance through hierarchical planning. Subsequently, the lower layers of the HAC agents are transferred to the Fetch environment variations to investigate the transferability of low-level subgoal policies.

7.1.3.1 Hierarchical Actor-Critic Experiments

With the HAC algorithm implementation outlined above, agents were trained on the same three Fetch benchmark environments as in chapter 4.3.2.2. Here, two hyperparameter choices have to be made: the number of layers and the maximum trajectory length for the individual layers. As determined in chapter 4.3.2.1, an optimal solution to the PickAndPlace task requires approximately between 15 and 25 steps. A hierarchical decomposition of two layers with a maximum lower-level trajectory length of 10 was chosen to keep the architecture as simple as possible while at the same time ensuring that at least two subgoals will need to be generated by the higher policy. The results are depicted in Figures 7.2, 7.3, and 7.6, whereby each configuration was run with 5 random seeds, and the average performance and standard deviation are reported. The figures contrast the performance of the HAC agents with the previously established baseline results using a standard, non-hierarchical DDPG agent.

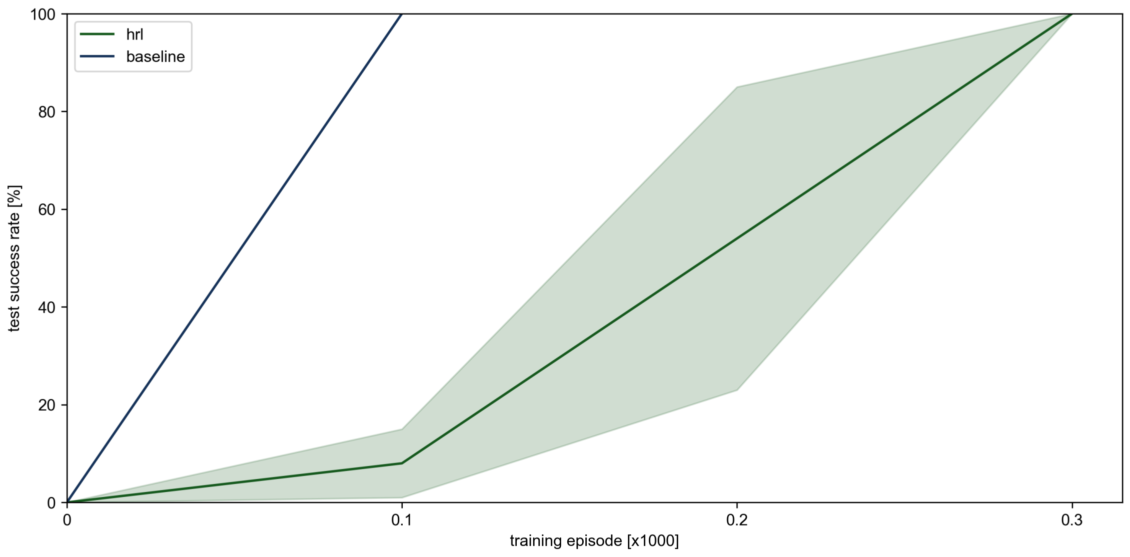


Figure 7.2 Performance of the HAC agents on the Reach task. The HAC agents perform significantly worse, as the added overhead of the hierarchical structure is not justified by the simplicity of the task.

Instead of applying the same design logic to tune the parameters to the Reach environment, which is solvable in less than 5 steps and thus would require a smaller spacing between subgoals, it was decided to use the same configuration for all tasks. The main reason is the clear focus is the PickAndPlace task as most relevant to the object picking use case, rendering any tuning efforts for the other benchmark environments out of scope for this investigation. Moreover, the deliberate presumed misconfiguration can be regarded as opportunity to explicitly investigate the effects on learning efficiency of HAC, which is a valuable insight for the applicability in real-world applications where knowledge of optimal control strategies is a privilege that cannot be presumed.

Notably, the HAC agents perform significantly worse on the Reach task, which is expected given the hypothesized suboptimality of the hyperparameter configuration (cf. Figure 7.2). Instead of solving the task in 100 episodes as the standard DDPG baseline, the HAC agents require 300 episodes to arrive at a solution. Intuitively, the optimal solution requires the highest layer to simply parrot the goal as subgoal, in order for the lower layer to reach the goal with a single trajectory. Therefore, the hierarchical structure does not provide any benefit. On the contrary, the results support that adding hierarchical decomposition where it is not needed introduces unnecessary complexity in form of communication overhead between the layers, which severely degrades the learning efficiency.

For the Push task, the results depicted in Figure 7.3 show a clear benefit of HAC over the standard DDPG baseline, improving the learning efficiency by more than 50%, from 15,000 to 7,000 episodes.

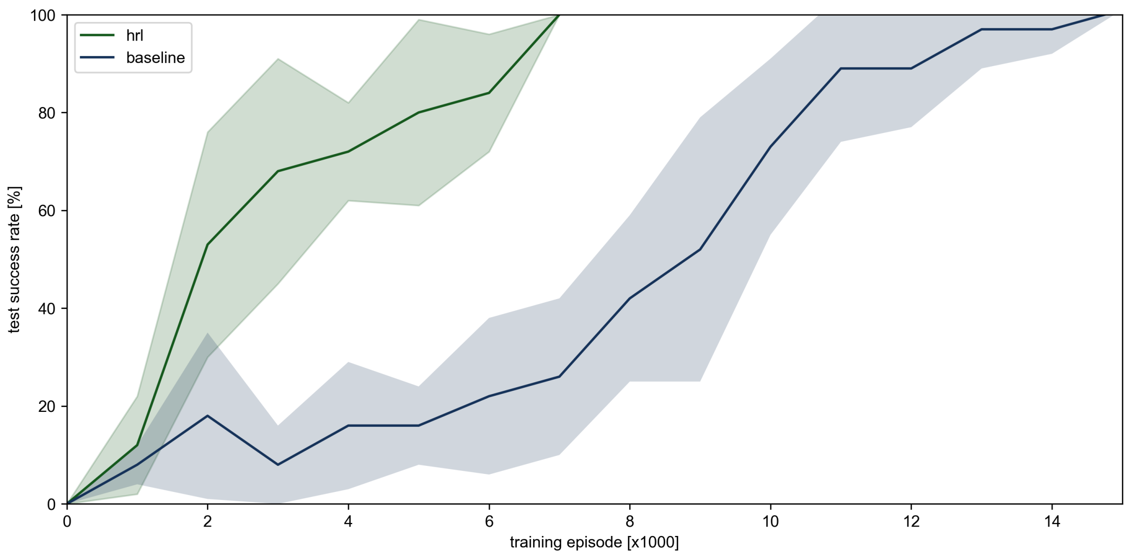


Figure 7.3 Learning curve of the HAC agents on the Push task. The HAC agents are able to solve the task significantly faster than the standard DDPG baseline.

Figure 7.4 further shows the exemplary behavior of a HAC agent for the Push task, which highlights a major benefit of interpretability of the hierarchical decision process. As the subgoals are derived from the goal, which constitutes the desired position of the cube, it is possible to visualize the subgoals. The visualization shows that the HAC agent learned to decompose the task by using a subgoal halfway to the final goal. Implicitly, the subgoal choice balances between shortening the distance to the final goal and the ability of the lower layer to consistently achieve the subgoal.

In addition to visualizing the subgoals in the environment, it is also possible to visualize the Q-value functions learned by the critics of the DDPG agents. This methodology is adopted from [17]. Of particular interest is the Q-value approximation through the high-level policy's critic. Since the critic aims to predict the

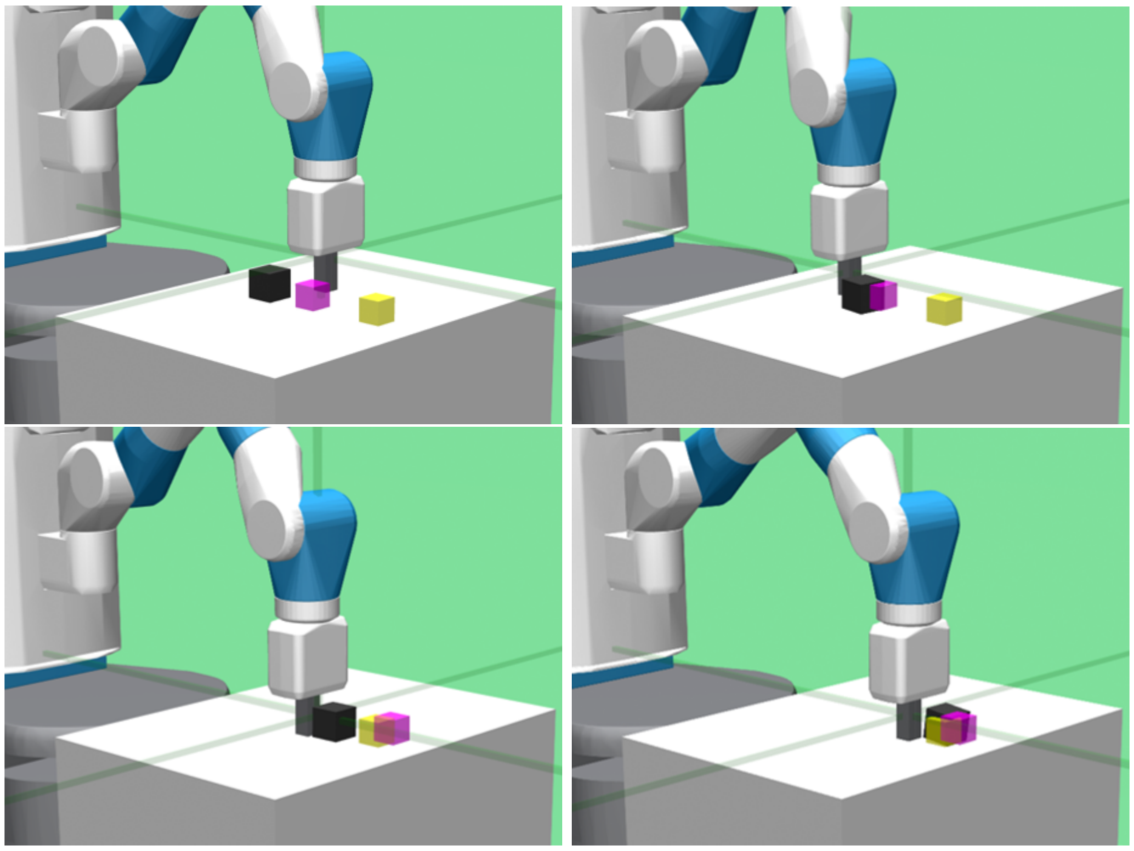


Figure 7.4 Exemplary solution trajectory of a HAC agent for the Push task. The agent successfully decomposes the task by using a subgoal halfway to the final goal.

expected cumulative future reward given a state-action pair, and the action of the high-level policy is the selection of a subgoal, it is possible to probe the critic for hypothetical subgoal choices and corresponding Q-values. This is exemplified in Figure 7.5, which overlays a fixed situation, i.e. cube, goal, and TCP position, with a heatmap of the subgoal Q-values. As expected, the Q-values increase from the current cube position towards the goal, however not fully reaching the goal as the high-level critic correctly learned that subgoals too far away are unreachable by the lower-level policy.

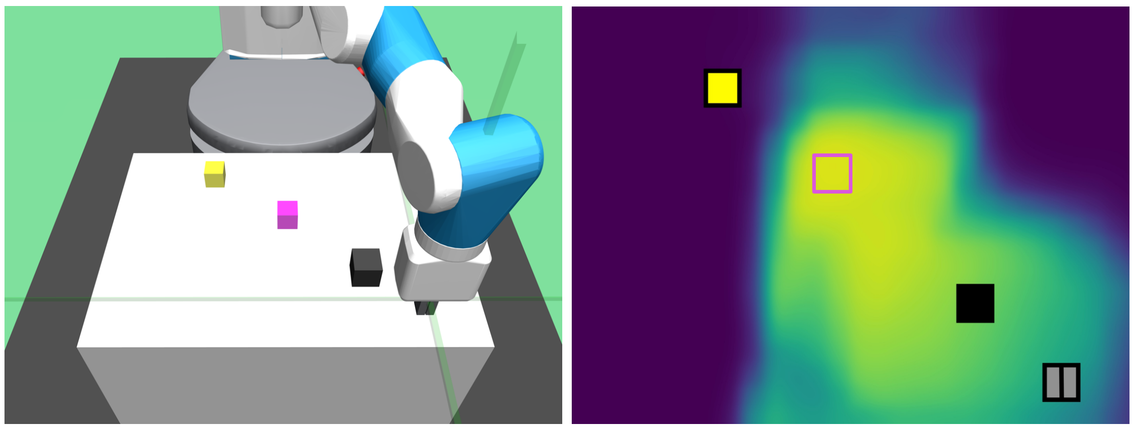


Figure 7.5 An exemplary state of the Push task and corresponding heatmap of the subgoal Q-values. Yellow regions indicate high, violet regions low Q-values.

For the PickAndPlace task, the results depicted in Figure 7.6 demonstrate a similar benefit of HRL. Unlike the previous baseline results, the HAC agents are able to solve the task reliably in 11,000 episodes, achieving a previously unattained success rate of 100%.

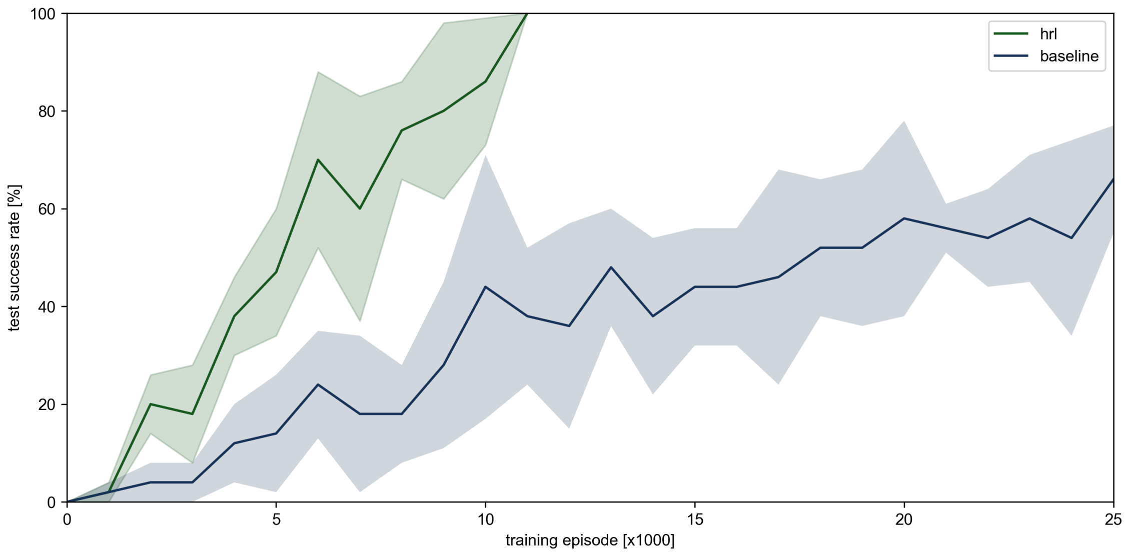


Figure 7.6 Learning curve of the HAC agents on the PickAndPlace task. The HAC agents are able to successfully learn the task, unlike the standard DDPG baseline.

An exemplary solution trajectory including subgoals is shown in Figure 7.7. As before, the agent successfully decomposes the task into two sections, using a subgoal halfway to the final goal. Thus, HRL has been demonstrated to be able to solve the pick-and-place task where standard DRL fails.

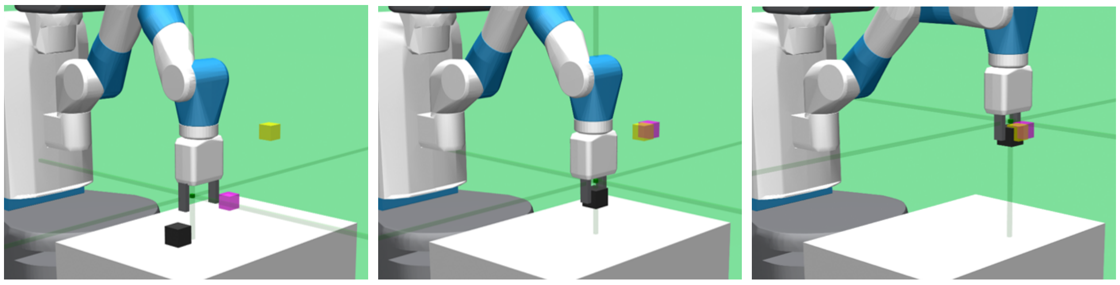


Figure 7.7 Exemplary solution trajectory of a HAC agent for the PickAndPlace task.

7.1.3.2 Low-Level Policy Transfer Experiments

Using the variations of the Fetch benchmark environments as target environments, an investigation into the transferability of the lower-level policies is conducted. First, HAC agents are trained from scratch to establish a comparative baseline. Subsequently, HAC agents previously trained on the standard environments are taken, and their high-level policy weights are re-initialized, effectively transferring the lower-level policies. Finally, training is continued on the target environments. Each training is performed with 16 random seeds, and the average performance and standard deviation are reported in Figures 7.8, 7.9, 7.12, and 7.13.

In the Push-Gate environment, the transferred agents exhibit a slightly steeper learning curve in the beginning of training than the agents trained from scratch (cf. Figure 7.8). However, this performance gap closes over time. Neither variant manages to solve the task entirely within the given episode limit of 10,000 which was imposed to limit the runtime of the experiments, and chosen based on the previous results of the Push source environment previously being solved in 7,000 episodes. Thus, no conclusive advantage of transferring pretrained lower-level policies can be observed.

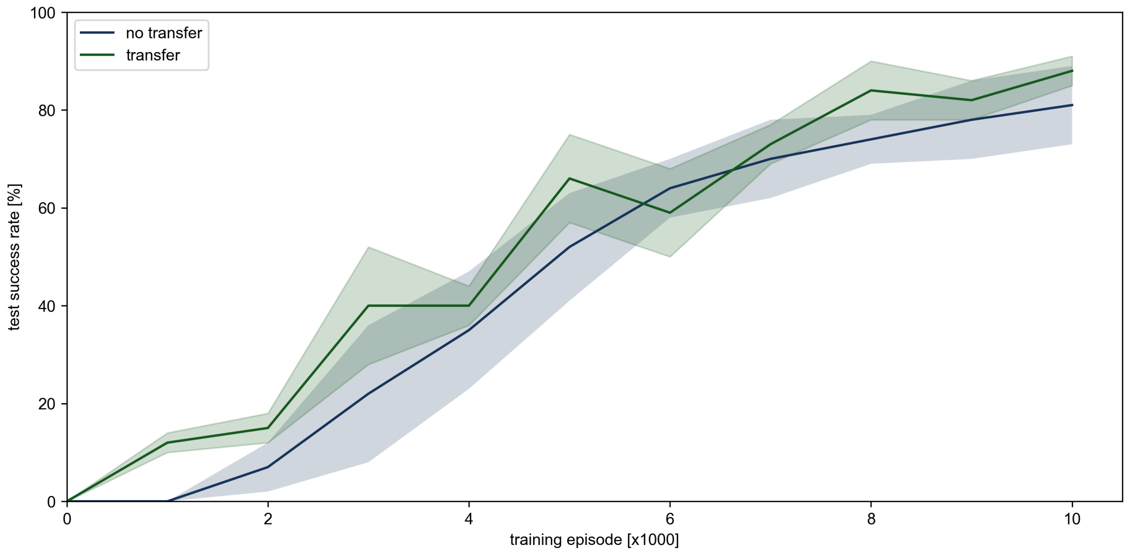


Figure 7.8 Transfer results for the Push-Gate Task. No conclusive advantage of transferring pretrained lower-level policies can be observed. Adapted from [156].

For the Push-Gap environment, the transfer strategy appears to even have a negative effect on the learning efficiency (cf. Figure 7.9). The transferred agents exhibit a similar learning curve as the agents trained from scratch, but gradually fall behind after 5,000 episodes.

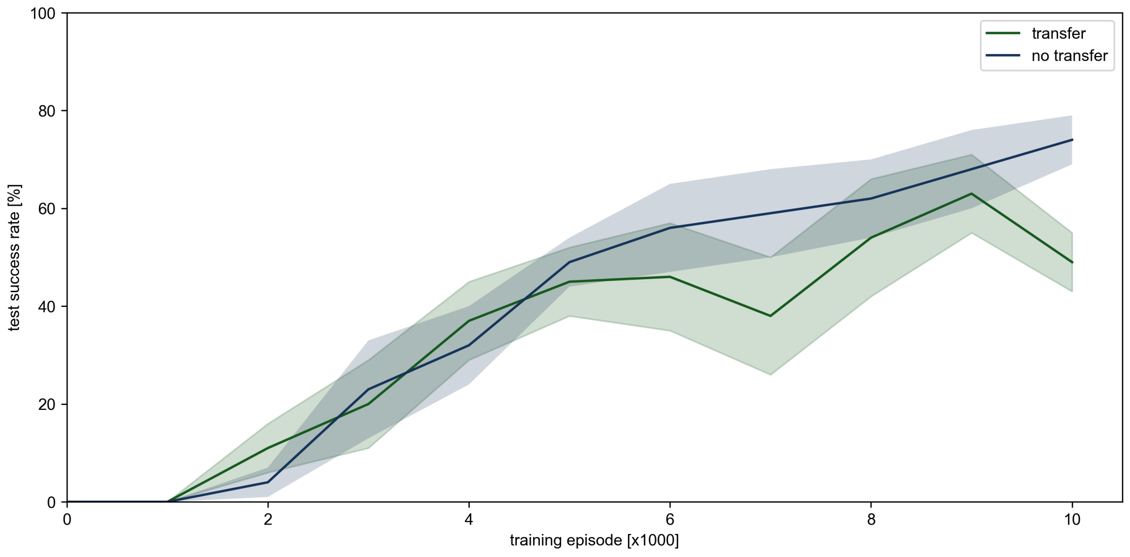


Figure 7.9 Transfer results for the Push-Gap Task. The transfer strategy appears to have a negative effect on the learning efficiency. Adapted from [156].

In an attempt to understand why the transfer in the Push scenarios is unsuccessful, a closer look at the decision-making of the agents is taken. Figure 7.10 shows a successful attempt of a hierarchical agent to place and reach the first subgoal. Here, it becomes apparent that the agent learned a rather risky strategy, skipping the gap in the tabletop.

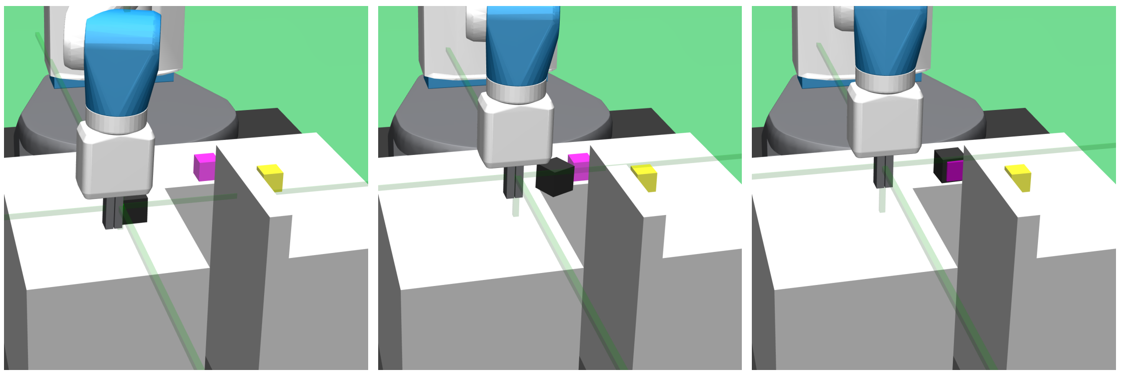


Figure 7.10 Strategy of a HAC agent on the Push-Gap Task. The agent learned a risky strategy, skipping the gap in the tabletop. Taken from [156].

Skipping the cube across the gap requires hitting the cube with high velocity. On closer inspection, it was found that the pretrained lower-level policy, optimized for obstacle-free environments, learned moderate movement patterns unsuitable for this aggressive maneuver. Surprisingly, this did not lead to the higher-level policy learning to take this into consideration and arrive at a more cautious strategy, which proposes more conservative subgoals around the gap. Instead, the results suggest that the higher-level policy pressured the lower-level policy to adapt the high-risk, high-reward strategy. As inferred by the suboptimal learning curve, this readjustment of the lower-level policy resulted in an increased instability of the learning process, leading to a worse performance than agents learning the same aggressive strategy from scratch. The analysis of the HAC agent behavior on the Push-Gap task can again be complemented by the visualization of the subgoal Q-values, as shown in Figure 7.11.

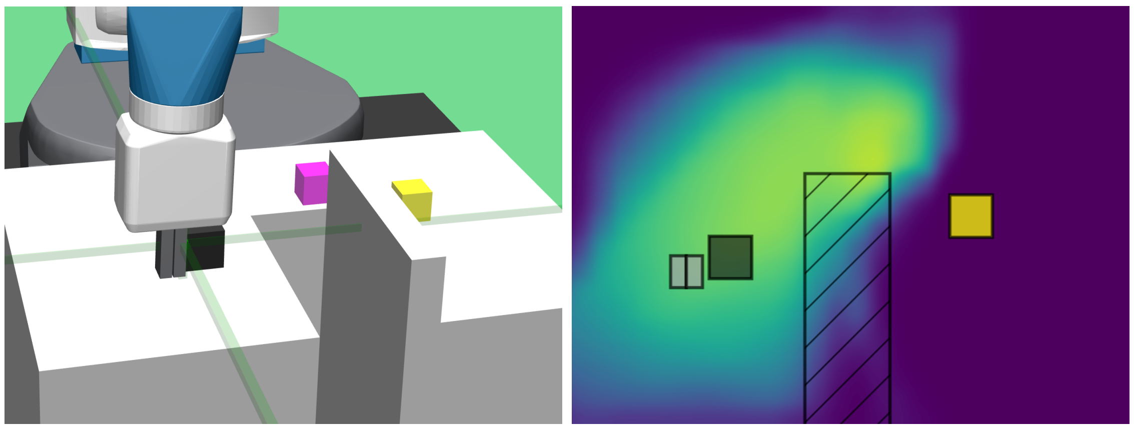


Figure 7.11 Q-value heatmap in the Push-Gap task. The highest Q-value is found over the gap, thereby incentivizing the lower level policy to attempt skipping the cube over the gap. Adapted from [156].

The heatmap confirms the previously observed behavior of the high-level policy, which places the subgoals across the gap. It additionally uncovers that the agent places some value in subgoals over the gap, which are clearly unfavorable. While this is likely due to the analyzed agent not being trained long enough to converge to an optimal strategy, it highlights a major weakness of DRL in general, which is the data inefficiency of replicating easily expressible domain knowledge, such as not placing subgoals over gaps, through pure end-to-end interaction with an environment. The designed human interpretability of the subgoal space however may be a promising interface to explicitly inject such domain knowledge into the learning process. However, as the Push scenario is not considered the main focus of this thesis, this line of research is not pursued further.

In contrast, the PickAndPlace transfer experiments show a clear advantage of transferring pretrained lower-level policies, as depicted in Figures 7.12 and 7.13. In both cases, the learning curves of the transferred agents show an immediate steep performance incline reaching test success rates above 95% within 4,000 episodes. The transferred agents thereby clearly outperform the agents trained from scratch, which take 2–3 times the training effort. Interestingly, the latter don't show any learning progress for the first 5,000 episodes, but then exhibit a similarly steep performance increase as the transferred agents, albeit with a higher variance. This is especially remarkable when revisiting the learning curves of the source environment, which continuously improved over the course of 11,000 episodes. This indicates that even when the performance metrics such as the test success rate suggest that a hierarchical agent is not progressing on the overall task, valuable learning is still taking place in the individual layers, which the agent is able to capitalize on later in the training process.

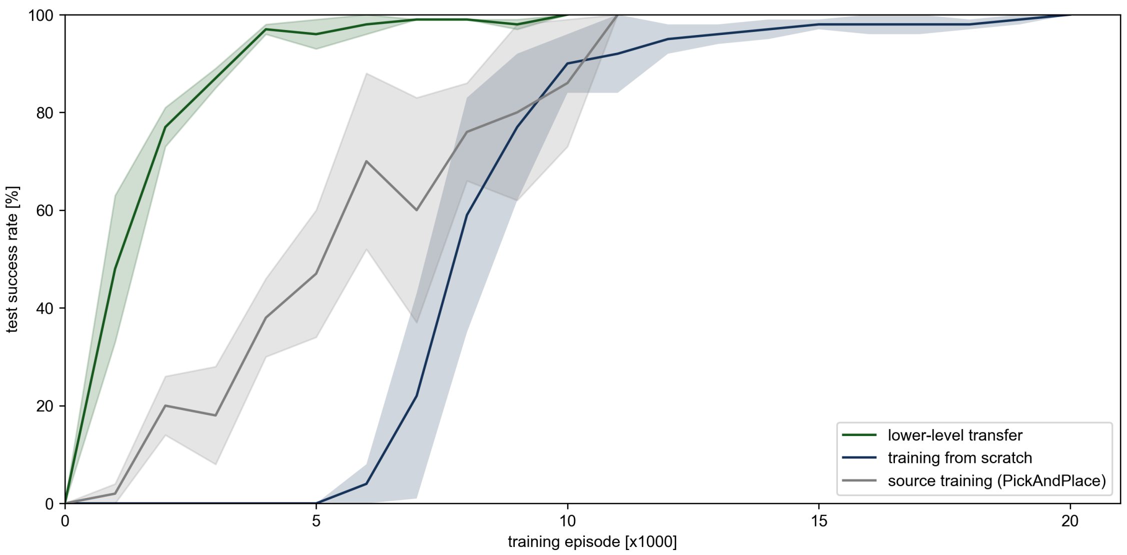


Figure 7.12 Transfer results for the PickAndPlace-Wall task. The transferred agents clearly outperform the agents trained from scratch. Adapted from [156].

An exemplary trajectory of a transferred agent in the PickAndPlace-Wall environment is shown in Figure 7.14, which demonstrates the ability to utilize the pretrained lower-level policy's ability to move a block in a straight line to towards a subgoal by using different subgoals, i.e. above the wall to circumvent the obstacle. With these positive results on the PickAndPlace environments, the potential of transferring lower-level HRL policies can be confirmed. However, considering the negative

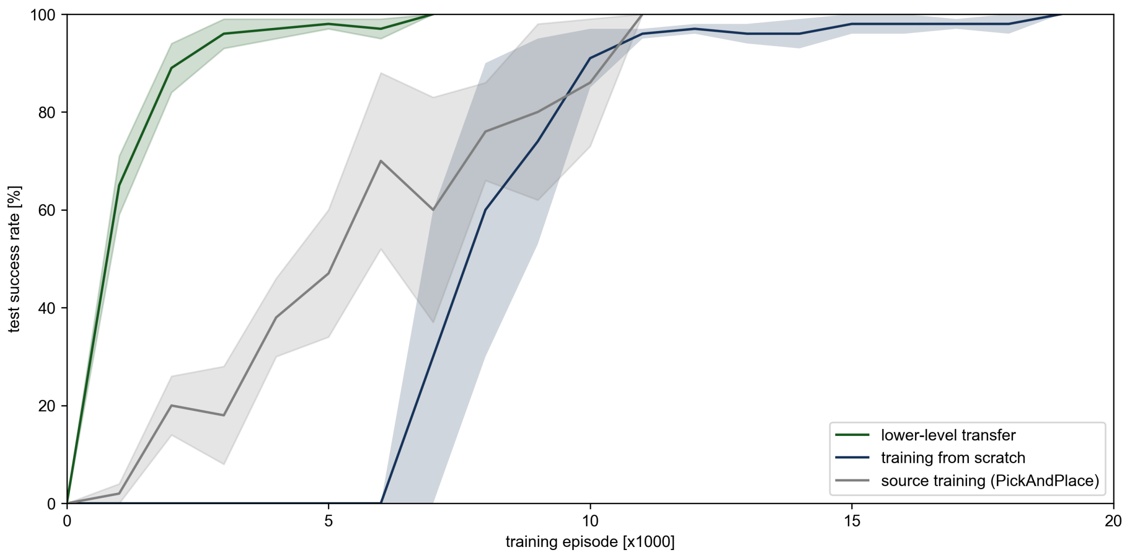


Figure 7.13 Transfer results for the PickAndPlace-Table task. The transferred agents clearly outperform the agents trained from scratch. Adapted from [156].

results on the Push environments, it has to be acknowledged that this transferability is not guaranteed. If the lower-level policy behavior does not match the higher-level policy's expectations, the resulting adaptation of the lower-level policy may destabilize the learning process compared to training from scratch.

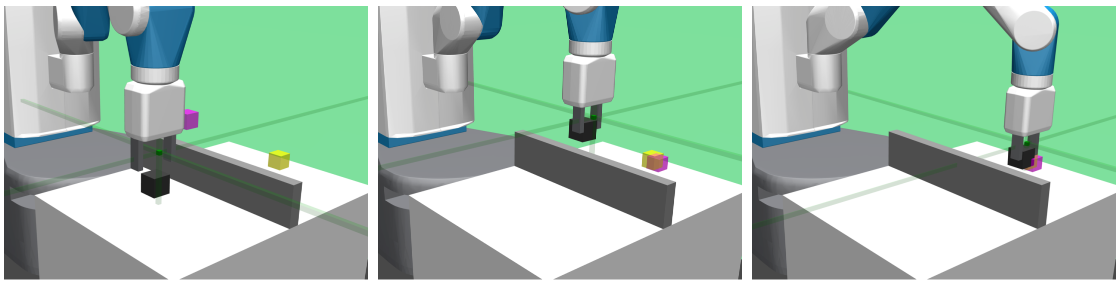


Figure 7.14 Visualization of a trajectory of a successfully transferred agent in the PickAndPlace-Wall task.

7.2 Assembly-by-Disassembly

In the clip assembly use case, the initial attempts to learn the task with DRL agents were unsuccessful. As previously mentioned, the agents learned to keep a safe distance between the clip and the shell to avoid likely collisions. To address this issue, the following investigation considers a simple, yet effective approach to prevent the agent from learning the avoidance strategy: reversing the task, i.e. disassembling the clip from its target location, and subsequently inverting the found trajectories. Once the disassembly task is solved, the trajectories are inverted and used to train an agent for assembly using BC.

7.2.1 Clip Disassembly Environment

While converting the environment implementation from an assembly task to a disassembly task is straightforward by switching the start and goal poses, the resulting disassembly environment is not necessarily fit for assembly-by-disassembly. Revisiting the assembly environment introduced in chapter 4.4.1, clip movement is realized with a force constraint which pulls the clip towards the next waypoint, determined by the chosen action. As depicted in Figure 7.15, this results in a simulation where trajectories may be non-reversible. Further, such non-reversible trajectories were found to be exploited by agents for reward hacking [6]. Reward hacking describes agents learning unintended strategies to maximize the reward, resulting from unforeseen environment mechanics such as simulation inaccuracies, or errors in the reward function.

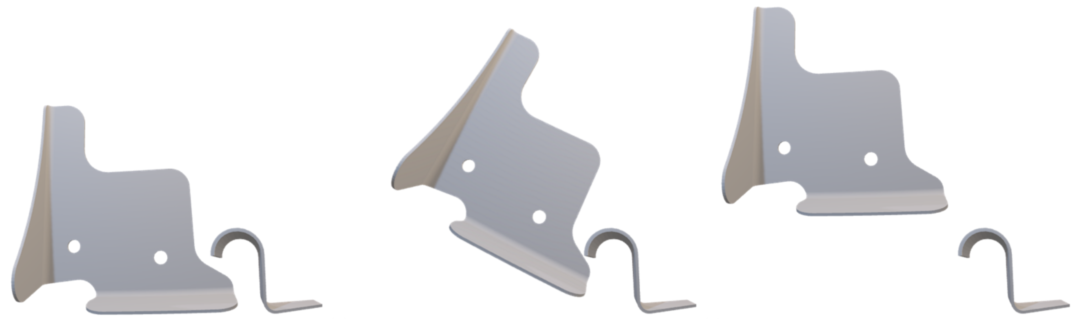


Figure 7.15 Reward hacking in the clip assembly environment. The agent is able to exploit the force constraint to move the clip out of the shell in a non-reversible manner.

In the case of the clip assembly environment, the force constraint does not account for the orientation changes of the clip along the trajectory, allowing an agent to forcefully yank the clip out of the shell. To address this issue, at each waypoint the trajectory is reversed and tested for validity. If the trajectory fails to insert the clip, the disassembly task is considered failed, resulting in a termination of the episode and an associated maximum negative reward.

As a waypoint is never perfectly reached, but considered reached if the distance to the waypoint is below a certain threshold, solely testing the current transition for reversibility is not sufficient due to stochasticity in the physics-based simulation. This changes the MDP formalism, as the reward function $\mathcal{R}(s, a, s'|\tau^{-1})$ is no longer independent of the previous history τ leading up to the state s. Empirical testing showed that retaining and checking the last two transitions for consecutive reversibility is sufficient to guarantee valid assembly trajectories in the given use case. Thus, the computational overhead for an episode length n from $\sum_{k=1}^{n} k$ to $2n-1$.

7.2.2 Experimental Evaluation

With the disassembly environment in place, DDPG agents are trained on the modified task. The actor and critic ANNs are realized as 3 layers of 64 neurons each, using ReLU activation functions. This configuration was chosen based on an intuitive estimation considering the low dimensionality of the state, goal and action space.

The training results are depicted in Figure 7.16, showing the mean learning performance of 10 training runs. The results clearly show that the agents are able to solve the task in 25,000 training episodes, which in the experimental setup was achieved in 10 minutes wall-clock time using 96 parallel environments and an Nvidia V100 GPU.

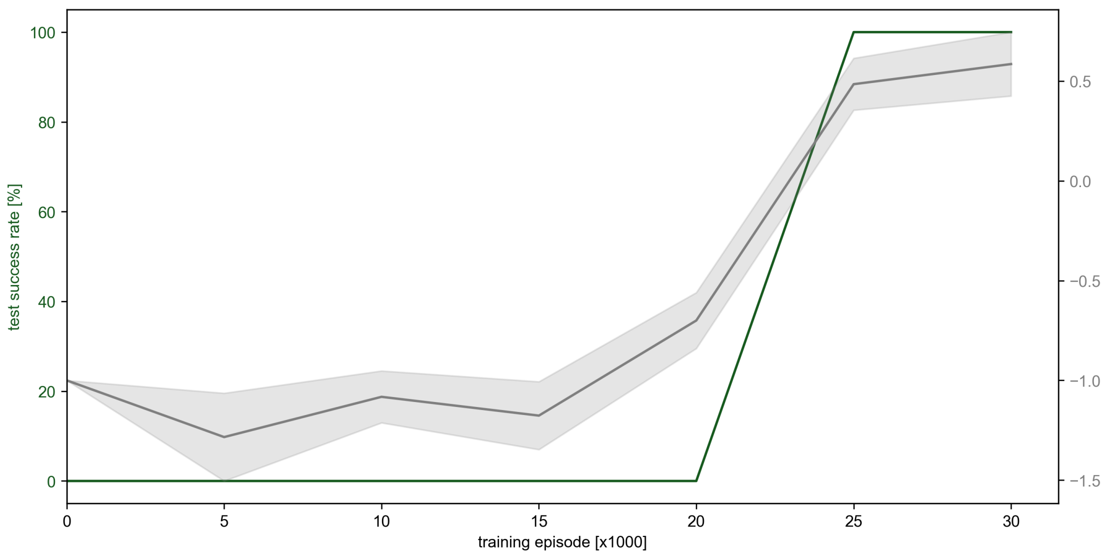


Figure 7.16 Performance of the DDPG agents on the clip disassembly task. The agents are able to solve the task in 25,000 training episodes.

To derive an agent capable of solving the original assembly task, the trained disassembly agent is used to generate a set of trajectories. These trajectories are then inverted and used to train an agent for assembly using BC. To evaluate the dependence of the transferability on the number of demonstration trajectories, different numbers of trajectories are used. The results are depicted in Table 7.1, which are based on 10 respective post-training runs on the assembly task. As the results show, a reliable zero-shot transfer is only possible with a sufficient number of 1,000 demonstration trajectories. While transfer with 100 trajectories is achievable with 5,000 additional post-training episodes, the success of this post-training is not guaranteed and only was achieved in 3 out of 10 runs. As expected, further reducing the number of demonstration trajectories drops the success rate to 0%.

BC Trajectories	0	10	100	1000
Success (%)	0	0	20	100
Episodes (x1000)	-	-	5	0

Table 7.1 Performance metrics for assembly task using BC with varying number of demonstration trajectories.

Given the high quality of the simulation due to the accurate underlying CAD model, the assembly agents were found to be directly applicable to the real-world demonstrator setup, given knowledge about the goal pose in the robot base coordinate system to correctly translate TCP measurements of the robot into the state space of the assembly agent, and subsequently translate the proposed action back into TCP displacement coordinates. Apart from this, no further fine-tuning was required to successfully automate the clip assembly process.

7.3 Summary

In this chapter, the third RQ was investigated, which asked for methods to exploit structure in robotic movement through hierarchical decomposition and reversibility of tasks.

In the object picking use case, the Fetch PickAndPlace environment was successfully solved using HRL with HAC, outperforming previous attempts using standard DDPG agents. Further, the lower-level policy was transferred to task variations including obstacles, demonstrating the potential of HRL to decompose a strategy into task-agnostic low-level subgoal-based execution and task-specific high-level subgoal selection. However, transfer was not found to be universally successful and may even have a negative effect when the lower-level policy does not match the higher-level policy's expectations regarding precision, resulting in significant performance degradation and additional training overhead to compensate for the mismatch.

For the clip assembly use case, assembly-by-disassembly was successfully applied. By learning the reverse of the assembly task, agents were no longer able to avoid contact with the shell, forcing them to learn meaningful trajectories. At the same time, the disassembly task is arguably easier than the assembly task, as the possible collision-free movements are much more constrained, leading to faster learning. It was discussed that not all trajectories are reversible, leading to a trajectory reversibility check to be included in the reward function to ensure that the learned disassembly trajectories can subsequently be used for assembly. Subsequently, the assembly task was successfully solved in a zero-shot manner by inverting the disassembly trajectories and using them as demonstration trajectories for BC.

8

Cross-Robot Execution Imitation

"Imitation is the sincerest form of flattery."
- Charles Caleb Colton

In industrial robotics, tasks are often times planned in TCP space, which reduces the planning space to the most critical element: how a tool should be moved to achieve the desired outcome. While this planning space is robot-agnostic, deploying a TCP trajectory to a specific robot traditionally requires specification of the robot configurations. This is necessary, since the mapping of a TCP pose to a joint configuration with IK is not unique, as for most TCP poses there are multiple viable joint configurations. This ambiguity is traditionally managed by manually specifying the desired joint configuration for each TCP waypoint in the trajectory, restricting the robot's workspace to regions where IK solutions are unique, or by imposing joint limits to reduce the joint solution space. Such manual restrictions however require elaborate manual analysis of the robotic system to on the one hand ensure safe operation without collisions, without on the other hand limiting the efficiency of the robot by restricting its workspace to only allow for suboptimal movement. The manual labor is especially amplified in the context of cross-robot TRL, where manually post-processing vast amounts of source domain demonstrations is not feasible. This chapter serves as an exploratory step toward answering **RQ-4: How can movement trajectories be transferred between different robot morphologies?**

The following investigation proposes a Cross-Robot Imitation Learning (CRIL) framework, which automatically selects target trajectories most similar to a given source trajectory based on a joint space similarity metric. The mapped trajectories are then used to pretrain a DRL agent for the target robot with IL techniques. Finally, the transferability is validated by contrasting the learning performance of the pretrained agent to that of an agent trained from scratch on the target robot. The CRIL framework and experimental validation were previously published by the author [19].

8.1 Robot-Task Environments

To investigate the cross-robot transferability of learned behaviors, the modular robot-task environment design of the framework introduced in chapter 4.1 is used to easily exchange between the source and robot models. The Pybullet physics simulation engine [35] was chosen due to its previous use in chapter 6 and thereof resulting familiarity with the framework. Deciding against the Mujoco physics engine [186] was a deliberate choice. Although it was previously used in the object picking use case, at the time the decision was made the Mujoco was only available under a restricted academic license, whereas Pybullet was freely available under an open-source license. Figure 8.1 depicts the resulting ability to quickly generate different robot-task combinations.

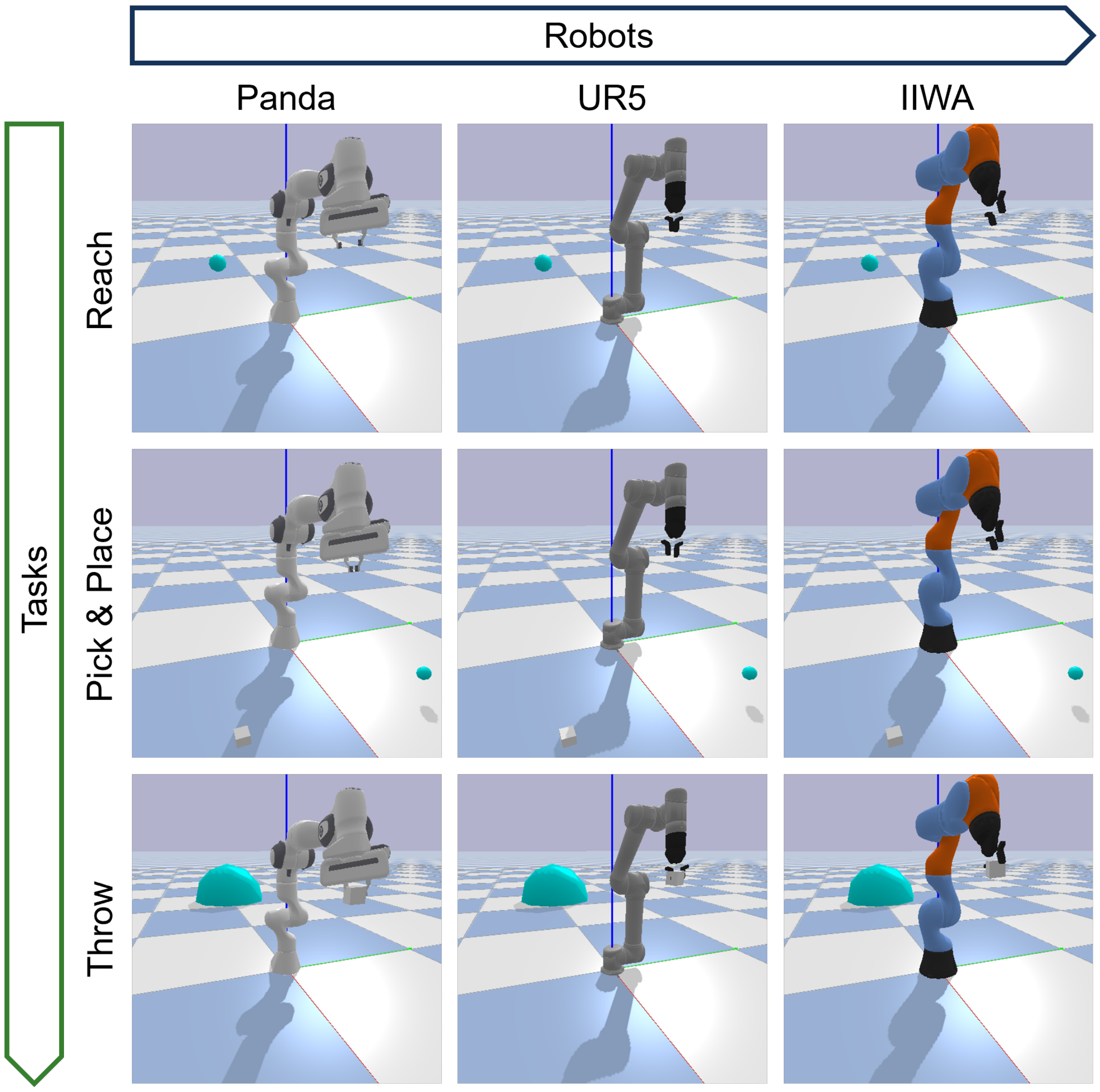


Figure 8.1 The modular robot-task environments allow for quick generation of different robot-task combinations to evaluate cross-domain transferability. Adapted from [21].

For the following experiments, a reach task is used as proof-of-concept task (cf. first row of Figure 8.1), which requires the robot to move its end-effector to a randomly sampled target position. The state space consists of the joint angles of the robot, and the action space translates into relative joint angle changes. The Panda and the UR5 are selected as source and target robot for two reasons. Besides being the robot models used in the three use cases, the robots also differ in degrees of freedom, with the Panda having 7 and the UR5 having 6 joints. Thus, the state and action spaces of the two domains also are of different dimensionality, which must be considered by the following CRIL framework.

8.2 Cross-Robot Imitation Learning Framework

Towards the development of a CRIL framework, the following questions must be addressed: First, how to define and compare behaviors of different robot arms. Second, how to map a source trajectory to a target trajectory. Third, how to use the mapped trajectories to pretrain a DRL agent in the target domain with IL techniques. The following sections will address these questions in detail.

8.2.1 Cross-Robot Behavior Similarity

To enable cross-robot behavior comparison, a similarity metric is developed which measures the likeness of two robot arm trajectories, i.e. their behavior. According to the American Psychological Association's dictionary, behavior is "any action or function that can be objectively observed or measured in response to controlled stimuli." [190]. Translating this definition to the MDP formalism, the stimuli can be understood as a current state, and the measured response as the resulting next state induced by the selected action.

Taking inspiration from related literature, the similarity metric proposed by von Eschenbach et al. [196] is a suitable starting point. The authors consider a simplified version of cross-robot behavior comparison, where they take the same robot model, but lock joints to create robot instances with different DOF. To compare these robots, they propose the following correspondence metric:

$$w_{s_t^S, s_t^T} = \sum_{i=0}^{D^S} \sum_{j=0}^{D^T} w_{i,j} \cdot d(p_{t,i}^S, p_{t,j}^T) \tag{8.1}$$

The correspondence metric takes the weighted sum of the distances between all key points of two robot arms. As key points, intermediate poses along the kinematic chains of the two robots are used to capture the general shape of the arms. The weighting matrix $w_{i,j}$ accounts for the comparability of different key points according to similar location in the kinematic chain. Von Eschenbach et al. [196] consider the same robot model and propose a distance metric which compares the position, rotation, velocity, and angular velocity of the robot joints.

To apply this metric to compare arbitrary robot models, this thesis will only consider the position of the key points, as joint angle values and velocities are not directly comparable between different robot models, or would require manual formalization. For example, the special case of identical robot models with inverted joint orientations would require explicit knowledge that joint angles of different sign are actually equivalent. The position of the key points in 3D space is however straightforward to compare and thus a suitable generalized abstraction space.

While the orientation and velocities are hard to compare for the robot joints, it can be applied to robot-agnostic TCP poses. Here, the entire pose, including orientation and position, can be compared. This results in the following adjusted correspondence metric, which combines the reduced joint distance metric of von Eschenbach et al. [196] with the TCP distance.

$$w^{EC}_{s^S_t, s^T_t} = d(p^S_{t,TCP}, p^T_{t,TCP}) + \sum_{i=0}^{D^S} \sum_{j=0}^{D^T} w_{i,j} \cdot |p^S_{t,i} - p^T_{t,j}| \tag{8.2}$$

This metric will be referred to as Embodiment Correspondence and is visualized in Figure 8.2.

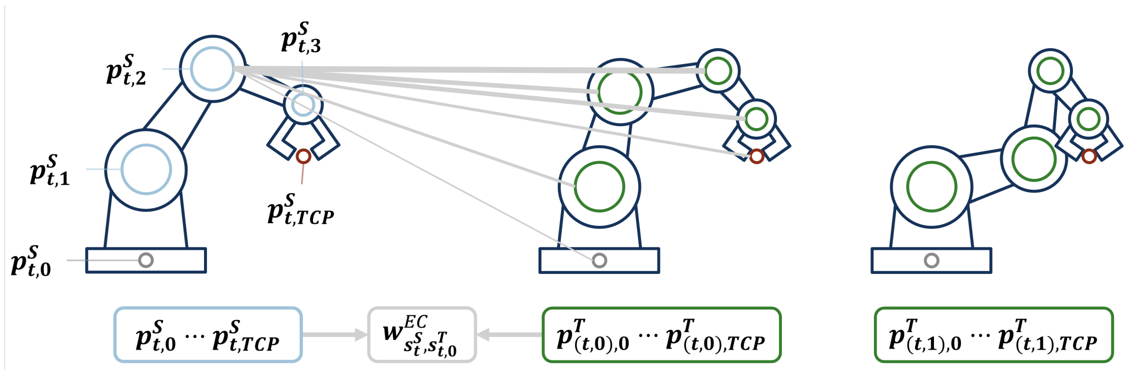

$p^S_{t,3}$
$p^S_{t,2}$
$p^S_{t,1}$
$p^S_{t,TCP}$
$p^S_{t,0}$
$p^S_{t,0} \cdots p^S_{t,TCP}$
$w^{EC}_{s^S_t, s^T_{t,0}}$
$p^T_{(t,0),0} \cdots p^T_{(t,0),TCP}$
$p^T_{(t,1),0} \cdots p^T_{(t,1),TCP}$

Figure 8.2 The Embodiment Correspondence (EC) metric is the weighted sum of the distances between all key points of two robot arms. As key points, intermediate poses along the kinematic chains of the two robots are used to capture the general shape of the arms. Taken from [19].

In order to apply the EC metric, the selection of suitable key points is crucial. This thesis follows the approach of von Eschenbach [196] and use one key point in each robot joint, which are easily computed as the intermediate solutions of FK. To measure the translation distance between two key points, the Euclidean distance is used. For the TCP correspondence, the orientation difference may be computed as the angular difference [127]. The angular difference between two orientations R_1 and R_2 is computed as

$$\theta = arccos((trace(R_1^T R_2) - 1)/2) \tag{8.3}$$

8.2.2 Cross-Robot Trajectory Mapping

With the ability to quantify the similarity between two robot kinematics, two general options can be considered to map a given source to a target joint configuration. The first option is to exploit the differentiability of the EC metric with respect to the robot joint angles and perform an iterative gradient descent optimization to find corresponding joint configurations. The second option exploits explicit knowledge of the robot kinematics to directly compute potential joint configurations, and subsequently use the EC metric to select the most similar one.

For this approach, the latter option was chosen, since for the two selected robot models, the analytical IK solutions are known and provided by previous works [68, 70, 184, 193]. While the availability of analytical IK solutions is not always guaranteed and poses a limitation for the applicability of the proposed approach, recent advances in automatically deriving IK solutions from the robot kinematics have been reported [129], which directly address this limitation. The proposed mapping framework is depicted in Figure 8.3.

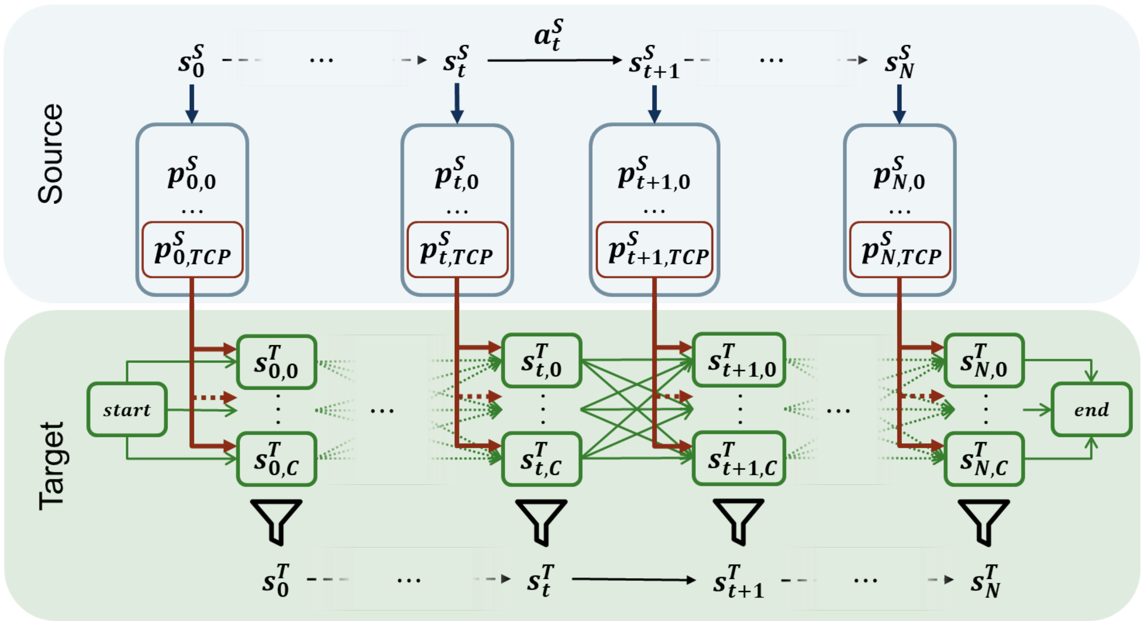
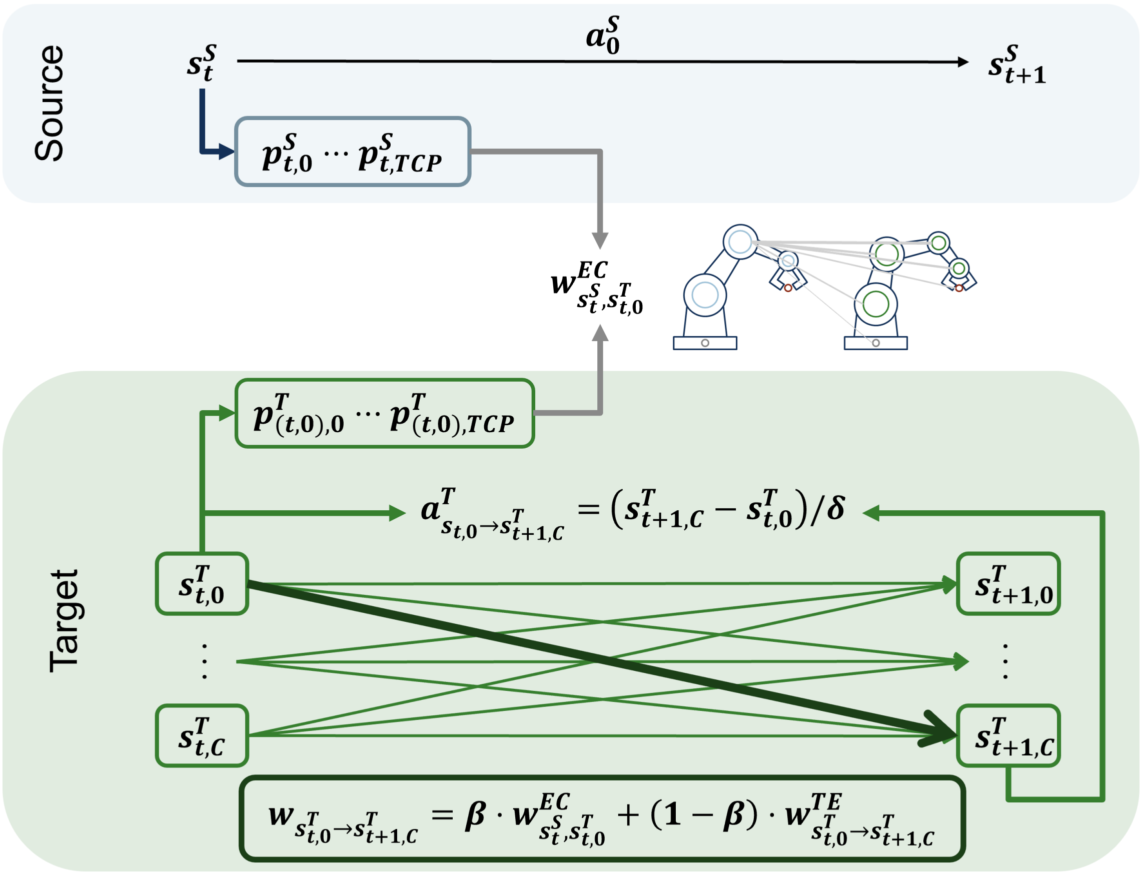


Figure 8.3 The mapping strategy of the CRIL framework. For each state of the source trajectory, the TCP pose is determined. Next, all possible joint configuration candidates are computed with IK. From all candidates, a graph is constructed, whose edge weights reflect the EC metric and the transition cost. The target trajectory with most similar behavior to the source trajectory is found by finding the shortest path through the graph. Taken from [19].

Using an explicit analytical IK model provides two advantages: On the one hand, the analytical solution ensures that the TCP poses are preserved across the mapping, which is a desired property since the TCP trajectory is the most important process-relevant key point. On the other hand, the analytical solution is much more efficient, as it deterministically computes all possible joint configurations in one step. In contrast, the gradient descent approach requires an iterative optimization per solution and thus may converge to a local minimum, as well as does not provide any guarantees for finding all possible joint configurations. Further, the iterative gradient descent approach is computationally more expensive than a one-shot analytical solution. Notably, this decision also simplifies the EC metric, as the distance between the TCP key points is by design always zero and can thus be omitted from the computation.

When mapping entire trajectories, it is important to consider that simply mapping the individual waypoint configurations to their respective best matching target configuration may result in a trajectory which is not feasible or suboptimal for the target robot. To address this, the transition between two consecutive waypoints must be considered in the selection process. As elaborated above, each state s_t^S describing a joint space configuration of a given source trajectory is mapped to a set of key



Source
s_t^S
a_0^S
s_{t+1}^S
$p_{t,0}^S \cdots p_{t,TCP}^S$
$w^{EC}_{s_t^S, s_{t,0}^T}$
Target
$p_{(t,0),0}^T \cdots p_{(t,0),TCP}^T$
$a^T_{s_{t,0} \to s^T_{t+1,C}} = (s^T_{t+1,C} - s^T_{t,0})/\delta$
$s_{t,0}^T$
$s_{t,C}^T$
$s_{t+1,0}^T$
$s_{t+1,C}^T$
$w_{s^T_{t,0} \to s^T_{t+1,C}} = \beta \cdot w^{EC}_{s_t^S, s_{t,0}^T} + (1-\beta) \cdot w^{TE}_{s^T_{t,0} \to s^T_{t+1,C}}$

Figure 8.4 Detailed view of the graph weight computation. The EC metric is computed from key points determined by FK. The transition cost is computed from an approximation of the required action between two consecutive candidates. Taken from [19].

points $p_{t,i}^S$ using FK. Subsequently, the last key point, i.e. the TCP pose, is fed into the IK solver of the target robot to compute all possible target joint configurations $s_{t,c}^T : 0 \leq c < C$, which are referred to as candidates. To now find the most similar target trajectory, a directed graph is constructed, whose nodes represent the candidates and edges represent the transition between two consecutive candidates. A start and end node is introduced, producing a graph where each path from start to end node represents a potential target trajectory. To quantify the suitability of potential target trajectories, the respective paths through the graph are evaluated. This requires introducing edge weights to reflect the previously introduced EC metric, as well as the cost of transition between two candidates. The edge weight computation is visualized in Figure 8.4 and is detailed in the following.

Since the robots use joint angle control $s_{t+1} = s_t + \alpha a_t$, where α is a scaling vector to linearly map between the action space and joint angle space, the required action between two given consecutive states can be approximated as $a_{s^T_{t,c}, s^T_{t+1,d}} \approx (s^T_{t+1,d} - s^T_{t,c})/\alpha^T$. This allows the computation of a Transition Efficiency (TE) metric

$$w^{TE}_{s^T_{t,c}, s^T_{t+1,d}} = \begin{cases} \|a_{s^T_{t,c}, s^T_{t+1,d}}\| & \text{if } a_{s^T_{t,c}, s^T_{t+1,d}} \in \mathcal{A}^T \\ \infty & \text{otherwise} \end{cases} \tag{8.4}$$

which effectively favors actions of small magnitude and additionally removes impossible transitions, i.e. actions that are not in the target action space $\mathcal{A}^T$, from the graph. This leads to an overall edge weight formulation

$$w_{s^T_{t,c}, s^T_{t+1,d}} = \beta \cdot w^{EC}_{s^S_t, s^T_{t,c}} + (1-\beta) \cdot w^{TE}_{s^T_{t,c}, s^T_{t+1,d}} \tag{8.5}$$

with the tuning parameter $\beta \in [0, 1]$ balancing between the two weight components.

As stated above, the selection of the best suited target trajectory requires finding the shortest path through the graph from the start to the end node. This can be efficiently computed using Dijkstra's algorithm, which is a well-established method in the field of graph theory [46]. It is important to note that the mapping can fail if there is no valid path from the start to the end node, i.e. if the source behavior cannot be mimicked by the target robot. This can happen if a source state cannot be mapped to any target state candidates, resulting in a disconnected graph. Mapping also fails if the action space of the target robot is too small to achieve the required transitions between consecutive target states, which in the graph means edges with infinite weight.

8.2.3 Cross-Robot Imitation Learning

With the mapping framework introduced above, it is possible to translate a set of source trajectories into a set of corresponding target trajectories which correspond in behavior. The final stage in the CRIL framework is to incorporate these mapped trajectories into the training process of a DRL agent with the goal of reducing the required amount of training in the target domain. Since the mapped trajectories constitute target domain demonstrations, this remaining challenge can be treated as a classical IL problem.

In the following, MARWIL is chosen as behavioral cloning method, which extends the standard BC method by using rewards to emphasize high-value transitions and thereby improves the quality of the pretrained policy [200]. Given the assumption that similar behavior results in similar rewards, the rewards collected in the source domain are directly transferable to the respective target domain transitions.

After pretraining, the agent is transferred and fine-tuned on the target environment. For this, a different DRL algorithm from the previously used DDPG is chosen, namely PPO. The main reasoning behind this switch is that MARWIL utilizes the same actor and critic architectures as PPO, namely a stochastic policy $\pi(a|s)$ and a state-value function $V(s)$. This allows for a seamless transfer, whereas conversion of MARWIL components into the deterministic policy $\mu(s)$ and state-action-value function $Q(s, a)$ of DDPG is non-trivial. It was considered to use Soft Actor-Critic (SAC), which like DDPG is an off-policy algorithm, while like PPO using a stochastic policy. However, SAC also uses a state-action-value based critic, leading to the same transferability issues as DDPG.

Since the required DRL training framework required a different algorithm in the form of PPO, as well as an implementation of MARWIL, it was deemed more efficient to not extend the previously developed framework, but instead utilize an existing open-source library. Specifically, RLlib [98] was identified as most suitable, as it is an actively maintained open-source library with a large community and comprehensive documentation, and a dedicated focus on industrial applicability. RLlib provides implementations of both MARWIL and PPO, which will be used in the following experiments.

8.3 Experimental Evaluation

In the following, the previously developed cross-robot imitation-learning framework is evaluated regarding its ability to transfer knowledge in the form of trajectory demonstrations from a source to a target robot. As a prerequisite for the evaluation, source trajectory demonstrations must be generated first. To this end a DRL agent is trained on the source Panda robot using PPO. The aggregated learning curves of five different agents are depicted in Figure 8.5. Training is performed until a success threshold of 98% is reached, which was chosen as a compromise between the expectation of a near-perfect performance on the given reach task, and the consideration that stochasticity in the initialization and dynamics of the environment may prevent an agent from achieving perfect performance. The results show a stable convergence of the learning curves, with agents achieving the target success rate in 4,000–5,000 episodes.

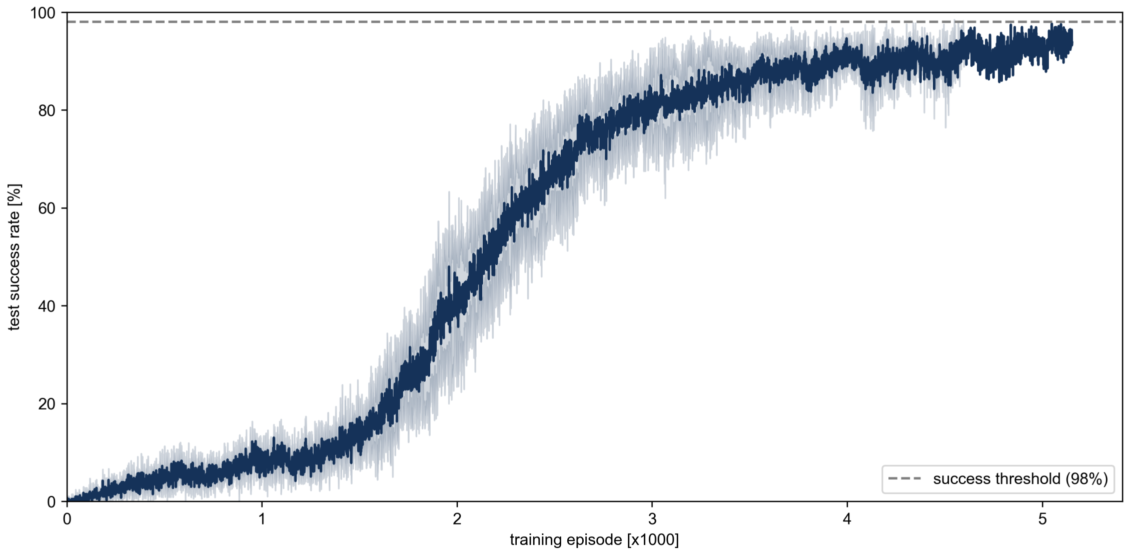
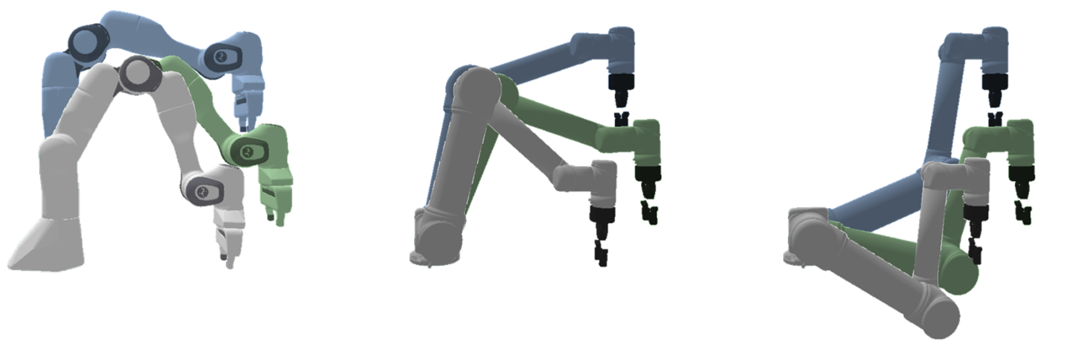


Figure 8.5 Source agent learning curve for the Panda robot with stable convergence.

The trained source agent is subsequently used to generate trajectory demonstrations within the source domain. To mitigate potential transferability issues arising from insufficient data, the previously observed number of required source domain episodes (5,000) are multiplied by a safety factor of 100, resulting in 500,000 source trajectory demonstrations. This deliberate decision to oversize the dataset ensures a diverse and extensive dataset that is highly likely to encompass the full behavioral repertoire of the source robot, while also accounting for expected mapping failures due to incompatibilities between the two robots behavioral capabilities.

Next, the source trajectory demonstrations are mapped to the target domain. Figure 8.6 depicts an exemplary trajectory mapping using the Panda robot as source and the UR5 robot as target. Figures 8.6b and 8.6c show the trajectories resulting from the most and least optimal path identified by the Dijkstra algorithm, respectively. As can be seen, the proposed mapping framework correctly produces trajectories which preserve the general shape of the arm across the two robots.



(a) Source trajectory of the Panda robot. **(b)** Best-match mapped target trajectory of the Panda robot. **(c)** Randomly mapped target trajectory of the Panda robot.

Figure 8.6 Example trajectory mapping demonstrating the mapping capabilities of the proposed CRIL framework. The general shape of the arm, i.e. the behavior, is preserved. Taken from [19].

Despite the two robots being similar in size and workspace, only 132,477 target trajectories could be successfully mapped, which is a yield of only 26.5%. As this mapping failure was accounted for by the initial oversizing of the source demonstration dataset, the resulting target trajectory dataset was deemed sufficient, still providing significantly more trajectories to the target agent than seen by the source agent.

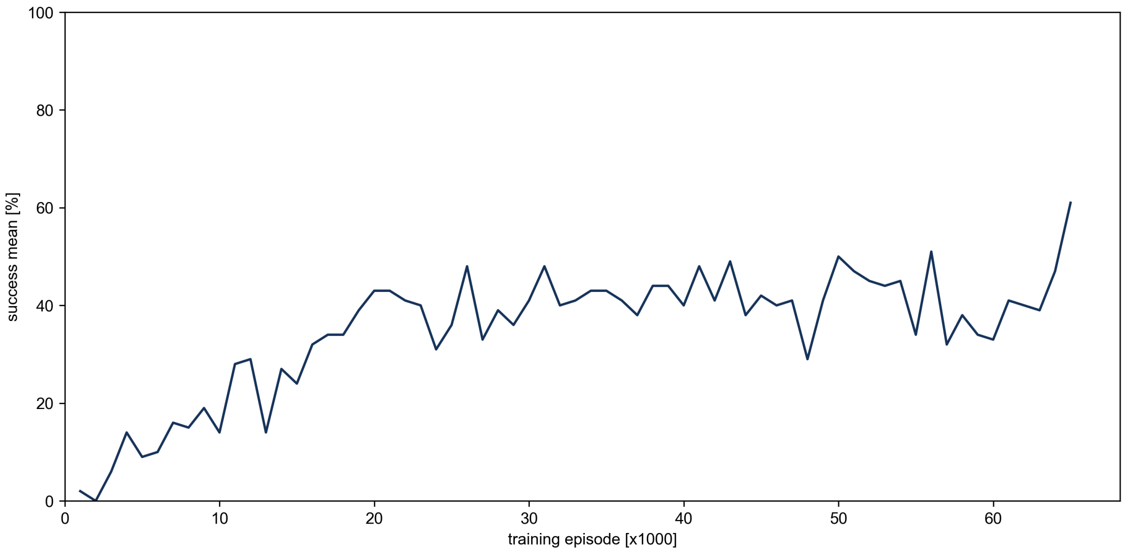


Figure 8.7 Pretraining performance of the MARWIL agent on the target UR5 robot using the mapped target trajectory dataset.

The agent pretraining learning curve with MARWIL is depicted in Figure 8.7. The performance is evaluated every 1,000 episodes by deploying the agent to 100 randomly initialized target environments and averaging the results. Note that MARWIL is a supervised learning algorithm, and thus an episode denotes one iteration through the entire target trajectory dataset. Training is performed for a total of 100,000 episodes, with an early stopping criterion of 20,000 episodes. The learning curve shows that MARWIL is able to reach a maximum success rate of 61% after 65,000 episodes.

Finally, the transfer through finetuning with PPO is evaluated, the results of which are shown in Figure 8.8. A baseline is established by training PPO agents from scratch on the target environment for 5,000 episodes. The maximum episode cutoff was selected based on the previous results on the source environment, as the learning performance was expected to be similar for both robots. Contrary to expectation however, the agents struggled more on the target environment, with learning curves converging to success rates below 80%. With this baseline, a success rate threshold of 90% is considered a significant performance improvement for the subsequent transfer attempts. As clearly visible in the figure, the proposed transfer strategy outperforms the baseline by a large margin, with the agents reaching success rates above 90% after only 1,000 episodes. This validates the effectiveness of the proposed cross-robot transfer framework.

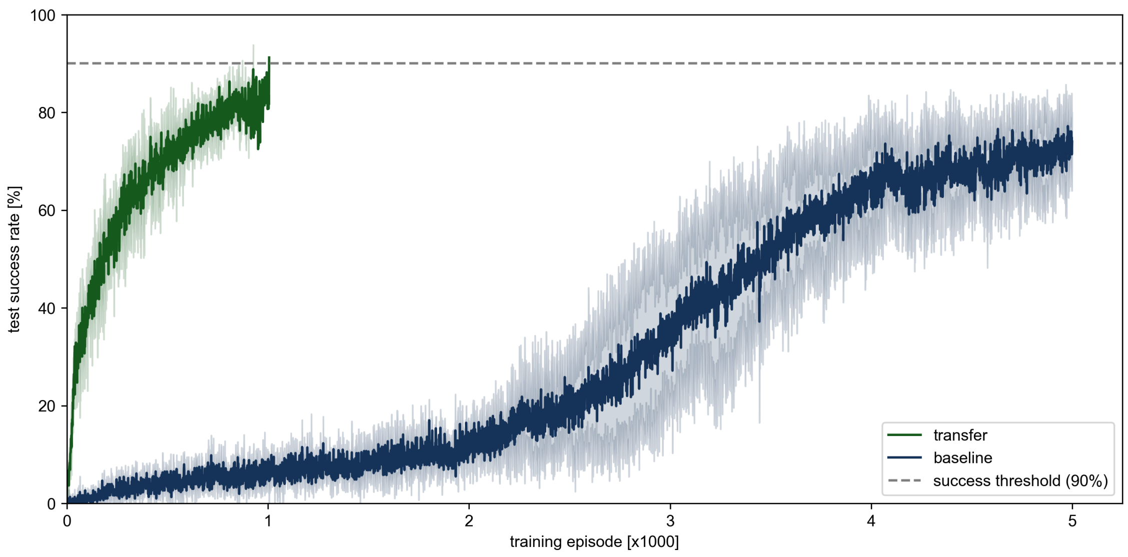


Figure 8.8 Finetuning performance of the pretrained agent on the target UR5 robot. The transfer strategy clearly outperforms agents trained from scratch. Taken from [19].

8.4 Summary

In this chapter, the fourth and final RQ was addressed, which prompted the transfer of movement trajectories across different robot morphologies. A CRIL framework was developed for transferring joint space trajectories. The approach consists of mapping the source states (joint angles) into a robot-agnostic representation space (TCP poses), which are then mapped into target state candidates using IK. A directed graph is constructed, whose nodes represent the IK candidates and edges represent the transition between two consecutive candidates. By using edge weights which capture the similarity between source and target key points (joint positions) as well as the transition cost, and subsequently finding the shortest path through the graph, a target trajectory is identified which is most similar to the source trajectory.

The experimental evaluation showed that the proposed framework is able to transfer trajectories from a source to a target robot and use these trajectories to pretrain a DRL agent with IL techniques. The results demonstrate a significant positive impact on the learning performance of the DRL agent on the target robot compared to a baseline trained from scratch.

9 Critical Reflection and Outlook

"Follow effective action with quiet reflection. From the quiet reflection will come even more effective action."
- Peter Drucker

This final chapter concludes the thesis by reflecting on the RQs and the conducted experiments. For each RQ, the main findings are summarized, and the limitations of the conducted experiments are discussed. Additionally, future research directions are proposed towards further advancing the field of DRL in industrial robotics.

9.1 Research Question 1

How can smooth, continuous movement be achieved within the RL framework?

The first RQ was addressed in chapter 5, where an asynchronous DRL framework was proposed and validated for the wire-loop use case. The main concept of the framework is based on the introduction of intermediate states, which are observed while an action is still executing, to enable a policy to preemptively determine the next action. Further, a suitable action space based on Bézier curves was proposed to enforce smoothness between individual trajectory segments. The final asynchronous framework was found to be effective in achieving smooth, continuous robot movement with position-based control, which was validated through the wire-loop use case. Despite the academic nature of the use case, the results regarding the reduction of process time and reduced wear and tear on mechanical parts through smoother motion are relevant to industrial applications. To address the non-Markovian nature of the asynchronous framework, a dynamics rollout model was proposed to explicitly model the system dynamics and detach the latency compensation from the non-delayed decision-making. The experimental evaluation revealed that the dynamics rollout model is naturally limited in its ability to predict the next state by the information provided in a given state, hallucinating hypothetical future states otherwise.

In addition, the system latencies were found to be low enough to allow for the use of intermediate states without significant performance degradation. Therefore, while the dynamics rollout model showed promise in its ability to produce future state predictions based on the current state and pose transformation, the results remain academic without a clear advantage for industrial applications.

The ineffectiveness of the dynamics rollout model in the wire-loop use case does not invalidate the concept altogether. Instead, using the wire-loop use case may be seen as a limitation in terms of not providing sufficient latencies and challenges to justify the additional complexity of the dynamics rollout model. The approach of using GANs to roll out high-dimensional state spaces can however be challenged based on the results of chapter 6, where it was shown that image-based states can be compressed into low-dimensional embeddings which contain all relevant information for a policy. Thus, reconstructing the next state, as done by the dynamics rollout model, adds additional complexity without a clear advantage apart from the interpretability of the original state space.

While lightweight policies such as the one used in the wire-loop use case are sufficiently small to run the agent on-edge, and reduce latencies to a point where their effects become negligible, it is conceivable that a growing need for intelligence in increasingly complex scenarios will require more powerful agents, and thereby more powerful hardware. Although the industrial readiness of such hardware is expected to continuously improve, becoming both more powerful and more affordable, it is also likely that trends such as Everything as a Service (XaaS) will drive applications towards distributed architectures, where the agent is running on a remote server [48, 157]. In the latter scenario, the question of handling latencies between the agent and the robotic system will stay relevant and a worthy research direction for future work. In this context, important questions are how to compress the state information to a minimum to reduce the required bandwidth but still retain the necessary information for the policy and how to learn a dynamics model in such a reduced latent space to offset latencies. Further, measuring and incorporating the stochasticity of latencies is an important topic which has been touched upon in the related work [201], but leaves room for further investigation beyond simple latency distributions.

9.2 Research Question 2

How can perception modules for vision-based robotic tasks be developed in simulation, and how can their structure be designed when the relevant low-level features are not immediately obvious?

The second RQ was investigated in chapter 6, where the methodology of DR was successfully applied to achieve sim2real transfer of a pose estimation model in the object picking use case. The main contribution focused on the wire-loop use case, where the task-relevant information is easily pointed out in the camera images, but a suitable encoding is not self-evident. In a first attempt to compress the state space, a CNN-based autoencoder was used to learn a latent space from a dataset of real-world images. While the resulting latent space was found to be usable as input for a DRL to improve on the initial baseline results, the unsupervised nature of the approach left room for improvement, such as the encoding of irrelevant features. In an exploratory study, the attention of trained DRL agents was visualized, supporting the hypothesis that agents focus on similar features as humans, i.e. the wire shape. To inject this knowledge into the training process of the perception module, segmentation maps were introduced as prediction targets to distill the autoencoder latent space on process-relevant parts. To generate sufficient data, simulations were employed, where the segmentation maps are easily generated. Although the autoencoder was able to learn the segmentation task on the simulation data, the transfer to real-world images was not successful. Based on related work that suggests focusing the DR parameter distribution improves transferability, a single real-world image was used to select more realistic pixel color values. With this final modification, the autoencoder was able to learn latent space compressions transferable to real-world images.

One aspect that was not exhaustively investigated is the amount of training data required to learn a suitable latent space. The comparative evaluation and ranking of the unsupervised, semi-supervised, and single real-world image approaches reveals that the latter objectively outperforms the others in transferability and real-world data requirements. Some of the shortcomings may however be attributed to limited data, especially in the unsupervised approach. Thus, a quantitative answer regarding how much real-world effort was saved by the proposed approach remains unclear. Further, the achievement of using a single real-world image for sim2real is heavily tied to the controlled use-case environment where a single image is enough to capture the entire color variation distribution and thus may not be claimed as an inherent property of the approach. For example, it cannot be expected that the policy will transfer seamlessly to scenarios without a black cardboard background because the state images would be outside the distribution of the simulation images. The proposed approach is expected to be very effective in industrial applications where the state space is controllable to ensure that the visual appearance does not deviate from the injected domain knowledge. However, for applications where this is not the case it is likely to fail or require additional effort to fully capture the required parameter randomization distribution.

Looking forward, several aspects remain open for future work. First, attention maps were found to be a valuable tool for qualitatively analyzing the sim2real transferability. Based on this analysis, quantitative metrics of the sim2real gap for vision-based state spaces could be developed. One potential approach could lie in the combination of attention and segmentation maps to measure how much attention is placed on manually identified relevant segments. Further, the human-in-the-loop domain knowledge injection approach by means of annotated real-world data leaves room for improvement and automation. On the one hand, the likely need for additional real-world annotations in more complex scenarios elaborated on above can be supported by the use of SOTA segmentation models, such as SAM [86], as well as methods to automatically identify which real-world images to annotate, e.g. by using anomaly detection techniques [149]. Mapping the real-world image segments to simulation objects might also be performed without human involvement, e.g. by random assignment. While this would expand the distribution away from the real-world data, it can be expected that the transferability of the final model is not affected, as the optimal color combination remains part of the source domain distribution. In fact, it could even have a positive effect on the previous point of generalizability through increased randomization.

Another important aspect to address is the limitation of using learned latent spaces with respect to true modularization of the perception module. As the latent space is arbitrarily learned by the perception module, the development of the execution module can only commence after the perception module is trained. In addition, every retraining of the perception module will lead to a different latent space, i.e. a completely different interface. Investigating how latent spaces can be constrained to be consistent, e.g. by regularization of the latent spaces, or alternatively finding ways to easily adapt two independently trained interfaces is a highly relevant topic for future work towards truly modular DRL policies and parallel development of such modules.

9.3 Research Question 3

How can decomposition of robotic movement be leveraged to modularize task-independent behavior policies, and how can reversibility of robotic movement be exploited to improve training efficiency?

The third RQ was investigated in chapter 7, where for the first part of the RQ, the HAC framework was applied to successfully learn the Fetch benchmark suite, most notably solving the PickAndPlace task which the previous non-hierarchical DRL approach failed on. To validate the hypothesis that lower layers in a hierarchical agent encapsulate task-independent behavior, they were transferred to task variations, whereby the higher level policy was re-initialized before finetuning the entire HAC agent on the respective new scenario. The results demonstrated mixed cross-task transferability. The transfer was found to be of significant benefit for harder tasks, such as the PickAndPlace variations. However, for easier tasks, such as the Reach and Push tasks, learning from scratch outperformed the transferred agents.

While the general suitability of HRL as a decomposition approach is confirmed as prompted by the RQ, the followed transfer methodology can be improved. In the experimental studies, transfer was conducted by re-initializing the high-level policy weights and continuing training on the target environment. This is a rather simple approach, which as observed bears risk of degrading performance, presumably due to the pretrained lower-layer adjusting to the subgoals provided by a newly trained high-layer. A more sophisticated transfer methodology would take this into account, potentially by initially freezing the lower-level policy weights during the transfer process, and only unfreezing them after the high-layer has managed to utilize the pretrained behavior. Balancing the preservation of existing skills with the potential necessity to have the lower layer acquire new behavior patterns required for a new task is non-trivial and prompts a dedicated investigation.

It must be acknowledged that the lower-level policies are not fully task-agnostic, but rather agnostic to task variations. This is on the one hand due to the subgoal space being derived from the original environments goal spaces, which for example describes the TCP position in the Reach task, and the cube position in the Push and PickAndPlace tasks. Thus, a transfer across different tasks is not expected to be successful. On the other hand, the interpretation of the subgoals differs between tasks. Where the lower-level policy in the PickAndPlace task implicitly learns to pick up the cube first, the lower-level policy in the Reach task learns to hit the cube towards the goal. Between the manual design of general-purpose subgoal spaces which only contain robot-specific information about the TCP and gripper on one end of the spectrum, and the application of unsupervised DRL to automatically learn task-agnostic skill embeddings on the other end, there are many possible research avenues worth exploring.

The investigations of this thesis also do not answer the question of when such a decomposition is actually beneficial, apart from a general observation that the benefit seems to correlate with increasing task complexity. With a suboptimally designed hierarchy, the learning performance is degraded. Importantly, this is true in both directions, as the hierarchy can be too shallow, e.g. a one-layer hierarchy in the harder PickAndPlace task, or too deep, e.g. a two-layer hierarchy in the easier Reach task. Instead of relying on human intuition to design the hierarchy, further research is required to guide the hyperparameter choice, such as the number of layers and maximum sub-trajectory length, in a more systematic way. One important first step would lie in a larger scale of experiments to investigate potential correlations between shared task complexity metrics, e.g. the episode length of an optimal policy, and the best-performing HRL architecture. This already would provide a valuable prior for the agent design process. As a complementary solution, finding a methodology that adapts the hierarchical structure throughout training by adding or removing temporal abstraction layers could address the issue in a curriculum learning fashion. Here, two important questions are to be investigated. On the one hand, how to determine when to change the architecture, which in its simplest form can follow a fixed curriculum. On the other hand, how to transfer knowledge between the architecture changes to improve on the naive approach of training from scratch. For the latter, IL techniques such as BC, which were successfully applied in this thesis, can serve as a suitable starting point.

For the second part of RQ-3, the concept of assembly-by-disassembly was applied in the context of DRL. The proposed approach of learning the inverted task of disassembly first and using the inverted trajectories as basis to pretrain a DRL agent for assembly using IL was successfully verified by solving the original assembly task of the clip assembly use case. The investigations revealed that partial reversibility has to be accounted for to avoid arriving at non-reversible trajectories. The implemented solution was to explicitly check for reversibility of the trajectory after every action, which could be reduced to checking the respective last two actions. While this validation approach is effective, it is not optimal in terms of computational efficiency, requiring three times the original number of simulation steps. One potential measure to avoid the need for this validation overhead is to use a different simulation environment, where movement is not achieved through force constraints, but e.g. by directly simulating position-based control.

To further improve on the proposed approach in future work, several aspects remain open for investigation. Instead of the chosen straightforward approach of sequentially learning the respective agents, a more sophisticated approach could be to learn both tasks at the same time, either by training both agents in parallel, or by learning a joint policy for disassembly and assembly. A potential training strategy could be to alternate between the two tasks in a curriculum or adversarial learning setup, where the goal of each agent can be viewed as resetting the environment for the respective other task. A potential advantage of this approach is that it is much better suited for training in real-world environments, where resetting the environment is a challenge in itself. In this context, another interesting aspect to consider is to allow for partial reversibility of the trajectory, instead of eliminating it by design. This would allow for the individual disassembly and assembly agents to find respective optimal policies, which may include non-reversible sub-trajectories. How to effectively identify and share reversible solutions between the two tasks, and at the same time leave room for task-specific optimal solutions, is an important question to explore in future research.

9.4 Research Question 4

How can movement trajectories be transferred between different robot morphologies?

The last RQ was addressed in chapter 8, where a CRIL framework was proposed. The approach uses FK to map the joint angles of a source robot to a set of key points along the kinematic chain, including the TCP pose. From this TCP pose, IK is computed to find corresponding joint angles for a target robot. To resolve the ambiguity of IK producing multiple potential joint angle candidates, an embodiment correspondence metric is used to capture the similarity of the source and target robot configurations. Further, a graph is constructed from mapping all the states of a source trajectory, where each path through the graph represents a potential trajectory for the target robot. By introducing edge weights based on the embodiment correspondence metric and estimated transition costs between individual states, the best target trajectory is found by means of shortest path search. The proposed approach was validated by using a set of mapped demonstrations for pretraining a target domain agent with IL, which significantly reduced the required training effort.

The proposed approach is a promising step towards cross-robot transferability, but leaves room for improvement. First, the embodiment correspondence metric is a rather simplistic measure, which does not fully capture the similarity between two robots. Further, matching the key points to each other based on the location in the kinematic chain is a reasonable strategy, but also just a rough approximation prone to errors. Therefore, exploring more sophisticated metrics is a valuable future research direction. With the availability of 3D mesh models, it is for example conceivable to calculate the true overlap between the two robots, or vice versa and arguably more important, identify differences which may lead to unexpected collisions of the target robot. Here, finding a balance between infinitely precise comparisons and reasonable approximations, e.g. convex hulls, to ensure computational feasibility will be a key challenge to consider.

Another key limitation of the presented approach lies in mapping individual states between source and target robots, which fails to capture the temporal dependencies and structure of entire behaviors. This state-wise mapping also leads to infeasible transitions in the target domain, explaining the high rate of unsuccessful mappings observed in the experiments. A more robust solution would be to employ sequence-to-sequence mapping, aligning and transferring entire trajectories rather than isolated states, thereby preserving the underlying behavioral intent and improving the success rate of cross-robot transfer. This goal of capturing behavior is closely related to the previously mentioned research direction regarding comprehensive skill embeddings.

9.5 Closing Remarks

The vision of modular DRL agents is one of flexible, reusable components promising to enable general-purpose autonomy in dynamic and heterogeneous environments, as well as potential for interpretability of decision-making processes. The findings of this thesis offer a foundational step in this direction. They highlight not only the feasibility but also the practical benefits of modularizing DRL architectures in industrial settings to enable knowledge transfer from simulations, across different task variations, and between different robot models.

Looking ahead, future work should delve deeper into the role of skills or behaviors as natural interfaces between task-specific and robot-specific modules. Key open questions include how to effectively train such skill embeddings and, equally important, how to ensure that these embeddings are maximally reusable and interpretable. While algorithmic advances are critical for building more capable agents, the human element remains indispensable. Developing efficient human-in-the-loop interaction strategies is a largely untapped challenge in DRL, where real-time guidance, correction, and teaching from humans could exponentially improve agent learning. Achieving this will demand progress not only in human-robot communication but also in adaptive learning algorithms that can seamlessly incorporate human feedback.

Ultimately, advancing modular DRL agents is crucial for building robotic systems that are not only intelligent and adaptive, but also maintainable, interpretable, and deployable at scale across diverse domains and applications.

Bibliography

[1] Ahmad, S., Lavin, A., Purdy, S., and Agha, Z. Unsupervised real-time anomaly detection for streaming data. *Neurocomputing 262* (Nov. 2017), 134–147.

[2] Akbulut, M., Oztop, E., Seker, M. Y., Hh, X., Tekden, A., and Ugur, E. Acnmp: Skill transfer and task extrapolation through learning from demonstration and reinforcement learning via representation sharing. In *Conference on Robot Learning* (2021), PMLR, pp. 1896–1907.

[3] Akkaya, I., Andrychowicz, M., Chociej, M., Litwin, M., McGrew, B., Petron, A., Paino, A., Plappert, M., Powell, G., and Ribas, R. Solving rubik's cube with a robot hand. *arXiv preprint arXiv:1910.07113* (2019).

[4] Ammar, H. B., Eaton, E., Ruvolo, P., and Taylor, M. Unsupervised cross-domain transfer in policy gradient reinforcement learning via manifold alignment. In *Proceedings of the AAAI Conference on Artificial Intelligence* (2015), vol. 29.

[5] Ammar, H. B., Mocanu, D. C., Taylor, M. E., Driessens, K., Tuyls, K., and Weiss, G. Automatically Mapped Transfer between Reinforcement Learning Tasks via Three-Way Restricted Boltzmann Machines. In *Advanced Information Systems Engineering*, D. Hutchison, T. Kanade, J. Kittler, J. M. Kleinberg, F. Mattern, J. C. Mitchell, M. Naor, O. Nierstrasz, C. Pandu Rangan, B. Steffen, M. Sudan, D. Terzopoulos, D. Tygar, M. Y. Vardi, G. Weikum, C. Salinesi, M. C. Norrie, and Ó. Pastor, Eds., vol. 7908. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, pp. 449–464.

[6] Amodei, D., Olah, C., Steinhardt, J., Christiano, P., Schulman, J., and Mané, D. Concrete Problems in AI Safety, July 2016.

[7] Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Pieter Abbeel, O., and Zaremba, W. Hindsight Experience Replay. In *Advances in Neural Information Processing Systems* (2017), vol. 30, Curran Associates, Inc.

[8] Andrychowicz, O. M., Baker, B., Chociej, M., Józefowicz, R., McGrew, B., Pachocki, J., Petron, A., Plappert, M., Powell, G., Ray, A., Schneider, J., Sidor, S., Tobin, J., Welinder, P., Weng, L., and Zaremba, W. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research 39*, 1 (Jan. 2020), 3–20.

[9] Bain, M., and Sammut, C. A Framework for Behavioural Cloning. In *Machine Intelligence 15* (1995), pp. 103–129.

[10] Ballard, D. H., and Brown, C. M. *Computer Vision*, 1st ed. Prentice Hall Professional Technical Reference, Mar. 1982.

[11] Baram, N., and Mannor, S. Inspiration Learning through Preferences, Sept. 2018.

[12] Barreto, A., Borsa, D., Hou, S., Comanici, G., Aygün, E., Hamel, P., Toyama, D., hunt, J., Mourad, S., Silver, D., and Precup, D. The Option Keyboard: Combining Skills in Reinforcement Learning. In *Advances in Neural Information Processing Systems* (2019), vol. 32, Curran Associates, Inc.

[13] Barto, A. G., and Mahadevan, S. Recent Advances in Hierarchical Reinforcement Learning. *Discrete Event Dynamic Systems 13*, 4 (Oct. 2003), 341–379.

[14] Beaussant, S., Lengagne, S., Thuilot, B., and Stasse, O. Towards zero-shot cross-agent transfer learning via latent-space universal notice network. *Robotics and Autonomous Systems 184* (2025).

[15] Bellgardt, M., Scheiderer, C., and Kuhlen, T. W. An Immersive Node-Link Visualization of Artificial Neural Networks for Machine Learning Experts. In *2020 IEEE International Conference on Artificial Intelligence and Virtual Reality (AIVR)* (Dec. 2020), pp. 33–36.

[16] Berner, C., Brockman, G., Chan, B., Cheung, V., Dębiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., and Hesse, C. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680* (2019).

[17] Beyret, B., Shafti, A., and Faisal, A. A. Dot-to-Dot: Explainable Hierarchical Reinforcement Learning for Robotic Manipulation. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (Nov. 2019), pp. 5014–5019.

[18] Bézier, P. Numerical control-mathematics and applications. *Translated by AR Forrest* (1972).

[19] Bitter, C., Peters, J., Tercan, H., and Meisen, T. Industrial Cross-Robot Transfer Learning. *Procedia CIRP 120* (Jan. 2023), 1297–1302.

[20] Bitter, C., Tercan, H., Meisen, T., Bodnar, T., and Meisen, P. When to Message: Investigating User Response Prediction with Machine Learning for Advertisement Emails. In *2021 4th International Conference on Artificial Intelligence for Industries* (Piscataway, NJ, 2021), K. Li and J. Shih, Eds., IEEE, pp. 25–29.

[21] Bitter, C., Thun, T., and Meisen, T. Karolos: An Open-Source Reinforcement Learning Framework for Robot-Task Environments, Dec. 2022.

[22] Bócsi, B., Csató, L., and Peters, J. Alignment-based transfer learning for robot models. In *The 2013 International Joint Conference on Neural Networks (IJCNN)* (Aug. 2013), pp. 1–7.

[23] Bogue, R. Bin picking: A review of recent developments. *Industrial Robot: the international journal of robotics research and application 50*, 6 (2023), 873–877.

[24] Böhm, P., Pounds, P., and Chapman, A. C. Non-blocking Asynchronous Training for Reinforcement Learning in Real-World Environments. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (Oct. 2022), pp. 10927–10934.

[25] Böhm, P., Pounds, P., and Chapman, A. C. Feature Extraction for Effective and Efficient Deep Reinforcement Learning on Real Robotic Platforms. In *2023 IEEE International Conference on Robotics and Automation (ICRA)* (May 2023), pp. 7126–7132.

[26] Bourou, A., Mezger, V., and Genovesio, A. GANs Conditioning Methods: A Survey, Sept. 2024.

[27] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. OpenAI Gym, June 2016.

[28] Brodtmann, N. L., and Schilberg, D. Dynamic Bézier Curves, New Findings on Reparameterization by Arc length. *Intelligent Human Systems Integration (IHSI 2022): Integrating People and Intelligent Systems 22*, 22 (2022).

[29] Brown, N., and Sandholm, T. Superhuman AI for multiplayer poker. *Science 365*, 6456 (2019), 885–890.

[30] Buchholz, D. *Bin-Picking—5 Decades of Research*, vol. 44. Springer International Publishing, Cham, 2016, pp. 3–12.

[31] Chai, J., Zeng, H., Li, A., and Ngai, E. W. Deep learning in computer vision: A critical review of emerging techniques and application scenarios. *Machine Learning with Applications 6* (2021), 100134.

[32] Chen, B., Xu, M., Li, L., and Zhao, D. Delay-Aware Model-Based Reinforcement Learning for Continuous Control. *Neurocomputing 450* (2020), 119–128.

[33] Chiuso, A., and Pillonetto, G. System Identification: A Machine Learning Perspective. *Annual Review of Control, Robotics, and Autonomous Systems 2*, Volume 2, 2019 (May 2019), 281–304.

[34] Christen, S., Jendele, L., Aksan, E., and Hilliges, O. Learning functionally decomposed hierarchies for continuous control tasks with path planning. *IEEE Robotics and Automation Letters 6*, 2 (2021), 3623–3630.

[35] Coumans, E., and Bai, Y. Pybullet, a python module for physics simulation for games, robotics and machine learning, 2016.

[36] Custode, L. L., and Iacca, G. Interpretable pipelines with evolutionary optimized modules for reinforcement learning tasks with visual inputs. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion* (Boston Massachusetts, July 2022), ACM, pp. 224–227.

[37] Dara, S., and Tumma, P. Feature Extraction By Using Deep Learning: A Survey. In *2018 Second International Conference on Electronics, Communication and Aerospace Technology (ICECA)* (Mar. 2018), pp. 1795–1801.

[38] de Koster, R., Le-Duc, T., and Roodbergen, K. J. Design and control of warehouse order picking: A literature review. *European Journal of Operational Research 182*, 2 (Oct. 2007), 481–501.

[39] De Mello, L. H., and Sanderson, A. C. A correct and complete algorithm for the generation of mechanical assembly sequences. In *1989 IEEE International Conference on Robotics and Automation* (1989), IEEE Computer Society, pp. 56–57.

[40] de Wit, C. C., Siciliano, B., and Bastin, G. *Theory of Robot Control.* Springer Science & Business Media, 2012.

[41] Denavit, J., and Hartenberg, R. S. A Kinematic Notation for Lower-Pair Mechanisms Based on Matrices. *Journal of Applied Mechanics 22*, 2 (June 1955), 215–221.

[42] Derman, E., Dalal, G., and Mannor, S. Acting in Delayed Environments with Non-Stationary Markov Policies. In *International Conference on Learning Representations* (2021).

[43] Devin, C., Gupta, A., Darrell, T., Abbeel, P., and Levine, S. Learning modular neural network policies for multi-task and multi-robot transfer. In *2017 IEEE International Conference on Robotics and Automation (ICRA)* (May 2017), pp. 2169–2176.

[44] Dhariwal, P., Hesse, C., Klimov, O., Nichol, A., Plappert, M., Radford, A., Schulman, J., Sidor, S., Wu, Y., and Zhokhov, P. OpenAI baselines, 2017.

[45] Digney, B. Skill transfer and training in emergent hierarchical control systems. In *Proceedings of the 1996 IEEE International Symposium on Intelligent Control* (Sept. 1996), pp. 74–79.

[46] Dijkstra, E. W. A note on two problems in connexion with graphs. *Numerische Mathematik 1*, 1 (Dec. 1959), 269–271.

[47] Downs, L., Francis, A., Koenig, N., Kinman, B., Hickman, R., Reymann, K., McHugh, T. B., and Vanhoucke, V. Google Scanned Objects: A High-Quality Dataset of 3D Scanned Household Items. In *2022 International Conference on Robotics and Automation (ICRA)* (May 2022), pp. 2553–2560.

[48] Duan, Y., Fu, G., Zhou, N., Sun, X., Narendra, N. C., and Hu, B. Everything as a service (XaaS) on the cloud: Origins, current and future trends. In *2015 IEEE 8th International Conference on Cloud Computing* (2015), IEEE, pp. 621–628.

[49] Dulac-Arnold, G., Mankowitz, D., and Hester, T. Challenges of Real-World Reinforcement Learning, Apr. 2019.

[50] Encyclopedia of Mathematics. Bézier curve. https://encyclopediaofmath.org/index.php?title=B%C3%A9zier_curve.

[51] Fachantidis, A., Partalas, I., Taylor, M. E., and Vlahavas, I. Transfer learning with probabilistic mapping selection. *Adaptive Behavior 23*, 1 (Feb. 2015), 3–19.

[52] Farin, G. *Curves and Surfaces for Computer-Aided Geometric Design: A Practical Guide*. Elsevier, 2014.

[53] Feng, Y., Yu, W., Chen, Y., Tan, X., Wang, R., and Madani, K. Option-based motion planning and ANFIS-based tracking control for wheeled robot in cluttered environment. In *2015 12th International Conference on Informatics in Control, Automation and Robotics (ICINCO)* (July 2015), vol. 01, pp. 287–293.

[54] Fickinger, A., Cohen, S., Russell, S., and Amos, B. Cross-Domain Imitation Learning via Optimal Transport, Apr. 2022.

[55] Firoiu, V., Ju, T., and Tenenbaum, J. At Human Speed: Deep Reinforcement Learning with Action Delay. *ArXiv abs/1810.07286* (2018), null.

[56] Florensa, C., Duan, Y., and Abbeel, P. Stochastic Neural Networks for Hierarchical Reinforcement Learning, Apr. 2017.

[57] Frans, K., Ho, J., Chen, X., Abbeel, P., and Schulman, J. Meta Learning Shared Hierarchies, Oct. 2017.

[58] Gallager, R. G. Reversible Markov chains. In *Discrete Stochastic Processes*, vol. 48 of *Journal of the Operational Research Society*. Taylor \& Francis, Jan. 1997, pp. 240–245.

[59] Gökçe, B., and Akin, H. L. Implementation of Reinforcement Learning by transfering sub-goal policies in robot navigation. In *2013 21st Signal Processing and Communications Applications Conference (SIU)* (Apr. 2013), pp. 1–4.

[60] Goodfellow, I., Bengio, Y., Courville, A., and Bengio, Y. *Deep Learning*, vol. 1. MIT press Cambridge, 2016.

[61] Grondman, I., Busoniu, L., Lopes, G. A., and Babuska, R. A survey of actor-critic reinforcement learning: Standard and natural policy gradients. *IEEE Transactions on Systems, Man, and Cybernetics, part C (applications and reviews) 42*, 6 (2012), 1291–1307.

[62] Guédon, A., and Lepetit, V. Sugar: Surface-aligned gaussian splatting for efficient 3d mesh reconstruction and high-quality mesh rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2024), pp. 5354–5363.

[63] Gui, H., Pang, S., Fan, Z., and Wang, T. Two-stage motion strategy transfer for robots across simulation and physical domains based on reinforcement learning. In *2024 6th International Conference on Frontier Technologies of Information and Computer (ICFTIC)* (2024), pp. 275–278.

[64] Gupta, A., Devin, C., Liu, Y., Abbeel, P., and Levine, S. Learning Invariant Feature Spaces to Transfer Skills with Reinforcement Learning, Mar. 2017.

[65] Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P., and Levine, S. Soft Actor-Critic Algorithms and Applications, Jan. 2019.

[66] Hao, C., Weaver, C., Tang, C., Kawamoto, K., Tomizuka, M., and Zhan, W. Skill-Critic: Refining Learned Skills for Hierarchical Reinforcement Learning. *IEEE Robotics and Automation Letters 9*, 4 (Apr. 2024), 3625–3632.

[67] Hartenberg, R., and Danavit, J. *Kinematic Synthesis of Linkages*. New York: McGraw-Hill, 1964.

[68] Hawkins, K. P. Analytic inverse kinematics for the universal robots UR-5/UR-10 arms. Tech. rep., Georgia Institute of Technology, 2013.

[69] Hayes, B., and Scassellati, B. Autonomously constructing hierarchical task networks for planning and human-robot collaboration. In *2016 IEEE International Conference on Robotics and Automation (ICRA)* (May 2016), pp. 5469–5476.

[70] He, Y., and Liu, S. Analytical inverse kinematics for franka emika panda–a geometrical solver for 7-dof manipulators with unconventional design. In *2021 9th International Conference on Control, Mechatronics and Automation (ICCMA)* (2021), IEEE, pp. 194–199.

[71] Heess, N., Wayne, G., Tassa, Y., Lillicrap, T., Riedmiller, M., and Silver, D. Learning and Transfer of Modulated Locomotor Controllers, Oct. 2016.

[72] Hermann, L., Argus, M., Eitel, A., Amiranashvili, A., Burgard, W., and Brox, T. Adaptive Curriculum Generation from Demonstrations for Sim-to-Real Visuomotor Control. In *2020 IEEE International Conference on Robotics and Automation (ICRA)* (May 2020), pp. 6498–6505.

[73] Hester, T., and Stone, P. TEXPLORE: Real-time sample-efficient reinforcement learning for robots. *Machine Learning 90*, 3 (Mar. 2013), 385–429.

[74] Hwang, J.-H., Arkin, R. C., and Kwon, D.-S. Mobile robots at your fingertip: Bezier curve on-line trajectory generation for supervisory control. In *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)(Cat. No. 03CH37453)* (2003), vol. 2, IEEE, pp. 1444–1449.

[75] Ibarz, J., Tan, J., Finn, C., Kalakrishnan, M., Pastor, P., and Levine, S. How to train your robot with deep reinforcement learning: Lessons we have learned. *The International Journal of Robotics Research 40*, 4-5 (Apr. 2021), 698–721.

[76] International Organization of Standardization. ISO 8373:2021 Robotics - Vocabulary, 2021.

[77] Isola, P., Zhu, J.-Y., Zhou, T., and Efros, A. A. Image-To-Image Translation With Conditional Adversarial Networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2017), pp. 1125–1134.

[78] Jaffri, A. Hype cycle for artificial intelligence 2024. https://www.gartner.com/en/articles/hype-cycle-for-artificial-intelligence, Nov. 2024.

[79] Jakobi, N. Evolutionary Robotics and the Radical Envelope-of-Noise Hypothesis. *Adaptive Behavior 6*, 2 (Sept. 1997), 325–368.

[80] James, S., Davison, A. J., and Johns, E. Transferring End-to-End Visuomotor Control from Simulation to Real World for a Multi-Stage Task. In *Proceedings of the 1st Annual Conference on Robot Learning* (Oct. 2017), PMLR, pp. 334–343.

[81] Jaritz, M., de Charette, R., Toromanoff, M., Perot, E., and Nashashibi, F. End-to-End Race Driving with Deep Reinforcement Learning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)* (May 2018), pp. 2070–2075.

[82] Josifovski, J., Auddy, S., Malmir, M., Piater, J., Knoll, A., and Navarro-Guerrero, N. Continual Domain Randomization. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (Oct. 2024), pp. 4965–4972.

[83] Katsikopoulos, K., and Engelbrecht, S. Markov decision processes with delays and asynchronous cost collection. *IEEE Transactions on Automatic Control 48*, 4 (Apr. 2003), 568–574.

[84] Kim, K., Gu, Y., Song, J., Zhao, S., and Ermon, S. Domain Adaptive Imitation Learning. In *Proceedings of the 37th International Conference on Machine Learning* (Nov. 2020), PMLR, pp. 5286–5295.

[85] Kim, N. H., Xie, Z., and Panne, M. Learning to correspond dynamical systems. In *Learning for Dynamics and Control* (2020), PMLR, pp. 105–117.

[86] Kirillov, A., Mintun, E., Ravi, N., Mao, H., Rolland, C., Gustafson, L., Xiao, T., Whitehead, S., Berg, A. C., and Lo, W.-Y. Segment anything. In *Proceedings of the IEEE/CVF International Conference on Computer Vision* (2023), pp. 4015–4026.

[87] Klump, R., Jurkat, A., and Schneider, F. *Tracking the Rise of Robots: A Survey of the IFR Database and Its Applications.* Munich Personal RePEc Archive, 2021.

[88] Krug, R., Stoyanov, T., Tincani, V., Andreasson, H., Mosberger, R., Fantoni, G., and Lilienthal, A. J. The Next Step in Robot Commissioning: Autonomous Picking and Palletizing. *IEEE Robotics and Automation Letters 1*, 1 (Jan. 2016), 546–553.

[89] Kwok, J., Lohstroh, M., and Lee, E. A. Efficient Parallel Reinforcement Learning Framework Using the Reactor Model. In *Proceedings of the 36th ACM Symposium on Parallelism in Algorithms and Architectures* (Nantes France, June 2024), ACM, pp. 41–51.

[90] LeCun, Y., Bengio, Y., and Hinton, G. Deep learning. *nature 521*, 7553 (2015), 436–444.

[91] LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. Backpropagation applied to handwritten zip code recognition. *Neural computation 1*, 4 (1989), 541–551.

[92] Leibovich, G., Jacob, G., Endrawis, S., Novik, G., and Tamar, A. Validate on Sim, Detect on Real - Model Selection for Domain Randomization. In *2022 International Conference on Robotics and Automation (ICRA)* (May 2022), pp. 7528–7535.

[93] Levine, S., Finn, C., Darrell, T., and Abbeel, P. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research 17*, 39 (2016), 1–40.

[94] Levine, S., Pastor, P., Krizhevsky, A., Ibarz, J., and Quillen, D. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *The International Journal of Robotics Research 37*, 4-5 (Apr. 2018), 421–436.

[95] Levy, A., Platt, R., and Saenko, K. Hierarchical actor-critic. *arXiv preprint arXiv:1712.00948 12* (2017).

[96] Li, P., Pei, Y., and Li, J. A comprehensive survey on design and application of autoencoder in deep learning. *Applied Soft Computing 138* (2023), 110176.

[97] Li, Y., Zhang, R., and Jiang, D. Order-Picking Efficiency in E-Commerce Warehouses: A Literature Review. *Journal of Theoretical and Applied Electronic Commerce Research 17*, 4 (Dec. 2022), 1812–1830.

[98] Liang, E., Liaw, R., Nishihara, R., Moritz, P., Fox, R., Goldberg, K., Gonzalez, J., Jordan, M., and Stoica, I. RLlib: Abstractions for distributed reinforcement learning. In *International Conference on Machine Learning* (2018), PMLR, pp. 3053–3062.

[99] Liang, E., Wu, Z., Luo, M., Mika, S., Gonzalez, J. E., and Stoica, I. Rllib flow: Distributed reinforcement learning is a dataflow problem. *Advances in Neural Information Processing Systems 34* (2021), 5506–5517.

[100] Liang, J., Makoviychuk, V., Handa, A., Chentanez, N., Macklin, M., and Fox, D. GPU-Accelerated Robotic Simulation for Distributed Reinforcement Learning, Oct. 2018.

[101] Liebers, C., Megarajan, P., Auda, J., Stratmann, T. C., Pfingsthorn, M., Gruenefeld, U., and Schneegass, S. Keep the Human in the Loop: Arguments for Human Assistance in the Synthesis of Simulation Data for Robot Training. *MULTIMODAL TECHNOLOGIES AND INTERACTION 8*, 3 (Mar. 2024), 18.

[102] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. Continuous control with deep reinforcement learning, July 2019.

[103] Limoyo, O., Konar, A., Ablett, T., Kelly, J., Hogan, F. R., and Dudek, G. Working Backwards: Learning to Place by Picking. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2024), IEEE, pp. 11155–11162.

[104] Liu, M., Tan, Y., and Padois, V. Generalized hierarchical control. *Autonomous Robots 40*, 1 (Jan. 2016), 17–31.

[105] Liu, Y., and Stone, P. Value-function-based transfer for reinforcement learning using structure mapping. In *Proceedings of the National Conference on Artificial Intelligence* (2006), vol. 21, Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, p. 415.

[106] Lu, Y., Fu, J., Tucker, G., Pan, X., Bronstein, E., Roelofs, B., Sapp, B., White, B., Faust, A., Whiteson, S., Anguelov, D., and Levine, S. Imitation Is Not Enough: Robustifying Imitation with Reinforcement Learning for Challenging Driving Scenarios, Dec. 2022.

[107] Makedon, V., Mykhailenko, O., and Vazov, R. Dominants and Features of Growth of the World Market of Robotics. *European Journal of Management Issues 29*, 3 (2021), 133–141.

[108] Mamou, K., Lengyel, E., and Peters, A. Volumetric hierarchical approximate convex decomposition. *Game engine gems 3* (2016), 141–158.

[109] Maschler, B., Vietz, H., Tercan, H., Bitter, C., Meisen, T., and Weyrich, M. Insights and example use cases on industrial transfer learning. *Procedia CIRP 107* (2022), 511–516.

[110] Matsuo, Y., LeCun, Y., Sahani, M., Precup, D., Silver, D., Sugiyama, M., Uchibe, E., and Morimoto, J. Deep learning, reinforcement learning, and world models. *Neural networks : the official journal of the International Neural Network Society 152* (2022), 267–275.

[111] Mazzotti, L., Angelini, M., and Carricato, M. Solving the Wire Loop Game with a reinforcement-learning controller based on haptic feedback. In *2024 20th IEEE/ASME International Conference on Mechatronic and Embedded Systems and Applications (MESA)* (Sept. 2024), pp. 1–8.

[112] McKinsey & Company. Automation at scale: The benefits for payers, 2019.

[113] Mehta, B., Diaz, M., Golemo, F., Pal, C. J., and Paull, L. Active Domain Randomization. In *Proceedings of the Conference on Robot Learning* (May 2020), PMLR, pp. 1162–1176.

[114] Meyes, R., Scheiderer, C., and Meisen, T. Continuous motion planning for industrial robots based on direct sensory input. *Procedia CIRP 72* (2018), 291–296.

[115] Meyes, R., Scheiderer, C., Thiele, T., and Meisen, T. Selbstlernende adaptive Robotersteuerung: Kontinuierliche Bewegungsplanung für Industrieroboter auf Basis von Sensordaten. *Fabriksoftware* (2018), 42–44.

[116] Meyes, R., Tercan, H., Roggendorf, S., Thiele, T., Büscher, C., Obdenbusch, M., Brecher, C., Jeschke, S., and Meisen, T. Motion Planning for Industrial Robots using Reinforcement Learning. *Procedia CIRP 63* (Jan. 2017), 107–112.

[117] Milani, S., Topin, N., Veloso, M., and Fang, F. Explainable Reinforcement Learning: A Survey and Comparative Review. *ACM Computing Surveys 56*, 7 (July 2024), 1–36.

[118] Morgenstern, D., and Bellman, R. Adaptive control processes: A guided tour. *Econometrica 30*, 3 (1962), 599.

[119] Mujhid, A., Surono, S., Irsalinda, N., and Thobirin, A. Comparison and combination of Leaky ReLU and ReLU activation function and three optimizers on deep CNN for COVID-19 detection. In *Fuzzy Systems and Data Mining VIII*. IOS Press, 2022, pp. 50–57.

[120] Müller, C. *World Robotics 2022 – Industrial Robots*. IFR Statistical Department, VDMA Services GmbH, Frankfurt am Main, Germany, 2022.

[121] Müller, C. *World Robotics 2024 – Industrial Robots*. IFR Statistical Department, VDMA Services GmbH, Frankfurt am Main, Germany, 2024.

[122] Müller, M., Dosovitskiy, A., Ghanem, B., and Koltun, V. Driving Policy Transfer via Modularity and Abstraction, Dec. 2018.

[123] Muratore, F., Eilers, C., Gienger, M., and Peters, J. Data-Efficient Domain Randomization With Bayesian Optimization. *IEEE Robotics and Automation Letters 6*, 2 (Apr. 2021), 911–918.

[124] Muratore, F., Gienger, M., and Peters, J. Assessing Transferability From Simulation to Reality for Reinforcement Learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence 43*, 4 (Apr. 2021), 1172–1183.

[125] Muratore, F., Treede, F., Gienger, M., and Peters, J. Domain randomization for simulation-based policy optimization with transferability assessment. vol. 87 of *Proceedings of Machine Learning Research*, pp. 700–713.

[126] Nachum, O., Gu, S. S., Lee, H., and Levine, S. Data-efficient hierarchical reinforcement learning. In *Advances in Neural Information Processing Systems* (2018), vol. 31.

[127] Nurlanov, Z. Exploring so(3) logarithmic map: Degeneracies and derivatives. Tech. rep., Tech. Report, 2021.

[128] O'Mahony, N., Campbell, S., Carvalho, A., Harapanahalli, S., Hernandez, G. V., Krpalkova, L., Riordan, D., and Walsh, J. Deep Learning vs. Traditional Computer Vision. In *Advances in Computer Vision*, K. Arai and S. Kapoor, Eds., vol. 943. Springer International Publishing, Cham, 2020, pp. 128–144.

[129] Ostermeier, D., Külz, J., and Althoff, M. Automatic Geometric Decomposition for Analytical Inverse Kinematics, Sept. 2024.

[130] Paul, R. P. *Robot Manipulators: Mathematics, Programming, and Control: The Computer Control of Robot Manipulators.* The MIT Press Series in Artificial Intelligence. MIT Press, Cambridge, Mass, 1981.

[131] Peng, X. B., Andrychowicz, M., Zaremba, W., and Abbeel, P. Sim-to-Real Transfer of Robotic Control with Dynamics Randomization. In *2018 IEEE International Conference on Robotics and Automation (ICRA)* (May 2018), pp. 3803–3810.

[132] Peng, X. B., Coumans, E., Zhang, T., Lee, T.-W., Tan, J., and Levine, S. Learning agile robotic locomotion skills by imitating animals. *arXiv preprint arXiv:2004.00784* (2020).

[133] Peters, J., Waubert de Puiseau, C., Tercan, H., Gopikrishnan, A., Lucas de Carvalho, G. A., Bitter, C., and Meisen, T. Emergent language: A survey and taxonomy. *Autonomous Agents and Multi-Agent Systems 39*, 1 (Mar. 2025), 18.

[134] Plappert, M., Andrychowicz, M., Ray, A., McGrew, B., Baker, B., Powell, G., Schneider, J., Tobin, J., Chociej, M., Welinder, P., Kumar, V., and Zaremba, W. Multi-Goal Reinforcement Learning: Challenging Robotics Environments and Request for Research, Mar. 2018.

[135] Possas, R., Barcelos, L., Oliveira, R., Fox, D., and Ramos, F. Online BayesSim for Combined Simulator Parameter Inference and Policy Improvement. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (Oct. 2020), pp. 5445–5452.

[136] Qi, X., Wei, Y., Mei, X., Chellali, R., and Yang, S. Comparative Analysis of the Linear Regions in ReLU and LeakyReLU Networks. In *Neural Information Processing* (Singapore, 2024), B. Luo, L. Cheng, Z.-G. Wu, H. Li, and C. Li, Eds., Springer Nature, pp. 528–539.

[137] Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., and Dormann, N. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of machine learning research 22*, 268 (2021), 1–8.

[138] Raj, R., and Kos, A. An Extensive Study of Convolutional Neural Networks: Applications in Computer Vision for Improved Robotics Perceptions. *Sensors 25*, 4 (Jan. 2025), 1033.

[139] Ramstedt, S., Bouteiller, Y., Beltrame, G., Pal, C., and Binas, J. Reinforcement Learning with Random Delays. *ArXiv abs/2010.02966* (2020), null.

[140] Ramstedt, S., and Pal, C. Real-Time Reinforcement Learning, Dec. 2019.

[141] Ranjan Das, A., and Koskinopoulou, M. Toward sustainable manufacturing: A review on innovations in robotic assembly and disassembly. *IEEE access : practical innovations, open solutions 13* (2025), 100149–100166.

[142] Raychaudhuri, D. S., Paul, S., Vanbaar, J., and Roy-Chowdhury, A. K. Cross-domain imitation from observations. In *International Conference on Machine Learning* (2021), PMLR, pp. 8902–8912.

[143] Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2016), pp. 779–788.

[144] Ren, A. Z., and Majumdar, A. Distributionally Robust Policy Learning via Adversarial Environment Generation. *IEEE Robotics and Automation Letters 7*, 2 (Apr. 2022), 1379–1386.

[145] Rohmer, E., Singh, S. P. N., and Freese, M. CoppeliaSim (formerly V-REP): A Versatile and Scalable Robot Simulation Framework. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems* (Nov. 2013), pp. 1321–1326.

[146] Ronneberger, O., Fischer, P., and Brox, T. U-Net: Convolutional Networks for Biomedical Image Segmentation. In *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, N. Navab, J. Hornegger, W. M. Wells, and A. F. Frangi, Eds., vol. 9351. Springer International Publishing, Cham, 2015, pp. 234–241.

[147] Rosenblatt, F. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological review 65*, 6 (1958), 386.

[148] Ross, S., Gordon, G., and Bagnell, D. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics* (2011), JMLR Workshop and Conference Proceedings, pp. 627–635.

[149] Ruff, L., Vandermeulen, R., Goernitz, N., Deecke, L., Siddiqui, S. A., Binder, A., Müller, E., and Kloft, M. Deep One-Class Classification. In *Proceedings of the 35th International Conference on Machine Learning* (July 2018), PMLR, pp. 4393–4402.

[150] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. Learning representations by back-propagating errors. *Nature 323*, 6088 (Oct. 1986), 533–536.

[151] Sadeghi, F., and Levine, S. CAD2RL: Real Single-Image Flight without a Single Real Image, June 2017.

[152] Salter, S., Hartikainen, K., Goodwin, W., and Posner, I. Priors, hierarchy, and information asymmetry for skill transfer in reinforcement learning. 11th International Conference on Learning Representations, ICLR 2023.

[153] Sandha, S. S., Garcia, L., Balaji, B., Anwar, F., and Srivastava, M. Sim2Real Transfer for Deep Reinforcement Learning with Stochastic State Transition Delays. In *Proceedings of the 2020 Conference on Robot Learning* (Oct. 2021), PMLR, pp. 1066–1083.

[154] Scheiderer, C., Dorndorf, N., and Meisen, T. Effects of Domain Randomization on Simulation-to-Reality Transfer of Reinforcement Learning Policies for Industrial Robots. In *Advances in Artificial Intelligence and Applied Cognitive Computing*, H. R. Arabnia, K. Ferens, D. De La Fuente, E. B. Kozerenko, J. A. Olivas Varela, and F. G. Tinetti, Eds. Springer International Publishing, Cham, 2021, pp. 157–169.

[155] Scheiderer, C., and Meisen, T. Auf eigenen Füßen: Machine Learning in der Industrie. *iX Special 2018 - Industrial Internet of Things* (2018), 48–51.

[156] Scheiderer, C., Mosbach, M., Posada-Moreno, A. F., and Meisen, T. Transfer of Hierarchical Reinforcement Learning Structures for Robotic Manipulation Tasks. In *2020 International Conference on Computational Science and Computational Intelligence (CSCI)* (2020), IEEE, pp. 504–509.

[157] Scheiderer, C., Thun, T., Idzik, C., Posada-Moreno, A. F., Krämer, A., Lohmar, J., Hirt, G., and Meisen, T. Simulation-as-a-service for reinforcement learning applications by example of heavy plate rolling processes. *Procedia Manufacturing 51* (2020), 897–903.

[158] Scheiderer, C., Thun, T., and Meisen, T. Bézier Curve Based Continuous and Smooth Motion Planning for Self-Learning Industrial Robots. *Procedia Manufacturing 38* (Jan. 2019), 423–430.

[159] Schuitema, E., Buşoniu, L., Babuška, R., and Jonker, P. Control delay in Reinforcement Learning for real-time dynamic systems: A memoryless approach. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems* (Oct. 2010), pp. 3226–3231.

[160] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms, 2017.

[161] Selvaraju, R. R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., and Batra, D. Grad-CAM: Visual Explanations From Deep Networks via Gradient-Based Localization. In *Proceedings of the IEEE International Conference on Computer Vision* (2017), pp. 618–626.

[162] Sharma, A., Ahn, M., Levine, S., Kumar, V., Hausman, K., and Gu, S. Emergent Real-World Robotic Skills via Unsupervised Off-Policy Reinforcement Learning, Apr. 2020.

[163] Sharma, A., Gu, S., Levine, S., Kumar, V., and Hausman, K. Dynamics-Aware Unsupervised Discovery of Skills, Feb. 2020.

[164] Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., and Hassabis, D. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science 362*, 6419 (Dec. 2018), 1140–1144.

[165] Simba, K. R., Uchiyama, N., and Sano, S. Real-time smooth trajectory generation for nonholonomic mobile robots using Bézier curves. *Robotics and Computer-Integrated Manufacturing 41* (2016), 31–42.

[166] Simonič, M., Ude, A., and Nemec, B. Hierarchical learning of robotic contact policies. *Robotics and Computer-Integrated Manufacturing 86* (Apr. 2024), 102657.

[167] Simonic, M., Zlajpah, L., Ude, A., and Nemec, B. Autonomous Learning of Assembly Tasks from the Corresponding Disassembly Tasks. In *2019 IEEE-RAS 19th International Conference on Humanoid Robots (Humanoids)* (Toronto, ON, Canada, Oct. 2019), IEEE, pp. 230–236.

[168] Singh, A., Yang, L., Hartikainen, K., Finn, C., and Levine, S. End-to-End Robotic Reinforcement Learning without Reward Engineering, May 2019.

[169] Soni, V., and Singh, S. Using homomorphisms to transfer options across continuous reinforcement learning domains. In *AAAI* (2006), vol. 6, pp. 494–499.

[170] Springenberg, J., Dosovitskiy, A., Brox, T., and Riedmiller, M. Striving for Simplicity: The All Convolutional Net. In *ICLR (workshop track)* (2015).

[171] Srinivas, A., Jabri, A., Abbeel, P., Levine, S., and Finn, C. Universal planning networks - Long version + supplementary. vol. 11 of *35th International Conference on Machine Learning, ICML 2018*, pp. 7536–7548.

[172] Stockman, G., and Shapiro, L. G. *Computer Vision*, 1st ed. Prentice Hall PTR, USA, 2001.

[173] Stooke, A., and Abbeel, P. Rlpyt: A Research Code Base for Deep Reinforcement Learning in PyTorch, Sept. 2019.

[174] Sutton, R. S., and Barto, A. *Reinforcement Learning: An Introduction*, second edition ed. Adaptive Computation and Machine Learning. The MIT Press, Cambridge, Massachusetts and London, England, 2018.

[175] Sutton, R. S., Precup, D., and Singh, S. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence 112*, 1 (Aug. 1999), 181–211.

[176] Taylor, M. E., Kuhlmann, G., and Stone, P. Autonomous transfer for reinforcement learning. In *AAMAS (1)* (2008), pp. 283–290.

[177] Taylor, M. E., and Stone, P. Cross-domain transfer for reinforcement learning. vol. 227 of *ACM International Conference Proceeding Series*, pp. 879–886.

[178] Taylor, M. E., and Stone, P. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research 10*, 7 (2009).

[179] Tekin, B., Sinha, S. N., and Fua, P. Real-Time Seamless Single Shot 6D Object Pose Prediction. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition* (Salt Lake City, UT, USA, June 2018), IEEE, pp. 292–301.

[180] Tercan, H., Bitter, C., Bodnar, T., Meisen, P., and Meisen, T. Evaluating a Session-based Recommender System using Prod2vec in a Commercial Application. In *Proceedings of the 23rd International Conference on Enterprise Information Systems* (2021), SCITEPRESS - Science and Technology Publications.

[181] Tian, Y., Willis, K. D. D., Al Omari, B., Luo, J., Ma, P., Li, Y., Javid, F., Gu, E., Jacob, J., Sueda, S., Li, H., Chitta, S., and Matusik, W. ASAP: Automated Sequence Planning for Complex Robotic Assembly with Physical Feasibility. In *2024 IEEE International Conference on Robotics and Automation (ICRA)* (May 2024), pp. 4380–4386.

[182] Tian, Y., Xu, J., Li, Y., Luo, J., Sueda, S., Li, H., Willis, K. D. D., and Matusik, W. Assemble Them All: Physics-Based Planning for Generalizable Assembly by Disassembly. *ACM Transactions on Graphics 41*, 6 (Dec. 2022), 1–11.

[183] Tiboni, G., Arndt, K., and Kyrki, V. DROPO: Sim-to-real transfer with offline domain randomization. *ROBOTICS AND AUTONOMOUS SYSTEMS 166* (Aug. 2023).

[184] Tittel, S. Analytical solution for the inverse kinematics problem of the franka emika panda seven-dof light-weight robot arm. In *2021 20th International Conference on Advanced Robotics (ICAR)* (2021), IEEE, pp. 1042–1047.

[185] Tobin, J., Fong, R., Ray, A., Schneider, J., Zaremba, W., and Abbeel, P. 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)* (Sept. 2017), pp. 23–30.

[186] Todorov, E., Erez, T., and Tassa, Y. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems* (2012), IEEE, pp. 5026–5033.

[187] Torrey, L., Walker, T., Shavlik, J., and Maclin, R. Using Advice to Transfer Knowledge Acquired in One Reinforcement Learning Task to Another. In *Machine Learning: ECML 2005* (Berlin, Heidelberg, 2005), J. Gama, R. Camacho, P. B. Brazdil, A. M. Jorge, and L. Torgo, Eds., Springer, pp. 412–424.

[188] Towers, M., Kwiatkowski, A., Terry, J., Balis, J. U., Cola, G. D., Deleu, T., Goulão, M., Kallinteris, A., Krimmel, M., KG, A., Perez-Vicente, R., Pierré, A., Schulhoff, S., Tai, J. J., Tan, H., and Younis, O. G. Gymnasium: A Standard Interface for Reinforcement Learning Environments, Nov. 2024.

[189] Tsai, Y.-Y., Xu, H., Ding, Z., Zhang, C., Johns, E., and Huang, B. DROID: Minimizing the Reality Gap Using Single-Shot Human Demonstration. *IEEE Robotics and Automation Letters 6*, 2 (Apr. 2021), 3168–3175.

[190] VandenBos, G. R. APA Dictionary of Psychology. *American Psycho-logical Association* (2007).

[191] Vezhnevets, A. S., Osindero, S., Schaul, T., Heess, N., Jaderberg, M., Silver, D., and Kavukcuoglu, K. FeUdal Networks for Hierarchical Reinforcement Learning. In *Proceedings of the 34th International Conference on Machine Learning* (July 2017), PMLR, pp. 3540–3549.

[192] Vietz, H., Maschler, B., Tercan, H., Bitter, C., Meisen, T., and Weyrich, M. Industrielles Transfer-Lernen: Von der Wissenschaft in die Praxis. *atp magazin 64*, 8 (2022), 86–93.

[193] Villalobos, J., Sanchez, I. Y., and Martell, F. Alternative Inverse Kinematic Solution of the UR5 Robotic Arm. In *Advances in Automation and Robotics Research* (Cham, 2022), H. A. Moreno, I. G. Carrera, R. A. Ramírez-Mendoza, J. Baca, and I. A. Banfield, Eds., Springer International Publishing, pp. 200–207.

[194] Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D. H., Powell, R., Ewalds, T., Georgiev, P., Oh, J., Horgan, D., Kroiss, M., Danihelka, I., Huang, A., Sifre, L., Cai, T., Agapiou, J. P., Jaderberg, M., Vezhnevets, A. S., Leblond, R., Pohlen, T., Dalibard, V., Budden, D., Sulsky, Y., Molloy, J., Paine, T. L., Gulcehre, C., Wang, Z., Pfaff, T., Wu, Y., Ring, R., Yogatama, D., Wünsch, D., McKinney, K., Smith, O., Schaul, T., Lillicrap, T., Kavukcuoglu, K., Hassabis, D., Apps, C., and Silver, D. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature 575*, 7782 (Nov. 2019), 350–354.

[195] Viviers, C. G. A., Filatova, L., Termeer, M., de With, P. H. N., and van der Sommen, F. Advancing 6-DoF Instrument Pose Estimation in Variable X-Ray Imaging Geometries. *IEEE Transactions on Image Processing 33* (2024), 2462–2476.

[196] von Eschenbach, M. E., Manela, B., Peters, J., and Biess, A. Metric-Based Imitation Learning Between Two Dissimilar Anthropomorphic Robotic Arms, 2020.

[197] Voulodimos, A., Doulamis, N., Doulamis, A., and Protopapadakis, E. Deep Learning for Computer Vision: A Brief Review. *Computational Intelligence and Neuroscience 2018* (2018), 1–13.

[198] WALSH, T. J., NOURI, A., LI, L., AND LITTMAN, M. L. Learning and planning in environments with delayed feedback. *Autonomous Agents and Multi-Agent Systems 18*, 1 (Feb. 2009), 83–105.

[199] WANG, G., NIU, H., ZHU, D., HU, J., ZHAN, X., AND ZHOU, G. A Versatile and Efficient Reinforcement Learning Framework for Autonomous Driving, Mar. 2022.

[200] WANG, Q., XIONG, J., HAN, L., SUN, P., LIU, H., AND ZHANG, T. Exponentially Weighted Imitation Learning for Batched Historical Data. In *Advances in Neural Information Processing Systems* (2018), vol. 31, Curran Associates, Inc.

[201] WANG, Z., XING, D., YANG, Y., AND WANG, P. Delayed dynamic model scheduled reinforcement learning with time-varying delays for robotic control. *IEEE Robotics and Automation Letters 10*, 3 (2025), 2646–2653.

[202] WATANABE, K., AND INADA, S. Search algorithm of the assembly sequence of products by using past learning results. *International Journal of Production Economics 226* (2020).

[203] WEISS, K., KHOSHGOFTAAR, T. M., AND WANG, D. A survey of transfer learning. *Journal of Big Data 3*, 1 (May 2016), 9.

[204] WENG, J., LIN, M., HUANG, S., LIU, B., MAKOVIICHUK, D., MAKOVIYCHUK, V., LIU, Z., SONG, Y., LUO, T., AND JIANG, Y. Envpool: A highly parallel reinforcement learning environment execution engine. *Advances in Neural Information Processing Systems 35* (2022), 22409–22421.

[205] XIAO, T., JANG, E., KALASHNIKOV, D., LEVINE, S., IBARZ, J., HAUSMAN, K., AND HERZOG, A. Thinking While Moving: Deep Reinforcement Learning with Concurrent Control. In *International Conference on Learning Representations* (Apr. 2020).

[206] YANG, M., CAO, H., ZHAO, L., ZHANG, C., AND CHEN, Y. Robotic Sim-to-Real Transfer for Long-Horizon Pick-and-Place Tasks in the Robotic Sim2Real Competition, Mar. 2025.

[207] YANG, Q., STORK, J. A., AND STOYANOV, T. Learn from robot: Transferring skills for diverse manipulation via cycle generative networks. In *2023 IEEE 19th International Conference on Automation Science and Engineering (CASE)* (2023), pp. 1–6.

[208] YANG, Y., CALUWAERTS, K., ISCEN, A., ZHANG, T., TAN, J., AND SINDHWANI, V. Data efficient reinforcement learning for legged robots. In *Conference on Robot Learning* (2020), PMLR, pp. 1–10.

[209] ZAKKA, K., ZENG, A., LEE, J., AND SONG, S. Form2Fit: Learning Shape Priors for Generalizable Assembly from Disassembly. In *2020 IEEE International Conference on Robotics and Automation (ICRA)* (May 2020), pp. 9404–9410.

[210] Zhang, Q., and Li, M. The knowledge transfer methods and its application in robot based on subtask hierarchical reinforcement learning. *Advances in Information Sciences and Service Sciences 4*, 20 (2012), 119–128.

[211] Zhang, X., and Saniie, J. Unsupervised Learning for 3D Ultrasonic Data Compression. In *2021 IEEE International Ultrasonics Symposium (IUS)* (Sept. 2021), pp. 1–3.

[212] Zhang, Z. A flexible new technique for camera calibration. *IEEE Transactions on pattern analysis and machine intelligence 22*, 11 (2002), 1330–1334.

[213] Zhao, X., Wang, L., Zhang, Y., Han, X., Deveci, M., and Parmar, M. A review of convolutional neural networks in computer vision. *Artificial Intelligence Review 57*, 4 (Mar. 2024), 99.

[214] Zhu, R., Dai, T., and Celiktutan, O. Cross domain policy transfer with effect cycle-consistency. In *2024 IEEE International Conference on Robotics and Automation (ICRA)* (2024), pp. 9471–9477.

[215] Zolna, K., Rostamzadeh, N., Bengio, Y., Ahn, S., and Pinheiro, P. O. Reinforced Imitation in Heterogeneous Action Space, Aug. 2019.

[216] Zou, Z., Chen, K., Shi, Z., Guo, Y., and Ye, J. Object Detection in 20 Years: A Survey. *Proceedings of the IEEE 111*, 3 (Mar. 2023), 257–276.