Reinforcement Learning for Real-world Production Systems

Reinforcement Learning in realen Produktionssystemen

Von der Fakultät für Maschinenwesen der
Rheinisch-Westfälischen Technischen Hochschule Aachen
zur Erlangung des akademischen Grades
einer Doktorin der Ingenieurwissenschaften
genehmigte Dissertation

vorgelegt von
Emma Anna Elisabeth Cramer, geb. Pabich

Berichter: Univ.-Prof. Dr. sc. Sebastian Trimpe
Univ.-Prof. Dr.-Ing. Jakob Lukas Andert

Tag der mündlichen Prüfung: 24.09.2025

Diese Dissertation ist auf den Internetseiten der Universitätsbibliothek online verfügbar.

Acknowledgments

Keep on keeping at what you love.

Alicia Keys, Underdog

This dissertation was developed during my time as a research associate at the Institute for Data Science in Mechanical Engineering (DSME) at RWTH Aachen University.

First and foremost, I would like to express my heartfelt gratitude to my doctoral supervisor, Sebastian Trimpe, for his guidance and support throughout my academic journey. I am especially grateful for the exceptional research environment he fostered—one filled with diverse personalities, transparency, productive discussions, and collaboration. Under his mentorship, I have gained invaluable insights into both research and leadership. He taught me how to pursue excellence without compromising ideals, how to push myself to be better while supporting others, and to pay attention to details while recognizing that we all have limits. My time at DSME has been incredibly valuable, and I will always be thankful for having had Sebastian as my doctoral thesis supervisor.

Further, I thank Univ.-Prof. Dr.-Ing. Jakob Andert for willingly taking on the role of secondary reviewer and the head of the examination committee for presiding over my examination.

I would also like to thank my colleagues at DSME for creating a supportive and friendly research environment. Without you, I would not have enjoyed the last couple of years at DSME so much. While I cannot list everyone, each of you has contributed to my growth in unique ways. Paul, thank you for our limitless morning coffees and chats. Bernd, you were a great office mate and discussion partner for reinforcement learning. Thank you for being such a reliable TA partner for RLLBC. Henrik and Alexander, I am grateful for your willingness to answer my questions about robotics and control and for your rare gift of explaining complex concepts without making me feel inadequate. Friedrich, thank you for keeping an eye on my PhD progress and subtly checking in on me at just the right moments. Also, thank you to all the senior PhD students at DSME, Alex, Dominik, Mona, and many more, who taught me how this weird research world works. I am also grateful to Friedrich, Andres, Bernd, Lukas, and David for taking the time to read this dissertation, providing insightful feedback, and reminding me that it will indeed be finished one day.

Over the years, I had the pleasure of supervising fourteen student thesis projects and twelve student assistants. Working with them has been a truly rewarding experience, teaching me invaluable lessons about programming, ROS, robotics, communication, and

leadership. More than anything, it has been a privilege to collaborate with such motivated, intelligent, and kind individuals. While I cannot name everyone, I would like to extend a special thank you to Robin for supporting me through the transition from IfU to DSME and being my sparring partner in developing the spatial autoencoder ideas. Hannah and Jana, your structured and well-organized approach to work has been inspiring. Johannes and Lukas, thank you for believing in my vision during the early stages of the polishing robot project. Jonas, Ramil, and Lukas, I truly appreciate your dedication and perseverance throughout the often challenging process of writing papers.

I am forever grateful to my family for their support throughout my entire life. You helped so much in the big challenge of finishing two dissertations while raising a small human being.

Lastly, I want to thank my husband, Simon, for his continuous support, belief in me, and love. My life and this thesis are dedicated to you and our children, Arthur, Jasper, and whoever will join us in the future.

Emma Cramer
Aachen, Feb 12, 2025

Abstract

This dissertation envisions a future of robotic production capable of solving complex tasks in changing and uncertain environments. Such capabilities are critical as production trends shift toward smaller lot sizes, individualised products, and human-robot collaboration. Reinforcement learning (RL) methods hold promises for solving such challenges as they learn behaviours through interaction with their surrounding environments and ideally generalise to new unseen scenarios. However, realising this promise in real-world settings has been challenging. Key barriers include data collection efficiency and safety, the complexity and unpredictability of real-world dynamics, and usability challenges for domain experts with limited machine learning experience. The central challenge addressed in this dissertation lies in bridging the gap between existing RL methods and the challenges posed by real-world production. It proposes novel methods to improve data collection regarding safety and efficiency, consider changing and uncertain environments, and improve user interpretability and acceptance.

As a first contribution, we introduce a novel metric for evaluating the performance of spatial autoencoders, a neural network extracting spatial features from images. By quantifying how well such autoencoders track objects in a 3D space, we address key challenges in using image-based state representations in RL, particularly in unstructured environments. The proposed metric is applied to assess various spatial autoencoder architectures and their ability to track object positions. We show that this ability correlates with downstream RL performance. As a result, the developed evaluation metric offers a lightweight, computationally efficient method for predicting RL performance, providing a practical tool for selecting trained models in robotic RL tasks.

Next, we develop the Contextualized Hybrid Ensemble Q-learning (CHEQ) algorithm, a novel adaptive hybrid RL method. The algorithm blends the action from an RL agent with a classical model-based controller and dynamically adjusts the weighting between both based on the RL agent's current capabilities. CHEQ is evaluated on a simulated car racing task that requires precise control near stability limits. Compared to state-of-the-art adaptive hybrid RL methods, CHEQ demonstrates (i) enhanced data efficiency, (ii) safer exploration, and (iii) improved zero-shot transferability to unseen scenarios.

Lastly, this thesis explores the application of RL to robotic polishing, a contact-rich and dynamic task that exemplifies the complexities of manipulation and other contact-rich problems in industrial automation. In this context, the CHEQ algorithm is applied to learn variable impedance gains and path-following for polishing a 3D workpiece. In simulation, we demonstrate the potential of CHEQ to learn suitable gains while maintaining safe exploration behaviour. Building on this, we successfully deploy CHEQ on hardware. Direct hardware training introduces environment uncertainties and nonstationarities, as well as practical requirements for sample efficiency and safety. CHEQ effectively addresses these challenges, confirming its suitability for real-world industrial automation tasks.

Zusammenfassung

Der Trend zu kleineren Losgrößen, individualisierten Produkten und der Zusammenarbeit zwischen Menschen und Robotern stellt neue Herausforderungen für die roboterbasierte Produktion dar. Um diesen Anforderungen gerecht zu werden, müssen Roboter zukünftig in der Lage sein, komplexe Aufgaben in sich verändernden und unsicheren Umgebungen zu bewältigen. Reinforcement Learning (RL) stellt einen vielversprechenden Ansatz dar, indem es Roboter befähigt, Regelungsstrategien durch direkte Interaktion mit der Umgebung zu erlernen. Die reale Welt stellt jedoch bislang erhebliche Hürden für die Anwendung von RL-Algorithmen dar. Zu den wesentlichen Hindernissen zählen die Effizienz und Sicherheit der Datenerfassung, die Komplexität und Unvorhersehbarkeit realer Umgebungen sowie die Nutzbarkeit und Akzeptanz für Anwender im Produktionskontext. Die zentrale Zielsetzung dieser Dissertation besteht darin, die Lücke zwischen bestehenden RL-Methoden und den praktischen Anforderungen der industriellen Produktion zu verringern.

Als ersten Beitrag wird eine neue Metrik zur Bewertung von spatial Autoencodern (SAE) vorgestellt, einer speziellen Art von neuronalen Netzen, um Objektpositionen aus Bildern zu extrahieren. Die entwickelte Metrik ermöglicht es, zu quantifizieren, wie gut ein trainierter Autoencoder in der Lage ist, Objekte auf den Bildern über mehrere Zeitschritte zu verfolgen. Die vorgeschlagene Metrik wird genutzt, um verschiedene SAE-Architekturen zu bewerten. Darüber hinaus ermöglicht die Metrik eine effiziente Vorhersage der nachgeschalteten RL-Performance, wodurch sie sich als praktisches Werkzeug zur Auswahl geeigneter Autoencoder-Modelle für robotische RL-Aufgaben etabliert.

Im nächsten Schritt wird der Algorithmus Contextualized Hybrid Ensemble Q-learning (CHEQ) entwickelt, ein neuartiger adaptiver hybrider RL-Ansatz. Der Algorithmus kombiniert die Aktionen eines RL-Agenten mit der einer klassischen Regelung. Die Gewichtung zwischen beiden Teilen wird basierend auf den aktuellen Fähigkeiten des RL-Agenten angepasst. Im Vergleich zu aktuellen adaptiven hybriden RL-Methoden zeigt CHEQ (i) eine höhere Dateneffizienz, (ii) ein sichereres Explorationsverhalten und (iii) eine verbesserte Übertragbarkeit auf unbekannte Szenarien.

Abschließend wird die Praktikabilität des CHEQ-Algorithmus am Anwendungsfall des roboterbasierten Polierens untersucht, einer kontaktreichen und dynamischen Manipulationsaufgabe, die exemplarisch für die Herausforderungen in der industriellen Automatisierung steht. In der Simulation ist CHEQ in der Lage, gutes Polierverhalten zu erlernen und gleichzeitig sicheres Explorationsverhalten zu gewährleisten. Aufbauend darauf wird CHEQ erfolgreich auf realer Hardware eingesetzt. Das Training direkt auf Hardware berücksichtigt reale Unsicherheiten und Instationaritäten und stellt hohe Anforderungen an Effizienz und Sicherheit. CHEQ meistert diese Herausforderungen und bestätigt damit seine Eignung für industrielle Automatisierungsaufgaben.

Contents

1. Introduction **1**

1.1. Reinforcement Learning . . . 3
1.2. Gaps . . . 4
1.2.1. Real-world Data Collection . . . 6
1.2.2. Real-world Environments . . . 6
1.2.3. Real-world Users . . . 7
1.3. Main Contributions . . . 8
1.3.1. Outline . . . 8
1.3.2. Evaluation of Spatial Feature Extractors . . . 9
1.3.3. Adaptive Hybrid Reinforcement Learning . . . 10
1.3.4. Robotic Polishing . . . 12
1.3.5. Overview of Publications . . . 13
1.3.6. Additional Publications by the Author . . . 15
1.3.7. Student Thesis Supervisions . . . 15

2. Preliminaries **17**

2.1. Deep Learning . . . 17
2.1.1. Artificial Neural Networks . . . 18
2.1.2. Training a Neural Network . . . 19
2.2. Reinforcement Learning . . . 20
2.2.1. Reinforcement Learning Fundamentals . . . 20
2.2.2. Deep Reinforcement Learning . . . 23
2.2.3. Soft Actor-Critic . . . 25
2.3. Latent Feature Extraction with Autoencoders . . . 26
2.3.1. Autoencoders . . . 27
2.3.2. Spatial Autoencoders . . . 28
2.4. Serial Manipulators . . . 30
2.4.1. Force-Motion Control . . . 33

3. A Metric for Keypoint Detection in Reinforcement Learning **37**
3.1. Introduction . . . 38
3.2. Related Work . . . 39
3.3. Reinforcement Learning with Spatial Autoencoders . . . 40
3.3.1. Spatial Autoencoder Pre-Training . . . 41
3.3.2. Reinforcement Learning Training with Latent State Representations 42
3.3.3. Assumptions . . . 42
3.4. Problem Formalization . . . 43
3.5. A Metric to Evaluate Keypoints . . . 43
3.6. Evaluation . . . 47
3.6.1. Spatial Autoencoder Evaluation . . . 48
3.6.2. Reinforcement Learning Evaluation . . . 52
3.7. Conclusion . . . 54

4. Contextualized Hybrid Ensemble Q-learning **55**
4.1. Introduction . . . 56
4.2. Related Work . . . 57
4.3. Background . . . 58
4.4. A Unified View on Hybrid Reinforcement Learning . . . 59
4.4.1. Mixing Function . . . 60
4.4.2. Weight Adaption Function . . . 60
4.5. Contextualized Hybrid Reinforcement Learning . . . 61
4.5.1. General Concept of Contextualized Hybrid Reinforcement Learning 61
4.5.2. Illustrative Example . . . 63
4.5.3. Contextualized Hybrid Ensemble Q-learning (CHEQ) . . . 64
4.6. Racing Task . . . 67
4.7. Experiments . . . 68
4.7.1. Experimental Setup . . . 69
4.7.2. Evaluation on the Car Racing Environment . . . 69
4.8. Conclusion . . . 72

5. Safe Variable Impedance Learning for Robotic Polishing **73**
5.1. Introduction . . . 74
5.2. Related Work . . . 76
5.3. Robotic Polishing . . . 77

5.4. Learning Variable Impedance via CHEQ . . . 78
5.4.1. Impedance Controller (IC) . . . 78
5.4.2. Hybrid Reinforcement Learning-based Reference Controller (RC) . 79
5.5. Simulation Study . . . 82
5.5.1. Evaluating the Need for Variable Impedance . . . 83
5.5.2. Learning Variable Impedance Control with Reinforcement Learning 84
5.5.3. Zero Shot Transfer . . . 86
5.6. Case Study: Real-world 3D polishing . . . 89
5.6.1. Real-world Reinforcement Learning Challenges and Solutions . . . 91
5.7. Conclusion . . . 92

6. Conclusion **93**
6.1. Summary . . . 93
6.2. Future Work and Discussion . . . 94

Bibliography **97**

A. A Metric for Keypoint Detection in Reinforcement Learning **111**
A.1. Implementation Details . . . 111
A.1.1. SAE Setup and Hyperparameters . . . 111
A.1.2. RL Setup and Hyperparameters . . . 113
A.2. Additional Results and Ablation Study . . . 115

B. Contextualized Hybrid Ensemble Q-learning **119**
B.1. Implementation Details . . . 119
B.1.1. RL Setup and Hyperparameters . . . 119
B.1.2. Nominal Control Setup and Hyperparameters . . . 120
B.2. Additional Results and Ablation Study . . . 120

C. Safe Variable Impedance Learning for Robotic Polishing **125**
C.1. Polishing Environment . . . 125
C.1.1. Derivation of the Task Goal . . . 125
C.1.2. Safety Limits . . . 126
C.2. Implementation Details . . . 128
C.2.1. RL Setup and Hyperparameters . . . 128
C.2.2. Nominal Control Setup and Hyperparameters . . . 132

List of Acronyms **135**

List of Figures **137**

List of Tables **141**

1. Introduction

Imagine a robot precisely assembling a high-end mechanical watch, polishing the intricate surfaces of a musical instrument, or performing complex assembly tasks for highly customised products. These scenarios illustrate the vision of future robotic production — one that surpasses the limitations of traditional manufacturing paradigms. Achieving this vision requires production systems capable of handling tasks that are inherently difficult or even impossible to model using conventional methods [1]. One prime example is achieving near-human dexterity in robotic manipulation tasks, which is essential for performing fine-grained physical interactions and compliant behaviour. Such tasks involve contact-rich operations with underlying dynamics that are inherently complex and difficult to model. In addition to dexterous manipulation, future production systems must excel in dynamic and uncertain environments where adaptability is essential. This need arises from several evolving manufacturing trends, including smaller lot sizes, increasingly individualised products, and the growing demand for flexible assembly lines [2, 3, 4]. Furthermore, the shift toward collaborative production environments [5, 6], where humans and robots work side by side, amplifies these challenges. Summarising these demands, future robots must adapt to varying tasks, navigate dynamic production environments, and handle complex tasks.

Achieving this vision requires highly adaptive and flexible automation solutions [5]. However, the current state of robotic production falls short of this ambition. Most robotic tasks in today's industrial production are limited to scenarios where dynamics can be easily modelled or where simple control strategies suffice. Tasks that demand high dexterity or involve complex contact dynamics are still predominantly carried out by human workers. Robotics in production is also largely confined to highly controlled environments, where robots perform repetitive tasks with controllers designed for predefined and fixed operating conditions. While these controllers ensure consistent quality, they require meticulous tuning by process experts, typically a manual and resource-intensive process [7, 8]. This approach works well in traditional mass production settings, where repetitive tasks like pick-and-place operations dominate. However, as production environments become more complex and less structured, this reliance on static and highly

controlled systems creates a significant gap between the status quo and the envisioned future of robotic production. Bridging this gap requires developing new perception and control techniques that scale to unstructured and dynamic production settings.

One promising approach to addressing this challenge lies in the development of self-learning machine learning (ML) agents capable of performing complex tasks in dynamic and uncertain environments [9]. Reinforcement learning, a powerful technique within ML, offers significant potential in this regard. It enables agents to learn control policies from data through interaction with the environment, allowing them to master complex tasks without requiring explicit first-principle models. The agent explores different actions and receives feedback based on its performance. Through this process of exploration and optimisation, the agent iteratively refines its strategy to achieve a given task. This learning paradigm can be viewed as a form of trial and error, as the agent continually adjusts its actions to improve its performance. Reinforcement learning has demonstrated its ability to adapt to changing environments, learn complex nonlinear control policies, and achieve high performance across a range of tasks such as games [10, 11], healthcare [12, 13, 14], legged locomotion [15, 16] and robotics [17, 18, 19]. Despite these advances, the application of reinforcement learning (RL) in real-world production settings remains limited [20]. Training RL agents in such environments poses many unique hurdles that must be addressed to advance the field of RL towards production settings. In this thesis, we focus on three key challenges.

The first challenge lies in data collection. Collecting data in real-world environments has to be efficient to minimise the need for human supervision and to reduce strain on hardware. Moreover, data collection processes must be safe for both humans working alongside robots and for the hardware.

The second critical challenge stems from the inherent unpredictability of real-world production settings. Not all system dynamics can be accurately modelled due to factors such as sensor noise and uncertainty in the environment. In addition, non-stationary conditions, such as wear and tear on robots, variations in workpieces, or changes in production setups like lighting and temperature, can further complicate performance and reliability. These challenges are even more pronounced in future production scenarios involving individualised products, frequent reconfigurations of assembly lines, and highly flexible manufacturing processes.

As a third challenge, the introduction of ML solutions into production contexts requires careful consideration of the users interacting with these systems. ML algorithms rarely work out of the box; software engineers and production experts must collaborate to develop, tune, and monitor them effectively. To support this co-development

process, ML methods should be designed for interpretability, allowing their behaviour and outputs to be understood using domain-specific knowledge or metrics commonly used by process experts. Furthermore, gaining acceptance among stakeholders, including company leaders and production engineers, is essential for successful deployment. By prioritising usability and transparency, ML solutions can foster trust and adoption, ensuring their seamless integration into production environments.

This thesis explores the unique challenges of applying RL in production environments and proposes strategies and solutions that contribute toward addressing them. In this chapter, we begin with a brief introduction to RL and its key notations, followed by a detailed discussion of the associated challenges. Finally, we provide an overview of the thesis's contributions and outline its structure.

1.1. Reinforcement Learning

Reinforcement learning is learning through trial and error. Imagine a child learning to grasp an object. At first, they may try different motions, missing or holding the object awkwardly. But each time they succeed, they get positive feedback, extrinsic feedback from another person or intrinsic feedback from themselves, encouraging them to repeat the successful action. Over time, they learn the best way to grasp. Reinforcement learning works similarly; an agent interacts with an environment, takes actions, and receives rewards or feedback. Through trial and error, it learns which actions lead to the best outcomes, gradually improving its performance by maximising rewards.

More formally, this can be described by a discounted Markov decision process (MDP) defined by the tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, p, r, \rho_0, \gamma)$, with state space $\mathcal{S}$, action space $\mathcal{A}$, and start state distribution ρ_0. The commonly unknown transition function $p(s_{t+1}, r_{t+1} \mid s_t, a_t)$ describes transitions between states $s_t \in \mathcal{S}$ and actions $a_t \in \mathcal{A}$. During transitions, the agent receives a scalar reward $r_{t+1} = r(s_t, a_t) \in \mathbb{R}$. A trajectory starting from $t = 0$ until $t = T$ is denoted by $\tau = (s_0, a_0, r_1, \dots s_T)$. The objective of the RL agent is to learn a policy $\pi(a_t \mid s_t)$ that maximises the expected cumulative sum of rewards discounted by $\gamma \in (0, 1)$. This results in the RL objective

$$J(\pi) = \max_{\pi} \mathbb{E}_{\tau \sim \pi, \mathcal{M}} \left[\sum_{t=0}^{\infty} \gamma^t r_{t+1} \right]. \tag{1.1}$$

With $\gamma < 1$, the infinite sum in Equation (1.1) has a finite value. This formulation also includes episodic tasks with a finite horizon T.

An important paradigm in RL is the notion of exploration and exploitation [21]. The RL agent exploits its current knowledge about the world by acting according to its current best policy. At the same time, exploring unknown parts of the state action space is important to find a potentially better policy. There are different ways to ensure exploration of the RL agent; one common way in modern RL algorithms is adding an entropy maximisation term into the standard RL objective [22]. The agent then acts within the environment and continuously improves its policy. We call N the total number of samples until the RL agent has learned a sufficiently good policy. More details on the fundamental concepts of RL are given in Section 2.2.

Having established the fundamental RL notation, we now analyse and formalise the gaps between current RL research applications and their real-world production counterparts.

1.2. Gaps

As introduced, robotic production settings present three key challenges: data collection, environment unpredictability, and usability. These challenges are not well addressed in current deep RL research, and consequently, only limited RL applications exist in real-world production settings. While we will formulate the challenges in detail later, it is useful to first examine the domains where RL has been successfully applied in the past to better understand the gap between current RL methods and the requirements of RL for robotics in real-world manufacturing.

Deep RL methods have achieved notable success in tackling challenging control problems, such as board games [10, 23], video games [24, 25, 11], and healthcare [13]. Focusing on robotics, recent advances have enabled deep RL to succeed at some real-world robotic tasks such as grasping [26], locomotion [16] and drone racing [27]. These achievements highlight the promise of deep RL in managing complex robotic systems that traditional decision-making, planning, and control methods struggle to handle effectively. However, the maturity of state-of-the-art deep RL solutions varies significantly across different robotic applications, with applications in industrial manipulation remaining severely limited [28]. In fact, most significant breakthroughs in deep RL to date have primarily been realised within simulation or gaming environments. In typical simulations, the RL agent interacts with stationary environments, which can be reset to the same initial state, and the dynamics remain constant over time. While RL methods generally do not assume prior knowledge about the environment, using simulations to generate data necessitates a sufficiently accurate dynamics model. Furthermore, simulations usually offer

full access to the environment's state, with commonly used models being entirely deterministic or incorporating only simple noise settings. Interactions in these environments are inherently safe, and engaging with the environment is both inexpensive and fast. As a result, many RL successes rely on the availability of extensive, safe, and repeatable environment interactions. Given that simulated interactions are nearly unlimited, current RL research typically includes minimal prior knowledge of the environment within the policy, instead favouring learning from scratch through random exploration. When looking at this through the lens of an MDP, current RL research assumes little knowledge about the underlying dynamics distribution $p(s_{t+1}, r_{t+1} \mid s_t, a_t)$, unlimited or large amounts of samples N, the possibility of random exploration with the policy π and a stationary MDP $\mathcal{M}$.

In contrast, real-world environments, particularly production environments, differ significantly from the assumptions commonly made in RL research about the underlying MDP. Learning-based methods excel in modelling complex systems, where exact simulations are often unavailable. For example, accurately simulating dynamics involving contact and collisions is notoriously challenging. Even when precise dynamics simulations are feasible, certain physical parameters, such as friction coefficients, actuation delays, and measurement noise, are difficult to measure in real-world settings and replicate in simulation. This issue is often referred to as the sim-to-real gap [29]. Research on sim-to-real policy learning [30, 31] has shown that significant transfer from simulation to the real world is possible, even for manipulation policies. However, these approaches still depend on high-quality simulators, which require substantial effort to develop. As robotic applications become more diverse and flexible, creating a realistic simulator for every new task becomes increasingly impractical. Consequently, this thesis argues that to transition RL applications from controlled research settings to real-world production environments, we cannot rely solely on simulation environments. Instead, it is essential to address the unique challenges posed by real-world environments. Recognising this, we categorise the fundamental limitations currently impeding the application of RL methods in production settings into three main areas: (i) constraints in real-world data collection, (ii) complexity of real-world environments, and (iii) requirements of real-world users. The following sections explore these limitations in detail, outlining the research gaps this thesis aims to address.

1.2.1. Real-world Data Collection

Data can be collected quickly and without risk in simulated environments. However, this advantage diminishes in real-world scenarios, leading to several key challenges.

G1 **Sample Efficiency**: Simulated environments can efficiently generate vast amounts of data, often running up to thousands of times faster than real-time [32]. This is particularly true for simulations powered by advanced GPU-based frameworks, such as NVIDIA Isaac [33], which enable parallel execution of multiple environments simultaneously. This parallelism dramatically reduces the time required for training, making simulations highly efficient for RL. In contrast, real-world systems face significant bottlenecks in data collection. Resetting the environment in the real world can involve extended, unpredictable downtime, mainly when a robot must perform complex reset movements. Further, real-world interactions often require human supervision to ensure safety, significantly increasing resource demands and limiting parallel training. Consequently, while simulations can quickly generate millions of samples, real-world data collection is much more time-consuming and resource-intensive.

G2 **Challenges of Exploration**: A fundamental assumption in standard RL is that no prior knowledge of the system is required, leading to random exploration during the initial training phases. While this is acceptable in a simulated environment, it poses significant safety risks in the real world. Unrestricted exploration can cause excessive wear and tear on machinery and, in the worst case, lead to equipment failure or damage. At the same time, exploration is vital for RL to function effectively, as it allows the system to discover optimal policies and adapt to its environment. This makes it essential to balance safety and exploration by incorporating prior knowledge, such as safety mechanisms or guided exploration strategies, when deploying RL in production environments.

1.2.2. Real-world Environments

Real-world environments are inherently complex and uncertain, and capturing this complexity in a simulation is not straightforward. Simulations are typically designed to abstract and simplify reality, making them inadequate for representing the intricacies of real-world dynamics. As current RL methods are primarily designed for such simplified simulation environments, they are not well-equipped to address the challenges of real-world settings. These challenges include accurately capturing the environment and effectively detecting and managing non-stationary dynamics.

G3 **Environment Uncertainty**: In real systems, there is always some degree of

uncertainty. Factors such as hardware variability between different robots, sensor noise, and incomplete knowledge about the positions or characteristics of objects in the environment introduce unpredictability. For example, a robot in production may encounter subtle differences in part alignment or unexpected environmental factors that are not accounted for in simulations. Some parts of the environment might also not be easily measurable, such as the positions of parts that the robot has to interact with. Visuomotor control [34, 35] involves learning control policies from raw image data, often using neural networks to extract low-dimensional state representations. Finding a state representation reliable enough for robotic policies is a nontrivial task that can considerably hinder success in such environments.

G4 **Nonstationary Environments**: Unlike controlled or simulated settings, real-world environments evolve over time, introducing gradual and abrupt changes. Gradual changes occur as hardware components degrade due to wear and tear, altering system dynamics. Additionally, external factors like fluctuating temperatures, changing lighting conditions, or variations in material properties can also lead to gradual changes. In contrast, abrupt changes arise from events such as component replacements during maintenance or variations in the production task. These non-stationary conditions pose a major challenge for RL, which traditionally assumes a stationary environment.

1.2.3. Real-world Users

Reinforcement learning must be user-friendly for practitioners with limited ML experience to be effectively applied in production environments. Two main concerns regarding usability in production are interpretability and user acceptance.

G5 **Interpretability**: Current RL methods require extensive hyperparameter tuning, often performed by ML specialists who rely on ML specific metrics such as training loss or reward maximisation. However, in a production setting, the focus is on process performance rather than algorithmic details. Practitioners typically have deep expertise in the production process but may lack familiarity with ML techniques. Therefore, developing more interpretable RL systems that relate directly to application-specific metrics is crucial, allowing users to understand how adjustments in hyperparameters or model behaviour affect process outcomes. Achieving this will improve the accessibility of RL in industrial applications.

G6 **User Acceptance**: A significant barrier to adopting RL in production environments is practitioners' hesitation to replace trusted, traditional control systems with RL-based methods. Many users are more comfortable with deterministic control systems

that provide clear, explainable results. Realising RL 's potential for solving complex tasks often relies on black-box neural networks and self-supervised learning, which introduce challenges related to reliability, transparency, and accountability. Overcoming these challenges is essential for gaining user trust and ensuring successful RL deployment. This requires developing RL approaches that are not only more interpretable but also support incremental improvements to existing systems, facilitating a smoother adoption process.

1.3. Main Contributions

This dissertation systematically addresses the above-identified challenges of applying RL to real-world robotic environments. The following section provides an overview of the thesis structure, followed by a discussion of the individual contributions and their connections to the identified gaps. This thesis is based on several publications, as well as manuscripts that are currently under review. Parts of this thesis are taken verbatim from these publications. The publications are listed with the co-authors at the end of this section. Notwithstanding this list, all publications are cited in the corresponding chapters and can be found in the reference list at the end of the thesis.

1.3.1. Outline

This thesis is structured to explore the identified gaps and present the corresponding contributions. We begin in Chapter 2 by reviewing the preliminaries. Chapter 3 (SAE) focuses on spatial autoencoders (SAEs), a method for automatically extracting state space representations from images. In this chapter, we develop a novel evaluation metric for SAEs and demonstrate its effectiveness in downstream RL applications. Chapter 4 (CHEQ) introduces Contextualized Hybrid Ensemble Q-Learning (CHEQ), an adaptive hybrid RL algorithm that combines a learned RL agent with a control prior. In Chapter 5 (Polishing), we explore the application of CHEQ to robotic polishing, a prototypical task for future production environments. The related work relevant to each contribution is discussed in the corresponding chapters. Table 1.1 summarises the link between the individual chapters and the research gaps. Finally, Chapter 6 presents the conclusions and summarises the key findings of the thesis.

Table 1.1.: Overview and link between individual gaps and the chapters of this thesis.

Gap	Gap ID	Link to Chapter
Real-world Data Collection	G1	SAE (Chap. 3), CHEQ (Chap. 4), Polishing (Chap. 5)
	G2	CHEQ (Chap. 4), Polishing (Chap. 5)
Real-world Environments	G3	SAE (Chap. 3), Polishing (Chap. 5)
	G4	CHEQ (Chap. 4), Polishing (Chap. 5)
Real-world Users	G5	SAE (Chap. 3), CHEQ (Chap. 4)
	G6	SAE (Chap. 3), CHEQ (Chap. 4)

1.3.2. Evaluation of Spatial Feature Extractors SAE

In real-world robotic control tasks, the effectiveness of RL depends heavily on the quality of the state representation. This representation must capture all task-relevant objects and information within the environment. As highlighted in gap G3, this challenge becomes especially pronounced in unstructured environments or when the robot must interact with numerous objects. Cameras are rich, low-cost sensors that can be easily integrated into production settings. However, as raw image data is inherently unstructured and high-dimensional, learning from images poses significant challenges [36]. This makes raw image data unsuitable to capture an effective representation of the environment state. A promising solution to this issue involves extracting low-dimensional representations from images, facilitating more efficient and effective use of visual data in control tasks [37, 34].

Different state representations can be useful depending on the production setting. A particularly common case in robotic manipulation involves spatial problems, where robots operate in Cartesian space. In these scenarios, the state must effectively encode spatial information, such as the positions of task-relevant objects. This subsection focuses on these problems as a representative use case for evaluating state representations. A notable solution in this domain is the use of SAEs [38, 39], which effectively learn to extract low-dimensional representations from images in the form of 2D spatial keypoints. These keypoints can then serve as components of the RL agent's state. However, limited research has investigated the ability of SAEs to accurately extract positional information

from scenes.

Most existing evaluations focus on RL performance in downstream tasks [40, 41, 42]. While informative, such evaluations are computationally expensive and make it challenging to determine whether poor performance arises from deficiencies in the SAE or the RL training process itself. Alternative approaches often focus on the SAE 's training loss or conduct qualitative assessments of keypoint locations. However, these methods frequently neglect the temporal consistency of keypoints over trajectories. As noted in gap G5, developing physically meaningful and interpretable metrics is crucial for real-world robotic production. Such metrics can provide actionable insights to process experts and enhance their understanding of system performance. In response to these gaps, this part of the dissertation makes two contributions:

C1 **SAE Evaluation Metric**: We propose a quantitative metric for evaluating how effectively SAEs extract positional information for task-relevant objects. The metric allows practitioners to assess whether the feature extractor sufficiently captures all critical objects required for the task. This is particularly important, as feature extractors often must be fine-tuned for specific tasks to ensure accurate state representation in complex environments.

C2 **Prediction of Downstream RL Performance**: We demonstrate that SAE performance, as measured by our proposed metric, can serve as a reliable predictor of downstream RL performance. This enables practitioners to evaluate the quality of their feature extractors with far fewer computational resources than would be required for full RL training and without needing to run experiments on real robotic hardware. This metric significantly increases the usability of existing SAE approaches for RL in real-world settings by providing an efficient way to pre-evaluate SAE performance.

Our proposed metric serves as a cornerstone for thoroughly evaluating the SAE component within complex RL pipelines. Such a separate evaluation can significantly reduce the sample requirements for downstream RL training, addressing the concerns raised in gap G1. Additionally, evaluations based on physically interpretable quantities can foster greater user acceptance, as discussed in gap G6.

1.3.3. Adaptive Hybrid Reinforcement Learning CHEQ

One of the primary reasons for the limited application of RL in real-world settings is that many state-of-the-art RL algorithms [43, 22] are designed with a task-agnostic architecture. This means they do not incorporate prior knowledge about solving the task, forcing the agent to learn entirely from scratch. As a result, these approaches

are data-intensive (cf. gap G1) and particularly problematic in real-world applications, where initial exploration can lead to unsafe or damaging behaviour (cf. gap G2).

Control theory provides a rich set of methods for designing controllers that offer reliable performance in many applications. While some approaches, such as optimal control methods, can provide provably optimal solutions within their assumed models, many traditional controllers rely on simplified or linearised representations of system dynamics. When these modelling assumptions do not fully capture the complexity of the real system, the resulting controllers may be suboptimal. Nevertheless, they provide a safe and structured starting point for RL exploration. This motivates the drive for hybrid RL methods [44, 45] that blend control priors with deep RL policies. Hybrid algorithms thus combine the prior controller's generalisation capabilities and informed behaviour with the power of deep RL for solving general nonlinear problems.

The majority of prior work [44, 45, 35, 46] proposes a fixed weighting between the control prior and the RL agent. A fixed blending, however, disregards the fact that the capability of the RL agent depends on training time and state. As more data is observed, the RL agent improves its behaviour, ultimately outperforming the control prior in large portions of the domain. The core idea of our approach is to adapt the weighting between the RL agent and control prior based on the agent's confidence. While achieving this goal, we have found the following three contributions.

C3 **Unified Hybrid RL Framework**: We provide a unified view on hybrid RL, which allows for a systematic classification of previous work and exposes critical limitations in existing methods. This framework highlights the need for a novel adaptive hybrid RL formulation with descriptive, time-invariant dynamics.

C4 **Hybrid MDP Formulation**: To address the limitations found in contribution C3, we propose a novel formulation called the contextualised hybrid MDP. This formulation addresses structural problems of prior work in hybrid RL with adaptive weighting. The contextualised hybrid MDP is relevant for any non-stationary or adaptive combination of an existing policy with a learned RL policy.

C5 **Adaptive Hybrid RL**: Building on this new MDP formulation, we propose a novel adaptive hybrid RL algorithm called CHEQ. Contextualized Hybrid Ensemble Q-Learning dynamically adjusts the weighting between the RL agent and the control prior using an uncertainty estimate from a critic ensemble. This uncertainty-driven approach allows the algorithm to gradually shift control from the control prior to the RL agent as the agent becomes more confident in its learned policy. CHEQ also integrates ensemble-based acceleration techniques to improve data efficiency, enhancing its applicability to real-world settings. Compared to previous work in adaptive hybrid RL,

CHEQ offers several key advantages. It reduces the number of failures during training by leveraging the control prior in uncertain situations and improves sample efficiency, effectively addressing gaps G1 and G2. In this thesis, safety is not defined in terms of strict provable guarantees but rather as ensuring that the exploration behaviour remains within practical safety limits. This often means acting better, or at least not significantly worse, than the prior controller during training. Beyond improving training efficiency and safety, CHEQ enhances the agent's ability to transfer its learned behaviour to new, unseen tasks, making it more robust for generalisation in dynamic environments. This increased performance and safety outside the training data distribution is a practical step towards the application in changing real-world production environments (cf. gap G4). Lastly, effectively quantifying the RL agent's capabilities through the critic ensemble can increase interpretability and user acceptance, as discussed in gaps G5 and G6.

1.3.4. Robotic Polishing Polishing

The adaptive hybrid reinforcement learning (AHRL) algorithm, CHEQ (cf. Section 1.3.3), demonstrates significantly safer exploration behaviour, characterised by fewer violations of safety limits and accelerated learning compared to traditional model-free RL, residual RL, and prior work in AHRL. The properties of AHRL methods in general, and CHEQ in particular, make them promising candidates for real-world applications and direct hardware training. While some fixed-weight hybrid RL methods have been successfully trained on hardware [45, 35], advancements in AHRL have so far been validated only in simulation and with low-dimensional action spaces.

Real-world challenges, such as contact dynamics, friction, and state-dependent noise, are difficult to simulate accurately. On the other hand, training directly on hardware introduces a range of uncertainties and non-stationarities in the environment, aligning with the challenges described in gaps G3 and G4. To fully explore the potential and limitations of AHRL methods, it is essential to evaluate their performance on challenging hardware tasks.

In this part, we focus on polishing a 3D object with a robotic arm, a task that exemplifies the complexities of contact-rich manipulation in industrial automation. Polishing requires maintaining continuous contact with the surface while adhering to specific force and velocity profiles, necessitating precise control to ensure quality and consistency. Variable impedance control is particularly well-suited for such tasks [47, 48]. However, determining appropriate variable impedance gains is inherently complex, and manual

tuning becomes impractical for tasks involving high-dimensional or nonlinear dynamics. These factors make robotic polishing an ideal benchmark for evaluating RL-based control strategies.

C6 **Safe Exploration and Variable Impedance Learning**: We demonstrate the potential of CHEQ to learn variable impedance gains while maintaining safe exploration behaviour. In simulation, we show that variable impedance control (VIC) significantly enhances polishing performance, achieving more consistent and precise control. Comparing standard RL approaches with AHRL, we find that CHEQ effectively learns impedance gains while ensuring that the control prior guides exploration behaviour close to the desired task. This results in exploration that remains within practical safety limits. Although CHEQ does not provide theoretical safety guarantees, its adherence to practical safety constraints addresses gap G2 by reducing hardware wear and tear and creating a safer training environment for nearby humans.

C7 **Direct Hardware Deployment**: We demonstrate that the CHEQ algorithm can be deployed directly on hardware, exhibiting both safe and efficient learning behaviour. Applying CHEQ on hardware allows us to evaluate its contributions to sample efficiency (cf. gap G1), safety (cf. gap G2), and handling of environment uncertainty (cf. gaps G3, G4). Our results show that CHEQ effectively manages these challenges in a use case representative of future production scenarios, confirming its suitability for real-world industrial automation tasks. While multiple approaches exist which train a fixed-weight hybrid RL policy on hardware (see Section 4.2), this dissertation marks the first application of an AHRL policy weighted by the agent's uncertainty, to a hardware learning problem.

1.3.5. Overview of Publications

This thesis, including the introduction section, draws from multiple publications in ML and control conferences written during the author's time as a PhD student. The authors' contributions are reflected by the order of names in the publications. The first author contributed the most, and all authors were involved in editing and writing. In all cases, the last author, Prof. Sebastian Trimpe, provided active academic guidance, including feedback on research and manuscripts.

Evaluation of Spatial Autoencoder

The first publication introduces a method to evaluate the performance of SAEs by measuring how well extracted keypoints track ground truth objects in images.

- **E. Cramer**, J. Reiher, and S. Trimpe, 'Tracking object positions in reinforcement learning: A metric for keypoint detection', in *Proceedings of the 6th Annual Learning for Dynamics & Control Conference*, Oxford: PMLR, Jun. 2024, pp. 104–116. Available: `https://proceedings.mlr.press/v242/cramer24a.html`

The author of this thesis developed the method and problem formulation. Jonas Reiher implemented the evaluation metric and the validation in downstream RL, which was the basis for his master's thesis.

Contextualized Hybrid Ensemble Q-learning

The second publication introduces 'Contextualised Hybrid Ensemble Q-learning: Learning Fast with Control Priors' (CHEQ), which adaptively combines a classic control approach and an RL agent.

- **E. Cramer**, B. Frauenknecht, R. Sabirov, and S. Trimpe, 'Contextualised Hybrid Ensemble Q-learning: Learning Fast with Control Priors', in *Reinforcement Learning Journal*, Amherst Massachusetts, Aug. 2024, pp. 926–945.
  Available: `https://rlj.cs.umass.edu/2024/papers/Paper118.html`

The method CHEQ was co-developed with Ramil Sabirov and Bernd Frauenknecht, who contributed equally to this publication. Bernd and the author of this dissertation designed the problem formulation, contributed to the solution approach, and wrote the publication. Ramil implemented the method, contributed to the solution approach, and performed the systematic testing. For Ramil Sabirov, CHEQ was the basis for his master's thesis.

Robotic Polishing

Finally, the third publication presents an experimental demonstration of the hybrid RL algorithm CHEQ for robotic polishing with variable impedance.

- **E. Cramer**, L. Jäschke, and S. Trimpe, 'CHEQ-ing the Box: Safe Variable Impedance Learning for Robotic Polishing ', *Joint 10th IFAC Symposium on Mechatronic Systems and 14th Symposium on Robotics*, Paris, Jul 2025 (to appear).

The author of this thesis developed the method, problem formulation and overall control pipeline. During his master's thesis, Lukas Jäschke implemented the simulation environment and performed the experiments.

1.3.6. Additional Publications by the Author

In addition to the publications listed above, the author has also actively contributed to the following publications during the time of the PhD:

- J. Dierkes, **E. Cramer**, H. H. Hoos, and S. Trimpe, 'Combining Automated Optimisation of Hyperparameters and Reward Shape', in *Reinforcement Learning Journal*, Amherst Massachusetts, Aug. 2024, pp. 1441–1466.
  Available: `https://rlj.cs.umass.edu/2024/papers/RLJ_RLC_2024_188.pdf`
- P. Ennen, **E. Pabich**, R. Kupper, P. Benmoussa, and R. Vossen, 'Leitfaden Selbstlernende Produktionsprozesse -Einführungsstrategie für Reinforcement Learning in der industriellen Praxis', VDMA, *Forum Industrie 4.0*, 2019.

These publications are not part of this thesis, but the results partly relate to the contributions herein, and the author contributed to them during her time as a PhD student, partly during her time before joining DSME.

1.3.7. Student Thesis Supervisions

During the PhD, the author supervised many student thesis projects, often aligned with the subject of this work. Some of these have led to publications with the student as a co-author and are included in this thesis (cf. Section 1.3.5). All other results are not included in this thesis.

The author served as the primary supervisor for the following thesis projects:

- L. Jäschke, 'Flexible Robotic Polishing via Adaptive Hybrid Reinforcement Learning', master thesis, RWTH Aachen University, 2024.
- R. Sabirov, 'Uncertainty-based Blending of Deep Reinforcement Learning and Classic Control', master thesis, RWTH Aachen University, 2024.
- A. Pradhan, 'Achieving Compliant Motion Control for Continuous-Contact Task with Reinforcement Learning', master thesis, RWTH Aachen University, 2023.
- J. Reiher, 'Unsupervised State Representation Learning from Video for Reinforcement Learning Applications in Robotics', master thesis, RWTH Aachen University, 2023.
- H. Markgraf, 'Toward Safety Guarantees for Deep Neural Network Controllers: Stability Analysis of Reinforcement Learning Policies', master thesis, RWTH Aachen University, 2021.

- J. N. Büscher, 'Lung Cancer Segmentation with Guided Neural Networks for Computed Tomography Images', master thesis, RWTH Aachen University, 2020.
- L. Merten, 'Learning Fair Trading in Financial Markets with Multi-Agent Reinforcement Learning', master thesis, RWTH Aachen University, 2020.

In addition, the author served as a secondary supervisor for the following thesis projects:

- T. H. Sandermann, 'Centralized Voltage Control in Distribution Grids Using Hybrid Reinforcement Learning', master thesis, RWTH Aachen University, 2025.
- C. Hauer, 'Reinforcement Learning for Decentralized Voltage Control in Distribution Grids', bachelor thesis, RWTH Aachen University, 2025.
- D. Singh, 'Combining Visual Servoing & Force Control with Reinforcement Learning for Gear Assembly', master thesis, RWTH Aachen University, 2024.
- M. A. Farook Deen, 'Investigating Robot Learning of Quadrupedal Locomotion on Deformable Terrain', master thesis, RWTH Aachen University, 2024.
- K. Sachdev, 'Analysis of Observation Spaces for Reinforcement Learning in Robotic Manipulation', master thesis, RWTH Aachen University, 2023.
- J. Middendorf, 'Performance Analysis and Scalability of Reinforcement Learning - Algorithms for Climatisation of Multi-Zone Buildings', master thesis, RWTH Aachen University, 2023.
- M. Waqar, 'Analysis of Field Test Data and Development of a Data-driven Model to Describe the Behaviour of Ice Melting Probes', master thesis, RWTH Aachen University, 2022.

2. Preliminaries

In this thesis, we investigate and address challenges on how RL can be applied to production scenarios. This chapter introduces the necessary background for the following chapters. We introduce neural networks (NNs) and their training procedure, which are at the core of deep RL. Next, we introduce the fundamentals of deep RL and build up towards the soft actor-critic (SAC) algorithm. SAC is a modern model-free actor-critic architecture, which will be the base algorithm used in all subsequent chapters. We then introduce autoencoders, a key deep learning approach, which will be used as a feature extraction method in Chapter 3. Lastly, we introduce the necessary background in robotic manipulation, which we require to formalise the polishing task in Chapter 5.

2.1. Deep Learning

At the core of deep learning are artificial neural networks (ANNs), computational models inspired by the structure and function of biological neural networks. ANNs, often also called NNs, consist of layers of interconnected nodes, or neurons, that learn to approximate functions by adjusting internal parameters (weights and biases) during training. There are various types of NNs, each tailored to specific domains and data structures. Feedforward neural networks (FNNs) are the simplest form, where information moves in one direction from input to output without feedback loops. Convolutional neural networks (CNNs) specialise in processing grid-like data, such as images, by applying a moving filter over the input image to extract a feature map that captures spatial relations. Recurrent neural networks (RNNs), such as long short-term memory (LSTM) networks, are designed for sequential data. They allow information to persist through loops in the network, making them suitable for tasks like time-series forecasting and natural language processing.

The following introduces the general notation for ANNs and details the two types of layers used in this thesis, feedforward and convolutional layers. Then, we present gradient-based optimisation techniques commonly used to train these networks.

2.1.1. Artificial Neural Networks

An ANN is the fundamental architecture in deep learning. It consists of an input layer, one or more hidden layers, and an output layer. In mathematical terms, an ANN can be represented as a function $f_\theta : \mathcal{X} \to \mathcal{Y}$, where the network is parameterised by a set of parameters θ, and $\mathcal{X}$ and $\mathcal{Y}$ represent the sets of possible inputs and outputs, respectively.

For a given input $x \in \mathcal{X}$, the network computes a predicted output y_{pred}. This mapping from input to output is computed through the sequential evaluation of the hidden layers $l_1 \dots l_n$, such that $y_{\text{pred}} = f_\theta(x) = f_{l_n} \circ \cdots f_{l_2} \circ f_{l_1}(x)$.

In a supervised learning setting, this predicted output y_{pred} should approximate a target or ground-truth value y_{target} as closely as possible, assuming the weights θ are sufficiently optimised. A scalar loss function $L(f_\theta(x), y_{\text{target}})$ quantifies the discrepancy between the predicted output and the target value over all samples (x, y_{target}) in a dataset D [49].

For example, in a regression problem where the outputs and targets are continuous variables $y \in \mathbb{R}$, the mean squared error (MSE) is a common choice for the loss function. It is defined as

$$L_{\text{MSE}}(y_{\text{pred}}, y_{\text{target}}) = \mathbb{E}\left[(y_{\text{pred}} - y_{\text{target}})^2\right], \tag{2.1}$$

where the expectation is typically approximated using the empirical mean over the training dataset. The goal of training the NN is to minimise this loss by adjusting the network's parameters.

Feedforward Layers

A feedforward layer in a NN transforms its inputs via a weighted sum followed by a nonlinear activation function. This structure allows feedforward networks to be highly flexible and capable of approximating complex functions, making them fundamental in many ML applications. The basic operation for a given layer l_i is given by

$$a_{l_i} = f_{l_i}(x) = \sigma(w_i^T x + b_i), \tag{2.2}$$

where $x \in \mathbb{R}^{n_i}$ is the input vector of the layer l_i, $w_i \in \mathbb{R}^{m_i \times n_i}$ are the weights of the layer, $b_i \in \mathbb{R}^{m_i}$ are the biases, $a_{l_i} \in \mathbb{R}^{m_i}$ is the output of the layer, and σ denotes the non-linear activation function. Each layer's output serves as the next layer's input.

A network consisting of feedforward layers is called a FNN, and its parameters θ are the set of weights and biases of all layers in the network. Despite their representational

power, fully connected feedforward layers do not scale well to high-dimensional data, as the number of parameters grows rapidly with input size. This increases computational cost, making them less suitable for tasks involving structured data such as images or sequences.

Convolutional Layers

Due to the high dimensionality and spatial properties of image data, a specialised type of neural network layer, known as a convolutional layer, is used to process image inputs. Specifically, images $x \in X \subseteq \mathbb{R}^{h \times w \times c}$ are represented as tensors with height h, width w, and c channels. These high-dimensional inputs can lead to extremely large parameter spaces for NNs, making traditional feedforward layers computationally impractical.

To address this, convolutional layers replace simple weighted sums with convolutional operations. Operations between layers are computed over local receptive fields, and the same set of weights, a kernel, is shared across different regions. The convolution operation for a 2D input is mathematically expressed as

$$S_{(i,j)} = (x * K)_{(i,j)} = \sum_m \sum_n x_{(m,n)} K_{(i-m,j-n)} \tag{2.3}$$

where $x \in \mathbb{R}^2$ represents the input image matrix, $K \in \mathbb{R}^2$ is the kernel, and $S \in \mathbb{R}^2$ is the output feature map. By applying the same kernel across different regions, this operation efficiently captures spatial dependencies while significantly reducing the number of free parameters and maintaining spatial equivariance. A neural network primarily composed of convolutional layers is referred to as a CNN.

2.1.2. Training a Neural Network

Training a NN involves solving an optimisation problem where the network's parameters θ are adjusted to minimise a loss function $L(f_\theta(x), y_{\text{target}})$ over the dataset D. The optimisation can be represented as

$$\theta^* = \arg\min_\theta \frac{1}{|D|} \sum_{(x, y_{\text{target}}) \in D} L(f_\theta(x), y_{\text{target}}). \tag{2.4}$$

Finding a closed-form solution to this problem is generally infeasible due to the high-dimensional, non-convex nature of the loss landscape and the sheer number of parameters in modern networks. Additionally, the large size of typical datasets further complicates the optimisation process, making exact computations impractical.

To address these challenges, NNs are optimised using an iterative approach based on gradient-based methods. Rather than processing the entire dataset at once, training is typically performed using mini-batches, randomly sampled subsets of the dataset, allowing for efficient updates while maintaining computational feasibility. A widely adopted method is stochastic gradient descent (SGD) [49], which updates the weights based on the gradient of the loss function with respect to each parameter.

In the naive case, the parameter update at iteration j is given by:

$$\theta_i^{(j+1)} = \theta_i^{(j)} - \eta \left. \frac{\partial L(f_\theta(x), y_{\text{target}})}{\partial \theta_i} \right|_{\theta^{(j)}}, \tag{2.5}$$

where θ_i is a particular parameter, η is the learning rate, and the gradient $\nabla_\theta L$ indicates the direction in which the parameter should be adjusted to reduce the loss. This process is repeated iteratively for all parameters in the network until the loss converges to a minimum.

More complex optimisation algorithms propose adapting the learning rate to improve convergence speed and stability. For instance, Adam [50] extends SGD by incorporating momentum and adaptive learning rates, maintaining moving averages of both the gradients and their squared values. This approach empirically results in more stable and faster convergence, particularly in highly non-convex loss landscapes.

2.2. Reinforcement Learning

In RL, an agent learns a policy, a map from situations to actions. Instead of being explicitly told what actions to take in a supervised manner, an RL agent learns through trial and error, experimenting with different actions and receiving feedback in the form of rewards [21].

While Section 1.1 provided an overview of RL notation, this section delves deeper into its foundational concepts leading up to the SAC algorithm [22], which is used as the base algorithm within this thesis.

2.2.1. Reinforcement Learning Fundamentals

RL is commonly formalized as a discounted MDP, represented by the tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, p, r, \rho_0, \gamma)$. Here, $\mathcal{S}$ denotes the state space, $\mathcal{A}$ the action space, $p(s_{t+1}, r_{t+1} \mid s_t, a_t)$ is the unknown transition function that defines the probability of transitioning to a new state s_{t+1} and receiving a reward r_{t+1} after taking action a_t in state s_t. The distribution

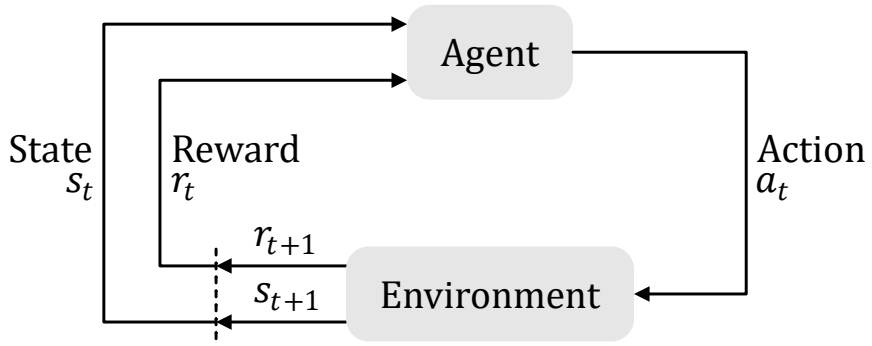


Figure 2.1.: The agent-environment interaction in an MDP, adapted from Sutton and Barto [21]. In each time step, the agent observes a state s_t and a reward r_t and chooses an action a_t. This action leads to the next state state s_{t+1}, and the agent receives reward r_{t+1} for the transition.

of the initial state s_0 given by $\rho_0(s_0)$, and $\gamma \in (0, 1)$ is the discount factor applied to future rewards.

At each timestep t, the agent interacts with the environment by selecting an action $a_t \in \mathcal{A}$ according to its policy $\pi(a_t \mid s_t)$. In return, it receives a scalar reward $r_{t+1} = r(s_t, a_t) \in \mathbb{R}$ from the reward function. Figure 2.1 illustrates this process. A sequence of state transitions and actions, or a trajectory, is denoted by $\tau = (s_0, a_0, r_1, s_1, \ldots, s_T)$. The goal of the RL agent is to learn an optimal policy $\pi^*(a_t \mid s_t)$ that maximises the expected cumulative discounted reward over time, formalised as:

$$J(\pi) = \mathbb{E}_{\tau \sim \pi, \mathcal{M}} \left[\sum_{t=0}^{\infty} \gamma^t r_{t+1} \right]. \tag{2.6}$$

This is known as the return, and the optimisation problem can be written as maximising this return with respect to the policy π. With $\gamma < 1$, the infinite sum in Equation (2.6) has a finite value. This formulation also includes episodic tasks with a finite horizon T. For a finite episode length T, the sum is also well defined for a discount factor $\gamma = 1$.

Key functions in MDPs include the value function

$$V^{\pi}(s_t) = \mathbb{E}_{\tau \sim \pi, \mathcal{M}} \left[\sum_{t=0}^{\infty} \gamma^t r_{t+1} \mid s_0 = s_t \right], \tag{2.7}$$

which estimates the expected return starting from state s under policy π, and the action-

value function, or Q-function,

$$Q^{\pi}(s_t, a_t) = \mathbb{E}_{\tau \sim \pi, \mathcal{M}} \left[\sum_{t=0}^{\infty} \gamma^t r_{t+1} \mid s_0 = s_t, a_0 = a_t \right], \tag{2.8}$$

which estimates the expected return starting from state s and taking action a.

The RL agent's goal is to find the policy π that maximises the expected return, which is equivalent to finding the optimal policy π^* that satisfies:

$$\pi^*(a_t \mid s_t) = \arg\max_{\pi} J(\pi). \tag{2.9}$$

There are several approaches for finding the optimal policy in RL. These methods can generally be grouped into value-based, policy-based, and actor-critic. Value-based methods [51, 24], such as Q-learning, focus on learning a value function that helps select optimal actions indirectly. Policy-based methods [52, 53, 43], on the other hand, directly optimise the policy without explicitly learning value functions. Actor-critic methods [54, 55, 56, 22] combine the strengths of both by learning both a value function and an explicit policy. Additionally, RL approaches can be divided into model-free methods, which do not rely on a model of the environment, and model-based methods, which use a model to predict future states and rewards. Furthermore, algorithms can be classified as on-policy, where the policy being improved is also used for decision-making, or off-policy, where the policy used for learning differs from the one used to explore the environment. This thesis focuses on model-free RL and uses SAC, an off-policy actor-critic algorithm.

Q-Learning and Mean Squared Bellman Error

Modern offline actor-critic algorithms stem from the idea of tabular Q-learning [57]. The starting point for the update rule of tabular Q-Learning is the Bellman optimality equation [21] for the optimal Q-function $Q^*(s_t, a_t)$

$$\begin{aligned} Q^*(s_t, a_t) &= \mathbb{E}\left[R_{t+1} + \gamma V^*(S_{t+1}) \mid S_t = s_t, A_t = a_t\right] \\ &= \mathbb{E}\left[R_{t+1} + \gamma \max_{a'} Q^*(S_{t+1}, a') \mid S_t = s_t, A_t = a_t\right] \\ &= \sum_{s_{t+1}, r_{t+1}} p(s_{t+1}, r_{t+1} \mid s_t, a_t)(r_{t+1} + \gamma \max_{a'} Q^*(s_{t+1}, a')), \end{aligned} \tag{2.10}$$

where s_t denotes the current state and s_{t+1} the next state. If the MDP transition dynamics $p(s_{t+1}, r_{t+1} \mid s_t, a_t)$ would be known, this equation can, in principle, be solved by dynamic programming (DP) approaches [58]. Since this is generally not the case, RL

algorithms are utilized to iteratively find an approximate solution to $Q^*(s_t, a_t)$ based on experienced transition samples $\langle s_t, a_t, s_{t+1}, r_{t+1} \rangle$.

Equation (2.10) must be satisfied for every experienced transition $\langle s_t, a_t, s_{t+1}, r_{t+1} \rangle$ in a trajectory τ [21]. Inserting the current estimation $Q(s_t, a_t)$ for $Q^*(s_t, a_t)$, the right-hand side resembles a one-step-ahead estimate closer to $Q^*(s_t, a_t)$ with observed reward r_{t+1}. Using a learning rate η allows formulating the Q-learning update rule

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \eta(r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t)). \tag{2.11}$$

Exploration and Exploitation

A key challenge in RL is striking the right balance between exploration, which involves trying new actions to discover potentially better strategies, and exploitation, which involves leveraging the current policy to maximise rewards. Overemphasising exploration can lead to inefficient use of resources, while excessive exploitation risks overlooking better strategies. In Q-learning, this tradeoff is handled using the ϵ-greedy policy, where the agent selects an action at random with probability ϵ and chooses the action deemed optimal with probability $1 - \epsilon$. The ϵ-greedy policy can be written as

$$\pi_{\epsilon\text{-greedy}}(s_t) = \begin{cases} \arg\max_{a_t \in \mathcal{A}} Q(s_t, a_t), & \text{with probability } 1 - \epsilon \\ a_t \sim \mathcal{U}(\mathcal{A}), & \text{with probability } \epsilon. \end{cases} \tag{2.12}$$

2.2.2. Deep Reinforcement Learning

As the state space grows, classical Q-learning becomes impractical due to the significant memory required to store a tabular representation of the Q-function. In many real-world problems, the state space is continuous or extremely large, making traditional Q-learning infeasible. To address this, Q-learning can be combined with function approximators like deep NNs.

Mean-squared Bellman Error

We approximate the Q-function $Q_\theta(s_t, a_t)$ by a NN parameterized by θ. A suitable loss function for gradient-based optimisation is required to train this NNs, as mentioned in Section 2.1.2. The right-hand side of Equation (2.10) can again be used as a target value $y_{\text{target}}(s_{t+1}, r_{t+1})$ and be combined with the MSE loss function in Equation (2.1). With a set of experienced transitions $\mathcal{D}$, collected over trajectories τ, this allows the

formulation of the mean squared Bellman error (MSBE) as a loss [21]

$$\begin{aligned} L_{\text{MSBE}}(\phi, \mathcal{D}) &= \mathbb{E}_{\langle s_t, a_t, s_{t+1}, r_{t+1}\rangle \sim \mathcal{D}} \left[\left(y_{\text{target}}(s_{t+1}, r_{t+1}) - Q_\theta(s_t, a_t) \right)^2 \right] \\ &= \mathbb{E}_{\langle s_t, a_t, s_{t+1}, r_{t+1}\rangle \sim \mathcal{D}} \left[\left(r_{t+1} + \gamma \max_{a'} Q_\theta(s_{t+1}, a') - Q_\theta(s_t, a_t) \right)^2 \right]. \end{aligned} \tag{2.13}$$

This loss forms the basis for most deep Q-learning algorithms, such as the Deep Q-Network (DQN) [24] algorithm.

Deep Q-learning

In DQN [24], the Q-function $Q_\theta(s_t, a_t)$ is approximated by a NN parameterized by θ, commonly referred to as a Q-network. To learn this NN, a dataset $\mathcal{D}$ of transitions $\langle s_t, a_t, r_{t+1}, s_{t+1}\rangle$ is collected by rolling out some non-optimal policy π. This dataset is known as the experience replay buffer. In addition to the standard Q-network $Q_\theta(s_t, a_t)$, DQN maintains an additional Q-network called the target network $Q_{\theta_{\text{target}}}(s_t, a_t)$.

The parameters θ are updated at each timestep by minimising the loss function

$$\mathcal{L}_{\text{DQN}}(\theta) = \mathbb{E}_{\langle s_t, a_t, s_{t+1}, r_{t+1}\rangle \sim \mathcal{D}} \left[\left(r_{t+1} + \gamma \max_{a' \in \mathcal{A}} Q_{\theta_{\text{target}}}(s_{t+1}, a') - Q_\theta(s_t, a_t) \right)^2 \right]. \tag{2.14}$$

The parameters θ_{target} are not updated directly through gradient descent. Instead, they are periodically copied from the main Q-network after a fixed number of training steps.

The use of experience replay helps to reduce the correlation between consecutive data samples. At the same time, the target network stabilises the learning process by preventing rapid changes in the Q-value estimates. Together, these mechanisms improve the training stability of DQN, addressing earlier challenges of combining Q-learning with deep NNs [59].

Deep Deterministic Policy Gradient

Directly applying DQN to continuous action spaces is impossible due to the maximisation operator in Equation (2.14). Lillicrap *et al.* [55] extend the DQN algorithm to handle continuous action spaces by introducing a deterministic policy, $\pi_\phi(s_t)$, represented by a NN with parameters ϕ. Additionally, they employ a separate target policy network, $\pi_{\phi\text{target}}(s_t)$. This approach is known as Deep Deterministic Policy Gradient (DDPG). Algorithms that simultaneously learn both a policy and a Q-function, like DDPG, are called actor-critic methods.

With this architecture, the Q-learning target $y_{\text{target}}(s_{t+1}, r_{t+1})$ then becomes

$$y_{\text{target}}(s_{t+1}, r_{t+1}) = r_{t+1} + \gamma Q_\theta(s_{t+1}, \pi_{\phi\text{target}}(s_{t+1})), \tag{2.15}$$

which approximates the maximum operator through the deterministic policy.

The policy NN itself can be updated via stochastic gradient ascent (SGA). Since both $\pi_\phi(s_t)$ and $Q_\theta(s_t, a_t)$ are differentiable NNs, gradients with respect to θ can be determined with the deterministic policy gradient (DPG) theorem [54] while $Q_\theta(s_t, a_t)$ is held fixed. The desired policy $\pi_\phi(s_t)$ determines the action maximizing $Q_\theta(s_t, a_t)$, which leads to the optimization objective

$$\max_\phi \mathbb{E}_{s_t \sim \mathcal{D}} \left[Q_\theta(s_t, \pi_{\theta_{\text{target}}}(s_t)) \right]. \tag{2.16}$$

Instead of updating the target networks every few steps like DQN does, DDPG performs continuous Polyak averaging [60]. Target parameter updates for both $\pi_{\theta_{\text{target}}}(s_t)$ and $Q_{\phi_{\text{target}}}(s_t, a_t)$ are then performed in every step with a Polyak update coefficient ρ close to 1 via

$$\theta_{\text{target}} \leftarrow \rho\theta_{\text{target}} + (1-\rho)\theta \quad \text{with} \quad 0 < \rho < 1. \tag{2.17}$$

Twin Delayed DDPG

Twin Delayed DDPG (TD3) by Fujimoto *et al.* [61] further improves upon DDPG [55] by addressing the practical problem of the Q-network $Q_\theta(s_t, a_t)$ overestimating Q-values [62] and the policy exploiting these errors. This is implemented using clipped double Q-learning, where two distinct Q-functions, $Q_{\phi_1}(s_t, a_t)$ and $Q_{\phi_2}(s_t, a_t)$, are maintained, and the minimum of the two Q-values is used to compute the MSBE target

$$y_{\text{target}} = r(s_t, a_t) + \gamma \min_{j=1,2} Q_{\theta_{j,\text{target}}}(s_{t+1}, \tilde{a}_{t+1}(s_{t+1})). \tag{2.18}$$

In this process, TD3 's target action $\tilde{a}_{t+1}(s_{t+1})$ adds clipped noise to the action produced by the deterministic policy to mitigate the exploitation of Q-function errors. Further, the policy updates are delayed to every other Q-function update step.

2.2.3. Soft Actor-Critic

Haarnoja *et al.* [22] propose the SAC algorithm as a model-free and off-policy RL algorithm for continuous state and action spaces. It builds upon the concepts of DQN [24], DDPG [55], and TD3 [61]. The main difference lies in its way of managing the exploration-

exploitation tradeoff. SAC is based on entropy-regularized RL [63], trading off explorative randomness and expected return.

In maximum entropy RL (MERL) [64, 63], the reward of the MDP gets augmented by a term that is proportional to the differential entropy $\mathcal{H}$ of the policy

$$r_{\text{MERL}}(s_t, a_t) = r(s_t, a_t) + \alpha \mathcal{H}(\pi(\cdot \mid s_t)), \tag{2.19}$$

with a temperature parameter $\alpha > 0$. The Q-function is slightly redefined to

$$Q^{\pi}_{\text{MERL}}(s_t, a_t) = \mathbb{E}_{\tau \sim \pi, \mathcal{M}} \left[r(s_t, a_t) + \sum_{k=1}^{\infty} \gamma^k r^{\pi}_{\text{MERL}}(s_k, a_k) \mid s_t = s, a_t = a \right], \tag{2.20}$$

excluding the entropy term for the first state. This new Q-function fulfils the soft Bellman equation and is called *soft Q-function*.

Similar to TD3, the SAC algorithm learns three NNs, two Q-networks Q_{ϕ_1} and Q_{ϕ_2} and one policy network π_θ. Additionally, it uses two target Q-networks $Q_{\phi_1,\text{target}}$ and $Q_{\phi_2,\text{target}}$ which are updated using Polyak averaging [60]. The stochastic policy is modelled as a NN outputting the mean and covariance of a Gaussian distribution. Unlike DDPG and TD3, SAC does not use a separate target network and is instead used directly for selecting target actions.

SAC showed to be consistently more sample efficient and stable than previous model-free RL algorithms [22] such as DDPG [55], TD3 [61] or PPO [43].

2.3. Latent Feature Extraction with Autoencoders

The effectiveness of RL algorithms largely depends on having an appropriate state representation, which is often not readily available. In many real-world scenarios, available sensors provide high-dimensional data that may not be directly interpretable in the context of state representation. To address this, unsupervised state representation learning methods can extract meaningful features from raw observations, such as video sequences, and derive compact representations. In Section 2.3.1, we describe autoencoders, a key tool for this type of feature extraction. Autoencoders condense high-dimensional inputs into lower-dimensional representations, making the extracted state more compact and useful for RL. Section 2.3.2 introduces SAEs, a particular type of autoencoder which capture positional information. This dual benefit of reducing dimensionality while improving understanding makes SAEs especially relevant for state representation learning in RL.

2.3.1. Autoencoders

Autoencoders have been a key component of ML techniques for several decades [49]. At their core, autoencoders learn to map a given input onto itself through a lower-dimensional bottleneck representation, creating an intermediate representation that is of interest for downstream tasks. This intermediate or latent representation can be useful for various applications, including dimensionality reduction, feature extraction, noise filtering, and anomaly detection [65]. Initially introduced by Kramer [66] as a nonlinear extension of the principal component analysis, their most recognised form today is as multilayer feedforward NNs featuring a bottleneck layer.

An autoencoder consists of two key components: the encoder and the decoder. The encoder function, denoted as $E : \mathcal{X} \to \mathbb{R}^n$, maps an input $x \in \mathcal{X}$ to a latent representation $z = E(x)$, while the decoder function, $D : \mathbb{R}^n \to \mathcal{X}$, reconstructs an approximation of the original input from this latent space, yielding $\hat{x} = D(z) = D(E(x))$.

In contemporary implementations, both the encoder $E_{\psi_{\text{enc}}}(x)$ and decoder $D_{\psi_{\text{dec}}}(z)$ are typically modelled using NNs, with their parameters represented as ψ_{enc} and ψ_{dec}, respectively. The objective is to minimise the reconstruction error such that

$$\min_{\psi_{\text{enc}},\psi_{\text{dec}}} \mathbb{E}\left[L_{\text{reconstruction}}(D_{\psi_{\text{dec}}}(E_{\psi_{\text{enc}}}(x)), x)\right], \tag{2.21}$$

where $L_{\text{reconstruction}}(\hat{x}, x)$ denotes the reconstruction loss function. Using a differentiable loss, such as the MSE [65] (cf. Equation (2.1)), this optimisation problem can be efficiently solved through gradient-based methods. Autoencoders are trained in an unsupervised manner, as their goal is to reconstruct the input data, eliminating the need for labelled examples.

Figure 2.2 illustrates the typical architecture of an autoencoder applied to an input image I and its reconstructed output $\hat{I}$.

Without additional constraints, both the encoder $E_{\psi_{\text{enc}}}(x)$ and decoder $D_{\psi_{\text{dec}}}(z)$ could simply learn an identity mapping, which would not produce any meaningful latent representation [49]. To address this, various constraints can be imposed on the latent space z to ensure that the representation learned is useful for the intended application [65]. A common approach is introducing a low-dimensional bottleneck between the encoder and decoder, which forces a compressed latent representation. This encourages the encoder to capture meaningful, variable aspects of the input, while the decoder may learn biases to reconstruct more stationary elements across samples.

However, simple dimensionality reduction may not be sufficient for controlling the

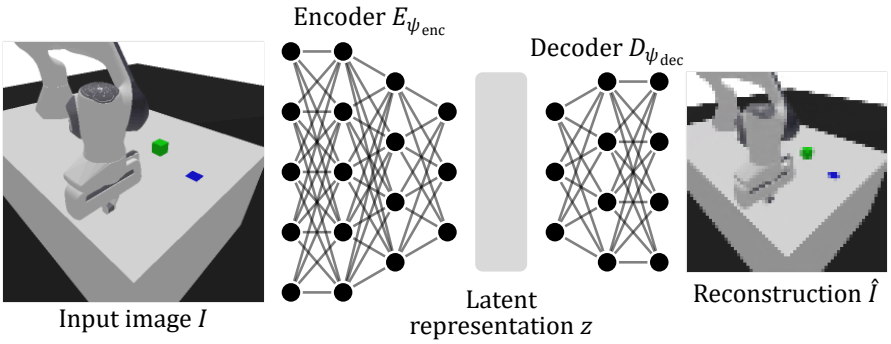


Figure 2.2.: Structure of a typical autoencoder for image data, consisting of an encoder $E_{\psi_{\text{enc}}}$ to map the image input I to a latent representation $z = E_{\psi_{\text{enc}}}(I)$ and a decoder reconstructing the input approximatively from the latent representation as $\hat{I} = D_{\psi_{\text{dec}}}(z)$.

specific features encoded in z. Depending on the application, different aspects of the input might be important. Regularisation techniques, such as denoising autoencoders [67], variational autoencoders (VAEs) [68], or SAEs [38], are often employed to achieve more structured and meaningful representations. In the following sections, we focus on SAEs, which are particularly effective at capturing positional information in images. These autoencoders serve as the foundation for state extraction in Chapter 3.

2.3.2. Spatial Autoencoders

SAEs are a specialised type of autoencoder where the latent bottleneck is constrained to capture 2D positional information in image space. These autoencoders have the typical architecture consisting of an encoder, a latent bottleneck, and a decoder. The encoder, usually a CNN, extracts features from images, which are then encoded into a spatially meaningful representation using a spatial soft-argmax operation. Finally, a convolutional or feedforward decoder reconstructs the input from the latent representation.

The spatial soft-argmax operation, also known as spatial softmax or spatial expectation, was first proposed by Levine *et al.* [34] as part of a feature extraction network for end-to-end RL. Building on this, Finn *et al.* [38] integrated the spatial soft-argmax into an autoencoder, positioning it as the final encoder layer after multiple convolutional layers. The spatial soft-argmax serves as a differentiable approximation of the argmax operator, applied to the activation maps of the encoder.

Specifically, it takes the N activation maps M_n with $n = 1, \ldots, N$ of the encoder's last convolutional layer as its input and performs a 2D softmax normalisation on each

map, resulting in a 2D probability mass function (PMF) S_n. The expectation of each softmax-normalized map yields a 2D coordinate $[x_n \quad y_n]^\top$, computed as the weighted mean of pixel coordinates [34]

$$z_n = \begin{bmatrix} x_n & y_n \end{bmatrix}^\top = \begin{bmatrix} \sum_{h,w} h \cdot s_{hwn} & \sum_{h,w} w \cdot s_{hwn} \end{bmatrix}^\top \tag{2.22}$$

$$\text{with} \quad s_{hwn} = \frac{\exp(\frac{m_{hwn}}{\alpha})}{\sum_{h',w'} \exp(\frac{m_{h'w'n}}{\alpha})}. \tag{2.23}$$

Here, $h = 1, \ldots, H$ and $w = 1, \ldots, W$ represent the spatial indices of the height and width for an entry m_{hwn} in the n^{th} activation map M_n. The corresponding entry in the softmax-normalised map S_n is denoted as s_{hwn}. The spatial softmax layer ensures that the latent encoding can be interpreted as 2D coordinates of important features in the input image. The number of filters N in the final convolutional layer determines the number of resulting 2D coordinates, forming the latent representation $z = [z_1 \quad z_2 \quad \ldots \quad z_N] \in \mathbb{R}^{C \times 2}$. This process is illustrated in Figure 2.3.

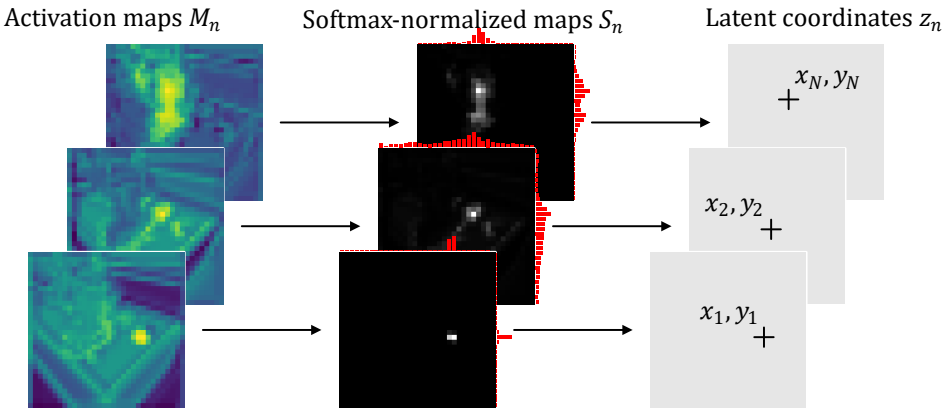


Figure 2.3.: The spatial soft-argmax layer takes the activation maps M_n of the preceding convolutional layer as input. It then applies a 2D softmax-normalisation to each map to obtain a PMF S_n before taking its expectation, resulting in a 2D weighted mean pixel coordinate [34].

With only 2D coordinates passing through the bottleneck, the decoder must learn both object appearance and spatial positioning based on z. The SAE is trained using the reconstruction loss introduced in Equation (2.21). To accelerate training, the target image is often downscaled to a lower resolution $\tilde{I}$ compared to the original input I [38].

2.4. Serial Manipulators

A manipulator can be represented as a kinematic chain of rigid bodies, so-called links, that are connected through joints. In this section, we first give a representation of the manipulator by defining both its task space configuration and its joint space configuration. Next, we describe the relation between both via forward and inverse kinematics. Then, we give the equations of motions for this system, which define the dynamics of the manipulator. Lastly, we introduce the general concept of impedance control, which is used as the low-level control approach in Chapter 5.

Robot Configuration

A manipulator consists of a series of links connected by joints. Joints can be either revolute or prismatic; in the following, we focus on revolute joints. Usually, an end-effector is attached to the last link. The mechanical structure of this manipulator is characterised by a number of degrees of freedom (DOFs). Each DOF is typically associated with a joint variable. The position and orientation of a rigid body in space are fundamental elements for describing its pose relative to a reference frame. The term pose encompasses both the spatial location and the angular orientation of the body, providing a complete representation of its configuration.

Consider an open-chain robotic manipulator consisting of $n+1$ rigid links connected by n revolute joints. By convention, the first link, referred to as link 0, is fixed to a base, establishing a fixed reference frame. This arrangement gives the manipulator n DOFs. The joints, numbered sequentially from 1 to n, are positioned between adjacent links, with joint i located between link $i-1$ and link i.

To describe the configuration of the manipulator, either joint space or task space can be used. In joint space, the configuration is expressed as $q = (q_1, q_2, \ldots, q_n) \in \mathbb{R}^n$, where q_i represents the angular position of joint i. The angular velocities and accelerations are denoted by $\dot{q}$ and $\ddot{q}$, respectively.

Task space provides a representation of the robot's end-effector pose. The position of the end-effector with regard to the base frame b in Cartesian coordinates can be described as

$$x^b = (x_x, x_y, x_z) \in \mathbb{R}^3. \tag{2.24}$$

To simplify notation, we omit the index b whenever we refer to the base frame and simply write $x^b = x$. The orientation of the end-effector can be represented in multiple ways. One common method involves the use of ZYX (Roll-Pitch-Yaw) Euler angles, denoted

as

$$\phi^{\text{Euler}} = (\phi_x, \phi_y, \phi_z), \tag{2.25}$$

which specify the rotations about the base frame axes x, y and z. Alternatively, orientation can be expressed using unit quaternions,

$$\phi^{\text{quat}} = (\phi_1, \phi_2, \phi_3, \phi_4), \tag{2.26}$$

which offer a singularity-free and computationally efficient representation of rotation. The complete pose of the end-effector is thus described as $p = (x, \phi^{\text{Euler}})$ when Euler angles are used, or $p = (x, \phi^{\text{quat}})$ when quaternions are preferred. Geometrically, ϕ^{quat} is a point lying on the three-dimensional unit sphere in $\mathbb{R}^4$. Consequently, unit quaternions are also identified with the 3-sphere, denoted S^3.

Direct Kinematics

The direct (or forward) kinematics problem for a serial-chain manipulator is to find the position and orientation of the end-effector relative to the base as a function of the joint variables [69].

The transformation of a vector from frame 1 of reference to frame 2 can be expressed by the homogenous transformation matrix A_1^0 containing the rotation matrix of frame 1 with respect to frame 0 and the translation vector from the origin of frame 0 to the origin of frame 1 [70]. Defining $\tilde{p} = (p, 1)$, the coordinate transformation from p^1 to p^0 can then be expressed as

$$\tilde{p}^0 = A_1^0 \cdot \tilde{p}^1. \tag{2.27}$$

The coordinate transformation describing the position and orientation of frame n with respect to frame 0 is then given by

$$T_n^0(q) = A_1^0(q_1)A_2^1(q_2)\cdots A_n^{n-1}(q_n). \tag{2.28}$$

Using this, the coordinate transformation describing the position and orientation of the end-effector frame e with respect to the base frame b can be obtained as

$$T_e^b(q) = T_0^b T_n^0(q) T_e^n, \tag{2.29}$$

where T_0^b and T_e^n describe the position and orientation of frame 0 with respect to the base frame and of the end-effector frame with respect to frame n, respectively.

Inverse Kinematics

The inverse kinematics problem for a serial-chain manipulator involves determining the values of the joint variables given the position and orientation of the end-effector [69]. While the forward kinematics, which map joint variables to the pose of the end-effector, can be easily computed once the joint variables are known (see Equation (2.29)), solving the inverse kinematics problem is significantly more complex [70]. Solving this problem requires the solution of a system of nonlinear equations. Depending on the manipulator's configuration and constraints, the equations may yield no solution or multiple solutions. Even when a unique solution exists, it is often not obtainable in closed form and must instead be computed using numerical methods.

Differential Kinematics

While forward and inverse kinematics describe the relation between joint variables and end-effector pose, differential kinematics present the relationship between joint velocities and end-effector velocity. This mapping is described by the Jacobian matrix $J(q)$, which depends on the robot configuration q.

The differential kinematics equation describes the end-effector velocity v_e in the base frame as

$$v_e = J(q)\dot{q}. \tag{2.30}$$

The inverse differential kinematics can be obtained via

$$\dot{q} = J^{-1}(q)v_e. \tag{2.31}$$

When a manipulator is redundant, i.e., it has more joints than the dimension of the task space, the Jacobian matrix has more columns than rows and cannot be inverted. A viable solution is to formulate the problem as a constrained linear optimisation problem, which minimises the quadratic cost of joint velocities. With the assumption of an equally weighted cost function, the joint velocities can be obtained as

$$\dot{q} = J^{\dagger}v_e, \tag{2.32}$$

where the matrix

$$J^{\dagger} = J^T(JJ^T)^{-1} \tag{2.33}$$

is the right pseudo-inverse of J.

Serial Manipulator Dynamics

The dynamic model of a manipulator describes the relationship between the joint actuator torques and the motion of the manipulator. There are two methods for the derivation of the equations of motion, the Lagrange formulation and the Newton-Euler formulation [70]. The Lagrangian method derives the dynamics from energy principles, where the Lagrangian is defined as the difference between the system's total kinetic and potential energy.

The Newton-Euler formulation recursively describes the motion of each link in terms of a balance of forces and links acting on it. This formulation is computationally more efficient and is used in most physics engines, including MuJoCo [32].

For a serial manipulator with n degrees of freedom as defined above, the joint space dynamics model can be written as

$$M(q)\ddot{q} + C(q,\dot{q})\dot{q} + g(q) = \tau, \tag{2.34}$$

where $M(q)$ is the $n \times n$ symmetric, positive-definite joint space inertia matrix, $C(q,\dot{q}$ is an $n \times n$ matrix such that $C(q,\dot{q})\dot{q}$ is the vector of Coriolis and centrifugal terms and $g(q)$ is the vector of gravity terms. For ease of notation, the dependence on q is often omitted in these terms. The vector of actuation torques is denoted by τ. More terms can be added to this equation, as required, to account for dynamical effects such as friction. The effects of a force h_e exerted on the mechanism at the end-effector can be accounted for by adding the term $J^T h_e$ to the right side of Equation (2.34).

In analogue to the equations of motion in joint space, the task space formulation can be written as

$$M^{\text{eef}}(p)\ddot{p} + \mu(p,\dot{p}) + \rho(p) = f, \tag{2.35}$$

where p denotes the pose of the robot's end-effector; $\dot{p}$ is the velocity of the end-effector, and f is the force exerted on the end-effector. The operational-space inertia matrix is denoted by $M^{\text{eef}}(p)$, and $\mu(p,\dot{p})$ and $\rho(p)$ are vectors of velocity-product and gravity terms, respectively.

2.4.1. Force-Motion Control

Many robotic tasks involve the robot's end-effector manipulating objects or operating on surfaces, with common industrial applications including polishing, deburring, machining, and assembly. Effective control of the physical interaction between a robotic manipulator and its environment is crucial for the successful execution of these tasks.

Relying solely on motion control to manage such interactions is often insufficient. This approach can only succeed if the task is meticulously planned, which requires precise models of both the robot's kinematics and dynamics as well as the environment's geometry and mechanical properties [69]. While robot manipulators can typically be modelled with reasonable accuracy, obtaining a detailed and precise representation of the environment is significantly more challenging.

To address this limitation, it is essential to ensure compliant behaviour during interactions. Compliance is critical for enabling robotic systems to operate effectively in dynamic and unpredictable environments. It enhances a robot's ability to adapt to unforeseen circumstances, improves robustness, and facilitates safer and more intuitive human-robot interaction.

Compliance can be achieved either passively or actively. In passive interaction control, compliance arises from the structural properties of the robot, such as elastic joints or flexible links, which are specifically designed for this purpose. Active compliance, on the other hand, is achieved through control strategies. Active interaction control strategies can be categorised into two main types: indirect force control and direct force control. Indirect force control ensures compliant behaviour without explicitly closing the force feedback loop.

In the following sections, we focus on impedance control, a method of indirect force control.

Impedance Control

Impedance control [71] is a widely used framework in robotic manipulation that models the interaction between the robot and its environment as a mass–spring–damper system with adjustable stiffness and damping parameters. The nature of this control scheme depends on the robot's response to external disturbances. When the system reacts to motion deviations by generating corrective forces, the control is categorised as impedance. Conversely, when the system responds to interaction forces by adjusting the motion trajectory, it operates as admittance control. This duality enables impedance control to bridge the gap between achieving precise motion tracking and ensuring safe, compliant interactions in dynamic environments.

We denote the pose error between the target and the current pose at time t as $\Delta p = p^{\text{ref}} - p$. End-effector velocity and acceleration are defined as $\dot{p}$ and $\ddot{p}$ and their errors as $\Delta\dot{p}$ and $\Delta\ddot{p}$ respectively.

The goal of impedance control is to model the behaviour of a mass-spring-damper

system in task space such that

$$M^{\text{eef}}\Delta\ddot{p} + D\Delta\dot{p} + K\Delta p = f_{\text{ext}}. \tag{2.36}$$

Here M^{eef}, D and $K \in \mathbb{R}^{6\times6}$ represent the Cartesian inertia, damping and stiffness matrices, and f_{ext} is a force applied to the robot.

It is important to highlight that selecting appropriate impedance parameters is a challenging task [69]. This difficulty arises from the fact that the closed-loop system exhibits different dynamics depending on whether the robot operates in free space or is in physical interaction with its environment. Additionally, the dynamics of the system during the interaction are influenced not only by the controller but also by the properties and behaviour of the environment itself.

3. A Metric for Keypoint Detection in Reinforcement Learning SAE

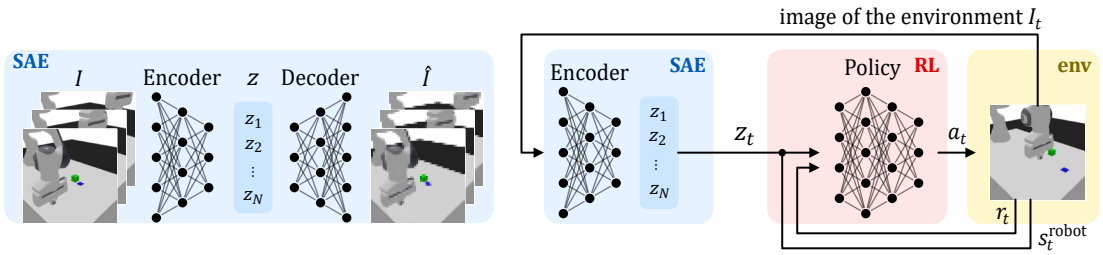
As robotic production systems increasingly operate in complex, uncertain, and unstructured environments, finding a suitable representation of the environment is critical for enabling the effective application of RL. This chapter focuses on unsupervised state space extraction, investigating how SAEs can be leveraged to detect and track task-relevant objects in RL environments.

By extracting a structured state space representation from raw sensory data, SAEs provide a way to infer missing information, thereby reducing environment uncertainty in unstructured environments where not all relevant state variables can be directly measured (G3). To assess and improve their application in RL, this chapter introduces a metric for evaluating the performance of SAEs in detecting task-relevant objects in contribution C1 (Section 3.5). By quantifying the tracking accuracy of SAEs, the chapter addresses key challenges in using SAEs as part of a larger RL pipeline.

The proposed metric is applied to assess various SAE architectures and their ability to track object positions (Section 3.6.1), which is shown to correlate with downstream RL performance in contribution C2 (Section 3.6.2). A key advantage of this metric is that it enables evaluation of the extracted state space before the actual RL training takes place, providing an early assessment of representation quality. This helps lower data requirements by reducing the need for extensive RL experiments to determine whether a given state representation is suitable (G1). We find significant differences in SAE performance and propose architectural improvements that enhance tracking capability.

The developed metric offers a physically interpretable measure of the SAE performance. This interpretability makes the evaluation process more transparent and improves usability by allowing practitioners to better understand and refine their learned representations before deploying them in RL applications (G5, G6).

Together, these contributions provide insights into the relationship between state representation quality and RL performance, offering a step toward the application of RL in unstructured production environments. This chapter is based on the paper [72].



(a) SAE trained on reconstruction loss. (b) Policy trained to maximise expected return.

Figure 3.1.: SAE extracts 2D positions from images via a spatial bottleneck (Figure a). The SAE encoder is then integrated into an RL framework to obtain a state representation for immeasurable objects (Figure b).

3.1. Introduction

In real-world control tasks like robotics, successful RL often hinges on a comprehensive state representation. This necessitates including all task-relevant objects in the scene. This issue is particularly prominent in tasks involving unstructured environments or interactions with numerous objects, where defining the state space without significant prior knowledge is difficult. Image data provides a potential solution, either through direct end-to-end learning of the control signal or by first learning a low-dimensional representation of the high-dimensional data [37, 34]. For practical applications, interpretability in terms of physical quantities is usually advantageous. SAEs have been effective in learning low-dimensional representations, expressed as 2D points on the image plane, referred to as keypoints. This latent representation can be used, e.g., as part of the state representation of the RL agent. Figure 3.1 shows the complete learning pipeline.

This chapter proposes a straightforward metric for quantitatively evaluating the extraction of positional information of task-relevant objects in the latent space of SAEs. The proposed metric is applied to (i) train multiple base SAE architectures and compare their tracking performance, (ii) explore various improvements of these architectures, and (iii) learn a suitable RL task using keypoints from different SAEs as the state. The evaluation reveals significant variations in the spatial extraction capability of common SAEs, emphasising the importance of a thorough evaluation before incorporating them into RL states. Building on these insights, we propose three key modifications to substantially improve the tracking performance of common SAE architectures, resulting in, e.g., a 30 % increase in tracking capability for the commonly used KeyNet architecture [39]. We demonstrate that the metric allows us to judge SAEs with regard to capturing physically interpretable positional features and that this metric is a good indicator of

downstream RL performance. For the considered robotic manipulation task, SAE training takes approximately an order of magnitude less computational resources than RL training, making our metric a practical and lightweight indicator of RL performance before expensive RL training.

3.2. Related Work

Various approaches utilise deep NNs to extract state representations from images or videos [73, 74]; we review the ones most related to this work.

Unsupervised State Representation Learning for Reinforcement Learning

Applications span from general continuous control [73, 75] to robotic manipulation [76, 77]. Many models follow an autoencoder structure with a low-dimensional bottleneck, optimising for input reconstruction [38, 78]. Some of these learn world models and recurrently capture environment dynamics in the latent representation [79, 74, 80]. Generally, autoencoders constrain the dimension, but not *what* is captured in the latent space [78, 77]. In contrast, SAEs are constrained to capture 2D keypoint positions [38].

Spatial Autoencoder Architectures

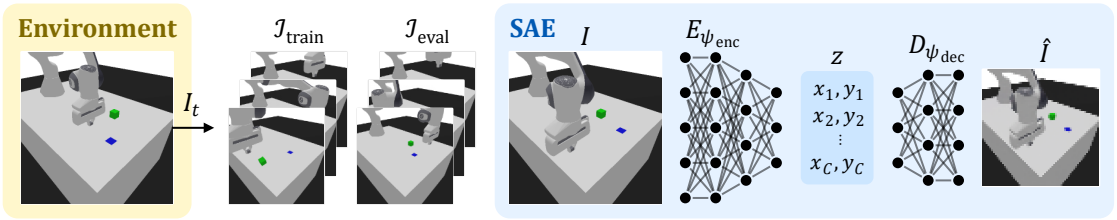
SAE keypoints have successfully been used as RL state representations in robotic control [81, 42], to play Atari games [40], or to provide a goal description [82]. Central to SAEs is the spatial soft-argmax layer first proposed by Levine *et al.* [34] to train an end-to-end deep visuomotor policy. Finn *et al.* [38] modified this approach to obtain a standalone deep SAE architecture consisting of a convolutional encoder and a fully connected decoder. Jakab *et al.* [39] propose the KeyNet architecture, incorporating a convolutional decoder. These two elements, the encoder-decoder structure and the spatial soft-argmax layer, are essential to all SAEs. Many architectures build upon these blocks; incorporating feature transport mechanisms [40], working on error maps [83], and reconstructing segmentation masks [81] or frame differences [84]. Recently, SAEs have been extended to learn 3D points [85, 86]. While these approaches differ in how they are trained, all aim to represent positional information. We focus our investigation on two of the most common base architectures [38, 39] as (i) they form the basis for many more complex architectures and (ii) we found that if trained correctly, they can serve as reliable feature extractors. In principle, our evaluation procedure can be applied to all of the above architectures.

Evaluation of Spatial Autoencoders

Typically, SAEs are evaluated indirectly through compute-intensive RL or control performance [82, 87, 88] or qualitative visual assessments [89, 81]. In general, latent representations can be evaluated via reconstruction loss [38], disentanglement measurements [90] or mutual information estimates [91], all of which neglect the spatial 2D keypoint structure and thus do not assess the physical meaningfulness of the features. In the computer vision domain, keypoints for image matching are evaluated by reprojecting from different views with known camera transformation [92], which is not applicable for SAEs. Jakab *et al.* [39] approximate labelled ground truth points as linear combinations of all keypoints. Their KeyNet SAE is evaluated with the percentage of these predicted points within a fixed distance from the labels. The same linear combination has been used by others to compute mean errors to ground truth points [89, 93, 84]. Kulkarni *et al.* [40] match keypoints to ground truth points via a min-cost assignment and compute precision and recall over trajectories. Although quantitative, the above approaches cannot assess the quality of keypoints over trajectories and allow no statement about whether all task-relevant objects are represented. We find that both aspects are critical for use in control or RL, and our method, described in Section 3.5, addresses these key limitations in existing evaluation approaches.

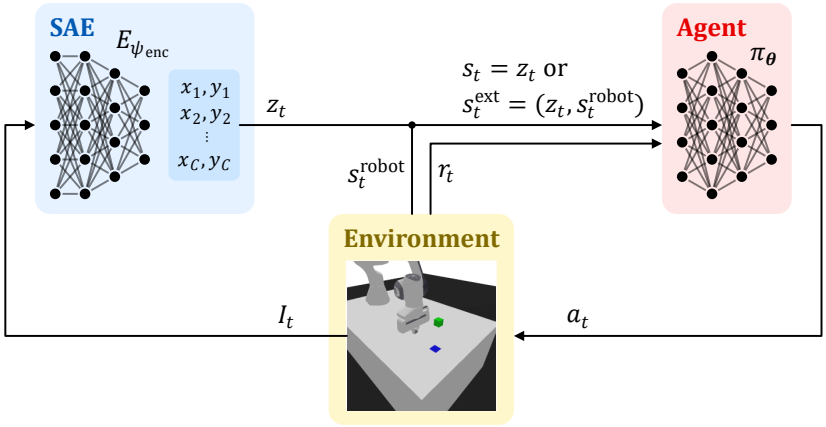
3.3. Reinforcement Learning with Spatial Autoencoders

To develop our SAE evaluation, we investigate a simulated robotic task wherein either the full or parts of the state come from image observations. When attempting to learn state representations from images, the ultimate goal is successfully solving a downstream RL task. This relies on a state s containing all task-relevant information, usually including the state of objects not directly measurable. In the considered framework, a camera is positioned next to the RL setup to capture the environment state visually. An SAE is pre-trained to derive a low-dimensional representation z of the environment state from high-dimensional image data I. An RL training run can be conducted using this representation. The complete framework is illustrated in Figure 3.2. Next, we describe the two main components, SAE pretraining and RL with latent state representations, in more detail.



(a) Dataset collection.

(b) Training of the SAE.



(c) RL training using the encoder part of the trained SAE.

Figure 3.2.: In this Figure, we show the complete learning framework. First, a video dataset is collected using a random policy (Figure a). An SAE is then trained on this dataset (Figure b), learning to capture 2D positions of moving objects in its latent space. Finally, the SAE's encoder is used to extract a state representation from live images in the RL process (Figure c).

3.3.1. Spatial Autoencoder Pre-Training

An initial image dataset $\mathcal{I}$ is recorded from the RL environment using a random policy. The collected video snippets are then split into separate training and evaluation sets $\mathcal{I}_{\text{train}}$ and $\mathcal{I}_{\text{eval}}$.

We use a smoothed random policy to ensure a diverse image dataset of environment interactions. In robotic scenarios, actions are typically in a continuous range and related to a change in position [94], for example, by controlling the end-effector displacement in every time step t.

Starting from a random initial action $a_0 \in \mathcal{A}$, smooth trajectories are achieved through small changes in the action's direction. We therefore sample the next action a_{t+1} from an isotropic Gaussian distribution with standard deviation σ centred on the previous action a_t via

$$a_{t+1} \sim \mathcal{N}(a_t, \sigma I) \quad \text{such that} \quad a_{t+1} \in \mathcal{A}. \tag{3.1}$$

From the training dataset $\mathcal{I}_{\text{train}}$, image batches are randomly sampled to train the encoder and decoder NNs $E_{\psi_{\text{enc}}}$ and $D_{\psi_{\text{dec}}}$ via the deep-learning-based optimisation techniques discussed in Section 2.1.2. The optimization objective as introduced in Section 2.3.1 is to reconstruct downscaled versions $\tilde{I}$ of the input images I as $\hat{I} = D_{\psi_{\text{dec}}}(E_{\psi_{\text{enc}}}(I)) \approx \tilde{I}$, minimizing the reconstruction loss via

$$\min_{\psi_{\text{enc}},\psi_{\text{dec}}} \mathbb{E}\left[L_{\text{reconstruction}}(D_{\psi_{\text{dec}}}(E_{\psi_{\text{enc}}}(I)), \tilde{I})\right]. \tag{3.2}$$

Through the SAE's spatial bottleneck, this leads to 2D positions being captured in the latent representation z that are supposed to locate non-stationary elements in the image plane. For the subsequent RL task, only the encoder is used, with its parameters fixed and no longer optimised.

3.3.2. Reinforcement Learning Training with Latent State Representations

With a trained SAE encoder $E_{\psi_{\text{enc}}}$, a state representation for the RL task can be formulated.

The objective of an RL task has been formulated in Section 2.2.1 as finding the optimal policy π^* that maximises the expected return. To find this policy, we choose the SAC algorithm [22] as introduced in Section 2.2.3.

In the resulting RL loop, an image I_t of the current environment configuration is passed through the SAE encoder $E_{\psi_{\text{enc}}}$ to obtain $z_t = E_{\psi_{\text{enc}}}(I_t)$. This latent image encoding z_t is then passed to the agent's policy $\pi(a_t \mid s_t)$. In our experimental results, we consider two possible state formulations s_t,

(i) using latent feature points only such that $s_t = z_t$, and

(ii) extending the state with robot state measurements such that $s_t^{\text{ext}} = (z_t, s_t^{\text{robot}})$.

3.3.3. Assumptions

For the framework outlined above to be applicable, several assumptions must be met. These are that

(i) the task is solvable using visual information since otherwise visually capturing all task-relevant aspects of the current environment state is not possible,

(ii) the camera is stationary, such that recurring 3D object positions are projected onto the same 2D position in the image plane, and

(iii) a reward signal supplying r_t is given, which can also be a sparse reward solely upon task completion.

3.4. Problem Formalization

We consider a setup with K rigid objects that shall be tracked. The projection of the 3D ground-truth position $o_k \in \mathbb{R}^3$ of object k, for example, given by its centre of mass, into the 2D image space is given by $x_k \in \mathbb{R}^2$ and referred to as a ground-truth point. The ground-truth points for all K objects are collectively denoted by $x = (x_1, \ldots, x_k) \in \mathbb{R}^{2\times K}$.

Further we consider a pretrained spatial autoencoder $I \xrightarrow{E_{\psi_{\text{enc}}}} z \xrightarrow{D_{\psi_{\text{dec}}}} \hat{I}$, operating on an input image I, which is mapped to a latent representation z via an NN encoder $E_{\psi_{\text{enc}}}$. The spatial softmax layer of the encoder ensures that the latent space z can be interpreted as N keypoints in the image plane with $z = (z_1, z_2, \ldots, z_N) \in \mathbb{R}^{2\times N}$.

An ideal SAE should track x with its latent representation z in *some sense*. However, how to evaluate the tracking performance is unclear, and proposing a method that quantifies this is our main objective:

Problem 1. *We seek to quantify how well the keypoints z represent the ground truth objects x.*

SAEs are often used as feature extractors for RL tasks, where keypoints z are then part of the state representation. The general understanding in literature [38, 41, 40, 87] is that well-tracking SAEs will yield better RL performance, such as higher expected return $J(\pi)$ or episode success rates. We investigate whether this holds true for the metric proposed for Problem 1:

Problem 2. *Is SAE performance (according to Problem 1) an indicator for RL performance?*

If this hypothesis holds true, SAE performance can be evaluated before actual RL training, usually at a significantly lower computational cost.

3.5. A Metric to Evaluate Keypoints

In this section, we propose a metric to quantify the tracking performance of an SAE, addressing Problem 1.

As RL makes decisions sequentially over time, we are interested in the object tracking performance over multiple time steps. Therefore, we denote by $x_{t,k} \in \mathbb{R}^2$ the ground truth position of the k-th object at time step t, and by $x_t \in \mathbb{R}^{2\times K}$ the positions of all K objects collectively. Furthermore, we denote the trajectory of these objects over time steps $t = 0, \ldots, T-1$ as

$$x_\tau = (x_0, x_1, \ldots, x_{T-1}) \in \mathbb{R}^{2\times K\times T}. \tag{3.3}$$

We use analogous notation for the latent keypoints; that is, we denote the trajectory of the N keypoints as

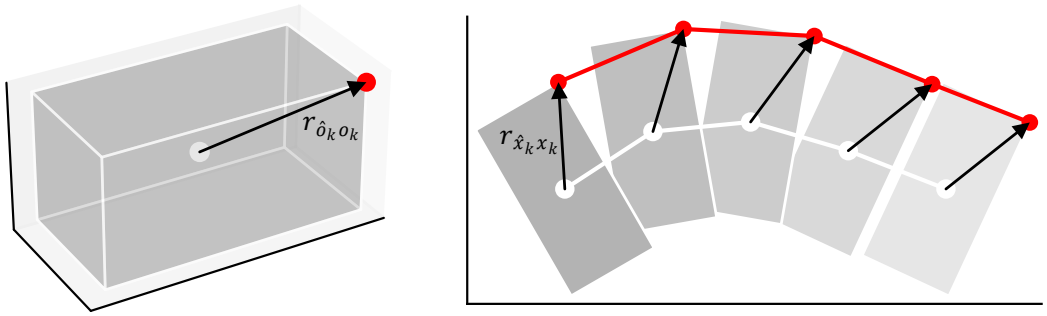
$$z_\tau = (z_0, z_1, \ldots, z_{T-1}) \in \mathbb{R}^{2\times N\times T}. \tag{3.4}$$

Given this notion of trajectories, Problem 1 translates to measuring how well the keypoint trajectory z_τ follows the ground truth trajectory x_τ for a given instance of an SAE. A naive approach would be to compute the Euclidean distance between point pairs along these trajectories directly. However, this will not yield satisfactory results. The keypoints are learned unsupervised, which provides no guarantee about which part of an object is tracked. As points on a rigid object have a fixed relation to each other, it is reasonable to assume that, for downstream RL training, any point on the object is an equally suitable representation. For example, if a ground truth point and a keypoint are on the same object at a constant offset, this offset would accumulate to a tracking error when naively taking the difference between the two points. Thus, we need to account for offsets by an appropriate transformation. Finally, the SAE extracts many keypoints (usually $N > K$), and the association of keypoints to ground truth points is unknown. Taking these together, an evaluation protocol of keypoints will thus require (i) accounting for the offset between any point on the object and ground truth, (ii) associating keypoints with ground truth points, and (iii) developing a quantitative measure to evaluate the capability of tracking all relevant ground truth points. The following subsections explain the three challenges in more detail and specify how our evaluation metric handles them.

Transformation

The natural choice for comparing a feature point trajectory $z_{\tau,n}$ to a ground-truth point trajectory $x_{\tau,k}$ is to consider the points' Euclidian distances. However, this naïve approach does not consider the unsupervised learning process and the 3D nature of the underlying environment. Keypoints are coordinates in the 2D image space, which are supposed to track objects in 3D space. Often, the centre of mass (CM) o_k is taken as

the ideal point to represent the 3D position of an object in the world frame. However, for the downstream RL task, the keypoints do not have to track the CM, but any fixed point on the object. Thus the point's offset from the CM should be constant in the object's 3D frame of reference, i.e., $\hat{o}_k = o_k + r_{\hat{o}_k o_k}$ where $r_{\hat{o}_k o_k}$ is constant. Such an offset is illustrated in Figure 3.3a.



(a) Constant 3D offset $r_{\hat{o}_k o_k}$.

(b) Varying 2D offset $r_{\hat{x}_k x_k}$ in time.

Figure 3.3.: Figure a shows an exemplary 3D offset from the CM and Figure b illustrates the resulting time-varying 2D offset for a moving object. If any fixed point of an object is tracked, the feature point trajectory $z_{\tau,n}$ that coincides with $\hat{x}_{\tau,k}$ (red) is considered to be equally informative as the ground-truth point trajectory $x_{\tau,k}$ of the CM (white). However, this constant 3D offset $r_{\hat{o}_k o_k}$ results in a time-variant 2D offset $r_{\hat{x}_k x_k}$, yielding a large tracking error.

For a moving object, this constant offset from the object's CM in its 3D reference frame induces an offset between the 2D projections x_k and $\hat{x}_k$. However, due to object motion and camera projection, the 2D offset $r_{\hat{x}_k x_k}$ is time-variant and nonlinearly related to the 3D offset $r_{\hat{o}_k o_k}$. Figure 3.3b shows such a 2D offset between trajectories $x_{\tau,k}$ and $\hat{x}_{\tau,k}$.

This 2D offset is generally unknown; it depends on the unknown 3D offset, object position, orientation, and camera view. This means even if keypoints were to track a fixed point on an object perfectly, this offset would falsely suggest a tracking error in 2D.

Instead of capturing the complete geometry of the problem, which requires additional problem insight, we propose a lightweight approach that eliminates the main offsets between keypoints and ground truth. We propose a time-invariant affine transformation of keypoints, such that the transformed keypoint $\hat{z}_n$ can be expressed as

$$\hat{z}_n = Az_n + b, \tag{3.5}$$

where $A \in \mathbb{R}^{2\times 2}$ describes the scaling, rotation, and skew and $b \in \mathbb{R}^2$ describes the

translation of feature point z_n in the 2D image plane. The parameters A and b are determined for every pair (n, k) of feature points z_n and ground-truth points x_k. They are fit via an ordinary least squares (LS) optimization [95] such that

$$\begin{aligned} A_{n,k}, b_{n,k} &= \arg\min_{A,b} \sum_{m=1}^{M} \|\hat{z}_{m,n} - x_{m,k}\|_2^2 \\ &= \arg\min_{A,b} \sum_{m=1}^{M} \|Az_{m,n} + b - x_{m,k}\|_2^2, \end{aligned} \tag{3.6}$$

where m indexes feature points over M time steps from multiple episodes. We use a held-out test set for this parameter fit, containing random time steps from trajectories unseen in SAE training.

We note that this comparatively simple and time-invariant linear transformation can generally not cancel the mentioned time-variant 2D offset exactly and represents only an approximation. Even more straightforward transformations can be found by reducing the transformation to translation and scaling or even to translation only. A pure translation is given by $\hat{z}_n = z_n + b$ while additional scaling gives $\hat{z}_n = \hat{A}z_n + b$ allowing nonzero entries in $\hat{A}$ only on the diagonal. However, these transformations reach obvious limitations even in a two-dimensional setting, as changing offsets due to basic rotary object motion cannot be eliminated. The affine transformation with dense A can rotate trajectories $z_{\tau,n}$ and deal with such rotations around a fixed axis. On the other hand, a more general nonlinear mapping $\hat{z}_n = f(z_n)$ would be possible to align feature points z_n with ground-truth points x_k. However, depending on the method used, such a transformation may be prone to overfitting or may not have a closed-form solution anymore. In our evaluations, we find that the chosen affine transformation is easy to compute, requires no additional information about the ground truth objects, and works well in practice.

Association and Tracking Error

Using the affine transformation introduced above, we consider the trajectories x_τ, $\hat{z}_\tau$ of K ground truth objects and N transformed keypoints. We propose an error metric between the trajectory of one ground truth object $x_{\tau,k}$ and the transformed trajectory of one keypoint $\hat{z}_{\tau,n}$. We define the tracking error $e_{n,k}$ between the trajectories of keypoint n and ground truth point k as

$$e_{n,k} = \sum_{t=1}^{T} \|\hat{z}_{t,n} - x_{t,k}\|_2^2. \tag{3.7}$$

The error $e_{n,k}$ measures how well a specific keypoint tracks a ground truth object over time.

Using the tracking error, we determine the index of the keypoint $z_{n_k^*}$ that best tracks object x_k as

$$n_k^* = \arg\min_n e_{n,k}. \tag{3.8}$$

Once we assign the most suitable keypoint for each ground truth object, we give the tracking error of the associated keypoint as $e_{n_k^*,k}$. For our evaluation, we always consider the tracking error of the best keypoint. The lower this tracking error, the better the ground truth point x_k is represented by the keypoint $z_{n_k^*}$. The error measure enables a comparison of different SAE architectures and individual training runs of the same architecture broken down into objects.

For the later evaluation of SAEs, we now define indicators for an SAE's overall tracking performance. As will be seen in Section 3.6.1, the object tracking error tends to jump from around the initial value to close to zero at some point in the SAE training process. This represents the object at first not being tracked and then suddenly being tracked. Using this knowledge, we classify an object x_k as correctly tracked if the tracking error of the most suitable keypoint $z_{n_k^*}$ is below an application-specific threshold $\mu_k > 0$. The index set $\mathcal{X}_c$ of all correctly tracked objects is given by $\mathcal{X}_c = \{k : e_{n_k^*,k} \leq \mu_k\}$. We then define the tracking capability TC of one trained SAE as the percentage of tracked ground truth objects, i.e.,

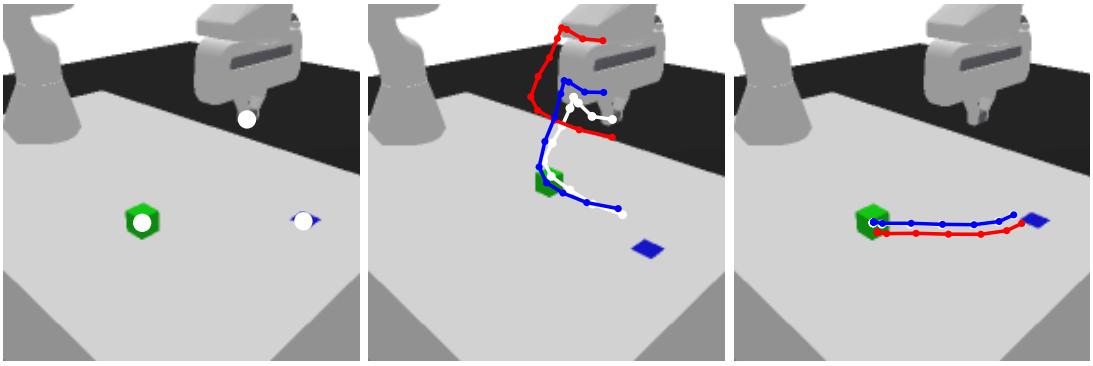
$$\mathrm{TC} = |\mathcal{X}_c| / K. \tag{3.9}$$

An ideal tracking capability of $\mathrm{TC} = 1$ means that for a specific SAE, the position of all ground truth objects is correctly encoded in the latent space.

A quantitative evaluation should consider the distribution of the tracking error and the tracking capability over multiple training runs. We look at the tracking error's mean, median, and variance over multiple runs. Similarly, we evaluate the mean tracking capability $\overline{\mathrm{TC}}$ over multiple runs. Intuitively, $\overline{\mathrm{TC}}$ gives the mean percentage of all ground truth objects captured by keypoints. An SAE with a high mean tracking capability is a reliable feature extractor for RL scenarios. For individual ground truth objects, we denote the mean tracking capability for object k as $\overline{\mathrm{TC}}_k$.

3.6. Evaluation

We first use our proposed metrics (3.7) and (3.9) to evaluate the tracking performance of base SAE architectures commonly used in RL and propose architecture modifications to



(a) (b) (c)

Figure 3.4.: Figure a shows the PandaPush-v3 task with the positions x_k of the three objects of interest marked. We show selected ground truth, keypoint, and transformed keypoint trajectories in white, red, and blue for the end effector (Figure b) and cube (Figure c).

improve tracking. We then investigate how tracking performance links to performance in a downstream RL task. The empirical results reveal the following main insights:

1. The proposed metric is able to quantify the tracking performance of SAEs.
2. The combination of the best baseline SAE with our proposed modification yields $\overline{\text{TC}} = 0.99$, and it can thus be considered a reliable and precise spatial feature extractor.
3. Our proposed metric for SAE tracking performance is indicative of the performance of RL; that is, the architecture with the best SAE metric also achieves the best asymptotic return.
4. The best-found architecture in terms of SAE tracking achieves an RL return comparable to training with ground truth points.

3.6.1. Spatial Autoencoder Evaluation

We demonstrate the suitability of the tracking error and tracking capability introduced in Section 3.5 to evaluate the performance of SAEs. We visualise our quantitative results at youtu.be/8KqFXQiWa9w.

Spatial Autoencoder Experiment Setup

We use the PandaPush-v3 environment from the panda-gym [94] simulation. The robot's task is to push a cube toward a target (cf. Figure 3.4). We identify three task-relevant objects in this environment, (i) the green *cube* to be moved, (ii) the blue square indi-

cating the *target*, and (iii) the tip of the *end effector*. The objects' different sizes and motion behaviour make them a suitable selection for evaluating tracking performance. We consider three standard SAE architectures and our own combination of modifications:

Basic: We design the Basic architecture to be a simple and efficient SAE baseline incorporating the key components that all SAEs typically share. The CNN encoder has six convolutional layers and max-pooling operations in between. The decoder uses KeyNet's Gaussian kernel maps, followed by three convolutional layers. We look at two versions of this SAE with $N = 16$ (Basic) and $N = 32$ keypoints (Basic-kp32).

DSAE [38]: DSAE introduced the spatial soft-argmax bottleneck, still used in many other architectures [89, 96, 83, 81, 88]. This was the first SAE to be successfully used for RL training. $N = 16$ keypoints are captured between a CNN encoder and a fully connected decoder.

KeyNet [39]: KeyNet is a widely used and built upon SAE architecture [40, 97, 83], consisting of a CNN encoder and decoder with $N = 30$ keypoints. Input to the decoder are N feature maps with isotropic Gaussian kernels at the corresponding keypoint locations.

Vel-std-bg modifications: We propose a set of modifications to the above architectures, combining ideas from existing and new works. Analogously to DSAE, we add a velocity loss term to the reconstruction loss with a weighting factor β. The velocity loss encourages temporal consistency by penalising a change of keypoint velocities in subsequent frame pairs. KeyNet uses Gaussian heatmaps as input to the first CNN decoder layer. We propose making the standard deviation σ of these heatmaps trainable. This enables the decoder to control the radius of influence of a keypoint. Finally, we add a bias with the dimensions of the target image to the decoder's output, giving the decoder a straightforward way to reconstruct a stationary background and allowing time-varying keypoints to focus on moving objects. For the modified architectures, we call the combinations of the KeyNet or Basic architecture combined with our proposed modifications KeyNet-vel-std-bg and Basic-vel-std-bg, respectively.

While many more architectures exist in the literature (cf. Section 3.2), we deliberately choose baseline architectures maintaining the usual autoencoder setup without auxiliary networks such as adversaries or feature transport mechanisms. We choose modifications that we believe are beneficial for the primary goal of SAEs, spatial tracking of keypoints over time. For SAE tracking evaluation, we conduct 24 training runs with different random seeds. The tracking thresholds need to be chosen heuristically. Here we choose $\mu_{\text{cube}} = \mu_{\text{target}} = 0.015$ and $\mu_{\text{eef}} = 0.1$. Intuitively larger objects result in a larger

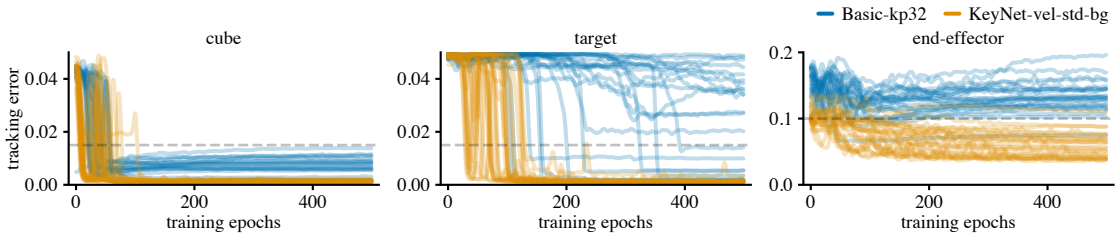


Figure 3.5.: Basic-kp32 and KeyNet-vel-std-bg tracking errors $e_{n_k^*,k}$ for $K = 3$ objects over epochs.

tracking error due to the possible offset to the centre of mass. We find a good heuristic to be related to the SAE reconstruction. Objects appear in the reconstruction when the tracking error falls below μ_k. Additional modifications, an ablation study, and all experimental details, such as hyperparameters, can be found in the appendix.

Evaluating Accuracy via the Tracking Error

First, we study the tracking error of individual runs during training. We observe a sudden drop in tracking error whenever the SAE has learned to track an object. To understand this behaviour, we look at the tracking error over episodes for an SAE with medium performance, the Basic-kp32, in Figure 3.5. All runs for Basic-kp32 show the drop in tracking error for the cube, which is the easiest to track. For the target, which is slightly harder to track due to its smaller size and rare movement, only a few runs show the expected drop below our threshold, resulting in a correctly tracked target. Instead of a sharp drop, the tracking error for the end-effector shows a shallow decrease over training epochs. We interpret this observation as follows: The end-effector occupies considerably more pixels in the image than the cube and target. Thus, the reconstruction first focuses on these areas, resulting in early vague tracking and reconstruction. However, consistently tracking a point on the end-effector is achieved only by a few runs. Looking at the tracking error of KeyNet-vel-std-bg, Figure 3.5 shows a distinct drop below the threshold for the cube and target. Even for the end-effector, the tracking error consistently falls below the threshold, indicating successful tracking. We find that the tracking error is useful in examining exactly how accurately a trained SAE architecture instance can track individual objects.

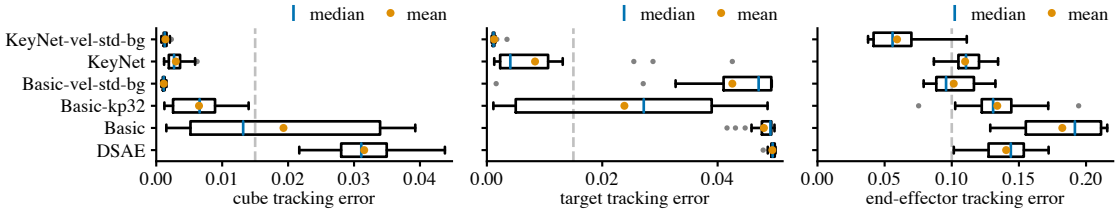


Figure 3.6.: Box plots of the tracking error $e_{n_k^*,k}$ for $K = 3$ ground truth objects.

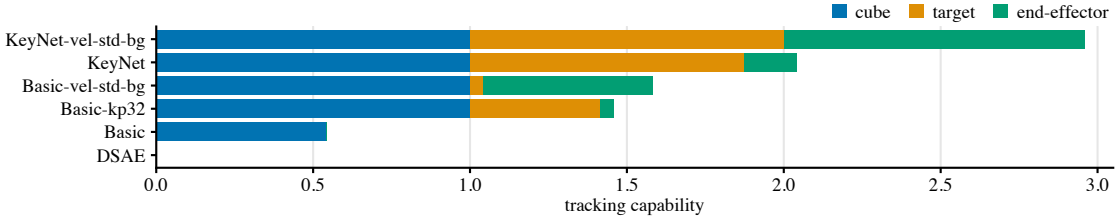


Figure 3.7.: Tracking capabilities $\overline{\mathrm{TC}_k}$ for $k = 3$ ground truth objects of the baseline architectures.

Evaluating Reliability via the Tracking Error

Our results indicate differing tracking performance for random seeds within the same architecture, showing that the SAE architectures need to be evaluated over multiple training runs. The tracking error's distribution over 24 training runs is illustrated in Figure 3.6. We remark that the tracking error varies among (i) architectures, (ii) random seeds, and (iii) objects. Among the standard architectures, KeyNet attains the lowest mean tracking error and smallest variance, indicative of good overall tracking performance. For DSAE and Basic, larger tracking errors with greater variance are observed, marking them less reliable. The KeyNet-vel-std-bg architecture shows the lowest mean tracking error and variance for all three objects. We identify the criteria for well-performing architectures as low mean tracking error and small variance over runs.

Evaluating Overall Performance via the Tracking Capability

Figure 3.7 shows the sum of mean object tracking capabilities $\overline{\mathrm{TC}_k}$ over architectures, further demonstrating the varying tracking performance across SAE architectures. Examining the tracking capability with regard to the individual ground truth objects, we further substantiate our hypothesis that the target and end-effector are more difficult to track than the cube. The tracking performance of all base architectures has potential for improvement as none is close to the theoretical maximum of 3.0. The combination vel-std-bg yields consistent improvement in tracking capability. KeyNet already tracks

cube and target well and has a $\overline{\text{TC}} = 0.681$. KeyNet-vel-std-bg has a near-perfect mean tracking capability of $\overline{\text{TC}} = 0.986$. The biggest change can be seen in end effector tracking, which improved from $\overline{\text{TC}}_{\text{eef}} = 0.167$ to 0.958. We see that the tracking capability is a compact description of how well task-relevant objects are tracked. This information is critical for downstream control and RL tasks.

Combining the insights from the tracking error and tracking capability answers Problem 1.

3.6.2. Reinforcement Learning Evaluation

We run RL experiments with SAE architectures selected by their tracking performance and find that this is a good indicator of downstream RL performance.

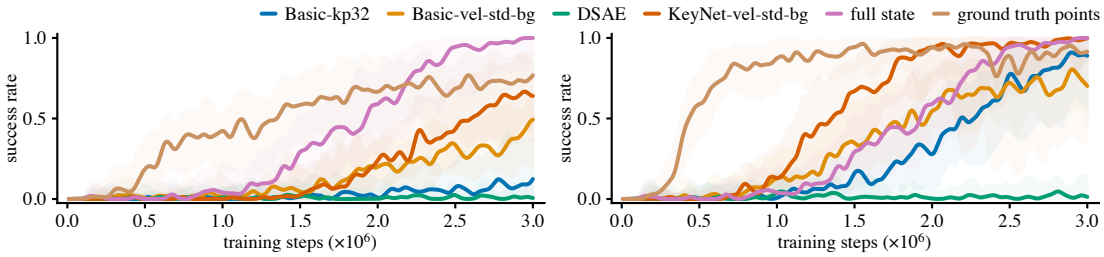
Reinforcement Learning Experiment Setup

For RL experiments with SAEs as state, we randomly sampled five trained SAEs per architecture and conducted two randomly seeded RL training runs with each of them, yielding a total of 10 runs per SAE architecture. We use the SAC [22] implementation from stable-baselines3 [98]. The complete list of hyperparameters can be found in Appendix A.1

We consider two types of state representation for RL with SAE-encoded keypoints: (i) latent keypoints as state $s_t = z_t$, (ii) latent keypoints combined with robot 3D position o^{eef} and velocity $\dot{o}^{\text{eef}}$, giving $s_t^{\text{ext}} = (z_t, o_t^{\text{eef}}, \dot{o}_t^{\text{eef}})$. The second scenario is relevant since end-effector position and velocity are often available as robot state measurements. As additional benchmarks, we include state representations with ground truth points x_t, which are usually not available in practice, obtaining $s_t = x_t$ and $s_t^{\text{ext}} = (x_t, o_t^{\text{eef}}, \dot{o}_t^{\text{eef}})$. Finally, we compare to RL runs with the full 3D simulation state, including positions, velocity, and orientation of the cube and target. Actions consist of 3D displacements $a_t = (\Delta o^{\text{eef}})_t$ of the end effector at every time step. We use a sparse reward with $r_t = -1$ and $r_T = 0$ on episode success. Following [99], we report interquartile mean (IQM) success rates with bootstrapped 95 % percentile confidence intervals.

Reinforcement Learning with Keypoints

Figure 3.8a shows the RL performance with state representation s_t. We observe varying success rates depending on the SAE architecture, and the gradations in RL performance follow the order of SAEs by tracking capability, as seen in Figure 3.7. The DSAE architecture shows no RL progress. Although Basic-vel-std-bg and Basic-kp32 have similar



(a) state s_t (b) state s_t^{ext}

Figure 3.8.: Success rate for different states: keypoints only in Figure a and extended state in Figure b.

total tracking capabilities (cf. Figure 3.7), the former performs better on the RL task. This is due to its ability to track the end-effector reasonably well, while Basic-kp32 tracks the target instead. End-effector tracking is critical, as moving the cube is otherwise impossible. The best-tracking KeyNet-vel-std-bg dominates the RL with learned keypoints. Still, this architecture does not achieve full-state performance. This is to be expected since the representation is limited to 2D space and lacks velocity information. The runs using 2D ground truth points, mimicking a perfect SAE, learn significantly earlier than KeyNet-vel-std-bg but only achieve a slightly higher final success rate.

Figure 3.8b shows the RL runs using state s_t^{ext}, i.e., including the end-effector's 3D position and velocity in addition to keypoints. RL performance with KeyNet-vel-std-bg and the full state show comparable final success rates close to 1.0, indicating that 2D keypoints are a useful state representation. KeyNet-vel-std-bg additionally learns notably faster. We presume that the reduced 2D representations of target and cube positions accelerate RL training. As expected with DSAE, tracking neither target nor cube learning progress is impossible. Compared to the first experiment setup, Basic-kp32 and Basic-vel-std-bg switch positions in the final RL performance. Although Basic-vel-std-bg improves with the more precise 3D end-effector position, it is still unable to track the target and, therefore, is limited in performance. For Basic-kp32, the missing end-effector tracking is now compensated with ground truth 3D information. Using its notable target tracking capability, it achieves better final performance. Initially, Basic-vel-std-bg learns faster, supporting the assumption that 2D representations can accelerate RL training. These kinds of insights are facilitated by the tracking capability and would not have been possible via traditional SAE evaluation. The IQM success rates for runs with ground truth instead of keypoints show faster learning but do not quite reach the maximum of full state and KeyNet-vel-std-bg.

Answering Problem 2, we find a link between SAE tracking capability, including the tracking capability for individual objects, and downstream RL performance.

3.7. Conclusion

In this chapter, we propose a metric to evaluate SAE performance with respect to task-relevant objects. Employing this metric, we show that well-performing SAE architectures actually track the positions of task-relevant objects. We find notable performance differences in SAE architectures and identify three components that reliably improve performance, leading to almost perfect object tracking. We show that SAE tracking performance is indicative of downstream RL performance for a representative robotic manipulation task. This allows for identifying suitable SAEs after comparatively lightweight SAE pretraining and before computationally expensive RL training. In addition, troubleshooting is greatly facilitated by the ability to evaluate the performance of an SAE as a key component of the RL pipeline. We observe that an RL agent using keypoints as part of its state achieves RL performance comparable to an agent with full simulation state. Thus, we consider keypoints a suitable state representation for robotic RL. We have demonstrated that this straightforward metric is effective in evaluating SAE architectures. The metric can be used to analyse any 2D keypoint extractor and is not restricted to SAEs. Investigating alternative keypoint extractors and extensions to 3D keypoints is thus a promising avenue for future research.

4. Contextualized Hybrid Ensemble Q-learning CHEQ

This chapter proposes the Contextualized Hybrid Ensemble Q-learning (CHEQ) algorithm, which is the main methodological contribution of this thesis. CHEQ is an AHRL method that dynamically blends actions from a deep RL agent with those from a control prior. By combining a prior controller with RL, this method simultaneously addresses multiple key challenges in the application of RL in production environments: unstructured exploration behaviour, the need to learn from scratch, and the limited generalisation of NNs, which only provide reliable predictions near their training data.

As a foundational contribution (C3), we present a unified AHRL framework that categorises existing approaches (Section 4.4). This framework allows us to reveal structural limitations of prior related work in Section 4.5.2. Building on these insights, we propose the contextualised hybrid MDP formulation (C4) in Section 4.5.1, which resolves key issues in prior hybrid RL methods related to time-variant weighting. This formulation applies to any adaptive or non-stationary combination of learned RL policies with existing controllers and does not rely on a specific weighting mechanism. Leveraging this formulation, we introduce CHEQ (C5) in Section 4.5.3, an AHRL algorithm that systematically adjusts the weighting between the RL agent and the control prior based on the RL agent's uncertainty. CHEQ estimates the uncertainty as the standard deviation of predictions from a critic ensemble.

We evaluate CHEQ in a simulated car racing task requiring precise control near stability limits. Compared to state-of-the-art AHRL methods, CHEQ reduces failures during training by relying on the control prior in unknown situations and significantly improves sample efficiency, addressing gaps G1 and G2. Additionally, its ability to detect out-of-distribution scenarios improves zero-shot transferability to unseen environments. This capability highlights CHEQ 's potential for application in dynamic and unpredictable real-world settings (G4). Finally, by quantifying the RL agent's capabilities through the critic ensemble, CHEQ enhances interpretability and fosters user acceptance, addressing gaps G5 and G6. This chapter is based on the paper [100].

4.1. Introduction

Deep RL methods have shown great success in challenging control problems such as gameplay [24, 10, 25] and robotic manipulation [101, 102]. Despite the great potential of RL methods, their data inefficiency, unstructured exploration behaviour, and inability to generalise to unknown scenarios represent a significant hurdle to their application to real-world problems.

A prime reason for limited real-world applications is the task-agnostic architecture of state-of-the-art RL approaches [43, 22] that does not incorporate prior knowledge on how to solve the task at hand. In contrast, control theory provides a rich set of methods for deriving near-optimal controllers in many applications. This motivates the drive for hybrid RL methods [44, 45] that blend control priors with deep RL policies. Hybrid algorithms thus combine the prior controller's generalisation capabilities and informed behaviour with the power of deep RL for solving general nonlinear problems.

Notwithstanding the conceptual benefit of hybrid RL formulations, how to systematically combine the control prior with the RL agent largely remains an open problem. The majority of prior work [44, 45, 35, 46] proposes a fixed weighting between the control prior and the RL agent. A fixed blending, however, disregards the fact that the capability of the RL agent depends on training time and state. In general, as more data is observed, the RL agent improves its behaviour, ultimately outperforming the control prior in large portions of the domain. The core idea of our approach is to adapt the weighting between the RL agent and control prior based on the agent's confidence. As the RL agent improves over time, this induces a time-variant weighting mechanism. This time dependency leads to structural problems of prior formulations in uncertainty-adapted hybrid RL [103, 104].

We provide a unified view on hybrid RL that allows us to classify prior work within a general framework. Analysing this framework highlights the necessity for a novel AHRL formulation with descriptive, time-invariant dynamics. We define the contextualised hybrid MDP, introducing the adaptive weight as a context variable. Building upon this MDP formulation, we propose the *Contextualized Hybrid Ensemble Q-learning* (CHEQ) algorithm that systematically adapts the weighting between RL agent and control prior based on an uncertainty estimate of a critic ensemble. CHEQ combines the contextualised hybrid RL formulation with uncertainty-based weight adaption and ensemble-based acceleration techniques for data-efficient RL.

We evaluate our algorithm on a car racing task [105], which requires operating a car close to its stability limits in order to achieve maximum return. We find that compared

to prior work in AHRL, the CHEQ algorithm shows (i) reduced failures during training, (ii) increased sample efficiency, and (iii) improved transfer behaviour on unseen race tracks.

In summary, our main contributions are:

- A unified framework that allows us to classify existing approaches and reveal key limitations.
- A hybrid MDP formulation, introducing the adaptive weight as a context variable and thus addressing structural problems of prior work in hybrid RL with adaptive weighting.
- A novel hybrid RL algorithm, CHEQ, that systematically adapts the weighting between RL agent and control prior based on Q-ensemble disagreement.

4.2. Related Work

In this section, we discuss relevant prior work combining RL and a control prior. We distinguish two types: hybrid RL with fixed and with adaptive weighting between the RL agent and controller.

Hybrid Reinforcement Learning with Fixed Weighting

Two concurrent works [44, 45] first combined RL and control prior and introduced the term residual RL. In residual RL, the control prior is assumed to be fixed, and the RL agent learns a residual on top of this. In this thesis, we use the general term hybrid RL to include approaches that adapt the controller's weight. Silver *et al.* [10] and Johannink *et al.* [45] show advantages of hybrid RL, such as sample efficiency, improved sim-to-real transfer, and robustness towards uncertainties. Hybrid RL with fixed weights has then successfully been applied to real robot insertion tasks [35, 106], peg insertion under uncertainty [107], driving [108] and to learn a residual RL policy for hardware on top of a simulation trained policy [46].

Hybrid Reinforcement Learning with Adaptive Weighting

Daoudi *et al.* [109] assume a given controller confidence function, employing a controller in instances of high confidence and an RL agent in other scenarios. Our work focuses on the RL agent's confidence and proposes to estimate the confidence based on a critic ensemble. Similar to our approach, Hoel *et al.* [110, 111] train an ensemble of bootstrapped Q-networks for a driving task with discrete actions. They evaluate the uncertainty as

the coefficient of variation of Q-estimates and resort to safe fallback actions in case of high uncertainty. However, they do not combine the controller and RL agent but switch between both. In this work, we investigate a seamless blending approach for continuous control. Rana *et al.* [112] estimate the policy uncertainty using Monte-Carlo dropout, and based on this uncertainty, either sample from a residual policy or the controller alone. Rana *et al.* [113] directly fuse a prior control distribution with an RL policy in a multiplicative fashion and anneal the influence of the control prior over training time. Rana *et al.* [104] use a policy ensemble to estimate how certain the RL agent is in the current action. The combined action is then computed as the Bayesian posterior of control prior and policy distribution. Cheng *et al.* [103] use the TD-error as an uncertainty estimate and combine controller and RL agent based on this. Both Rana *et al.* [104] and Cheng *et al.* [103] base their adaption mechanism on a form of policy uncertainty. However, they train their agents based on the combined action, which obscures the blending mechanism between the RL agent and control prior. We further discuss this limitation in Section 4.5.

4.3. Background

The following introduces the key components and the general concept of hybrid RL.

Reinforcement Learning

We use the standard RL formulation introduced in Section 2.2.1. The environment is modelled as a discounted Markov decision process (MDP) defined by the tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, p, r, \rho_0, \gamma)$, with state space $\mathcal{S}$, action space $\mathcal{A}$, and start state distribution ρ_0. The unknown transition function $p(s_{t+1}, r_{t+1} \mid s_t, a_t^{\mathrm{RL}})$ describes transitions between states $s_t \in \mathcal{S}$ and actions $a_t^{\mathrm{RL}} \in \mathcal{A}$. During transitions, rewards $r_t \in \mathbb{R}$ are emitted according to a reward function $r_{t+1} \sim r(s_t, a_t^{\mathrm{RL}})$. As defined in Equation (2.6), the objective of the RL agent is to learn a policy $\pi^{\mathrm{RL}}(a_t^{\mathrm{RL}} \mid s_t)$ that maximizes the expected cumulative sum of rewards such that

$$J(\pi^{\mathrm{RL}}) = \max_{\pi^{\mathrm{RL}}} \mathbb{E}_{\pi^{\mathrm{RL}}, \mathcal{M}} \left[\sum_{t=0}^{\infty} \gamma^t r_{t+1} \right].$$

The discounted sum of rewards is referred to as return and is accumulated along trajectories under the policy π^{RL} and the environment MDP $\mathcal{M}$. As defined in Equation (2.7),

the state value functions condition expected return on a particular state

$$V^{\pi^{\mathrm{RL}}}(s_t) = \mathbb{E}_{\pi^{\mathrm{RL}},\mathcal{M}}\left[\sum_{k=t}^{\infty} \gamma^{k-t} r_{k+1} \mid s_t\right].$$

In contrast, the action value or Q-function (cf. Equation (2.8)) conditions expected return on specific state action pairs

$$Q^{\pi^{\mathrm{RL}}}(s_t, a_t^{\mathrm{RL}}) = \mathbb{E}_{\pi^{\mathrm{RL}},\mathcal{M}}\left[\sum_{k=t}^{\infty} \gamma^{k-t} r_{k+1} \mid s_t, a_t^{\mathrm{RL}}\right].$$

Control Prior

The prior policy $\pi^{\mathrm{prior}}(a_t^{\mathrm{prior}} \mid s_t)$ represents prior knowledge for solving the RL objective (2.6), while typically not providing the optimal policy over the whole domain $\mathcal{S} \times \mathcal{A}$. This work focuses on control priors based on classic control theory. These can be derived with limited effort in many applications and often provide a good baseline for interaction with $\mathcal{M}$. We use s_t to denote the general state of the environment. However, the state formulations for the control prior and the RL agent do not need to be identical. Whenever they explicitly differ, we will denote them with s_t^{RL} and s_t^{prior}.

Hybrid Reinforcement Learning

Hybrid RL combines the control prior and the RL agent by blending their actions via some mixing function $a_t^{\mathrm{mix}} = f(a_t^{\mathrm{prior}}, a_t^{\mathrm{RL}}, \lambda_t)$ depending on a weight λ_t.

4.4. A Unified View on Hybrid Reinforcement Learning

Next, we develop a unified view on hybrid RL that allows classifying prior methods (cf. Section 4.2). In the standard RL setup depicted in Figure 4.1a, the RL agent π^{RL} interacts with the time-invariant MDP $\mathcal{M}$ with dynamics $p(s_{t+1}, r_{t+1} \mid s_t, a_t^{\mathrm{RL}})$ that represents the controlled system. Hybrid RL (see Figure 4.1b) incorporates a control prior π^{prior} which requires reformulating the standard framework. Here, π^{RL} and π^{prior} apply a combined action a_t^{mix} to $\mathcal{M}$. The mixing function f generates a combined action by blending the individual actions based on a weighting vector λ_t provided by a weight adaption function Λ. Within this generalised framework, prior work in hybrid RL can be categorised based on the choice of f and Λ.

4.4.1. Mixing Function

We consider mixing functions based on a weighted sum with a weighting vector $\lambda_t = [\lambda_t^{\text{prior}}, \lambda_t^{\text{RL}}]^\top$

$$a_t^{\text{mix}} = f(a_t^{\text{prior}}, a_t^{\text{RL}}, \lambda_t) = \lambda_t^{\text{prior}} \cdot a_t^{\text{prior}} + \lambda_t^{\text{RL}} \cdot a_t^{\text{RL}}. \tag{4.1}$$

This formulation allows to distinguish a *residual* and a *regularised* setting.

In the residual setting, λ_t^{prior} is typically constant while λ_t^{RL} can be variable, i.e., $\lambda_t = [1, \lambda_t^{\text{RL}}]^\top$ [44, 45, 35]. Thus, the RL agent interacts with the closed control loop between π^{prior} and $\mathcal{M}$ and learns a residual action on top of a_t^{prior}. Consequently, λ_t modulates the RL agent's impact on the closed loop dynamics. As the control prior is not scaled down, it might interpret the RL agent as a disturbance and counteract it [107], which can limit the overall performance of residual formulations.

In the regularised setting, both weights are adaptable such that $\lambda_t^{\text{prior}} + \lambda_t^{\text{RL}} = 1$ [103, 104]. This results in a mixing function of the form

$$a_t^{\text{mix}} = f(a_t^{\text{prior}}, a_t^{\text{RL}}, \lambda_t^{\text{RL}}) = (1 - \lambda_t^{\text{RL}}) \cdot a_t^{\text{prior}} + \lambda_t^{\text{RL}} \cdot a_t^{\text{RL}}. \tag{4.2}$$

with $\lambda_t^{\text{RL}} \in [0, 1]$. In the limit $\lambda_t^{\text{RL}} = 0$, the control prior interacts with $\mathcal{M}$ without the interference of the RL agent, while the regularised setting reduces to the standard RL problem for $\lambda_t^{\text{RL}} = 1$. Thus, λ_t^{RL} indicates not only the impact of a_t^{RL} but also whether the RL agent interacts with the open loop dynamics of $\mathcal{M}$ or the closed loop dynamics as in the residual setting. Consequently, the control prior can be interpreted as a regularisation of the RL agent. Our proposed algorithm operates in the regularised setting, allowing the agent to take over complete control when $\lambda_t^{\text{RL}} = 1$.

4.4.2. Weight Adaption Function

Hybrid RL approaches can further be classified based on the choice of the weight adaption function Λ modulating the weighting vector λ_t of the mixing function f.

A large body of work, which we refer to as *fixed-weight hybrid RL* [44, 45, 35, 107, 46] chooses λ_t fixed throughout training. This neglects the time- and state-dependent capabilities of the RL agent.

Approaches that adapt λ_t, which we refer to as AHRL methods, rely on different mechanisms. Scheduling approaches [113] change the weight explicitly with time, i.e. $\lambda_t = \Lambda(t)$, typically increasing the weight of the RL agent as training progresses. Domain-based approaches [114, 109] adapt the weights based on the point of operation within the domain $\mathcal{S} \times \mathcal{A}$, i.e. $\lambda_t = \Lambda(s_t, a_t)$. Uncertainty-based approaches [103, 104]

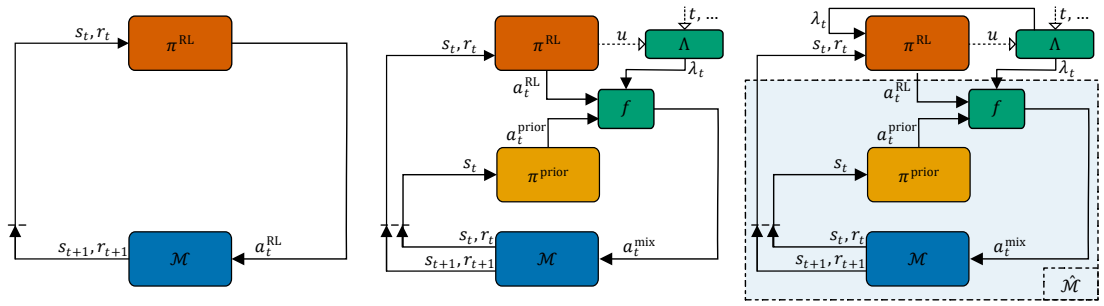


Figure 4.1.: We show a standard RL setting in Figure a, hybrid RL settings from prior work in Figure b and our contextualized hybrid RL setting based on RL action a_t^{RL} and weighting factor λ_t in Figure c.

adapt the weight based on the confidence of the RL agent, indicated by an uncertainty estimate $u(s_t, a_t, t)$, giving more weight to the RL agent when it has high confidence. Thus, they aim to leverage the benefits of the RL agent whenever possible while resorting to a safe controller in situations where the RL agent has not seen enough data. The time dependency of the uncertainty estimate, however, increases the complexity of the hybrid RL setting, requiring a reformulation of the learning problem. In Section 4.5, we discuss the shortcomings of prior formulations and propose our own.

4.5. Contextualized Hybrid Reinforcement Learning

In the following, we propose a novel contextualised formulation of the AHRL problem in Section 4.5.1 and illustrate its benefits over prior approaches in Section 4.5.2. Based on that framework, we propose the Contextualized Hybrid Ensemble Q-learning (CHEQ) algorithm in Section 4.5.3.

4.5.1. General Concept of Contextualized Hybrid Reinforcement Learning

Based on the unified view provided in Section 4.4, we propose a general formulation for the hybrid RL problem we call *contextualised hybrid RL*.

The environment in the hybrid setting not only consists of the controlled system $\mathcal{M}$ but also comprises the control prior, the mixing function, and the weight adaption function. While we consider both the control prior and the mixing function time-invariant, the weight adaption function Λ can have time-varying behaviour, i.e. $\lambda_t = \Lambda(t, \dots)$,

as discussed in Section 4.4.2. This leads to time-varying dynamics of the hybrid environment, violating the assumption of time-invariance in the MDP formulation [115]. Instead, we exclude Λ from the definition of the hybrid environment and introduce the adaptive weight vector λ_t as a context variable to the agent and the environment (see Figure 4.1c). We model the hybrid environment introducing the contextualized hybrid MDP $\hat{\mathcal{M}} = (\mathcal{S}, \mathcal{A}, \mathcal{W}, \hat{p}, r, \rho_0, \gamma)$ with $\mathcal{W}$ the set of weighting vectors λ_t and the contextualized dynamics function $\hat{p}(s_{t+1}, r_{t+1} \mid s_t, a_t^{\mathrm{RL}}, \lambda_t)$.

The MDP formulation $\hat{\mathcal{M}}$ induces the contextualized hybrid RL objective

$$\hat{J}(\pi^{\mathrm{RL}}) = \max_{\pi^{\mathrm{RL}}} \mathbb{E}_{\pi^{\mathrm{RL}}, \hat{\mathcal{M}}} \left[\sum_{t=0}^{T} \gamma^t r_{t+1} \right], \tag{4.3}$$

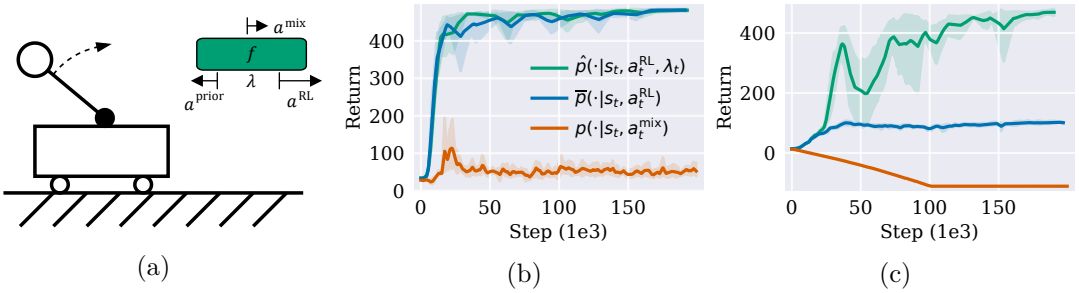
which enforces to learn a policy $\hat{\pi}^{\mathrm{RL}}(a_t^{\mathrm{RL}} \mid s_t^{\mathrm{RL}}, \lambda_t)$ that maximizes expected return in $\hat{\mathcal{M}}$. Introducing λ_t as a context variable further yields the contextualised hybrid value functions $\hat{V}^{\pi^{\mathrm{RL}}}(s_t, \lambda_t) = \mathbb{E}_{\pi^{\mathrm{RL}}, \hat{\mathcal{M}}} \left[\sum_{k=t}^{\infty} \gamma^{k-t} r_{k+1} \mid s_t, \lambda_t \right]$ and $\hat{Q}^{\pi^{\mathrm{RL}}}(s_t, a_t^{\mathrm{RL}}, \lambda_t) = \mathbb{E}_{\pi^{\mathrm{RL}}, \hat{\mathcal{M}}} \left[\sum_{k=t}^{\infty} \gamma^{k-t} r_{k+1} \mid s_t, a_t^{\mathrm{RL}}, \lambda_t \right]$. Thus, we can optimise (4.3) using standard RL methods by additional conditioning on λ_t. The general mechanism of contextualised hybrid RL is illustrated in Algorithm 1.

Algorithm 1 Contextualized Hybrid Reinforcement Learning

Require: RL policy $\hat{\pi}_\phi^{\mathrm{RL}}(a_t^{\mathrm{RL}} \mid s_t, \lambda_t)$, control prior $\pi^{\mathrm{prior}}(a_t^{\mathrm{prior}} \mid s_t)$, mixing function $f(a_t^{\mathrm{RL}}, a_t^{\mathrm{prior}}, \lambda_t)$, weight adaption function Λ, replay buffer $\mathcal{D} \leftarrow \emptyset$.

for each episode **do**
  Sample initial state $s_0 \sim \rho_0$, initialize λ_0.
  for each step **do**
    Sample RL action $a_t^{\mathrm{RL}} \sim \pi_\phi^{\mathrm{RL}} \left(a_t^{\mathrm{RL}} \mid s_t, \lambda_t \right)$.
    Sample control prior action $a_t^{\mathrm{prior}} \sim \pi_\phi^{\mathrm{prior}} \left(a_t^{\mathrm{RL}} \mid s_t \right)$.
    Get combined action $a_t^{\mathrm{mix}} = f(a_t^{\mathrm{RL}}, a_t^{\mathrm{prior}}, \lambda_t)$.
    Observe state transition $s_{t+1}, r_{t+1} \sim p\left(\cdot, \cdot \mid s_t, a_t^{\mathrm{mix}}\right)$.
    Store $\left(s_t, a_t^{\mathrm{RL}}, \lambda_t, s_{t+1}, r_{t+1}\right)$ into replay buffer $\mathcal{D}$.
    Get next adaptive weight $\lambda_{t+1} = \Lambda$.
    Sample set of transitions $(s, a, \lambda, s', r) \sim \mathcal{D}$.
    Optimize ϕ with respect to (4.3) using RL with sampled transitions.
  end for
end for

Prior work takes different approaches to formulating the hybrid learning problem. Approaches with a time-invariant weight adaption function, such as fixed-weight hybrid



(a) (b) (c)

Figure 4.2.: We illustrate different hybrid RL formulations on a cart pole system with a biased control prior, pushing to the left. The environment is shown in Figure a. Figure b shows the return of different hybrid agents in the case where λ_t^{RL} is fixed, and Figure c shows the return with variable λ_t^{RL}. Only the contextualized hybrid RL agent observing dynamics $\hat{p}$ can cope with both scenarios.

methods, include Λ in the definition of the environment MDP $\bar{\mathcal{M}} = (\mathcal{S}, \mathcal{A}, \bar{p}, r, \rho_0, \gamma)$ with dynamics $\bar{p}(s_{t+1}, r_{t+1} \mid s_t, a_t^{\text{RL}})$. This formulation directly applies to the standard RL objective (2.6); however, it does not generalise to time-varying adaption mechanisms such as uncertainty-adapted methods. Approaches with a time-varying adaption mechanism [103, 104] typically formulate the hybrid RL problem concerning the controlled system MDP and the combined action with dynamics $p(s_{t+1}, r_{t+1} \mid s_t, a_t^{\text{mix}})$. This likewise yields a formulation that is directly applicable to (2.6); however, it introduces a structural mismatch in the learning problem, as the agent optimises for the combined action, neglecting the influence of the mixing process.

4.5.2. Illustrative Example

We exemplify the strength of the contextualised hybrid RL formulation based on $\hat{\mathcal{M}}$ introduced in Section 4.5.1 by comparing it to prior approaches on the cart pole system depicted in Figure 4.2a. The goal is to balance the pole upright while keeping the cart close to its initial position. The system is controlled via continuous forces on the cart. We select the prior distribution, denoted as π^{prior}, such that it applies a constant force to the left. This choice is intended to destabilise formulations that are unconscious of the mixing process. We investigate a time-invariant fixed weight setting and a time-varying schedule-based weight adaption setting to highlight the capability of the respective formulations to deal with both scenarios. This simple example illustrates that the contextualised hybrid MDP formulation can deal with destabilising control priors and time-varying weight adaption functions while prior formulations fail.

Fixed Weighting

First, we consider a residual setting with fixed weights $\lambda_t^{\mathrm{RL}} = \lambda_t^{\mathrm{prior}} = 0.5$. Figure 4.2b depicts the performance of RL agents trained under $\hat{\mathcal{M}}$, $\bar{\mathcal{M}}$, and $\mathcal{M}$ with respective dynamics $\hat{p}(s_{t+1}, r_{t+1} \mid s_t, a_t^{\mathrm{RL}}, \lambda_t)$, $\bar{p}(s_{t+1}, r_{t+1} \mid s_t, a_t^{\mathrm{RL}})$, and $p(s_{t+1}, r_{t+1} \mid s_t, a_t^{\mathrm{mix}})$. While agents trained under $\hat{\mathcal{M}}$ and $\bar{\mathcal{M}}$ learn to stabilise the cart pole, the hybrid formulation concerning $\mathcal{M}$ fails. When formulating the RL problem concerning a_t^{mix}, the RL agent observes the combined action in its data and, therefore, learns the combined action in its policy. This, however, neglects the fact that the policy action is mixed with the controller action before being applied to $\mathcal{M}$. Assuming the cart pole is not moving, and the pole is upright, an agent trained under $\mathcal{M}$ provides the optimal combined action, namely applying no force, while a_t^{prior} pushes the pole to the left. This results in a_t^{mix} pointing to the left, causing the pole to fall while giving the agent no mechanism to observe and counteract this phenomenon. Instead, formulating the hybrid RL problem concerning a_t^{RL} allows the agent to observe the mixing mechanism and compensate for the destabilising control prior.

Adaptive Weighting

Second, we consider an AHRL problem with time-varying Λ. We choose a schedule-based approach with a regularising mixing function Equation (4.2) and $\lambda_t^{\mathrm{RL}} \in [0, 1]$ linearly increasing over time. Figure 4.2c shows the performance of agents trained under formulations based on $\hat{\mathcal{M}}$, $\bar{\mathcal{M}}$, and $\mathcal{M}$. While agents trained under $\bar{\mathcal{M}}$ and $\mathcal{M}$ fail, the formulation based on $\hat{\mathcal{M}}$ succeeds. The formulation under $\mathcal{M}$ suffers from the occlusion of the mixing process as discussed in the fixed weight setting, and the formulation under $\bar{\mathcal{M}}$ fails due to the missing information about the time-variant behaviour of the mixing process. The proposed contextualised hybrid RL formulation solves these issues by formulating the task concerning a_t^{RL} and introducing the context variable λ_t.

4.5.3. Contextualized Hybrid Ensemble Q-learning (CHEQ)

Based on the contextualised hybrid RL formulation introduced in Section 4.5.1, we propose the Contextualized Hybrid Ensemble Q-learning (CHEQ) algorithm. At the heart of CHEQ is a critic ensemble that (i) provides an uncertainty estimate enabling an uncertainty-adapted hybrid RL mechanism and (ii) allows the incorporation of ensemble-based acceleration techniques for data-efficient RL.

We base CHEQ on the SAC [22] algorithm and a regularizing mixing function Equation (4.2) with $\lambda_t = [(1 - \lambda_t^{\mathrm{RL}}), \lambda_t^{\mathrm{RL}}]^\top$. The weight adaption mechanism relies on a critic

ensemble comprising of E contextualised Q-functions with parameters θ_e, $e \in \{1, \ldots, E\}$ and corresponding target Q-functions with parameters $\bar{\theta}_e$, $e \in \{1, \ldots, E\}$. We update the critics with the mechanism of randomized ensemble double Q-learning (REDQ) [116] and enforce sufficient independence between Q-estimates using Bernoulli masking of the training data [117, 118, 119]. Model ensembles estimate epistemic uncertainty from disagreement between individual models within the ensemble. If different critics disagree about the outcome of taking action a_t^{RL} in s_t while weighting with λ_t, this indicates a weak understanding of the task in the particular area of $\mathcal{S} \times \mathcal{A} \times \mathcal{W}$. Thus, the control prior should be prioritised over the RL agent in such situations. Therefore, epistemic uncertainty represents a suitable quantity for adapting the weighting of control prior and RL agent. We define epistemic uncertainty as the standard deviation of critic predictions

$$u(s_t, a_t^{\mathrm{RL}}, \lambda_t^{\mathrm{RL}}) = \sqrt{\frac{1}{E} \sum_{e=1}^{E} \left(\hat{Q}_{\theta_e}(s_t, a_t^{\mathrm{RL}}, \lambda_t^{\mathrm{RL}}) - \mu(s_t, a_t^{\mathrm{RL}}, \lambda_t^{\mathrm{RL}}) \right)^2} \tag{4.4}$$

with $\mu(s_t, a_t^{\mathrm{RL}}, \lambda_t^{\mathrm{RL}}) = \frac{1}{E} \sum_{e=1}^{E} \hat{Q}_{\theta_e}(s_t, a_t^{\mathrm{RL}}, \lambda_t^{\mathrm{RL}})$. We aim to give low weight to the RL agent in areas of high uncertainty and vice versa. Thus, the weight adaption function $\Lambda(u(s_t, a_t^{\mathrm{RL}}, \lambda_t^{\mathrm{RL}}))$ maps the critics epistemic uncertainty to the weighting factor $\lambda_t^{\mathrm{RL}} \in [\lambda_{\min}, \lambda_{\max}] \subseteq [0, 1]$ via the piece-wise linear function

$$\lambda_{t+1}^{\mathrm{RL}} = \begin{cases} \lambda_{\max} & \text{if } u(s_t, a_t^{\mathrm{RL}}, \lambda_t^{\mathrm{RL}}) < u_{\min} \\ \frac{u(s_t, a_t^{\mathrm{RL}}, \lambda_t^{\mathrm{RL}}) - u_{\max}}{u_{\min} - u_{\max}} (\lambda_{\max} - \lambda_{\min}) + \lambda_{\min} & \text{if } u(s_t, a_t^{\mathrm{RL}}, \lambda_t^{\mathrm{RL}}) \in [u_{\min}, u_{\max}] \\ \lambda_{\min} & \text{if } u(s_t, a_t^{\mathrm{RL}}, \lambda_t^{\mathrm{RL}}) > u_{\max}. \end{cases} \tag{4.5}$$

Besides providing an uncertainty estimate of the RL agent, the critic ensemble used in CHEQ has proven effective in mitigating overestimation bias [120] in Q-learning-based approaches [121, 122, 116]. Consequently, the critic ensemble allows for increasing the update-to-data (UTD) ratio while maintaining stable learning. This substantially improves the data efficiency of value-based actor-critic methods [116]. Algorithm 2 shows the pseudocode of the Contextualized Hybrid Ensemble Q-learning algorithm.

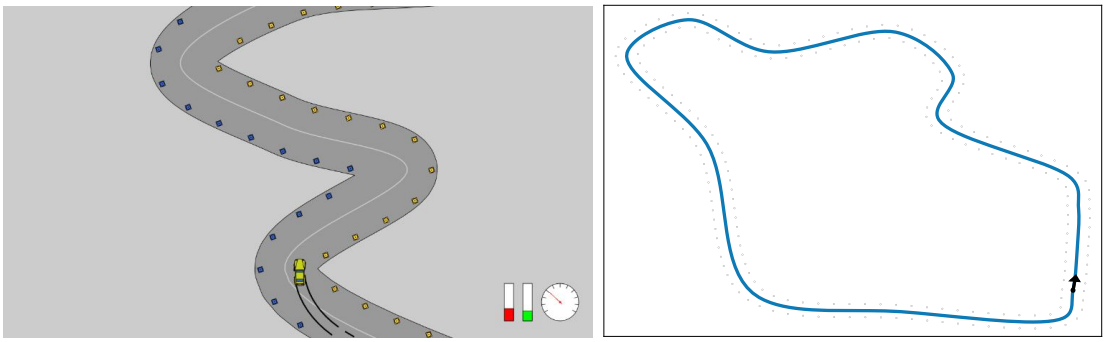
Algorithm 2 CHEQ

Initialize control prior $\pi^{\text{prior}}(a_t^{\text{prior}} \mid s_t)$, contextualized RL policy $\hat{\pi}_\phi^{\text{RL}}(a_t^{\text{RL}} \mid s_t, \lambda_t^{\text{RL}})$, contextualized critic ensemble $\hat{Q}_{\theta_e}(s_t, a_t^{\text{RL}}, \lambda_t^{\text{RL}})$, $e \in \{1, \ldots, E\}$, contextualized target critic ensemble $\hat{Q}_{\bar{\theta}_e}(s_t, a_t^{\text{RL}}, \lambda_t^{\text{RL}})$, $e \in \{1, \ldots, E\}$, replay buffer $\mathcal{D} \leftarrow \emptyset$, weighting interval $[\lambda_{\min}, \lambda_{\max}]$, uncertainty limits $[u_{\min}, u_{\max}]$, UTD ratio G, Bernoulli masking rate κ, minimization targets F, Polyak averaging factor τ

for each epoch **do**
  $s_0 \sim \rho_0$, $\lambda_0^{\text{RL}} = \lambda_{\min}$
  for each epoch step **do**
    $a_t^{\text{RL}} \sim \hat{\pi}_\phi^{\text{RL}}(\cdot \mid s_t, \lambda_t^{\text{RL}})$
    $a_t^{\text{prior}} \sim \pi^{\text{prior}}(\cdot \mid s_t)$
    $a_t^{\text{mix}} = (1 - \lambda_t^{\text{RL}}) a_t^{\text{prior}} + \lambda_t^{\text{RL}} a_t^{\text{RL}}$
    $u(s_t, a_t^{\text{RL}}, \lambda_t^{\text{RL}})$ according to (4.4)
    $\lambda_{t+1} = \Lambda(u(s_t, a_t^{\text{RL}}, \lambda_t^{\text{RL}}))$ according to (4.5)
    $s_{t+1}, r_{t+1} \sim \hat{p}(\cdot, \cdot \mid s_t, a_t^{\text{RL}}, \lambda_t^{\text{RL}})$
    for $e = 1, \ldots, E$ **do**
      Sample Bernoulli Mask $m_t^e \sim Ber(\kappa)$
    end for
    $\mathcal{D} \leftarrow \mathcal{D} \cup \{(s_t, a_t^{\text{RL}}, \lambda_t^{\text{RL}}, s_{t+1}, r_{t+1}), m_t^1, \ldots, m_t^E\}$
    for G updates **do**
      Sample mini-batch $\mathcal{B} = \{(s, a^{\text{RL}}, \lambda^{\text{RL}}, s', r\}$ from $\mathcal{D}$
      Sample a set $\mathcal{F}$ with $|\mathcal{F}| = F$ uniform at random from $\{1, \ldots, E\}$
      $\tilde{a}'^{\text{RL}} \sim \hat{\pi}_\phi^{\text{RL}}(\cdot \mid s', \lambda^{\text{RL}})$
      $y = r + \gamma \left(\min_{e \in \mathcal{F}} \hat{Q}_{\bar{\theta}_e}(s', \tilde{a}'^{\text{RL}}, \lambda^{\text{RL}}) - \alpha \log \hat{\pi}_\phi^{\text{RL}}(\tilde{a}'^{\text{RL}} \mid s', \lambda^{\text{RL}}) \right)$
      for $e = 1, \ldots, E$ **do**
        Update θ_e with gradient descent using
        $\mathbb{1}_{m^e} \nabla_{\theta_e} \frac{1}{|\mathcal{B}|} \sum_{(s, a^{\text{RL}}, \lambda^{\text{RL}}, r, s') \in \mathcal{B}} \left(\hat{Q}_{\theta_e}(s, a, \lambda^{\text{RL}}) - y \right)^2$
        $\bar{\theta}_e \leftarrow \tau \bar{\theta}_e + (1 - \tau) \theta_e$
      end for
    end for
    update ϕ with gradient ascent using
    $\tilde{a}_{\text{RL}} \sim \hat{\pi}_\phi^{\text{RL}}(\cdot \mid s, \lambda^{\text{RL}}))$
    $\nabla_\phi \frac{1}{|\mathcal{B}|} \sum_{s \in \mathcal{B}} \left(\frac{1}{E} \sum_{e=1}^{E} \hat{Q}_{\theta_e}(s, \tilde{a}_{\text{RL}}, \lambda^{\text{RL}}) - \alpha \log \hat{\pi}_\phi^{\text{RL}}(\tilde{a}_{\text{RL}} \mid s, \lambda^{\text{RL}})) \right)$
  end for
end for

4.6. Racing Task

We base our experiments on a car racing setting adapted from [105]. Achieving high returns in this task requires advanced trajectory planning and control while operating the vehicle close to stability limits. Figure 4.3 shows an example of the environment.



(a) (b)

Figure 4.3.: Racing task and the training track.

The vehicle is a front-wheel drive, which we model using a dynamic single-track model with a coupled Dugoff tyre model. For racing around the track, we control the throttle a_t^{throt}, the brake a_t^{brake}, and the steering angle δ_t. The mixed action is then defined as $a_t^{\text{mix}} = (a_t^{\text{throt,mix}}, a_t^{\text{brake,mix}}, \delta_t^{\text{mix}})$.

We define the RL agent's state as $s_t^{\text{RL}} = (v_{x,t}, v_{y,t}, \omega_t, \beta_t, o_{\text{track},t})$, where the ego vehicle's velocity vector is $v_{\text{ego},t} = (v_{x,t}, v_{y,t})$ in the vehicle's reference frame, β_t is the steering angle, and ω_t is the yaw rate. The track observation, $o_{\text{track},t} = (x_t, y_t)^T$, is represented as a vector of 20 Cartesian distances $(x_{i,t}, y_{i,t})$ to the track centre line, sampled equidistantly from a 60 m segment ahead.

We adopt the original reward formulation from Schier *et al.* [105], where the RL agent incurs a penalty $r_{\text{collision}}$ upon colliding with the track boundary and a penalty r_{fail} for leaving the track with its centre of mass. The latter event also terminates the episode. A positive reward is granted for driving at higher speeds, calculated as the scalar projection of the velocity vector v_{ego} onto the forward track direction n_{track}. The complete reward function is given by

$$r(s_t^{\text{RL}}, a_t^{\text{mix}}) = -r_{\text{fail}} - 0.2 \cdot r_{\text{collision}} + 0.01 \cdot n_{\text{track},t} \cdot v_{\text{ego},t}. \tag{4.6}$$

The control prior follows the centre line of the racing track with adaptive speeds using a Stanley controller [123] for the lateral control and a proportional controller for the

longitudinal control.

For the lateral control, the Stanley controller applies the steering control law

$$\delta_t^{\text{prior}} = \psi_t + \frac{k_{\text{cross}} \cdot e_t}{v_t + k_{\text{soft}}},$$

where ψ_t represents the heading error, e_t denotes the cross-track error of the front axle, and v_t is the vehicle's velocity.

For the longitudinal control, we design two symmetric P-controllers, one for the brake and one for the throttle. The target velocity is computed based on the curve radius R_t of the track segment directly in front of the vehicle:

$$v_t^{\text{target}} = \min\{k_r \cdot R_t, v^{\max}\},$$

where $v^{\max}$ is the maximum desired velocity. The throttle and brake control laws are then defined as

$$a_t^{\text{throt,prior}} = \begin{cases} k_{v,t} \cdot (v_t^{target} - v_t), & v_t^{target} - v_t \geq 0 \\ 0, & \text{otherwise} \end{cases}$$

$$a_t^{\text{br,prior}} = \begin{cases} k_{v,t} \cdot (v_t - v_t^{target}), & v_t^{target} - v_t \leq 0 \\ 0, & \text{otherwise} \end{cases}$$

with a shared gain k_v. Under this control scheme, the control prior accelerates when the vehicle is too slow and brakes when it is too fast. It never simultaneously applies the brake and throttle.

To mitigate aggressive braking when the RL agent hands control over to the prior in high-speed cornering situations, we introduce a clipped linear gain schedule for k_v that attenuates braking at higher velocities:

$$k_{v,t} = \text{clip}\left(\frac{k_v^{\max} - k_v^{\min}}{v_{\text{low}} - v_{\text{high}}}(v_t - v_{\text{low}}) + k_v^{\max}; k_v^{\max}, k_v^{\min}\right).$$

Further details on the controller gains can be found in Appendix B.1.2.

4.7. Experiments

We evaluate CHEQ on the racing task introduced in Section 4.6 and compare it to standard RL, fixed-weighting hybrid RL, and state-of-the-art AHRL. In our experiments,

CHEQ yields substantial improvements in (i) data efficiency compared to other hybrid methods, as well as (ii) exploration safety and (iii) zero-shot transferability to unknown scenarios as compared to all competitor approaches.

4.7.1. Experimental Setup

We compare CHEQ against the standard RL approaches SAC [22] and REDQ [116], fixed-weighting hybrid RL based on SAC, and the state-of-the-art AHRL methods *Control Regularization for Reduced Variance Reinforcement Learning* (CORE) [103] and *Bayesian Controller Fusion* (BCF) [104]. In all experiments, we provide results for CHEQ with a high UTD ratio (CHEQ-UTD20) to demonstrate the capabilities of the approach and a low UTD ratio (CHEQ-UTD1) for a fair comparison to SAC-based methods. All implementations are based on either the Clean RL library [124] or the original paper implementation [104]. We refer the reader to Appendix B.1.1 for a detailed description of the hyperparameter settings.

We train all approaches on ten random seeds and one fixed race track. We report return and cumulative training failures. Runs are considered a failure when the car leaves the track. For zero-shot transfer, we evaluate ten trained models per algorithm on ten unseen racetracks. Return and failure plots show the respective mean (solid lines) and 95 % confidence interval (shaded areas).

4.7.2. Evaluation on the Car Racing Environment

We compare CHEQ to standard RL, fixed-weight hybrid RL, and AHRL concerning learning performance (see Figure 4.4) and zero-shot transfer to unknown tracks (see Figure 4.5).

Comparison against Fixed-Weight Hybrid RL and Standard RL

Comparing the CHEQ algorithm based on SAC (CHEQ-UTD1) to a standalone SAC agent and the control prior in Figure 4.4a illustrates the general benefit of hybrid RL. While the control prior operates safely without failing, it shows limited performance due to the conservative driving policy. SAC shows strong asymptotic performance at the cost of frequent failures throughout training. CHEQ-UTD1 considerably outperforms SAC regarding data efficiency and exploration safety, learning faster with fewer failures while yielding comparable asymptotic performance. The comparison of CHEQ to fixed-weight hybrid RL methods further illustrates the advantage of an adaptive weighting scheme. The fixed-weight hybrid RL approaches (0.5-SAC, 0.7-SAC) combine the control prior

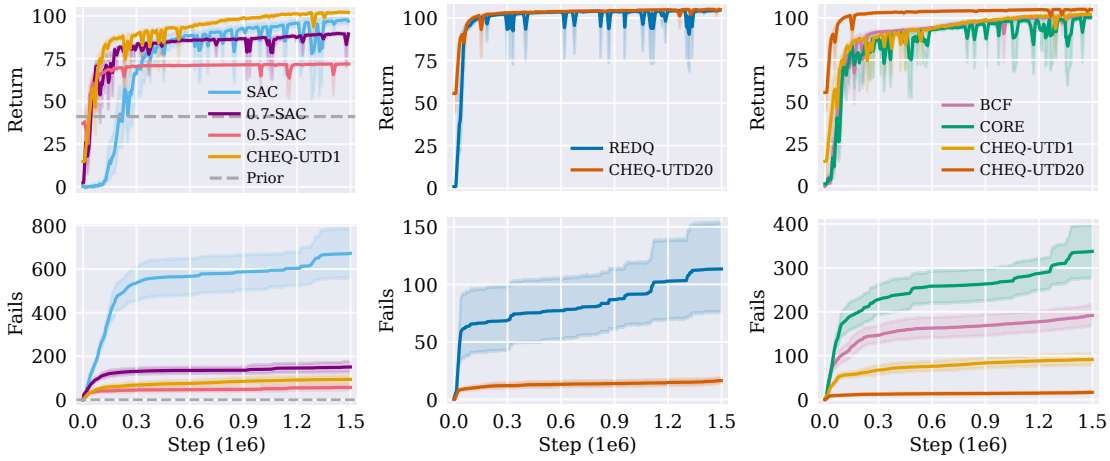
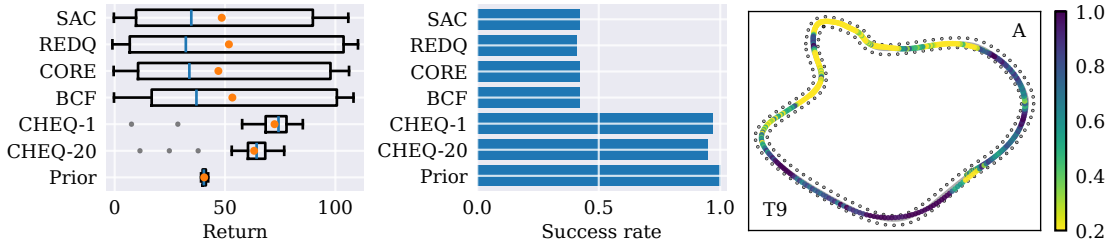


Figure 4.4.: Performance of all trained RL approaches in terms of evaluation return and training failures on the training track. We compare the training performance of CHEQ with fixed-weight hybrid RL and standard RL in Figure a. Figure b shows a comparison of our hybrid CHEQ-UTD20 agent to a standalone REDQ agent using the same UTD ratio. Lastly, we compare CHEQ to prior work in AHRL in Figure c.

with a SAC agent using the mixing function in Equation (4.2) with $\lambda_t^{\text{RL}} = 0.5$ and $\lambda_t^{\text{RL}} = 0.7$, respectively. Here, the choice of λ_t^{RL} represents a trade-off between exploration safety and asymptotic performance, where a higher λ_t^{RL} enables better performance while reducing safety. A fixed weight of $\lambda_t^{\text{RL}} = 0.5$ arguably reduces failures compared to CHEQ-UTD1. However, this comes at the cost of substantially lower performance.

Update-To-Data Ratio

As discussed in Section 4.5.3, the critic ensemble of CHEQ allows the use of acceleration techniques originally proposed in the REDQ algorithm. Increasing the UTD ratio to 20 notably improves data efficiency as compared to SAC, both as a standalone RL algorithm (REDQ) and as an AHRL algorithm (CHEQ-UTD20). The speed-up in training helps to reduce training failures as REDQ reports a drastically reduced number of failures compared to SAC. The benefit is further amplified in the adaptive hybrid formulation of CHEQ-UTD20 as indicated by its strong initial performance and the ability to reduce the mean cumulative fails to less than 20.



(a) (b) (c)

Figure 4.5.: Figures a and b show a comparison of the return and the failures of ten trained models per algorithm on ten transfer tracks. Figure c illustrates the development of λ^{RL} of the CHEQ-UTD20 agent for an exemplary transfer track.

Comparison against state-of-the-art Adaptive Hybrid RL

Next, Figure 4.4c compares CHEQ to state-of-the-art AHRL methods. CHEQ-UTD1 shows similar data efficiency and performance compared to CORE and BCF while considerably reducing accumulated fails. CHEQ-UTD20 substantially outperforms all competitor approaches in all performance metrics. A more detailed hyperparameter analysis of the prior approaches, as well as results for reformulations of CORE and BCF as contextualised hybrid RL methods, can be found in Appendix B.2.

Zero-shot Transfer

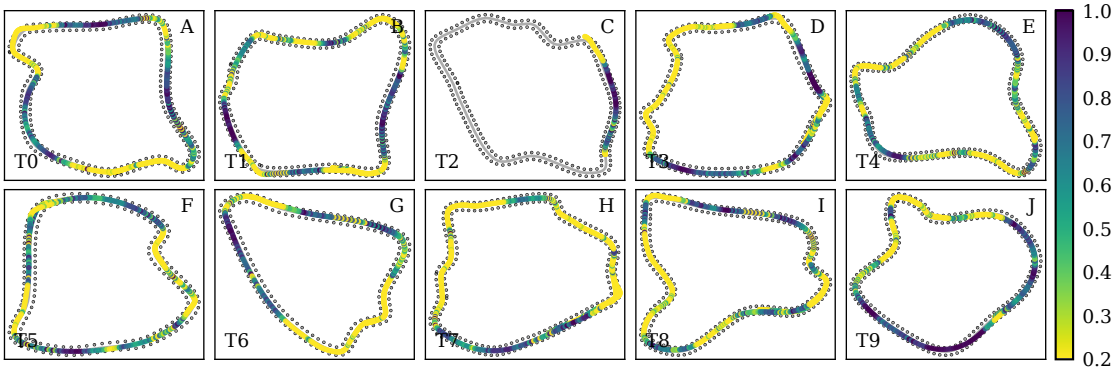


Figure 4.6.: The 10 transfer tracks and the corresponding zero-shot transfer of one seed of CHEQ-UTD20.

Finally, we perform a zero-shot transfer of the trained agents. Returns are depicted in Figure 4.5a while Figure 4.5b shows the success rate of the respective methods. CHEQ-UTD1, CHEQ-UTD20 and the control prior achieve a success rate of 97 %, 95 %, and

100 %, respectively. The other standard and hybrid RL methods frequently fail in unseen scenarios. While the CHEQ variants fail slightly more often than the controller, they drive notably faster, i.e., they achieve higher returns. Figure 4.5c illustrates the adaption mechanism of CHEQ on one example track. In challenging and unseen curves, the agent gradually hands over to the control prior. We find that in the few failure cases (3 out of 100 for CHEQ-UTD1 and 5 out of 100 for CHEQ-UT20), the agent correctly identifies its uncertainty and hands over to the controller, but the controller cannot navigate the situation safely. We illustrate all ten transfer tracks, as well as the weight adaption of CHEQ-UTD20 on these tracks in Figure 4.6. In summary, CHEQ shows strong zero-shot transfer behaviour, driving faster than the controller with only a few failure cases.

4.8. Conclusion

This chapter addresses how to systematically combine an RL agent with a control prior. We propose a novel formulation of the AHRL problem, which introduces the adaptive weighting parameter as a context variable of the MDP, and, based on this, propose the Contextualized Hybrid Ensemble Q-learning (CHEQ) algorithm. CHEQ combines a reliable critic uncertainty-based weight adaption mechanism with the data efficiency of critic ensemble methods, yielding substantially stronger results than state-of-the-art AHRL methods on a racing task concerning data efficiency, exploration safety, and transferability.

5. Safe Variable Impedance Learning for Robotic Polishing Polishing

This chapter investigates and demonstrates the application of RL to a hardware experiment motivated by real-world challenges in industrial production. Unlike most state-of-the-art approaches, which train in simulation and rely on sim-to-real transfer or domain adaptation, we train RL directly on hardware. This introduces additional challenges, such as uncertainties and non-stationarities, but ensures the learned policies are immediately applicable to real-world scenarios.

As a case study, we focus on robotic polishing, a contact-rich and dynamic task that exemplifies the complexities of manipulation in industrial automation. Polishing requires precise force and velocity control to achieve high-quality surface finishes while adapting to variations in the workpiece and tool interaction. These factors make robotic polishing an ideal benchmark for evaluating RL-based control strategies.

In this context, we investigate the application of the CHEQ algorithm introduced in Chapter 4. We first demonstrate the potential of CHEQ to learn variable controller gains and path-following behaviour for polishing a 3D workpiece with a robotic arm in simulation while maintaining safe exploration (cf. Section 5.5). This corresponds to contribution C6. Our results show that CHEQ outperforms standalone RL, achieving superior force and velocity tracking. The algorithm dynamically transitions from relying on the control prior to operating fully autonomously, introducing a natural curriculum learning approach. This gradual handover enables efficient and safe learning (G1, G2).

Building on this, we deploy the CHEQ algorithm on hardware, representing contribution C7 (cf. Section 5.6). Hardware training demonstrates that CHEQ can effectively learn in the presence of environment uncertainties and non-stationarities (G3, G4). Moreover, it enables a direct evaluation of whether the algorithm's sample efficiency and exploration safety remain within practical limits for real-world applications (G1, G2).

Our findings confirm that CHEQ effectively addresses these challenges in an industrially relevant use case, reinforcing its potential for real-world automation. This chapter is based on the paper [125].

5.1. Introduction

Robotics has become a cornerstone of industrial automation, playing a pivotal role in improving efficiency and precision across various domains. A long-term aspiration in robotics is to achieve manipulation capabilities that combine human-like dexterity with compliant behaviour, enabling robots for contact-rich tasks. In recent years, learning-based approaches such as deep RL have emerged to learn complex nonlinear control policies directly from data. While RL has shown great success in challenging control problems such as gameplay [24, 10] and robotic manipulation [102], its application to real-world problems remains limited mainly due to its data inefficient and unstructured exploration behaviour. Prior work often focuses on sim-to-real transfer to mitigate these challenges. However, for highly complex or contact-rich tasks, it is often impractical to design accurate simulations that capture all necessary dynamics and interactions. As a result, direct learning on hardware sometimes becomes unavoidable. This introduces a critical challenge: the reliance of RL on random exploration can lead to unsafe behaviours, risking damage to both the robot and its environment. Addressing safe exploration is essential to enable safe and efficient RL training on hardware.

A prime reason for the random exploration behaviour is the task-agnostic architecture of state-of-the-art RL approaches [22], which lack the incorporation of prior knowledge on how to solve the task at hand. In contrast, control theory offers a rich set of methods for deriving near-optimal controllers based on first-principles modelling. These methods provide a decent baseline that works well when the system dynamics can be accurately modelled but fall short when the dynamics are complex or poorly understood. This motivates hybrid RL [44, 45], blending control priors with deep RL policies. Hybrid algorithms thus combine the informed behaviour of the prior with task-specific RL optimization to tackle complex, nonlinear problems.

As discussed in Section 4.2, the majority of prior work in hybrid RL [44, 45, 35, 46] proposes a fixed weighting between control prior and RL agent. A fixed blending, however, disregards the fact that the capability of the RL agent depends on training time and state. As more data is collected, the agent refines its behaviour, ultimately outperforming the prior across larger portions of the task domain. AHRL methods [103, 104] adapt the weighting between the RL agent and control prior based on the agent's confidence.

The CHEQ algorithm, as introduced in Chapter 4, dynamically adjusts the weighting based on the parametric uncertainty of a critic ensemble. On the simulated racing task, this algorithm has demonstrated significantly safer exploration behaviour, i.e., fewer vio-

lations of safety limits, and accelerated learning than traditional model-free RL, residual RL, and prior work in AHRL (cf. Section 4.7.2). The properties of AHRL methods, in general, and CHEQ, in particular, are promising for real-world applications and direct training on hardware. Hardware evaluations are critical, as real-world challenges like contact dynamics, friction, and state-dependent noise are difficult to simulate accurately. While some fixed-weight hybrid RL methods have trained on hardware [45, 35], the advancements of AHRL have so far been validated exclusively in simulation and with low-dimensional action spaces. We argue that the full potential and the true challenges of these algorithms can only be revealed through evaluation on a challenging hardware problem.

In this chapter, we focus on polishing a 3D object with a robotic arm, a task that exemplifies the challenges of contact-rich manipulation in industrial automation. Polishing requires continuous contact with a surface while adhering to specific force and velocity profiles, demanding precise control to ensure quality and consistency. VIC is particularly well-suited for such tasks [47, 48], as it enables robots to dynamically adapt their stiffness and damping to interact effectively with their environment. However, defining suitable variable impedance gains is inherently complex, and manual tuning becomes impractical in scenarios with high-dimensional or nonlinear dynamics. This makes robotic polishing a compelling benchmark for evaluating RL-based control strategies.

This work is the first to apply AHRL, specifically the CHEQ algorithm, to a hardware learning problem. The hybrid RL agent outputs the impedance gains and the end-effector position and orientation. This output is then fed into a cascaded impedance controller. The goal is to (i) investigate the potential of CHEQ for learning variable impedance gains and (ii) explore its capability to learn such a policy safely, directly on hardware.

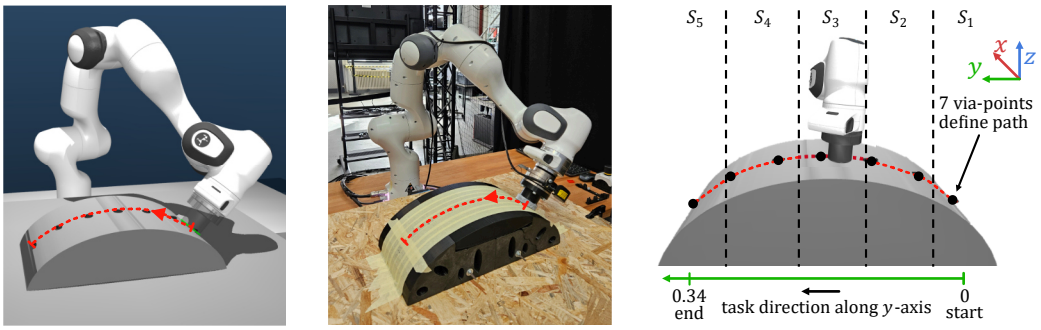
In simulation, we demonstrate that VIC significantly improves polishing behaviour. We compare standard RL with AHRL, showing that CHEQ can learn effective impedance gains while maintaining exploration within safety limits. While CHEQ cannot provide theoretical safety guarantees, the control prior guides the exploration behaviour to be close to the desired task. Finally, we deploy CHEQ on hardware and show that good polishing behaviour can be achieved within eight hours and only five failures, underscoring its efficiency and safety in real-world scenarios.

5.2. Related Work

In this section, we discuss relevant prior works focusing on RL-based VIC and polishing. For a discussion of prior work in hybrid RL, we refer to Section 4.2.

Buchli *et al.* [126] first demonstrated that RL can be used to learn VIC. Some works have focused on learning VIC from data gathered on hardware. Luo *et al.* [127] combine a higher model-based RL controller with a lower-level impedance controller, showing that this approach can learn real-world insertion tasks data efficiently. In [128], a model-based RL approach is trained to model human-robot interaction dynamics, and then a controller based on model predictive control (MPC) is used to optimize the VIC parameters. Anand *et al.* [18] learn VIC with a model-based RL agent through real-world interactions. They evaluate their approach on water-pouring with a robotic arm. While prior work emphasizes data-efficient model-based RL, both model-based and model-free algorithms are commonly used in RL. Here, we use model-free hybrid RL to remain safe during exploration.

Some prior work specifically focused on learning VIC for robotic polishing. Martín-Martín *et al.* [47] investigated vision-based surface wiping. While we aim to track a specific target force, they want to achieve an arbitrary force sufficient to wipe dirt. They trained in simulation and then transferred the policy to a real robot. Bogdanovic *et al.* [48] examined the simulated task of a circular end-effector motion while in contact with a table and applying a desired constant vertical force. Anand *et al.* [129] consider wiping a 2D surface with a robotic arm, demanding force tracking in the vertical direction and motion tracking in the remaining five DOF. The model-based RL agent tunes the two-dimensional impedance gains while a cascaded controller handles motion tracking. This approach was directly trained on hardware, focusing on data efficiency. All approaches above consider wiping a 2D surface, which reduces the action space of the RL agent substantially. In this work, we want to use a more complex 3D task which requires adapting impedance gains for all Cartesian axes and orientations. Both Martín-Martín *et al.* [47] and Bogdanovic *et al.* [48] investigate different choices of action spaces and conclude that a cascaded RL approach with variable impedance in end-effector space is most suited for the considered wiping task. We thus use this approach as our control pipeline and investigate the potential of AHRL for training directly on hardware.



(a) (b) (c)

Figure 5.1.: Figures a and b show the robot setup in simulation and on the real hardware. The robot's task is to polish along a path r^{path} defined via seven distinct via-points. The polishing path runs along the y-axis. For our later analysis of fixed-gain controllers, we split the task into five segments shown in Figure c.

5.3. Robotic Polishing

In this chapter, we explore robotic polishing as a continuous contact force control task that is solvable with VIC. A non-rotating polishing tool is attached to a 7-DOF Franka Panda robotic arm, with a 3D-printed bridge as the workpiece. The workpiece is positioned in front of the robot arm. The task involves polishing along a path r^{path} defined via seven distinct via-points. Figure 5.1 shows the simulated and real robot setup and the desired polishing path.

As discussed in Section 5.2, similar setups have been considered before [47, 48, 129], but only for 2D end-effector motions. We specifically choose a 3D workpiece for its multiple challenges. First, the 3D movement showcases the need for variable impedance. Further, the task's movement direction does not align with Cartesian axes, and the workpiece features curvature variation along the y-axis, adding complexity.

This research aims to achieve fine material removal through precise, low-force polishing, maintaining consistent material removal. Assuming a constant contact area, the material removal rate [130] is

$$\frac{dh}{dt} = c \cdot F_N \cdot v_T, \tag{5.1}$$

with material thickness h, normal force on the workpiece F_N, translational velocity v_T along r^{path}, and c a proportionality constant. We refer the reader to Appendix C.1.1 for a more detailed derivation of this equation. Consistency in material removal rate thus requires maintaining constant force and translational velocity throughout the task.

While direct force control approaches do exist, they suffer from several challenges.

Measuring forces directly requires highly accurate force-torque sensors. These sensors can be noisy, expensive, and have limited resolution and bandwidth. Noise in the measurement can make the control unstable. Also, forces may vary rapidly and unpredictably due to compliance or surface properties, making them hard to control precisely [131]. In contrast, impedance control achieves a desired relationship between position, velocity, and force, i.e., compliant behaviour, enabling natural adaptation to environment variations even with imperfect models. It also depends less on high-bandwidth force measurements, leveraging position or velocity data alongside lower-frequency, lower-precision force-torque sensing. As noted in Section 5.2, prior studies [47, 48] have shown that variable impedance in end-effector space is particularly effective for continuous contact polishing tasks. Accordingly, we adopt this approach.

5.4. Learning Variable Impedance via CHEQ

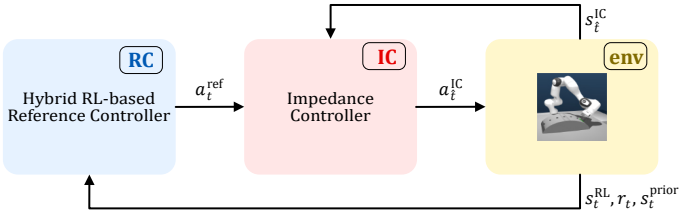
The control pipeline (cf. Figure 5.2) comprises a hybrid RL-based reference controller RC and cascaded impedance controller IC in end-effector space [132]. The RC is based on our AHRL algorithm, CHEQ. This controller outputs a_t^{ref}, including the next Cartesian end-effector pose and variable impedance gains. The task of the IC is to map the end-effector pose error to desired joint torques while keeping compliant end-effector behaviour. The output of the IC is then $a_{\hat{t}}^{\text{IC}}$.

5.4.1. Impedance Controller (IC)

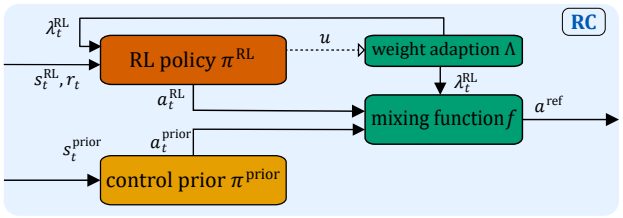
We specify the robot end-effector pose $p \in \mathbb{R}^6$ by its position $x = (x_x, x_y, x_z)$ in Cartesian space and orientation $\phi = (\phi_x, \phi_y, \phi_z)$ in Euler angles. The complete end-effector pose can be written as $p = (x, \phi)$. We denote the pose error between the target and the current pose at time t as $\Delta p = p^{\text{ref}} - p$. End-effector velocity and acceleration are defined as $\dot{p}$ and $\ddot{p}$ and their errors as $\Delta\dot{p}$ and $\Delta\ddot{p}$ respectively. We denote the joint space configuration by $q \in \mathbb{R}^7$. As discussed in Section 2.4.1, the goal of impedance control is to model the behaviour of a mass-spring-damper system in task space such that

$$M^{\text{eef}}\Delta\ddot{p} + D\Delta\dot{p} + K\Delta p = f_{\text{ext}}. \tag{5.2}$$

Here M^{eef}, D and $K \in \mathbb{R}^{6\times6}$ represent the Cartesian inertia, damping and stiffness matrices, and f_{ext} is a force applied to the robot. Commonly, $M^{\text{eef}} = 0$ is assumed to mitigate noisy acceleration measurements [131]. Moreover, we set $\dot{p}^{\text{ref}} = 0$. Under these assumptions, the desired joint torques to achieve this impedance behaviour can be



(a)



(b)

Figure 5.2.: Figure a shows the complete control pipeline with higher-level reference controller RC and lower-level impedance controller IC. Figure b gives a detailed view of the RC, an adaptive hybrid RL agent based on CHEQ.

calculated as

$$\tau_{\text{IC}}(q) = J(q)^{\intercal}(K\Delta p - D\dot{p}), \tag{5.3}$$

where J is the Jacobian matrix. The controller also applies a null-space torque $\tau_{\text{null}}(q)$, Coriolis and centrifugal forces $C(q, \dot{q})$, and gravity compensation $g(q)$. The IC then outputs

$$a_{\hat{t}}^{\text{IC}} = \tau_{\text{IC}}(q_{\hat{t}}) + \tau_{\text{null}}(q_{\hat{t}}) + C(q_{\hat{t}}, \dot{q}_{\hat{t}}) + g(q_{\hat{t}}). \tag{5.4}$$

5.4.2. Hybrid Reinforcement Learning-based Reference Controller (RC)

We use the CHEQ algorithm as introduced in Chapter 4 as the RC, adaptively combining actions from an RL agent and a nominal controller.

Reinforcement Learning Agent

First, we describe the RL agent for our hybrid setup. CHEQ uses the popular model-free soft-actor critic (SAC) architecture [22] as a base RL agent and combines this with an ensemble of E critic networks.

To reduce the action space, we define the stiffness matrix as a diagonal matrix $K = \text{diag}(k_x, k_y, k_z, k_{\phi_x}, k_{\phi_y}, k_{\phi_z})$ and the damping by a damping factor ζ proportional to K

such that $D = 2\zeta \cdot \sqrt{K}$. The agent then outputs

$$a_t^{\mathrm{RL}} = (\Delta p_t^{\mathrm{RL}}, K_t, \zeta_t). \tag{5.5}$$

For the state space of the agent, we define $\tilde{p} = (x, \phi^{\mathrm{quat}})$ where $\phi^{\mathrm{quat}} \in \mathbb{R}^4$ describes the orientation in unit quaternions. The complete state space can then be written as

$$s_t^{\mathrm{RL}} = (q_t, \cos(q_t), \sin(q_t), \dot{q}_t, \tilde{p}_t, \dot{\tilde{p}}_t, F_t, \Delta r_t^{\mathrm{path}}) \in \mathbb{R}^{74}. \tag{5.6}$$

Here, $F_t \in \mathbb{R}^3$ is the contact force of the end-effector, and $\Delta r_t^{\mathrm{path}} \in \mathbb{R}^{5\times 6}$ describes the position and velocity errors of the end-effector to the next five points along the path r^{path}. The contact force is sensed via a force-torque sensor.

At each time step t, the reward is a weighted sum of terms designed to encourage path-following, as well as velocity and force tracking such that

$$r_t = c_{c^\perp} r_{c^\perp,t} + c_{c^\parallel} r_{c^\parallel,t} + c_d r_{d,t} + c_v r_{v,t} + c_f r_{f,t}. \tag{5.7}$$

Here, $r_{c^\perp}$ and $r_{c^\parallel}$ reward path-following in perpendicular and parallel path direction, r_d directional alignment, and r_v and r_f velocities and forces close to the respective targets. Each term is weighted with a constant $c_i \in [0, 1]$, and we choose these constants such that $\sum_i c_i = 1$. Since we normalize all reward terms, this choice of weighting factor makes it easy to interpret the relation between each reward term. Importantly, to discourage truncation, we give a positive reward per time step instead of a penalty. For this, the deviation from the target value is mapped towards a positive reward. Further details on this can be found in Appendix C.2.1. In addition, we give a penalty r_{trunc} for episode truncation and a final termination reward r_{term} when all via-points are successfully wiped.

Nominal Control

The nominal controller outputs a_t^{prior}, including the next desired position and orientation and fixed predefined impedance gains

$$a_t^{\mathrm{prior}} = (\Delta p_t^{\mathrm{prior}}, K^{\mathrm{prior}}, \zeta^{\mathrm{prior}}). \tag{5.8}$$

Following a simple and established approach, we define our reference path r^{path} by interpolating between seven predefined via-points for the position [131] and using the spherical linear interpolation (SLERP) method [133] for the orientation. As the action

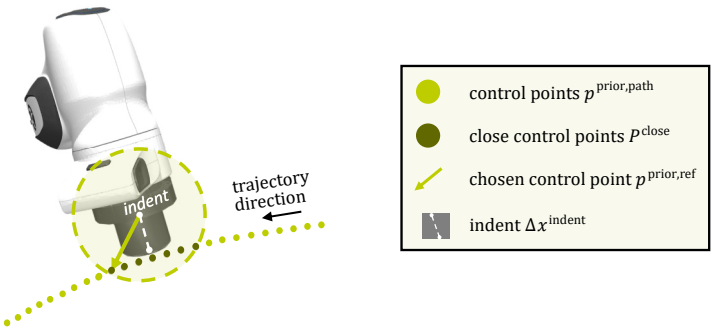


Figure 5.3.: Generation of the next desired control point. The next desired control point $p^{\text{prior,ref}}$ is chosen as the farthest point along the task direction out of the set of close control points.

space is defined in task space, the path r^{path} is defined via Cartesian positions and orientations. For the positions, we apply the Via-Points algorithm [131] using a third-degree interpolation method. The algorithm requires the specification of N via-points with their corresponding positions $\tilde{x}_n \in \mathbb{R}^3$ and directional vectors $\tilde{d}_n \in \mathbb{R}^3$ of the path, for all $n \in N$ via-points. The directional vectors $\tilde{d}_n$ are necessary hyperparameters for the Via-Points algorithm that represent the current path direction at the position of each via-point. Changing the directional vectors of a via-point leads to different curvatures of the resulting positional path. The position and the directional vectors act as boundary constraints for the path, resulting in a well-defined analytical problem. The algorithm then outputs a piecewise function between each neighbouring via-point pair. For the orientation, we apply the SLERP algorithm for quaternions [133]. For SLERP, we specify the orientation of the end-effector $\tilde{\phi}_n \in \mathbb{R}^4$ at the positions of the N via-points. SLERP then results in the definition of piecewise functions between each orientation specification $\tilde{\phi}_n$ and $\tilde{\phi}_{n+1}$. After creation, the functions for the position and orientation are combined to form a coherent and continuous path function r^{path}. This path is then discretized into $M \gg N$ equally spaced control points $p_m^{\text{prior,path}} \in \mathbb{R}^7$.

The control points build the foundation for the motion control, which outputs the action a_t^{prior}. Given the current pose $p_t = (x_t, \phi_t)^\intercal$ of the robotic arm, we search for close control points that lie within a predefined radius $a_r \in \mathbb{R}$. Close control points can be grouped in the set P^{close}. From this set, we choose the control point $p^{\text{prior,ref}}$ that is furthest along the task direction. The chosen control point is then used as a reference for the controller. Figure 5.3 visualizes this process.

After determining the reference control point $p^{\text{prior,ref}} = \left(x^{\text{prior,ref}}, \phi^{\text{prior,ref}}\right)$, the ac-

tion a_t^{prior} is computed using the following two equations for position and orientation

$$\Delta x_t^{\text{prior}} = x_t - x^{\text{prior,ref}}, \tag{5.9}$$

$$\Delta \phi_t^{\text{prior}} = \phi^{\text{prior,ref}} \cdot \phi_t^{-1}. \tag{5.10}$$

Since the positional part of the path is defined directly on the workpiece surface, we shift the robot's position x_t to be located inside the end-effector. For this purpose, we define an indentation parameter Δx^{indent} that describes the absolute distance from position x_t to the end-effector's surface. Changing the indentation modifies the force behaviour, with higher values leading to a stronger movement toward the workpiece.

CHEQ

As introduced in Chapter 4, CHEQ adaptively blends the action of the RL agent a_t^{RL} and the nominal controller a_t^{prior} (cf. Figure 5.2b). Following the definition in the previous chapter (cf. Equation (4.2)), the combined action a_t^{ref} is then a linear weighted sum

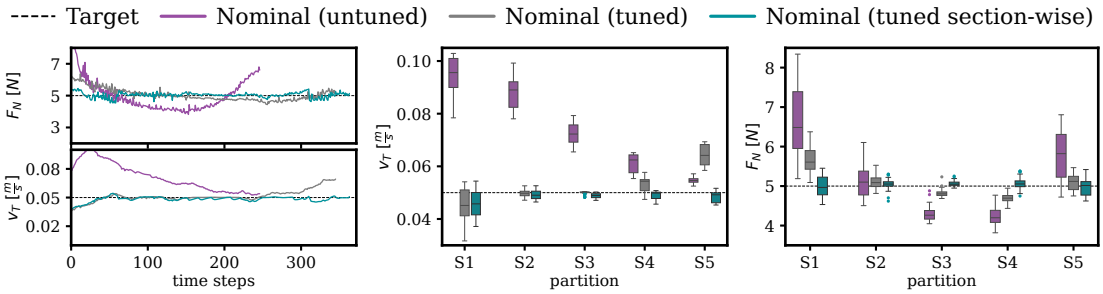
$$a_t^{\text{ref}} = (1 - \lambda_t^{\text{RL}}) \cdot a_t^{\text{prior}} + \lambda_t^{\text{RL}} \cdot a_t^{\text{RL}}, \tag{5.11}$$

with the adaptive weight $\lambda_t^{\text{RL}} \in [0, 1]$. The weight is adapted based on the uncertainty $u(s_t^{\text{RL}}, a_t^{\text{RL}}, \lambda_t^{\text{RL}})$ of an ensemble of E Q-networks Q_{θ_e}, with $e = 1 \dots E$. To simplify notation we introduce $z_t = (s_t^{\text{RL}}, a_t^{\text{RL}}, \lambda_t^{\text{RL}})$. The uncertainty is estimated as the standard deviation of the E critic predictions $u(z_t) = (\frac{1}{E} \sum_{e=1}^{E} (Q_{\theta_e}(z_t) - \mu(z_t))^2)^{-1}$ with $\mu(z_t) = \frac{1}{E} \sum_{e=1}^{E} Q_{\theta_e}(z_t)$. The weighting factor λ_t^{RL} is then a clipped linear function between a minimum uncertainty $u_{\min}$ and maximum uncertainty $u_{\max}$.

5.5. Simulation Study

We first conducted a simulation study to show that (i) variable gains benefit the polishing task and that (ii) CHEQ can learn such gains with safe and efficient exploration. Then, we evaluated a trained agent on zero-shot transfer polishing objects. Our main findings are

1. Variable impedance gains improve force and velocity tracking on the 3D workpiece.
2. CHEQ safely and efficiently explores the task space during training and, with this, can learn a good polishing policy.



(a) Force-velocity behaviour. (b) Velocity per section. (c) Force per section.

Figure 5.4.: Comparison of force and velocity over one episode in Figure a and boxplots for the five sections in Figures b and c. The fixed gain controller tuned with BO shows increased performance, and tuning per section further improves behaviour.

3. On zero-shot transfer tasks, the trained agent detects workpiece changes and refers responsibility to the nominal controller.

5.5.1. Evaluating the Need for Variable Impedance

To show the benefits of variable impedance, we first evaluate the nominal controller alone using tuned IC gains. We investigate the behaviour of fixed control gains for the whole task and partly fixed gains defined per task section.

Experimental Setup

We tune the gains of the IC using Bayesian Optimization (BO) in Weights&Biases [134]. Since the important impedance gains are expected to be k_y and k_z and the damping factor ζ, we focus on these parameters in our optimization. To ensure a fair comparison of learning capabilities, the search space is defined to match the action space of the RL agent. Specifically, the search space is spanned as $k_y \in [50, 200], [\mathrm{N\,m^{-1}}]$, $k_z \in [30, 130], [\mathrm{N\,m^{-1}}]$, and $\zeta \in [0.8, 1.2]$.

The other impedance gains are fixed at values high enough to enable the nominal controller to perform the movement. These are set to $k_x = 500\,\mathrm{N\,m^{-1}}, k_{\phi_x} = k_{\phi_y} = k_{\phi_z} = 500\,\mathrm{N\,rad^{-1}}$. We rank all runs according to a predefined task score:

$$\text{score} = w_c \cdot \frac{N_{\text{con}}}{N_{\text{tot}}} + w_w \cdot \frac{n_{\text{wipes}}}{7} - w_f \cdot \overline{\Delta F_N} - w_v \cdot \overline{\Delta v_T}, \tag{5.12}$$

where $N_{\text{tot}}, N_{\text{con}}$ are the numbers of total episodic steps and steps with contact to the workpiece, n_{wipes} is the number of wiped via-points, and $\overline{\Delta F_N}, \overline{\Delta v_T}$ represent the average

deviations of force and velocity from the specified targets. The weighting factors are configured as $w_c = 0.18$, $w_w = 0.52$, $w_f = 0.03$, and $w_v = 0.27$.

The run parameters yielding the highest score are chosen as the impedance gains of our tuned baseline, referred to as the *Nominal (tuned)* controller. This controller was identified after approximately 500 runs.

Next, we partition the workpiece into 5 sections (see Figure 5.1) and applied BO to optimize individual gains per section. Gains for each section were tuned iteratively using BO, incorporating the best-found gains from previous sections. This procedure is less data efficient, requiring 500 episodes per section. The resulting controller is referred to as *Nominal (tuned section-wise)*.

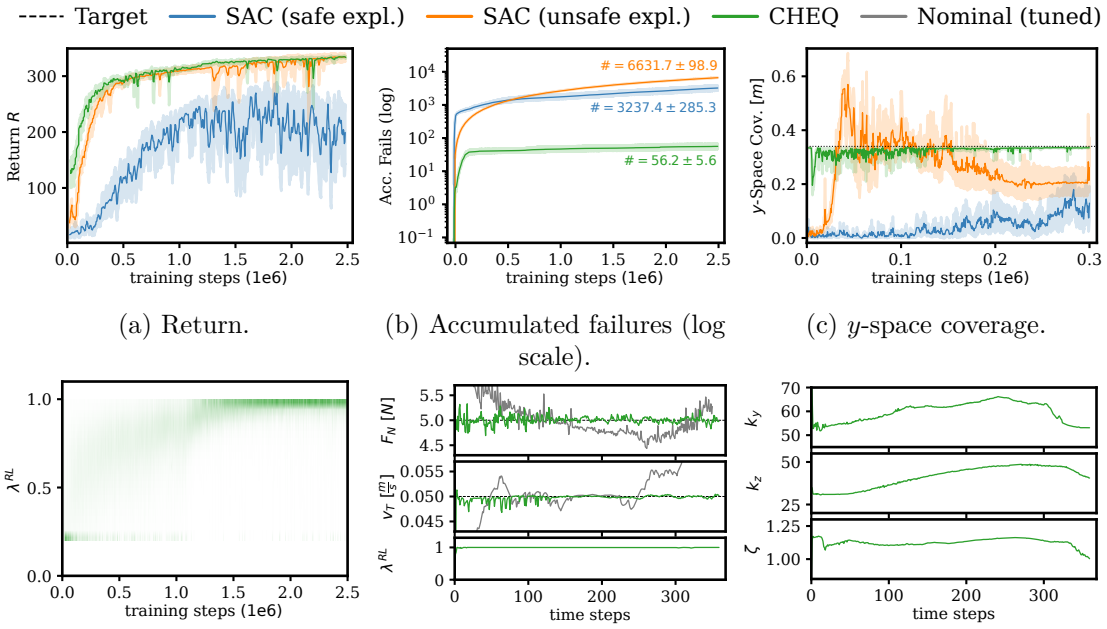
Lastly, we define an untuned control baseline by rounding up the values of the *Nominal (tuned)* controller. We refer to this controller as *Nominal (untuned)*. The selected parameters for all three controllers are provided in Appendix C.2.2.

Results

Figure 5.4 compares the force-velocity behaviour of the three controllers over an episode, along with boxplots for each of the five object sections. The dotted line marks the force and velocity targets of $5\,\mathrm{N}$ and $0.05\,\mathrm{m\,s^{-1}}$. The untuned controller fails to meet the targets, while the BO-tuned fixed gain controller performs significantly better but shows larger errors at the task's beginning and end, where the object curves steeply. The section-wise tuned controller effectively reduces errors in these regions, showing that variable impedance gains can yield superior performance for the 3D polishing movement. This motivates learning more sophisticated state-dependent impedance gains. However, dividing the controller into arbitrary sections is impractical with manual methods. Further, computational efforts grow linearly with the number of sections. RL offers the potential to tune state-dependent gains.

5.5.2. Learning Variable Impedance Control with Reinforcement Learning

Next, we evaluate the effectiveness of RL in learning state-dependent VIC. Our findings show that CHEQ not only enables safe exploration during training but also successfully learns effective polishing. In contrast, standard RL only learns when unsafe behaviour is permitted.



(a) Return.

(b) Accumulated failures (log scale).

(c) y-space coverage.

(d) Weighting factor λ^{RL} distribution.

(e) Force-velocity-lambda behaviour.

(f) Impedance gains.

Figure 5.5.: Figures a-c show a comparison of return, fails and y-space coverage of CHEQ and the SAC variants. CHEQ achieves the highest performance with only 56 fails. Over training, the distribution of the weighting factor λ^{RL} rises to 1 as shown in Figure d. Figures e and f show the trained agent after 2.5×10^6 steps. The agent achieves good polishing behaviour by adapting the gains.

Experimental Setup

We compared two SAC variants, *SAC (unsafe exploration)* and *SAC (safe exploration)*, along with our hybrid *CHEQ* method. The latter two enforce safe exploration by applying velocity and acceleration limits, bounding the end-effector's position, and constraining orientation to task-appropriate values. We provide details on our safety limits in Appendix C.1.2. Episodes are truncated upon constraint violations to ensure minimal hardware wear and to safeguard nearby humans. For *SAC (unsafe exploration)*, all restrictions are lifted. The CHEQ agent uses the *Nominal (untuned)* controller as its prior, arguing that an easy-to-define, untuned controller suffices for hybrid RL.

Results

Figures 5.5a-c show the performance of RL agents during training, averaged over 10 runs with 95 % quantiles. *SAC (safe exploration)* achieves low returns, failing to learn the

task effectively. In contrast, *SAC (unsafe exploration)* learns the task but accumulates over 6×10^3 failures. We count only the first violation per episode, though actual failures are much higher, as we do not truncate episodes. This high failure rate makes it unsuitable for hardware due to excessive wear and tear. The *CHEQ* agent achieves high returns, slightly outperforming unsafe SAC while reducing failures by three orders of magnitude. Differences between approaches are evident when investigating the y-space coverage (see Figure 5.5c). Since the polishing path is along the y-axis (see Figure 5.1), y-space coverage is a proxy for task space exploration. In the first 300×10^3 training steps, the safe SAC agent covers less than half the task, as safety constraints and random exploration limit progress. The unsafe SAC agent covers the full space but overshoots the endpoint. For CHEQ, the nominal controller guides exploration efficiently towards the full task space, and the small λ^{RL} at the beginning ensures the agent uses just enough random exploration to learn the task safely and efficiently. Looking at the λ^{RL}-distribution (see Figure 5.5d), the agent first learns a small residual on top of the nominal controller. This already provides good polishing behaviour and high returns. Over time, the agent transitions to full RL control, continuously improving performance. This induces a learning curriculum, ensuring safe exploration. Compared to the tuned baseline (Figure 5.5e), CHEQ achieves superior force and velocity tracking by adapting gains dynamically throughout the task (Figure 5.5f). Overall, CHEQ demonstrates significantly safer learning than standard RL while outperforming the tuned baseline.

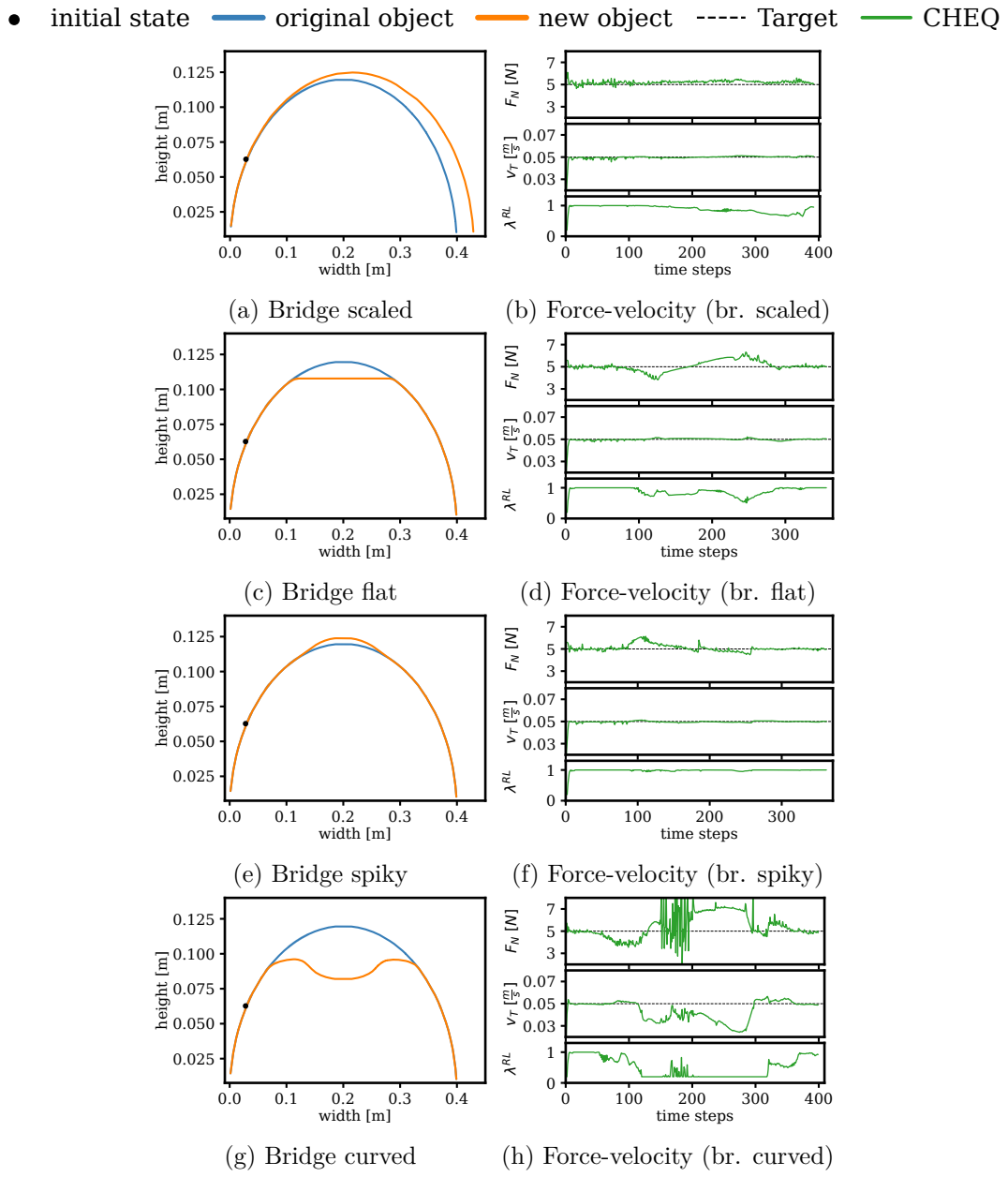
5.5.3. Zero Shot Transfer

In the following, we use the best-trained CHEQ policy and transfer it to four newly generated workpieces unseen during training. Each workpiece is a modified version of the original bridge workpiece. Figure 5.6 shows the profile difference compared to *bridge*.

We consider two smaller modifications, a scaled version of the original bridge (see Figure 5.6a) and a version with a slightly elevated middle part (see Figure 5.6e). Figures 5.6b and 5.6f show the force, velocity and weighting behaviour of the trained policy on these two objects. Noticeably, the model transfers for the scaled and spiky workpieces show good polishing behaviour close to the force and velocity targets. The weights λ^{RL} indicate that the model is mostly certain along the complete task. Especially for the scaled workpiece, the difference to the original workpiece is correctly identified, and the weight decreases over the episode.

We next consider two larger modifications, a flattened workpiece (see Figure 5.6c) and a workpiece with an inward curve (see Figure 5.6g). These transfer objects are

more challenging since the positional distances of these alterations are generally larger, and the end effector has to apply different orientations compared to the learned task. The force, velocity and weighting behaviour for these transfers can be seen in Figures 5.6d and 5.6h. The trained CHEQ agent correctly identifies the modified parts of the task, which results in low mixing weights. On the outer parts of the task, where no modifications are performed, CHEQ remains a high weighting. While the modifications significantly impact the final polishing performance, CHEQ maintains safety for both difficult transfer tasks as all of the contact forces of the episodes are below our predefined safety limits. Moreover, all the presented transfer objects result in continuous contact with the workpiece, which indicates promising conditions for continued learning.



(a) Bridge scaled (b) Force-velocity (br. scaled)

(c) Bridge flat (d) Force-velocity (br. flat)

(e) Bridge spiky (f) Force-velocity (br. spiky)

(g) Bridge curved (h) Force-velocity (br. curved)

Figure 5.6.: We show the profiles of the zero-shot transfer objects in orange and the original bridge object in blue. Figures b, d, f and h show the force, velocity and weighting behaviour of the trained policy on the respective zero-shot transfer objects. CHEQ detects whenever the object differs from the training object and reacts with a decrease in the weighting factor λ^{RL}.

5.6. Case Study: Real-world 3D polishing

In this section, we address the challenges and solutions of training CHEQ on hardware, demonstrated through a 3D polishing task with a robotic manipulator. This task, which is representative of contact-rich manufacturing problems, marks CHEQ 's first successful hardware implementation. We first describe our experimental setup and results and then discuss further details on the challenges that had to be overcome to achieve these results.

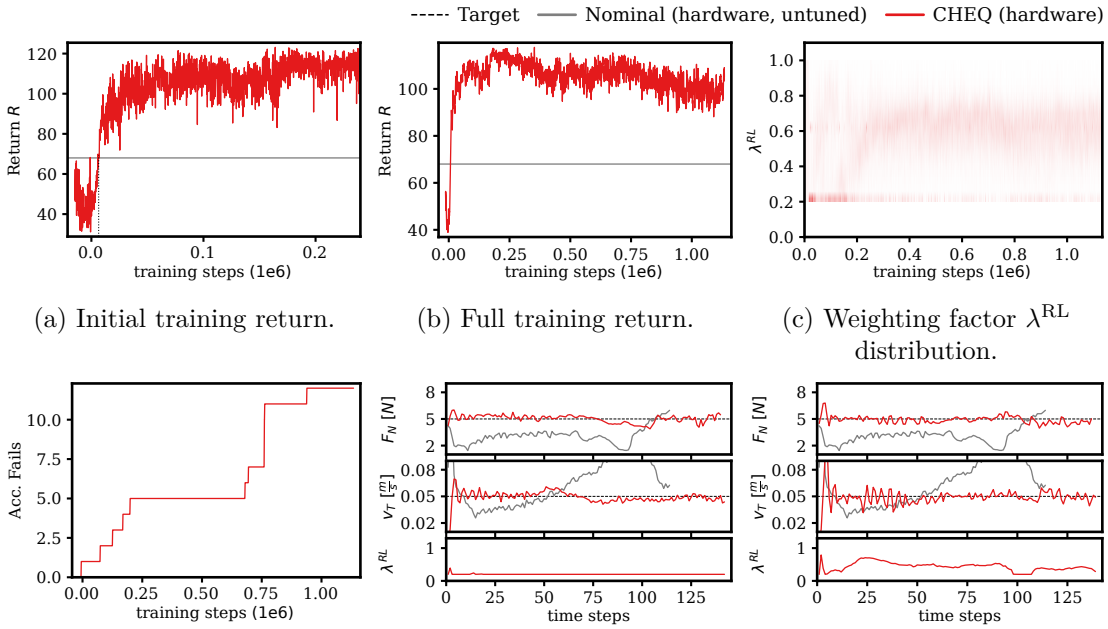
Experimental Setup

Additional challenges need to be considered to achieve stable training on hardware. On our hardware setup, we can achieve a maximum control frequency of 20 Hz while simultaneously dealing with measurement noise in the robot state and force measurements. Further, our simulation results revealed one critical aspect specific to CHEQ. In the initial training phase, the weighting factor fluctuates greatly between time steps, resulting in highly fluctuating IC gains. In simulation, this does not pose a problem. On real hardware, however, the gain fluctuations lead to a chattering end-effector motion. This doubly affects training as it complicates the control and increases the motion-dependent force sensor noise. To mitigate this, we use an average of ten time steps to compute our uncertainty, leading to smoother weights. Combining the mentioned challenges, we find that data collected on our hardware is less reliable than in simulation. This might lead to unstable training. CHEQ provides a mechanism to mitigate this by slowing the curriculum. Thus, for our hardware experiments, we set more conservative uncertainty limits. This reduces the agency of the RL agent in the beginning and leads to more stable, safer, albeit slower, learning. We use an untuned prior (*Nominal hardware (untuned)*) and set a UTD ratio of two to account for the higher cost of environment steps on hardware.

Results

On hardware, the return improves quickly, surpassing the *Nominal hardware (untuned)* controller after 6.5×10^3 steps. Even this early in the training process (250×10^3 steps and 8 hours of training), good polishing behaviour is achieved. The trained RC maintains smooth force and velocity close to the targets, achieving this with a low weight around 0.2 (see Figure 5.7e). Importantly, this behaviour is achieved while accumulating only five failures.

Figure 5.7b demonstrates that longer training does not improve the return. Addition-



(a) Initial training return.

(b) Full training return.

(c) Weighting factor λ^{RL} distribution.

(d) Accumulated fails (log scale).

(e) Force-velocity-lambda eval. ($\sim 90 \times 10^3$)

(f) Force-velocity-lambda eval. ($\sim 930 \times 10^3$)

Figure 5.7.: We show a close look at the initial training return in Figure a. Longer training does not further improve the return (cf. Figure b) but increases the weighting factor (cf. Figure c). Figure f shows the force and velocity behaviour of an early evaluation run after $\sim 90 \times 10^3$ training steps and a later evaluation run after $\sim 930 \times 10^3$ training steps. In both cases, the agent achieves good polishing behaviour. Further, the accumulated training fails are shown in Figure d.

ally, the force, velocity, and weight behaviour of an evaluation run after more training steps are presented in Figure 5.7f. This illustrates that, although the agent acts with a higher weighting λ^{RL}, it has not further improved its polishing behaviour. In our adaptive setting, the agent has to learn the complete state, action and weight space. Whenever the agent learns a higher weight, the return drops and rises again once the agent has mastered this part. Figure 5.7c shows the distribution of the weighting factor over training. With the slow curriculum, the agent has not yet learned the complete weight space after full 1.25×10^6 training steps. Still, we see no benefit in further training as our early results show that a small residual on the untuned controller suffices for effective polishing. Even though a residual term is enough to learn good polishing behaviour in this setting, CHEQ still works differently from traditional fixed-weight approaches. In Chapter 4, we demonstrated that using a fixed weight compromises safe learning and performance. Comparable final performance can only be achieved when the

weight is optimally tuned for a specific scenario and control prior. However, even then, training still benefits from the curriculum induced by adaptive weighting. We conclude that CHEQ learns good polishing behaviour safely and efficiently on hardware. The complete progression of training failures can be found in Figure 5.7d.

5.6.1. Real-world Reinforcement Learning Challenges and Solutions

Additional challenges needed to be considered to achieve stable training on hardware. In addition to the already mentioned fluctuations in λ^{RL}, we found (i) control frequency and (ii) sensor noise to be most important.

Simulation frameworks for RL, such as MuJoco [32], specify a control frequency, which determines the actual control frequency f^{RC}, i.e., the time between control actions as well as the waiting time $\Delta t_{(a_t, s_{t+1})}$ between applying an action a_t and collecting the next state s_{t+1}. These simulations assume no time spent on communication, action computation, or gradient updates, i.e., $\Delta t_{(a_t, s_{t+1})} = (f^{\mathrm{RC}})^{-1}$. These assumptions are unrealistic in the real world, and the waiting time can only use up some fraction of the control frequency. This poses an inherent dilemma, as a faster control frequency can only be achieved with a lower waiting time. Sufficient waiting times, however, are critical for allowing the RL agent to observe the consequences of its actions. To balance hardware limitations with the need for adequate waiting time, we selected a control frequency of 20 Hz and a waiting time of 0.033 s. In simulation, a higher control frequency of 50 Hz allowed for improved responsiveness to force changes. In addition, we defined a split actor and learner setup, similar to Luo *et al.* [19], to ensure that gradient updates do not lower the control frequency. This also allows the environment to reset parallel to the gradient steps.

The second challenge is the measurement of noise in the robot state and the force-torque sensor. In the latter, we encountered drift and motion-dependent sensor noise. This high noise level, combined with the low control frequency, complicates the training on hardware. We applied a low-pass filter with a cut-off frequency of 35 Hz. This simplified the optimization landscape and enabled the agent to focus on aspects within control.

Combining the abovementioned challenges, we find that data collected on our hardware setup is less reliable than in simulation, leading to unstable training. For our hardware experiments, we mitigate this by setting the uncertainty limits to be more conservative. This results in a heavier reliance on the nominal controller, leading to more stable, albeit slower, learning.

5.7. Conclusion

In this chapter, we applied the CHEQ algorithm, a novel AHRL algorithm, to robotic polishing. In a simulation study, we highlighted the benefits of variable impedance and showed that CHEQ outperforms standalone RL. The adaption mechanism ensured safe exploration by gradually transitioning from control priors to autonomous operation. On hardware, CHEQ achieved effective polishing performance. This study demonstrates the potential of AHRL for hardware training, combining safety with performance.

6. Conclusion

This chapter summarizes the results of this thesis and highlights future research directions.

6.1. Summary

Industrial applications of RL face significant challenges due to the inherent complexity and limitations of real-world environments. In this thesis, we identified three primary challenges specific to the real-world application of robotic manufacturing. First, data collection for such use cases often requires the use of physical hardware, imposing stringent requirements on sample efficiency (G1) and the safety of the data collection process (G2). Second, real-world environments are frequently characterized by uncertainties and non-stationarities (G3, G4). Finally, facilitating broader adoption of RL methods outside the research community necessitates improving the usability of RL pipelines through enhanced interpretability (G6) and user acceptance (G7). This dissertation proposes that carefully integrating domain-specific priors can address these challenges and bridge the gap between RL research and industrial applications.

The first part of this work (C1, C2) addresses uncertainties (G3) and improves the usability (G5, G6) of RL in production by leveraging the prior that spatial features are critical for effective robotic control. To this end, we developed a novel evaluation metric for SAEs, neural networks that extract spatial features from images. This metric quantifies how well an SAE tracks task-relevant objects, offering a computationally lightweight method to predict downstream RL performance. By enabling practitioners to select suitable feature extractors before engaging in expensive RL training, this approach improves sample efficiency (G1) and reduces the time required for RL model development. Experimental results confirm that well-performing SAEs provide effective state representations for RL-based robotic manipulation tasks.

The second part of this dissertation (C3, C5) introduces a novel AHRL method that blends an RL policy with a control prior. Traditional RL methods lack prior knowledge and rely heavily on random exploration, making them inefficient and unsafe

for real-world applications (G1, G2). To address this, the proposed CHEQ algorithm dynamically adjusts the weighting between the RL agent and the control prior using uncertainty estimation from a critic ensemble. This integration of prior knowledge safely guides the learning process, enabling the RL agent to transition gradually from the prior towards autonomous decision-making. The CHEQ algorithm, evaluated on a simulated car racing task, demonstrates improved sample efficiency (G1), reduced failure rates (G2), enhanced generalisation (G4), and increased usability (G5, G6).

To assess the real-world applicability of the CHEQ algorithm, the final part of this thesis (C6, C7) applies CHEQ to robotic polishing, a contact-rich, dynamic task representative of industrial automation. CHEQ successfully learns variable impedance gains, a key factor in achieving stable force and velocity tracking during polishing. This enables the agent to adapt to surface variations while maintaining safe exploration (G2), where safety is understood not as formal guarantees but as ensuring exploration remains within practical limits. CHEQ is successfully trained directly on robotic hardware, overcoming environment uncertainty (G3) and non-stationarity (G4). It demonstrates sample efficiency (G1) and safety (G2) in a real-world setting, making it a promising candidate for future industrial RL applications.

In summary, this thesis tackles critical challenges in applying RL to complex, real-world robotic production environments. Together, the proposed strategies demonstrate that the proper use of priors enables RL methods to be safe and generalizable, paving the way for their practical application in industrial automation. Continued efforts in this direction promise to enhance the safety, efficiency, and interpretability of RL for robotic systems, driving further advancements in industrial automation.

6.2. Future Work and Discussion

This thesis explores state space extraction, AHRL and robotic polishing, aiming to advance the application of RL in flexible production scenarios. The contributions presented towards this goal open up several promising directions for further exploration.

One important avenue for future work is the evaluation of other state space extractors. In our study, we focused primarily on spatial autoencoders, as positional information is ubiquitous in robotic state representations. However, the expanding use of foundation models across various domains has revolutionized many supervised learning tasks, including vision [135, 136]. Recent advances in robotic-specific foundation models [137, 138, 139] and the emergence of large-scale robot datasets [140, 141, 142] further underscore the potential for leveraging large datasets to construct world representations.

While the trend is toward training ever-larger models on vast datasets, it is important not to rely solely on data volume to ensure well-designed state space representations. Establishing robust evaluation metrics for transformer models could significantly enhance the data efficiency of downstream RL and simplify the troubleshooting process during training. Ideally, these metrics would be physically meaningful, allowing process experts to understand and interpret them easily. Addressing this challenge will likely require a multidisciplinary effort, bringing together expertise from process engineering, explainable artificial intelligence, foundation models, and RL.

A further line of investigation is the exploration of AHRL that can incorporate different types of prior policies. The results in this thesis demonstrate the potential of combining RL with a control prior. This approach is flexible, and other priors such as expert demonstrations [143] or pre-trained policies [144, 46] could be integrated depending on the task and available data. Just as the choice of prior is flexible, so too is the selection of the RL agent. Although our implementation of the CHEQ algorithm relies on the model-free SAC architecture, the underlying framework is agnostic to the choice of RL agent as long as the agent's uncertainty can be measured. In this context, model-based RL methods [145, 146, 147, 148] are particularly attractive due to their inherent sample efficiency when compared to model-free approaches [149]. In particular, dyna-style model-based RL methods [150, 151, 152] are promising, as they combine the strengths of both model-free agents and learned world models.

Another critical direction is the combined use of simulation and hardware experiments. Robotic RL methods must be validated on actual hardware to capture the effects of noisy data and complex dynamics that are often absent in simulations. Tang *et al.* [28] provide a framework that categorizes RL experiments into different levels of hardware testing, from simulation-only (Level 0) to deployment on commercial products (Level 5). In this thesis, we have demonstrated the potential of the CHEQ algorithm to learn an effective polishing policy in one hardware setting, corresponding to Level 1 validation. Future research should aim to extend these tests to various hardware platforms and operational conditions, moving toward Level 2 or 3. Recently proposed scalable hardware benchmarks for robotic manipulation tasks [153, 154] provide a promising direction for future hardware experiments.

At the same time, evaluation in simulation environments remains crucial, as it enables rapid experimentation across a wide range of environments, algorithms, and hyperparameters. A variety of well-established robotic manipulation benchmarks already exists [155, 156, 157, 33, 158]. Recent research has shown that simulation environments with high realism [159] allow for policy performance evaluations that correlate well with real-world

outcomes, providing a complementary approach to hardware testing. Given the challenges of reward shaping and hyperparameter tuning, especially as production use cases diversify, integrating AutoML frameworks [160, 161, 162] into simulation experiments presents a promising strategy to automate the search for optimal settings.

A key insight from this work is the importance of addressing key challenges in deploying RL for production, particularly by incorporating domain priors into learning without compromising the generality of deep RL approaches. To this end, we explored methods for integrating priors into state space extraction and implemented AHRL frameworks. This research demonstrates not only the feasibility of deploying RL in production environments but also opens a broader field of inquiry into balancing the integration of meaningful priors with the versatility of generalist RL agents. Future work should build on these foundational ideas by investigating new ways to integrate priors into state space extraction, advancing AHRL frameworks, and validating these approaches through rigorous hardware experiments.

Bibliography

[1] O. Kroemer, S. Niekum, and G. Konidaris, "A review of robot learning for manipulation: Challenges, representations, and algorithms," *Journal of machine learning research*, 2021.

[2] Y. Koren *et al.*, "Reconfigurable Manufacturing Systems," in *Manufacturing Technologies for Machines of the Future*, Berlin, Heidelberg: Springer, 2003.

[3] Z. L. Gan, S. N. Musa, and H. J. Yap, "A review of the high-mix, low-volume manufacturing industry," *Applied Sciences*, 2023.

[4] A. Kluge-Wilkes *et al.*, "Modular Control and Services to Operate Lineless Mobile Assembly Systems," in *Internet of Production: Fundamentals, Methods and Applications*, Springer International Publishing Cham, 2023.

[5] B. Corves *et al.*, "Robotic 4.0," in *Handbook Industry 4.0: Law, Technology, Society*, W. Frenz, Ed., Berlin, Heidelberg: Springer, 2022.

[6] C. Weidemann *et al.*, "Literature Review on Recent Trends and Perspectives of Collaborative Robotics in Work 4.0," *Robotics*, 2023.

[7] M. Neumann-Brosig, A. Marco, D. Schwarzmann, and S. Trimpe, "Data-Efficient Autotuning With Bayesian Optimization: An Industrial Control Study," *IEEE Transactions on Control Systems Technology*, 2020.

[8] M. Jelali, "An overview of control performance assessment technology and industrial applications," *Control Engineering Practice*, 2006.

[9] Y. Song, A. Romero, M. Mueller, V. Koltun, and D. Scaramuzza, "Reaching the Limit in Autonomous Racing: Optimal Control versus Reinforcement Learning," *Science Robotics*, 2023.

[10] D. Silver *et al.*, "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play," *Science*, 2018.

[11] P. R. Wurman *et al.*, "Outracing champion Gran Turismo drivers with deep reinforcement learning," *Nature*, 2022.

[12] T. Nair, D. Precup, D. L. Arnold, and T. Arbel, "Exploring uncertainty measures in deep networks for multiple sclerosis lesion detection and segmentation," *Medical image analysis*, 2020.

[13] X. Nie, E. Brunskill, and S. Wager, "Learning When-to-Treat Policies," *Journal of the American Statistical Association*, 2021.

[14] X. Wu, R. Li, Z. He, T. Yu, and C. Cheng, "A value-based deep reinforcement learning model with human expertise in optimal treatment of sepsis," *npj Digital Medicine*, 2023.

[15] J. Hwangbo *et al.*, "Learning agile and dynamic motor skills for legged robots," *Science Robotics*, 2019.

[16] J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter, "Learning quadrupedal locomotion over challenging terrain," *Science Robotics*, 2020.

[17] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *The International Journal of Robotics Research*, 2013.

[18] A. S. Anand, R. Kaushik, J. T. Gravdahl, and F. J. Abu-Dakka, "Data-Efficient Reinforcement Learning for Variable Impedance Control," *IEEE Access*, 2024.

[19] J. Luo *et al.*, *SERL: A Software Suite for Sample-Efficient Robotic Reinforcement Learning*, 2024.

[20] J. Ibarz *et al.*, "How to train your robot with deep reinforcement learning: Lessons we have learned," *The International Journal of Robotics Research*, 2021.

[21] R. S. Sutton and A. G. Barto, *Reinforcement Learning, second edition: An Introduction*. MIT Press, 2018.

[22] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International conference on machine learning*, PMLR, 2018.

[23] J. Schrittwieser *et al.*, "Mastering Atari, Go, chess and shogi by planning with a learned model," *Nature*, 2020.

[24] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, 2015.

[25] OpenAI *et al.*, *Dota 2 with Large Scale Deep Reinforcement Learning*, 2019.

[26] S. Levine, P. Pastor, A. Krizhevsky, J. Ibarz, and D. Quillen, "Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection," *The International Journal of Robotics Research*, 2018.

[27] E. Kaufmann *et al.*, "Champion-level drone racing using deep reinforcement learning," *Nature*, 2023.

[28] C. Tang *et al.*, "Deep reinforcement learning for robotics: A survey of real-world successes," *Annual Review of Control, Robotics, and Autonomous Systems*, 2024.

[29] P. Huang *et al.*, "What Went Wrong? Closing the Sim-to-Real Gap via Differentiable Causal Discovery," in *Proceedings of The 7th Conference on Robot Learning*, PMLR, 2023.

[30] H. Qi, A. Kumar, R. Calandra, Y. Ma, and J. Malik, "In-Hand Object Rotation via Rapid Motor Adaptation," in *Proceedings of The 6th Conference on Robot Learning*, PMLR, 2023.

[31] Y. Zhang, L. Ke, A. Deshpande, A. Gupta, and S. Srinivasa, *Cherry-Picking with Reinforcement Learning : Robust Dynamic Grasping in Unstable Conditions*, 2023.

[32] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *IEEE/RSJ international conference on intelligent robots and systems*, 2012.

[33] V. Makoviychuk *et al.*, *Isaac Gym: High Performance GPU-Based Physics Simulation For Robot Learning*, 2021.

[34] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *The Journal of Machine Learning Research*, 2016.

[35] G. Schoettler *et al.*, "Deep reinforcement learning for industrial insertion tasks with visual inputs and natural rewards," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020.

[36] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE transactions on pattern analysis and machine intelligence*, 2013.

[37] S. Bleher, S. Heim, and S. Trimpe, "Learning Fast and Precise Pixel-to-Torque Control: A Platform for Reproducible Research of Learning on Hardware," *IEEE Robotics & Automation Magazine*, 2022.

[38] C. Finn *et al.*, "Deep spatial autoencoders for visuomotor learning," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2016.

[39] T. Jakab, A. Gupta, H. Bilen, and A. Vedaldi, "Unsupervised learning of object landmarks through conditional image generation," in *Advances in Neural Information Processing Systems (NIPS)*, 2018.

[40] T. D. Kulkarni *et al.*, "Unsupervised learning of object keypoints for perception and control," in *Advances in Neural Information Processing Systems (NIPS)*, 2019.

[41] A. Ghadirzadeh, A. Maki, D. Kragic, and M. Björkman, "Deep predictive policy training using reinforcement learning," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017.

[42] L.-C. Chen, C.-K. Ho, and C.-T. King, "KeyState: Improving Image-based Reinforcement Learning with Keypoint for Robot Control," in *IEEE International Conference on Industrial Technology (ICIT)*, Orlando, FL, USA, 2023.

[43] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, *Proximal Policy Optimization Algorithms*, 2017.

[44] T. Silver, K. Allen, J. Tenenbaum, and L. Kaelbling, *Residual policy learning*, 2018.

[45] T. Johannink *et al.*, "Residual reinforcement learning for robot control," in *International Conference on Robotics and Automation (ICRA)*, 2019.

[46] F. Ceola, L. Rosasco, and L. Natale, "RESPRECT: Speeding-up Multi-Fingered Grasping With Residual Reinforcement Learning," *IEEE Robotics and Automation Letters*, 2024.

[47] R. Martín-Martín *et al.*, "Variable impedance control in end-effector space: An action space for reinforcement learning in contact-rich tasks," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019.

[48] M. Bogdanovic, M. Khadiv, and L. Righetti, "Learning variable impedance control for contact sensitive tasks," *IEEE Robotics and Automation Letters*, 2020.

[49] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[50] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," in *CoRR*, San Diego, USA, 2015.

[51] V. Mnih *et al.*, *Playing Atari with Deep Reinforcement Learning*, 2013.

[52] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, 1992.

[53] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International conference on machine learning*, PMLR, 2015.

[54] D. Silver *et al.*, "Deterministic Policy Gradient Algorithms," in *Proceedings of the 31st International Conference on Machine Learning*, PMLR, 2014.

[55] T. P. Lillicrap *et al.*, "Continuous control with deep reinforcement learning," in *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2016.

[56] V. Mnih *et al.*, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*, PMLR, 2016.

[57] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, 1992.

[58] R. Bellman, "Dynamic Programming," *Science*, 1966.

[59] J. Tsitsiklis and B. Van Roy, "An analysis of temporal-difference learning with function approximation," *IEEE Transactions on Automatic Control*, 1997.

[60] B. T. Polyak and A. B. Juditsky, "Acceleration of Stochastic Approximation by Averaging," *SIAM Journal on Control and Optimization*, 1992.

[61] S. Fujimoto, H. Hoof, and D. Meger, "Addressing Function Approximation Error in Actor-Critic Methods," in *Proceedings of the 35th International Conference on Machine Learning*, PMLR, 2018.

[62] H. v. Hasselt, A. Guez, and D. Silver, "Deep Reinforcement Learning with Double Q-Learning," *Proceedings of the AAAI Conference on Artificial Intelligence*, 2016.

[63] B. D. Ziebart, "Modeling Purposeful Adaptive Behavior with the Principle of Maximum Causal Entropy," thesis, Carnegie Mellon University, 2010.

[64] B. D. Ziebart, A. L. Maas, J. A. Bagnell, and A. K. Dey, "Maximum entropy inverse reinforcement learning.," in *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence*, Chicago, IL, USA, 2008.

[65] D. Bank, N. Koenigstein, and R. Giryes, "Autoencoders," in *Machine Learning for Data Science Handbook*, Cham: Springer International Publishing, 2023.

[66] M. A. Kramer, "Nonlinear principal component analysis using autoassociative neural networks," *AIChE Journal*, 1991.

[67] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders," in *Proceedings of the 25th international conference on Machine learning*, New York, NY, USA: Association for Computing Machinery, 2008.

[68] D. P. Kingma and M. Welling, *Auto-encoding variational bayes*, 2013.

[69] B. Siciliano and O. Khatib, *Springer handbook of robotics*, 2nd edition. Berlin ; Heidelberg: Springer, 2016.

[70] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo, *Robotics*, M. J. Grimble and M. A. Johnson, Eds. London: Springer, 2009.

[71] N. Hogan, "Impedance Control: An Approach to Manipulation: Part I—Theory," *Journal of Dynamic Systems, Measurement, and Control*, 1985.

[72] E. Cramer, J. Reiher, and S. Trimpe, "Tracking object positions in reinforcement learning: A metric for keypoint detection," in *Proceedings of the 6th Annual Learning for Dynamics & Control Conference*, PMLR, 2024.

[73] D. Dwibedi, J. Tompson, C. Lynch, and P. Sermanet, "Learning actionable representations from visual observations," in *IEEE/RSJ international conference on intelligent robots and systems (IROS)*, 2018.

[74] Y. Seo, K. Lee, S. L. James, and P. Abbeel, "Reinforcement learning with action-free pre-training from videos," in *International Conference on Machine Learning*, PMLR, 2022.

[75] D. Hafner *et al.*, "Learning latent dynamics for planning from pixels," in *International conference on machine learning*, PMLR, 2019.

[76] T. Lesort, M. Seurin, X. Li, N. Díaz-Rodríguez, and D. Filliat, "Deep unsupervised state representation learning with robotic priors: A robustness analysis," in *International Joint Conference on Neural Networks (IJCNN)*, 2019.

[77] R. Rafailov, T. Yu, A. Rajeswaran, and C. Finn, "Offline reinforcement learning from images with latent space models," in *Learning for Dynamics and Control*, PMLR, 2021.

[78] D. Yarats *et al.*, "Improving sample efficiency in model-free reinforcement learning from images," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2021.

[79] D. Ha and J. Schmidhuber, "Recurrent world models facilitate policy evolution," *Advances in neural information processing systems*, 2018.

[80] D. Hafner, J. Pasukonis, J. Ba, and T. Lillicrap, *Mastering Diverse Domains through World Models*, 2023.

[81] E. Y. Puang, K. P. Tee, and W. Jing, "Kovis: Keypoint-based visual servoing with zero-shot sim-to-real transfer for robotics manipulation," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020.

[82] Z. Qin, K. Fang, Y. Zhu, L. Fei-Fei, and S. Savarese, "Keto: Learning keypoint representations for tool manipulation," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2020.

[83] A. Gopalakrishnan, S. v. Steenkiste, and J. Schmidhuber, "Unsupervised Object Keypoint Learning using Local Spatial Predictability," 2020.

[84] J. J. Sun *et al.*, "Self-Supervised Keypoint Discovery in Behavioral Videos," 2022.

[85] Y. Li, S. Li, V. Sitzmann, P. Agrawal, and A. Torralba, "3d neural scene representations for visuomotor control," in *Conference on Robot Learning*, PMLR, 2022.

[86] J. J. Sun *et al.*, "BKinD-3D: Self-Supervised 3D Keypoint Discovery From Multi-View Videos," 2023.

[87] T. Wang, E. Y. Puang, M. Lee, W. Jing, and Y. Wu, "End-to-end Reinforcement Learning of Robotic Manipulation with Robust Keypoints Representation," in *Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*, 2022.

[88] R. Boney, A. Ilin, and J. Kannala, *Learning of feature points without additional supervision improves reinforcement learning from images*, 2021.

[89] Y. Zhang *et al.*, "Unsupervised discovery of object landmarks as structural representations," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[90] M.-A. Carbonneau, J. Zaidi, J. Boilard, and G. Gagnon, "Measuring Disentanglement: A Review of Metrics," *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

[91] A. L. Rezaabad and S. Vishwanath, "Learning Representations by Maximizing Mutual Information in Variational Autoencoders," in *IEEE International Symposium on Information Theory (ISIT)*, Los Angeles, CA, USA, 2020.

[92] X. Zhao *et al.*, "ALIKE: Accurate and Lightweight Keypoint Detection and Descriptor Extraction," *IEEE Transactions on Multimedia*, 2023.

[93] D. Lorenz, L. Bereska, T. Milbich, and B. Ommer, "Unsupervised Part-Based Disentangling of Object Shape and Appearance," 2019.

[94] Q. Gallouédec, N. Cazin, E. Dellandréa, and L. Chen, *Panda-gym: Open-source goal-conditioned environments for robotic learning*, 2021.

[95] G. S. Watson, "Linear Least Squares Regression," *The Annals of Mathematical Statistics*, 1967.

[96] S. Cabi *et al.*, *Scaling data-driven robotics with reward sketching and batch reinforcement learning*, 2019.

[97] M. Minderer *et al.*, "Unsupervised learning of object structure and dynamics from videos," *Advances in Neural Information Processing Systems*, 2019.

[98] A. Raffin *et al.*, "Stable-Baselines3: Reliable Reinforcement Learning Implementations," *Journal of Machine Learning Research*, 2021.

[99] R. Agarwal, M. Schwarzer, P. S. Castro, A. C. Courville, and M. Bellemare, "Deep Reinforcement Learning at the Edge of the Statistical Precipice," in *Advances in Neural Information Processing Systems*, Curran Associates, Inc., 2021.

[100] E. Cramer, B. Frauenknecht, R. Sabirov, and S. Trimpe, "Contextualized Hybrid Ensemble Q-learning: Learning Fast with Control Priors," in *Reinforcement Learning Journal*, Amherst Massachusetts, 2024.

[101] A. Gupta *et al.*, "Reset-Free Reinforcement Learning via Multi-Task Learning: Learning Dexterous Manipulation Behaviors without Human Intervention," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2021.

[102] D. Büchler *et al.*, "Learning to Play Table Tennis From Scratch Using Muscular Robots," *IEEE Transactions on Robotics*, 2022.

[103] R. Cheng *et al.*, "Control Regularization for Reduced Variance Reinforcement Learning," in *Proceedings of the 36th International Conference on Machine Learning*, PMLR, 2019.

[104] K. Rana *et al.*, "Bayesian controller fusion: Leveraging control priors in deep reinforcement learning for robotics," *The International Journal of Robotics Research*, 2023.

[105] M. Schier, C. Reinders, and B. Rosenhahn, "Learned Fourier Bases for Deep Set Feature Extractors in Automotive Reinforcement Learning," in *IEEE 26th International Conference on Intelligent Transportation Systems (ITSC)*, Bilbao, Spain, 2023.

[106] T. Davchev *et al.*, "Residual Learning From Demonstration: Adapting DMPs for Contact-Rich Manipulation," *IEEE Robotics and Automation Letters*, 2022.

[107] A. Ranjbar, N. A. Vien, H. Ziesche, J. Boedecker, and G. Neumann, "Residual Feedback Learning for Contact-Rich Manipulation Tasks with Uncertainty," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021.

[108] L. Kerbel, B. Ayalew, A. Ivanco, and K. Loiselle, "Residual Policy Learning for Powertrain Control," *IFAC-PapersOnLine*, 2022.

[109] P. Daoudi, B. Robu, C. Prieur, L. Dos Santos, and M. Barlier, "Enhancing Reinforcement Learning Agents with Local Guides," in *International Conference on Autonomous Agents and Multiagent Systems*, 2023.

[110] C.-J. Hoel, T. Tram, and J. Sjöberg, "Reinforcement Learning with Uncertainty Estimation for Tactical Decision-Making in Intersections," in *IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*, 2020.

[111] C.-J. Hoel, K. Wolff, and L. Laine, "Tactical Decision-Making in Autonomous Driving by Reinforcement Learning with Uncertainty Estimation," in *IEEE Intelligent Vehicles Symposium (IV)*, 2020.

[112] K. Rana, B. Talbot, V. Dasagi, M. Milford, and N. Sünderhauf, "Residual Reactive Navigation: Combining Classical and Learned Navigation Strategies For Deployment in Unknown Environments," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2020.

[113] K. Rana, V. Dasagi, B. Talbot, M. Milford, and N. Sünderhauf, "Multiplicative Controller Fusion: Leveraging Algorithmic Priors for Sample-efficient Reinforcement Learning and Safe Sim-To-Real Transfer," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020.

[114] P. Kulkarni, J. Kober, R. Babuška, and C. Della Santina, "Learning assembly tasks in a few minutes by combining impedance control and residual recurrent reinforcement learning," *Advanced Intelligent Systems*, 2022.

[115] M. G. Bellemare, W. Dabney, and M. Rowland, *Distributional reinforcement learning*. MIT Press, 2023.

[116] X. Chen, C. Wang, Z. Zhou, and K. W. Ross, "Randomized Ensembled Double Q-Learning: Learning Fast Without a Model," 2020.

[117] I. Osband, C. Blundell, A. Pritzel, and B. Van Roy, "Deep Exploration via Bootstrapped DQN," in *Advances in Neural Information Processing Systems*, Curran Associates, Inc., 2016.

[118] K. Lee, M. Laskin, A. Srinivas, and P. Abbeel, "Sunrise: A simple unified framework for ensemble learning in deep reinforcement learning," in *International Conference on Machine Learning*, PMLR, 2021.

[119] V. Mai, K. Mani, and L. Paull, "Sample Efficient Deep Reinforcement Learning via Uncertainty Estimation," 2022.

[120] S. Thrun and A. Schwartz, "Issues in Using Function Approximation for Reinforcement Learning," in *Proceedings of the Fourth Connectionist Models Summer School*, 1993.

[121] Q. Lan, Y. Pan, A. Fyshe, and M. White, *Maxmin Q-learning: Controlling the Estimation Bias of Q-learning*, 2021.

[122] H. Wang, S. Lin, and J. Zhang, "Adaptive Ensemble Q-learning: Minimizing Estimation Bias via Error Feedback," in *Advances in Neural Information Processing Systems*, Curran Associates, Inc., 2021.

[123] S. Thrun *et al.*, "Stanley: The robot that won the DARPA Grand Challenge," *Journal of Field Robotics*, 2006.

[124] S. Huang *et al.*, "CleanRL: High-quality Single-file Implementations of Deep Reinforcement Learning Algorithms," *Journal of Machine Learning Research*, 2022.

[125] E. Cramer, L. Jäschke, and S. Trimpe, "CHEQ-ing the Box: Safe Variable Impedance Learning for Robotic Polishing," in *Joint 10th IFAC Symposium on Mechatronic Systems and 14th Symposium on Robotics*, Paris, France, 2025.

[126] J. Buchli, F. Stulp, E. Theodorou, and S. Schaal, "Learning variable impedance control," *The International Journal of Robotics Research*, 2011.

[127] J. Luo *et al.*, "Reinforcement Learning on Variable Impedance Controller for High-Precision Robotic Assembly," in *International Conference on Robotics and Automation (ICRA)*, 2019.

[128] L. Roveda *et al.*, "Model-based reinforcement learning variable impedance control for human-robot collaboration," *Journal of Intelligent & Robotic Systems*, 2020.

[129] A. S. Anand, M. Hagen Myrestrand, and J. T. Gravdahl, "Evaluation of Variable Impedance- and Hybrid Force/MotionControllers for Learning Force Tracking Skills," in *IEEE/SICE International Symposium on System Integration (SII)*, Narvik, Norway, 2022.

[130] K. H. Fiedler, "Processing (Grinding and Polishing)," in *The Properties of Optical Glass*, H. Bach and N. Neuroth, Eds., Berlin, Heidelberg: Springer, 1998.

[131] K. M. Lynch and F. C. Park, *Modern Robotics*. Cambridge University Press, 2017.

[132] O. Khatib, "Inertial Properties in Robotic Manipulation: An Object-Level Framework," *The International Journal of Robotics Research*, 1995.

[133] K. Shoemake, "Animating rotation with quaternion curves," in *Proceedings of the 12th annual conference on Computer graphics and interactive techniques*, New York, NY, USA: Association for Computing Machinery, 1985.

[134] L. Biewald, *Experiment tracking with weights and biases*, 2020.

[135] A. Dosovitskiy *et al.*, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale," in *International Conference on Learning Representations*, 2021.

[136] K. Han *et al.*, "A Survey on Visual Transformer," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2023.

[137] R. Firoozi *et al.*, "Foundation models in robotics: Applications, challenges, and the future," *The International Journal of Robotics Research*, 2024.

[138] A. Brohan *et al.*, *RT-1: Robotics Transformer for Real-World Control at Scale*, 2023.

[139] O. Mees *et al.*, "Octo: An Open-Source Generalist Robot Policy," 2024.

[140] O. X.-E. Collaboration *et al.*, *Open X-Embodiment: Robotic Learning Datasets and RT-X Models*, 2024.

[141] A. Khazatsky *et al.*, "DROID: A large-scale in-the-wild robot manipulation dataset," in *Proceedings of Robotics: Science and Systems*, 2024.

[142] H. R. Walke *et al.*, "BridgeData V2: A Dataset for Robot Learning at Scale," in *Proceedings of The 7th Conference on Robot Learning*, PMLR, 2023.

[143] M. Vecerik *et al.*, *Leveraging Demonstrations for Deep Reinforcement Learning on Robotics Problems with Sparse Rewards*, 2018.

[144] H. R. Walke *et al.*, "Don't Start From Scratch: Leveraging Prior Data to Automate Robotic Reinforcement Learning," in *Proceedings of The 6th Conference on Robot Learning*, PMLR, 2023.

[145] M. Deisenroth and C. E. Rasmussen, "PILCO: A model-based and data-efficient approach to policy search," in *Proceedings of the 28th International Conference on machine learning (ICML-11)*, 2011.

[146] S. Levine and V. Koltun, "Guided policy search," in *International Conference on Machine Learning*, 2013.

[147] K. Chua, R. Calandra, R. McAllister, and S. Levine, "Deep Reinforcement Learning in a Handful of Trials using Probabilistic Dynamics Models," in *Advances in Neural Information Processing Systems*, Curran Associates, Inc., 2018.

[148] D. Hafner, T. Lillicrap, J. Ba, and M. Norouzi, "Dream to Control: Learning Behaviors by Latent Imagination," 2019.

[149] B. Frauenknecht, T. Ehlgen, and S. Trimpe, "Data-efficient Deep Reinforcement Learning for Vehicle Trajectory Control," in *IEEE 26th International Conference on Intelligent Transportation Systems (ITSC)*, 2023.

[150] M. Janner, J. Fu, M. Zhang, and S. Levine, "When to Trust Your Model: Model-Based Policy Optimization," in *Advances in Neural Information Processing Systems*, Curran Associates, Inc., 2019.

[151] B. Frauenknecht, A. Eisele, D. Subhasish, F. Solowjow, and S. Trimpe, "Trust the Model Where It Trusts Itself - Model-Based Actor-Critic with Uncertainty-Aware Rollout Adaption," 2024.

[152] B. Frauenknecht, D. Subhasish, F. Solowjow, and S. Trimpe, *On Rollouts in Model-Based Reinforcement Learning*, 2025.

[153] M. Heo, Y. Lee, D. Lee, and J. Lim, "FurnitureBench: Reproducible Real-World Benchmark for Long-Horizon Complex Manipulation," in *Robotics: Science and Systems XIX*, Robotics: Science and Systems Foundation, 2023.

[154] J. Luo *et al.*, "FMB: A functional manipulation benchmark for generalizable robotic learning," *The International Journal of Robotics Research*, 2024.

[155] Y. Zhu *et al.*, *Robosuite: A modular simulation framework and benchmark for robot learning*, 2020.

[156] K. Ehsani *et al.*, "ManipulaTHOR: A Framework for Visual Object Manipulation," 2021.

[157] T. Yu *et al.*, "Meta-World: A Benchmark and Evaluation for Multi-Task and Meta Reinforcement Learning," in *Proceedings of the Conference on Robot Learning*, PMLR, 2020.

[158] C. Li *et al.*, "BEHAVIOR-1K: A Human-Centered, Embodied AI Benchmark with 1, 000 Everyday Activities and Realistic Simulation," *CoRR*, 2024.

[159] X. Li *et al.*, *Evaluating Real-World Robot Manipulation Policies in Simulation*, 2024.

[160] F. Hutter, L. Kotthoff, and J. Vanschoren, Eds., *Automated Machine Learning: Methods, Systems, Challenges*. Springer Nature, 2019.

[161] J. Parker-Holder *et al.*, "Automated reinforcement learning (autorl): A survey and open problems," *Journal of Artificial Intelligence Research*, 2022.

[162] J. Dierkes, E. Cramer, H. H. Hoos, and S. Trimpe, "Combining Automated Optimisation of Hyperparameters and Reward Shape," in *Reinforcement Learning Journal*, Amherst Massachusetts, 2024.

[163] A. Paszke *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, 2019.

A. A Metric for Keypoint Detection in Reinforcement Learning

This chapter gives additional details for the methods introduced in Chapter 3. Details on the implementation and hyperparameters can be found in Appendix A.1, and additional results and ablations are discussed in Appendix A.2.

A.1. Implementation Details

The following section discusses further details regarding the SAE setup in Appendix A.1.1 and regarding the RL setup in Appendix A.1.2.

A.1.1. SAE Setup and Hyperparameters

For SAE tracking evaluation, we conduct 24 training runs with different random seeds for each considered model architecture.

Training

Hyperparameters are kept fixed and are listed in Table A.1. During SAE training, we add normally distributed noise with $\sigma = 0.001$ to the input images for regularization. Images contain floating point RGB values in the range $[0, 1]$.

Table A.1.: Hyperparameters for SAE training.

SAE Hyperparameter	Value
number of epochs	500
batch size	32
learning rate	0.001
optimizer	Adam [50]
input shape	$256 \times 256 \times 3$
target shape	$64 \times 64 \times 3$
latent spatial dimensions	$[-1, 1] \times [-1, 1]$

Datasets

The training, validation, and testing datasets consist of 5 000, 2 500, and 2 500 images, respectively. To keep all three datasets well-separated, we initially collect image sequences of 10 frames each and perform data splitting on these sequences. To capture sufficient variation in the randomly initialized object and target positions, episode lengths during image collection are limited to 20 frames. Images are collected using a smoothed random policy, starting with a random initial action in $[-10, 10] \times [-10, 10] \times [-10, 10]$ and sampling the next action from a Gaussian normal distribution with standard deviation $\sigma = 1.0$ centred on the previous action.

Evaluation

The tracking thresholds used for computing the tracking capability TC are chosen to be $\mu_{\text{cube}} = 0.015$, $\mu_{\text{target}} = 0.015$, and $\mu_{\text{eef}} = 0.1$. Tracking errors are evaluated in the latent spatial dimensions of size $[-1, 1] \times [-1, 1]$. Where point estimates are reported, they refer to values at the end of training. Figures showing the object-wise tracking errors over time are slightly smoothed by applying a Gaussian filter with $\sigma = 2.5$ steps.

SAE Architectures

Existing and new SAE architectures were implemented by us in PyTorch [163]. All details of this implementation can be found in the accompanying source code.

Architectures from literature: Our DSAE implementation follows the layout outlined by Finn *et al.* [38]. We keep all mentioned hyperparameters, including a CNN encoder with three convolutional layers, the number of keypoints $N = 16$ and a single fully connected layer as decoder. Our KeyNet implementation follows the implementation from Jakab *et al.* [39]. In their paper, the goal is to shift an object in one image with information from another image. For this, a second image is appended to the decoder input. We neglect this since we are assuming that the scene does not change. Where applicable, we keep all mentioned hyperparameters. We choose the number of keypoints to be $N = 30$, as primarily used in their experiments. We fix the number of convolutional blocks in the encoder as 4 and in the decoder as 3.

Basic: Our Basic SAE architecture is a CNN similar to KeyNet [39]. The encoder consists of three convolutional blocks. Each block consists of a 2D convolution followed by a 2D batch-normalization layer and another 2D convolution followed by a 2D max-pooling operation with kernel size and stride 2, and a 2D batch-normalization layer again. The kernel size for every convolution is 3. The number of output channels for

both convolutions in a block is identical, with 32 in the first, 64 in the second, and 128 output channels in the third block. The blocks are followed by a 2D 1×1-convolution to aggregate into 16 feature maps. A spatial soft arg-max layer extracts $N = 16$ 2D keypoints from these. Similarly to KeyNet, the decoder begins with 2D heatmaps with Gaussian kernels with $\sigma = 0.1$ at the keypoint location for each of the $N = 16$ feature maps. These maps are created at the target resolution of 64×64. Three convolutional layers follow, each with a kernel size of 3 and 64, 32, and 16 output channels, respectively. A 2D batch-normalization layer follows each convolution. A final 2D 1×1-convolution again aggregates the feature maps into 3 RGB image channels.

Modifications: We find three modifications to SAE architectures that improve tracking:

1. **Velocity loss term** (-vel). In Finn *et al.* [38], an additional loss term g_{slow} is added to the reconstruction loss with a weighting factor β. By penalizing a change of keypoint velocities in subsequent frame pairs, the velocity loss encourages the detection of temporally consistent keypoints. For the velocity loss term, we choose a weighting factor $\beta = 0.1$ while the reconstruction loss remains weighted with factor 1.0 and is here computed as the mean of the three samples passed through the encoder concurrently.
2. **Trainable Gaussian standard deviation** (-std). We follow Jakab *et al.* [39] and use Gaussian heatmaps as input to the first CNN decoder layer. We propose making the standard deviation σ trainable. This enables the decoder to control a keypoint's radius of influence. For the trainable Gaussian standard deviation, we use an initial value of $\sigma = 0.1$, equal to the fixed standard deviation in the unmodified case.
3. **Background bias layer** (-bg). We add a bias with the dimensions of the target image to the decoder's output, giving the decoder a straightforward way to reconstruct a stationary background. This usually happens within the first epochs, and the time-varying keypoints then focus on moving objects. The background bias layer is initialized to zeros.

A.1.2. RL Setup and Hyperparameters

For RL experiments with SAEs as state representation extractors, we randomly sample 5 trained SAEs of each architecture and conduct two randomly seeded RL training runs with each of them, yielding a total of 10 runs per SAE architecture.

Table A.2.: Hyperparameters for RL training.

RL Hyperparameter	Value
number of steps	3 000 000
steps after which learning starts	100 000
episode time limit (steps)	100
number of parallel environments	10
gradient descent steps per environment step	1

Table A.3.: Specifications of state representation vector configurations used for RL experiments.

Configuration	State
Keypoints	$s^t = z^t$
Keypoints + robot	$s^t_{\text{ext}} = (z^t, o_{\text{eef}}, \dot{o}_{\text{eef}})$
GT points	$s^t = x^t$
GT points + robot	$s^t_{\text{ext}} = (x^t, o_{\text{eef}}, \dot{o}_{\text{eef}})$
Full Simulation State	$s^t_{\text{full}} = (o_{\text{eef}}, \dot{o}_{\text{eef}}, o_{\text{cube}}, \dot{o}_{\text{cube}}, \phi_{\text{cube}}, \dot{\phi}_{\text{cube}}, o_{\text{target}})$

Training

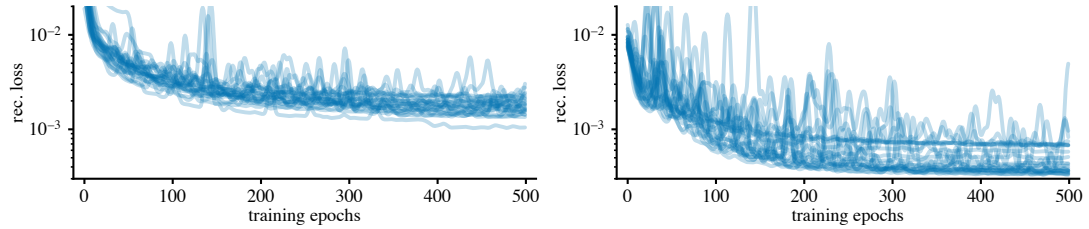
As an RL algorithm, we choose the SAC [22] implementation from stable-baselines3 [98]. Except for the values in Table A.2, hyperparameters are kept at default values.

Evaluation

Figures showing the evaluation success rate over time are slightly smoothed by applying a Gaussian filter with $\sigma = 2.5$ evaluation steps. Evaluation steps are performed every 10 000 training step with 10 evaluation episodes each.

State representation

In our RL evaluation, we consider different state representations. Table A.3 summarizes these configurations. We distinguish between using SAE-encoded keypoints and using ground truth (GT) points and between extending the state with end-effector state measurements (+ robot) or not doing this. Additionally, we consider the full simulation state, including target position and cube position, velocity, orientation, and angular velocity. Note that o and ϕ denote 3D positions and orientations, respectively, while $\dot{o}$ and $\dot{\phi}$ are their temporal derivatives.



(a) (b)

Figure A.1.: We show the reconstruction loss over training epochs for the Basic-kp32 architecture in Figure a and for the KeyNet architecture in Figure b.

A.2. Additional Results and Ablation Study

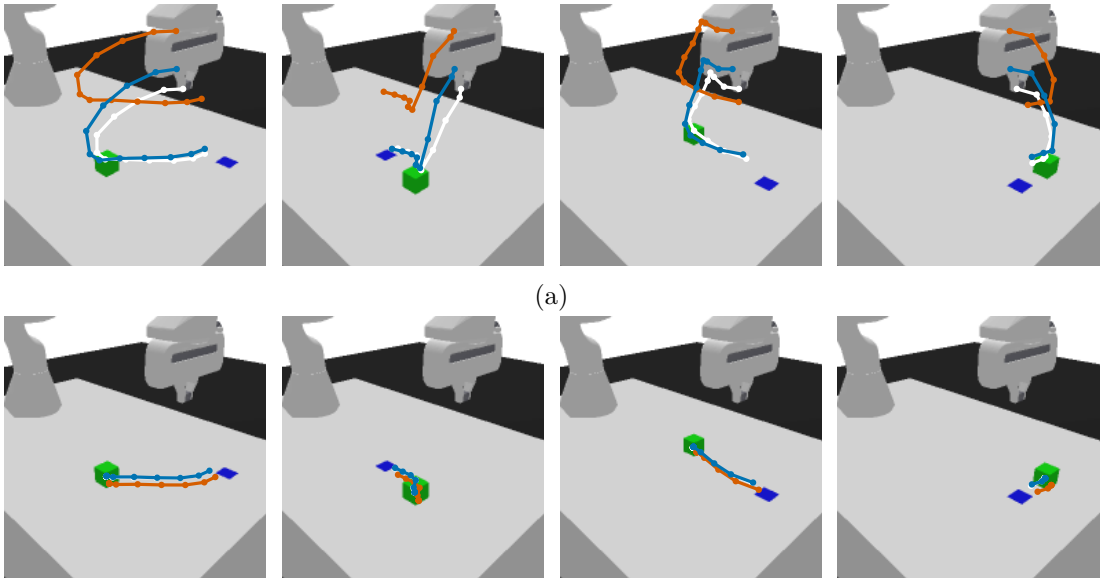
In the following, we discuss selected additional results we obtained. These are a comparison of our SAE metric with the reconstruction loss, qualitative results of applying the linear feature transformation, and a closer look at tracking error distributions for the considered SAE architecture modifications.

Reconstruction loss

In comparison to tracking errors and tracking capability, we examine the reconstruction loss of two exemplary architectures in Figure A.1. Overall, we observe the expected decrease in reconstruction loss over training epochs and a difference in final reconstruction loss between the two architectures. In contrast to the tracking error, the reconstruction loss cannot tell us if the two architectures effectively track the ground truth objects. No notable jumps are present, as they are for the tracking error over time, nor is there any way of differentiating between multiple objects. We, therefore, find the reconstruction loss to be an insufficient indicator for judging an SAE's tracking performance.

Qualitative transformation results

Figure A.2 shows trajectories from four example episodes for the end-effector and the cube. The trajectory of the best keypoint according to our tracking error formulation is shown in red. As expected, it is to be noted that this trajectory can significantly deviate from the white ground truth point trajectory. We obtain the blue trajectory when considering the proposed transformation for tracking evaluation. This trajectory comes much closer to the ground truth point trajectory we evaluate against. For the cube, these two trajectories coincide almost exactly. The larger deviation for the end-effector can be explained by its wider range of movement.



(a)

(b)

Figure A.2.: Trajectories from sample episodes for end-effector tracking in Figure a and cube tracking in Figure b. Ground truth points (white), untransformed keypoints obtained with KeyNet-vel-std-bg (red), and transformed keypoints (blue) are plotted over time.

Modifications and ablations

Figure A.3 shows the tracking capabilities of the Basic architecture with all combinations of modifications and the KeyNet architecture with the best-found combination. The velocity loss (-vel) generally improves the tracking capability for continuously moving objects, in our case, the end-effector. However, it can have a slightly negative effect on tracking more stationary objects. Making σ trainable (-std) and adding a background bias layer (-bg) almost always yields an improved tracking capability. The combination vel-std-bg yields a consistent improvement in tracking capability. For Basic, it is increased from $\overline{\mathrm{TC}} = 0.181$ to $\overline{\mathrm{TC}} = 0.528$. KeyNet already tracks the cube and target well and reaches $\overline{\mathrm{TC}} = 0.681$. KeyNet-vel-std-bg has a near-perfect mean tracking capability of $\overline{\mathrm{TC}} = 0.986$. The most significant change can be seen in end-effector tracking, improving from $\mathrm{TC}_{\mathrm{eef}} = 0.167$ to 0.958. The tracking error confirms this exceptional tracking performance. For cube and target, the tracking error consistently shows a distinct drop below our threshold (cf. Figure A.4). As Section 3.6.1 explains, the end-effector usually shows a shallower slope. Still, the tracking error consistently falls below the threshold, indicating that the KeyNet-vel-std-bg architecture learns to track the

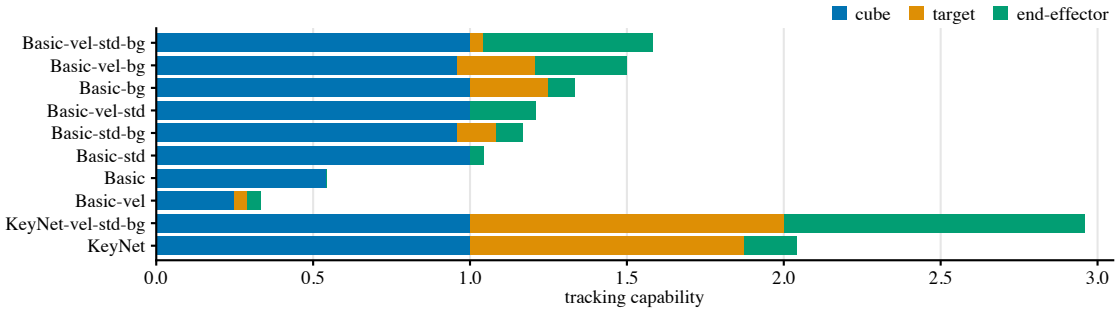


Figure A.3.: Model ablations: tracking capability $\overline{\mathrm{TC}_k}$ for $k = 3$ ground truth objects of the adapted architectures.

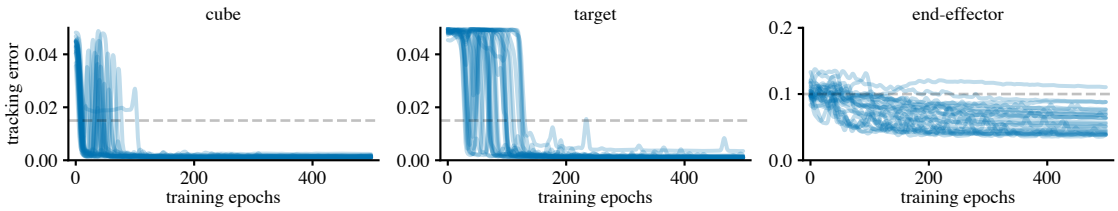


Figure A.4.: KeyNet-vel-std-bg tracking errors $\overline{\mathrm{TC}_k}$ for $k = 3$ ground truth objects over epochs.

gripper.

Due to its more directly interpretable nature, we primarily examine the tracking capability TC in our evaluation of architecture modifications. Figure A.5 additionally shows box plots of the object-wise tracking error distributions over 24 runs for each model. Again, we observe the effectiveness of the joined modification -vel-std-bg for both the Basic and the KeyNet architecture.

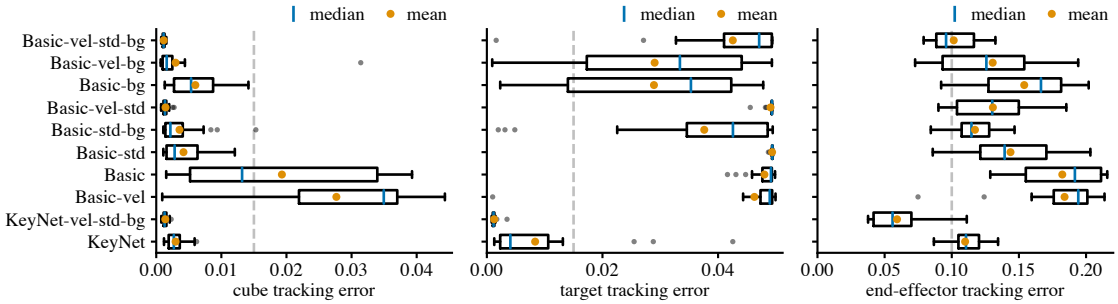


Figure A.5.: Box plots of tracking errors over 24 runs of the considered architecture modifications.

B. Contextualized Hybrid Ensemble Q-learning

This chapter gives additional details for the methods introduced in Chapter 4. We give details on the implementation in Appendix B.1 as well as selected additional results and ablations in Appendix B.2.

B.1. Implementation Details

This section gives implementation details and hyperparameters regarding the RL setup in Appendix B.1.1 and regarding the nominal controller in Appendix B.1.2.

B.1.1. RL Setup and Hyperparameters

We build our SAC implementation based on CleanRL [124]. All SAC-specific hyperparameters are kept consistent between all approaches and reported in Table B.1.

Table B.1.: Shared Hyperparameters for SAC and CHEQ.

Hyperparameter	Value
number of steps	1.5×10^6
batch size	256
learning rate actor	3×10^{-4}
learning rate critic	3×10^{-4}
target entropy H_t	-3
replay buffer size	1×10^6
discount factor γ	0.99
gradient update start	1×10^3 steps
Polyak averaging factor τ	0.005

CHEQ (UTD1 and UTD20) uses an ensemble of $E = 5$ critics. We set the upper bound of the uncertainty as $u_{\max} = 0.15$ and the lower bound as $u_{\min} = 0.03$. Further we set $\lambda_{\max} = 1.0$ and $\lambda_{min} = 0.2$. We use a Bernoulli masking rate of $\kappa = 0.8$ and $F = 2$ minimization targets.

BCF trains an ensemble of policy networks. We maintain the original ensemble size from [104], which uses ten policy networks. We set the standard deviation of the control prior in BCF as $\sigma^{\text{prior}} = 6.0$.

For the uncertainty estimate in CORE, we set $A = 7$, $C = 0.02$. Note that in the original paper, A is denoted as $\lambda_{\max}$, which we change to avoid ambiguous notation.

SAC uses a UTD ratio of 1. REDQ implementation uses an ensemble size of 5 and a UTD ratio of 20.

For all algorithms, we include a random sampling phase for the first 1×10^3 steps where we sample the RL action uniformly random and do not update our agent. In this setting, we keep λ_t^{RL} small for the hybrid agents. For CHEQ, we vary λ^{RL} between [0.2, 0.3]. As CORE and BCF are unable to observe changes in λ^{RL}, we use a fixed $\lambda^{\text{RL}} = 0.2$, which has shown to be favourable in our experiments. After the random sampling phase, agent training starts, but λ^{RL} is kept small for another 4×10^3 steps, and afterwards, λ adaption starts.

For performance evaluation, we conduct a greedy evaluation run every 20 episodes. Evaluation happens in an adapted setting, together with the controller, where the weight is calculated as in the training procedure.

B.1.2. Nominal Control Setup and Hyperparameters

For the racing task, we design a simple path-following controller with adaptive speeds. For the lateral control, we use a Stanley Controller [123], and for the longitudinal control, we design two symmetric P-controllers.

We tuned the controller gains and coefficients to $k_{\text{cross}} = 0.5[1/\text{s}]$, $k_{\text{soft}} = 1[\text{m}\,\text{s}^{-1}]$, $k_r = 0.4[1/\text{s}]$, $v_{\max} = 8[\text{m}\,\text{s}^{-1}]$, $k_v^{\max} = 0.25[\text{s}\,\text{m}^{-1}]$, $k_v^{\min} = 0.05[\text{s}\,\text{m}^{-1}]$, $v_{\text{low}} = 8[\text{m}\,\text{s}^{-1}]$ and $v_{\text{high}} = 28[\text{m}\,\text{s}^{-1}]$.

B.2. Additional Results and Ablation Study

In this section, we formulate contextualized hybrid variants of CORE and BCF, C-CORE and C-BCF. Further, we present additional hyperparameter sensitivity ablations and discuss the λ^{RL} distribution for CHEQ-UTD1, CORE, and BCF. For a reasonable comparison, we focus mainly on CHEQ-UTD1 using a UTD ratio of 1.

Contextualized Hybrid Variants of Prior Work

To further substantiate our claim that the contextualized hybrid RL formulation aids the training progress, we developed contextualized variants of the CORE and BCF algorithm, which we call C-CORE and C-BCF. To use the adaptive weight as a context variable, a weighting parameter λ_t^{RL} needs to be determined.

The CORE algorithm comes with a direct weight estimate[103], which can be written as

$$\lambda_t^{\mathrm{RL}} = \frac{1}{1 + \lambda_t^{\mathrm{CORE}}}$$

where $\lambda_t^{\mathrm{CORE}} = A(1 - e^{-C|\delta_{t-1}|})$ with the TD-error δ_{t-1} and C, A being tuning parameters[1].

For C-BCF, we derive a pseudo-weight, as the BCF algorithm [104] does not have a straightforward weighting factor λ^{RL}. In BCF, at timestep $t-1$, the fusion of the prior $\mathcal{N}_{\psi,t-1}(\mu_{\psi,t-1}, \sigma_{\psi,t-1})$ and the SAC-ensemble $\mathcal{N}_{\pi,t-1}(\mu_{\pi,t-1}, \sigma_{\pi,t-1})$ results in a Gaussian distribution with mean

$$\mu_{\mathrm{fuse},t-1} = \frac{\sigma_{\psi,t-1}^2}{\sigma_{\psi,t-1}^2 + \sigma_{\pi,t-1}^2} \cdot \mu_{\pi,t-1} + \frac{\sigma_{\pi,t-1}^2}{\sigma_{\psi,t-1}^2 + \sigma_{\pi,t-1}^2} \cdot \mu_{\psi,t-1}.$$

Thus, in BCF, the weight has the dimension of the action space, whereas the contextualized mechanism requires a scalar weight λ_t^{RL}. For C-BCF, we compute a scalar weight

$$\lambda_t^{\mathrm{RL}} = \frac{1}{N} \sum_{i=1}^{N} \left(\frac{\sigma_{\psi,t-1}^2}{\sigma_{\psi,t-1}^2 + \sigma_{\pi,t-1}^2} \right)_i .$$

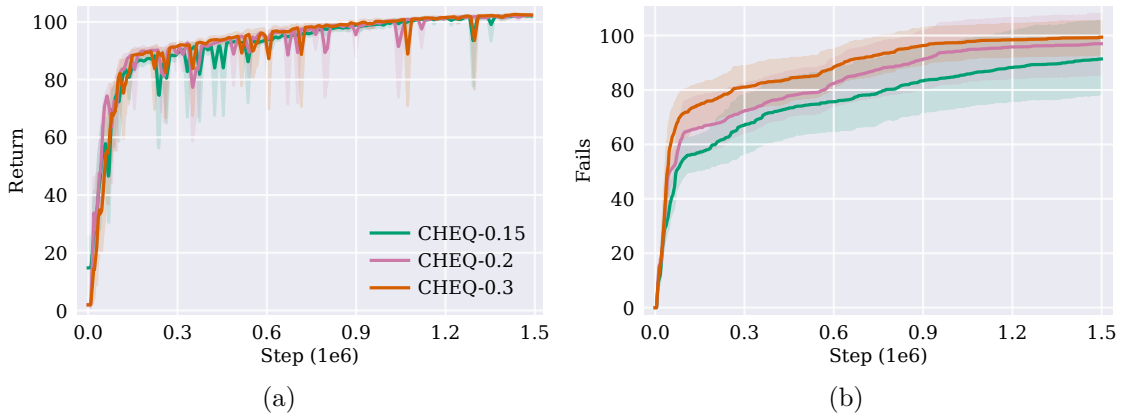
The next action a_t^{mix} is computed according to Equation 4.2 where $a_t^{\mathrm{RL}} \sim \mathcal{N}_{\pi,t}$ and $a_t^{\mathrm{prior}} = \mu_{\psi,t}$.

For C-BCF and C-CORE, we use the same warm-up phase as our CHEQ agent.

Additional Results and Hyperparameter Tuning for CHEQ

Our algorithm has only two important hyperparameters, upper and lower bounds of the uncertainty $u_{\max}$ and $u_{\min}$. We chose these hyperparameters by conducting one training run and investigating the uncertainty range within this run. CHEQ is generally robust against changes in these thresholds. We observe slightly lower final return and fewer fails for lower upper bounds $u_{\max}$. This is expected as frequent handover to the control prior

[1] CORE [103] uses the term $\lambda_{\max}$ instead of A. As we use $\lambda_{\max}$ in a different context, we stick to A here.



(a) (b)

Figure B.1.: Comparison of CHEQ-UTD1 with different $u_{\max}$ thresholds. Plotting return (a) and number of fails (b) for the racing environment.

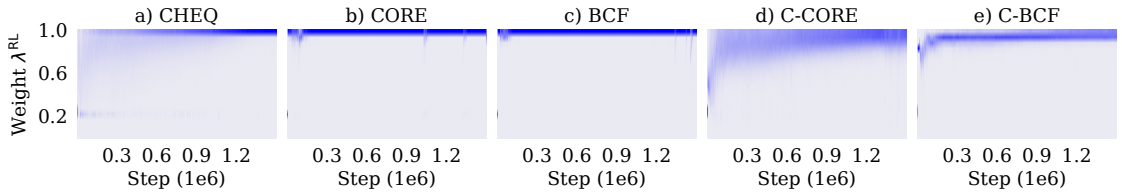


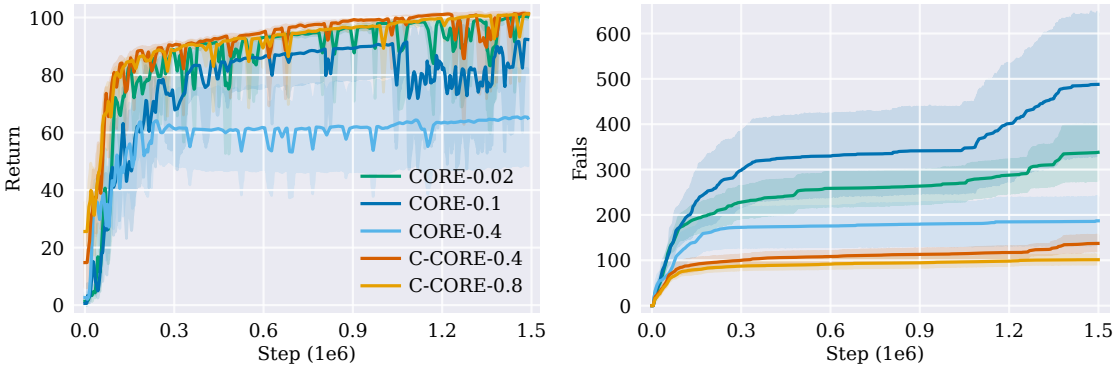
Figure B.2.: Distributions of λ^{RL} over training steps shown for all hybrid agents.

results in lower velocities and, thus, lower returns. Figure B.1 shows the return and the number of fails during training for our CHEQ-UTD1 variants. For CHEQ-UTD20, we were able to use the same upper and lower uncertainty bounds as for CHEQ-UTD1.

Figure B.2 shows the distribution of the weight λ^{RL} over the training progress. We find that for CHEQ-UTD1, the agent starts with an almost uniform weight distribution and slowly moves towards a $\lambda^{\mathrm{RL}} = 1$ regime. Even in later training stages, the agent hands over to the controller from time to time.

Additional Results and Hyperparameter Tuning for CORE and C-CORE

In Figure B.3, we compare different parameters C. We find stable training progress and high return for $C = 0.02$, but since this uses high λ^{RL} values, this setting results in many failures. Figure B.2 illustrates the high λ regime of the $C = 0.02$ agent. Using our contextualized hybrid framework, C-CORE notably outperforms CORE in terms of asymptotic return, training stability, and the number of training failures. Using the contextualized formulation, C-CORE can use a much wider λ^{RL} distribution (see Figure B.2).



(a) (b)

Figure B.3.: Comparison of return (a) and number of fails (b) CORE and C-CORE runs with different C parameters.

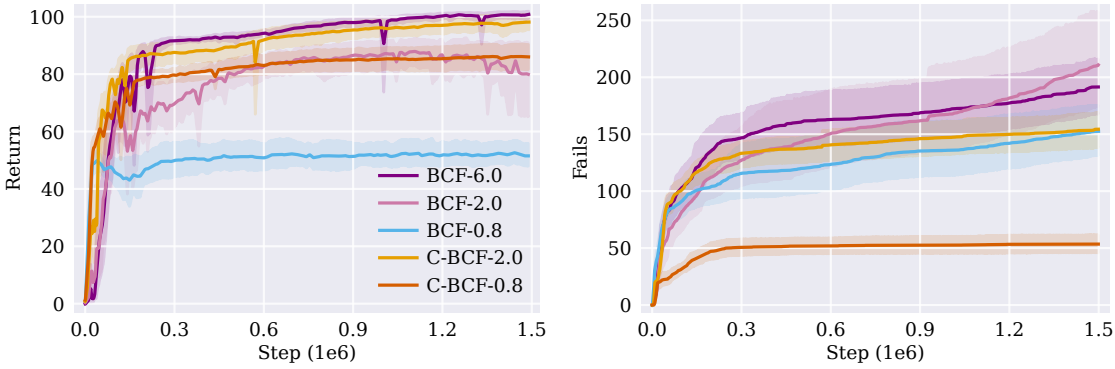
Additional Results and Hyperparameter Tuning for BCF and C-BCF

The BCF algorithm is sensitive to the parameters of the uncertainty threshold, which in this case is the variance of the control prior σ^{prior}. Higher variances lead to less weight on the control prior and thus high λ^{RL} regimes. In Figure B.4 we compared different parameters σ^{prior}. We find stable training progress and high return for $\sigma^{\text{prior}} = 6$. However, as expected, this setting uses a λ^{RL} regime close to one and thus results in many failures. Figure B.2 illustrates this regime.

Our C-BCF variant can resolve this problem only partially. Due to its construction, the BCF algorithm has a separate weighting factor for each action, of which we take the mean. In addition, our pseudo λ^{RL} factor is only a rough estimate of the actual mixing as BCF samples from the posterior distribution. Both factors result in information loss and make the weight λ^{RL} only a rough estimate. C-BCF-2.0 and BCF-6.0 achieve similar asymptotic performance, while C-BCF-2.0 leads to fewer failures.

Return vs Failure Comparison for all trained Models

Figure B.5 shows a scatter plot of the final return and the accumulated failures during training for all hybrid algorithms discussed in this paper. This final comparison shows that if prior methods are trained with the contextualized framework and tuned well (C-BCF-0.8, C-CORE-0.4, C-CORE-0.8), they achieve high returns while maintaining fewer failures than their non-contextualized counterparts. Our algorithm variants (CHEQ-UTD1, CHEQ-UTD20) achieve the highest return while maintaining the lowest number of cumulative failures.



(a) (b)

Figure B.4.: Comparison of return (a) and number of fails (b) of BCF and C-BCF runs with different σ_{prior} parameters.

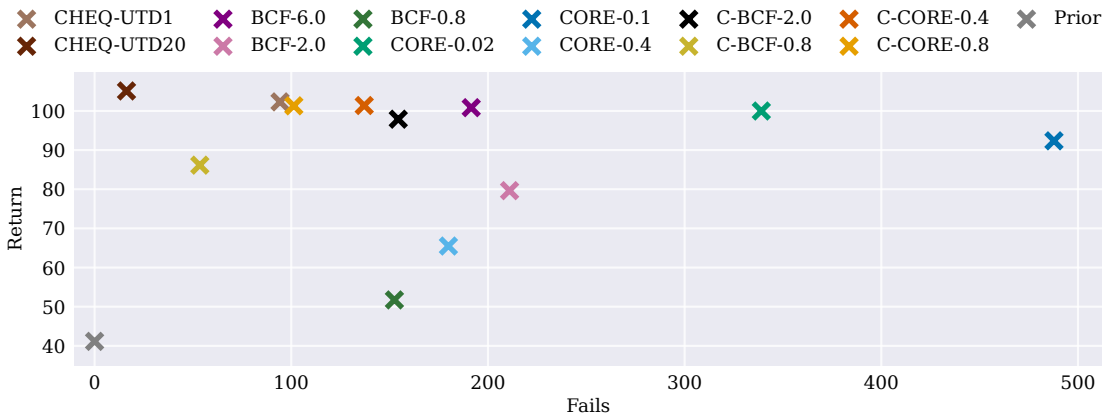


Figure B.5.: Scatter plot of the return and number of fails for different hyperparams for BCF, CORE, C-BCF, C-CORE. CHEQ-UTD20, CHEQ-UTD1 + Prior in Comparison

C. Safe Variable Impedance Learning for Robotic Polishing

This chapter gives additional details for the methods introduced in Chapter 5. Details on the polishing task setup are discussed in Appendix C.1, and a detailed list of hyperparameters can be found in Appendix C.2.

C.1. Polishing Environment

This section gives further details on the goal of the polishing task in Appendix C.1.1 and on the safety limits of the environment in Appendix C.1.2.

C.1.1. Derivation of the Task Goal

The goal of our task is to polish over the workpiece with a constant material removal rate (MRR). For this, we let a robotic arm with a polishing tool attached to the last joint follow a predefined path r^{path} over the bridge workpiece. To refine the requirements, further insights about the definition of MRR can be derived for our task setup.

The normal pressure p_N is defined as the normal force applied to a certain contact region of the surface $A \in \mathbb{R}$. In the case of low friction, the normed force is similar to the normal force F_N, as tangential forces can be omitted. When a spherical workpiece has contact with a plane polishing tool, the contact region has the geometry of an ellipsis. The size and shape of the ellipsis depend on the curvature of the workpiece, the applied force, and some material constants. Due to small expected variations in contact force, we approximate the area of contact to be constant. For this assumption to be valid, we require alignment between tool orientation and workpiece surface normal.

When the tool is moving along the path r^{path}, the relative velocity of the tool v_{rel} can be split into two parts: translational velocity $v_T \in \mathbb{R}$ and rotational velocity $v_B \in \mathbb{R}$. As we use a non-rotating polishing tool, we assume $v_B = 0$. The translational velocity v_T depicts the movement along the path.

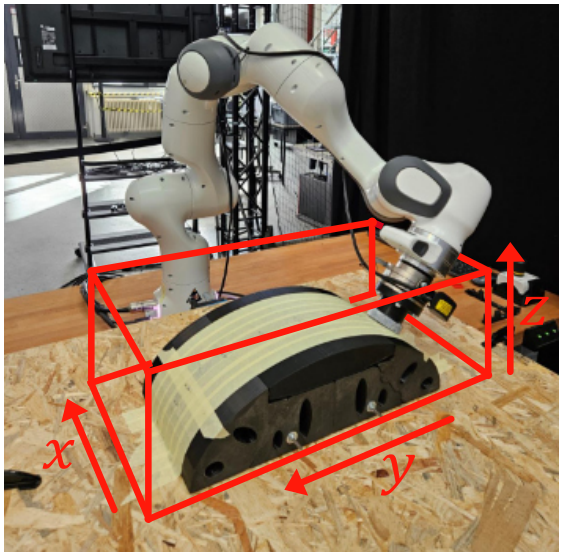


Figure C.1.: Positional safety limitations of the environment and the corresponding task space coordinate system.

Given these insights, we can calculate the material removal rate as

$$\begin{aligned} \frac{dh}{dt} &= K_p \cdot p_N \cdot v_{rel} \\ &= K_p \cdot \frac{F_N}{A} \cdot (v_B + v_T) \\ &= \frac{K_p}{A} \cdot F_N \cdot v_T \end{aligned} \tag{C.1}$$

The term $\frac{K_p}{A}$ is a constant factor. A constant MRR is therefore achieved if a constant force $F_N^{\text{target}} \in \mathbb{R}$ and constant translational velocity $v_T^{\text{target}} \in \mathbb{R}$ is maintained for the entire task.

C.1.2. Safety Limits

In the following, we describe the safety limitations of the polishing environment. A violation of one of the limitations will lead to truncation of the episode. We use four types of limits: positional, orientational, velocity and force limits.

Positional limitations

In some applications, robotic arms are placed close to other facilities or stations of an automation process. Our positional limits restrict the arm movement inside a bounding

box. We define the possible workspace of the robotic arm to be a cuboid, limiting the position $x = (x_x, x_y, x_z)$ of the end-effector close to the workpiece of the polishing process. The workspace cuboid is defined as

$$\begin{aligned} x_x &\in [0.05, 0.15]\,[\mathrm{m}], \\ x_y &\in [-0.23, 0.57]\,[\mathrm{m}], \text{ and} \\ x_z &\in [0.0, 0.2]\,[\mathrm{m}]. \end{aligned} \tag{C.2}$$

Here, the restriction of the x_x-position is tight compared to the other axes since the target path r^{path} is defined in the yz-plane. The asymmetric boundaries of the x_y-position arise because the origin of the y-axis is defined by the initial state of the polishing path. The z-axis has its origin on the table, and the origin of the x-axis is at the border of the workpiece. Note that the positional limitations also passively restrict the robotic arm's joint displacements, so we do not define additional joint limits in our environment. Figure C.1 shows the positional limits.

Orientational limitations

In addition to positional limits, we also restrain the end-effector orientation. Combined with the positional limits, this aims to avoid collision with the surroundings, particularly with the workpiece. We speak of collision when the robotic arm has non-planned contact with the environment, i.e., if a part other than the end-effector has contact with the environment. In Euler angle representation [°], the orientational limits are defined as

$$\begin{aligned} \phi_x &\in [-180, -110] \cup [110, 180], \\ \phi_y &\in [-10, 10], \text{ and} \\ \phi_z &\in [-10, 10]. \end{aligned} \tag{C.3}$$

Here, ϕ_x has a wider span compared to the other two rotation axes ϕ_y and ϕ_z since this rotation axis is used to perform the rotation movement of the end-effector.

Velocity limitations

To ensure safe movement of the robotic arm, we restrict the end-effector velocity $\dot{x}$. The limit of the velocity is defined as $\|\dot{x}\| < 0.5\,\mathrm{m\,s^{-1}}$. Anything above this threshold is considered dangerous for any setting of impedance gains.

Force limitations

We aim to avoid high-force interactions between the end-effector and the workpiece. To yield this, we limit the maximum contact force of the force sensor to a certain threshold $F_N < 25\,\text{N}$. In addition to that, no contact with the environment is allowed when the end-effector position is below $z < 0.01\,\text{m}$. Detecting a contact force in this position means the end-effector has undesired contact with the workpiece table.

C.2. Implementation Details

This section gives implementation details and hyperparameters regarding the RL setup in Appendix C.2.1 and regarding the nominal controller in Appendix C.2.2.

C.2.1. RL Setup and Hyperparameters

We give further details regarding the RL agents in the following.

Reward Definition and Weighting

We use two piecewise functions to transform an error value $e \in \mathbb{R}^+$ of the current state s_t into a reward. The function $l(e)$ is a purely linear mapping from error to reward and can be written as

$$l(e) = \begin{cases} 1 - \frac{e}{e_{max}}, & 0 \leq e \leq e_{max} \\ 0, & e > e_{max}. \end{cases} \tag{C.4}$$

The function $q(e)$ rewards values close to the targets quadratically and values far away from the targets linearly. The function is defined as

$$q(e) = \begin{cases} 1 - \frac{e}{2e_{mid}}, & 0 \leq e \leq e_{mid} \\ \frac{e_{max}^2 - e^2}{2(e_{max}^2 - e_{mid}^2)}, & e_{mid} < e \leq e_{max} \\ 0, & e > e_{max}. \end{cases}$$

Figure C.2 visualizes the two mapping functions.

The robot's path-following behaviour is reinforced by using four reward terms

1. the cross-error $e_{c\parallel}$ parallel to the path,
2. the cross-error $e_{c^\perp}$ perpendicular to the path,
3. the velocity error e_v,
4. and the directional error e_d.

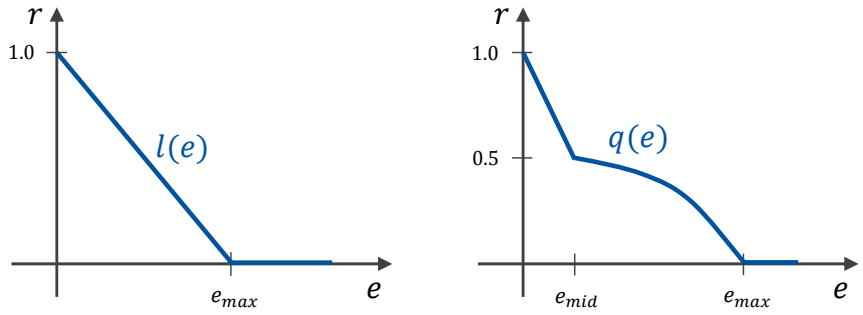


Figure C.2.: We define two functions to map an error to a reward. The left graph shows the linear transformation function $l(e)$. The right graph shows the transformation function $q(e)$ mapping outer values quadratically and values close to targets linearly.

The cross-error is the distance from the current position x_t to the path r^{path}. It can be split into the parallel part $e_{c\parallel}$ and the perpendicular part $e_{c^\perp}$ according to the surface of the workpiece. Here, perpendicular means the part that aligns with the surface normal of the workpiece, essentially displaying the height deviation of the robot from the path. This way, we can define separate weightings for both cross-error parts. Given the position of the closest control point $x_m^{\text{prior,path}}$ according to the current position x_t, the parallel part of the cross-error can be computed as

$$e_{c\parallel}(s_t) = \left(x_{x,t} - x_{x,m}^{\text{prior,path}}\right), \tag{C.5}$$

while the perpendicular part is computed as

$$e_{c^\perp}(s_t) = \sqrt{\left(x_{y,t} - x_{y,m}^{\text{prior,path}}\right)^2 + \left(x_{z,t} - x_{z,m}^{\text{prior,path}}\right)^2}. \tag{C.6}$$

Note that this logic only applies to our task setup, where the x-direction always describes the lateral deviation from the path. Both errors are transformed into rewards $r_{c\parallel}$ and $r_{c^\perp}$ using transformation function $l(e)$.

In addition to that, we reward good velocity behaviour according to the predefined velocity target v_T^{target}. Using the current velocity $\dot{p}$ from the state s, the velocity is computed as

$$e_v(s_t) = |\|\dot{x}_t\| - v_T^{\text{target}}|. \tag{C.7}$$

The error e_v is then transformed into a reward r_v using the linear function $l(e)$.

Since the information about the movement direction is lost when using the norm of

Table C.1.: Reward weightings for the polishing environment in simulation and on hardware.

weights	c_{trunc}	c_{term}	$c_{c^\perp}$	$c_{c^\parallel}$	c_v	c_d	c_f
Value (sim.)	-1	0.1	0.1	0.05	0.3	0.15	0.4
Value (real)	-1	0.03	0.1	0.05	0.35	0.2	0.3

the velocity $\|\dot{x}_t\|$ for the reward r_v, we define an additional movement direction reward. For this, we make use of the current movement direction $d_t = \frac{\dot{x}_t}{\|\dot{x}_t\|} \in \mathbb{R}^3$ and the path direction $d^{\text{path}} \in \mathbb{R}^3$ of the closest control point. The path direction d^{path} can be computed for every control point as the normed distance vector to the subsequent control point. Given these two directions, we can compute the directional error as their alignment angle

$$e_d(s_t) = \arccos\left(d_t \cdot d^{\text{path}}\right). \tag{C.8}$$

The final direction reward r_d can be computed using the piecewise linear function $l(e)$. Splitting the velocity and the direction into two rewards again enables independent reward weighting.

We define a force error based on the specified target force F_N^{target}. For this, the sensed force is normed $\|F_t\| = F_{N,t}$ and subtracted from the target

$$e_f(s_t) = |F_{N,t} - F_N^{\text{target}}|. \tag{C.9}$$

The force error e_f is transformed to the reward r_f using the linear-quadratic function $q(e)$. This transformation function emphasizes the reward for force values $F_{N,t}$ close to the target force F_N^{target}.

Table C.1 shows the weights of the reward terms, while Tab. C.2 defines the boundary values for the piecewise error transformation functions. We distinguish between simulated and real hardware environments. We define wider reward boundaries for the hardware environment because this environment is generally more challenging. Both SAC and CHEQ are trained using the same set of parameters.

SAC Hyperparameters

All SAC-specific hyperparameters for the simulated and hardware CHEQ agents, as well as the SAC variants, are described in Table C.3. To mitigate fluctuating λ^{RL} values on hardware, we choose a larger critic ensemble of 10 neural networks. For all algorithms,

Table C.2.: Reward transformation bounds for the polishing environment in simulation and on hardware.

reward	$r_{c^\perp}$	$r_{c^\parallel}$	r_v	r_d	r_f	r_f
boundaries	e_{max} [m]	e_{max} [m]	e_{max} [$\mathrm{m\,s^{-1}}$]	e_{max} [rad]	e_{max} [N]	e_{mid} [N]
Value (sim.)	0.01	0.01	0.01	0.6	2.0	0.5
Value (real)	0.015	0.03	0.03	0.6	4.0	1.5

Table C.3.: Hyperparameters for the SAC agents.

Environment	Simulation		Real Hardware
Parameter	Value (SAC)	Value (SAC-Ensemble)	
optimizer	ADAM	ADAM	ADAM
η	$1 \cdot 10^{-4}$	$3 \cdot 10^{-4}$	$3 \cdot 10^{-4}$
γ	0.99	0.99	0.99
ξ	0.005	0.005	0.005
$\lvert\mathcal{D}\rvert$	10^6	10^6	10^6
hidden-layers (critic NNs)	$[256, 256]$	$[256, 256]$	$[256, 256]$
hidden-layers (policy NN)	$[256, 256]$	$[256, 256]$	$[256, 256]$
# of critic networks	2	5	10
batch-size	256	256	256
UTD	-	1	2
non-linearity	ReLU	ReLU	ReLU
α	0.2	0.2	0.2
α-tuning	True	True	True
initial random steps	15000	15000	15000

we include a random sampling phase for the first 15×10^3 steps where we sample the RL action uniformly random and do not update our agent. In this setting, we vary λ^{RL} between [0.2, 0.3] for the CHEQ agent.

CHEQ Hyperparameters

Table C.4 provides the hyperparameters specific to the CHEQ algorithm.

As the control frequency $f^{\mathrm{RC,real}} = 20\,\mathrm{Hz}$ of the real hardware environment is lower than the control frequency of the simulation $f^{\mathrm{RC,sim}} = 50\,\mathrm{Hz}$, it takes less episodic steps to perform the task successfully. We thus choose different horizon lengths for the two environments. This results in less accumulated return, which alters the values of the Q-networks. Hence, the Q-networks show lower values in the hardware environment than in the simulation. The final return in simulation is approximately half the simulated return. Lower values result in lower standard deviations or total uncertainty values.

Table C.4.: Hyperparameters of CHEQ used for the weight adaption function λ^{RL} for the hybrid agent.

Parameter	Value (sim.)	Value (real)
$\lambda_{\min}$	0.2	0.2
$\lambda_{\max}$	1.0	1.0
$u_{\min}$	0.02	0.015
$u_{\max}$	0.2	0.1
horizon T	380	150

Table C.5.: Hyperparameters for the nominal controller in the simulation and on hardware. In addition, we define the suboptimal controller that is used for CHEQ. As optimal gains for the real process are unknown, we label the parameters for the real process as suboptimal as well.

Environment	Simulation		Real Hardware
Parameter	baseline	suboptimal	
a_r [m]	0.016	0.2	0.015
$\Delta x^{\mathrm{indent}}$ [m]	0.012	0.015	0.002
spacing [m]	$5 \cdot 10^{-4}$	$5 \cdot 10^{-4}$	$5 \cdot 10^{-4}$

Thus, lower uncertainty limits must be chosen for the hardware setup to achieve the same behaviour. Since we focus on safe exploration in our hardware experiments, we have increased the upper limit $u_{\max}$ to achieve safer exploration behaviour through lower λ^{RL} values.

C.2.2. Nominal Control Setup and Hyperparameters

Table C.5 provides the hyperparameters for the nominal controller for the simulated and real environment.

The spacing parameter describes the equidistant spacing between neighbouring control points used during path r^{path} creation. This setup consistently provides ~ 50 control points as close control points P^{close}. Further, note that we use a low value for the indentation $\Delta x^{\mathrm{indent}}$ in the real environment. This is due to the fact that the movement of the real robot is more cumbersome than in the simulated environment. Setting a lower indentation leads to a stronger movement in the direction of the path.

Baseline and Tuned Impedance Gains

To define our baseline control (*Nominal (tuned)*), we perform hyperparameter tuning with Bayesian Optimization to find the optimal static impedance gains for the baseline

controller in simulation. In the end, we choose the run resulting in the highest score as the impedance gains of the *Nominal (tuned)* controller, that could be found after ~ 500 runs. For the control prior used for the CHEQ agent, we further define a suboptimal controller (*Nominal (untuned)*) determined by rounding up the values.

Table C.6.: Definition of impedance gains for the fixed gain nominal controllers.

Controller	k_x	k_y	k_z	k_{ϕ_x}	k_{ϕ_y}	k_{ϕ_z}	ζ
Nominal (untuned)	500	160	50	500	500	500	1.0
Nominal (tuned)	500	107	68	500	500	500	0.9562
Nominal (hardware, untuned)	800	1085	900	150	80	25	1.0

For the partially tuned nominal controller *Nominal (tuned section-wise)*, we divide the workpiece into 5 distinct sections. The division of the task is based on the current y-position of the end-effector. We find the gains for the partitioned controller by iteratively tuning with Bayesian Optimization for each section and using the best-found gains for the previous sections. Table C.7 provides the final impedance gains configurations using this iterative method.

Table C.7.: Impedance profile for the *Nominal (tuned section-wise)* controller.

partition	$k_y\,[\mathrm{N\,m^{-1}}]$	$k_z\,[\mathrm{N\,m^{-1}}]$	ζ
S1	98	59	~ 0.9445
S2	91	69	~ 0.8563
S3	108	71	~ 0.9452
S4	137	72	~ 1.0294
S5	101	65	~ 1.1946

List of Acronyms

AHRL	Adaptive Hybrid Reinforcement Learning
ANN	Artificial Neural Network
CHEQ	Contextualized Hybrid Ensemble Q-Learning
CM	Center of Mass
CNN	Convolutional Neural Network
DDPG	Deep Deterministic Policy Gradient
DOF	degree of freedom
DP	Dynamic Programming
DPG	Deterministic Policy Gradient
DQN	Deep Q-Network
FNN	Feedforward Neural Network
IQM	Interquartile Mean
LS	Least Squares
LSTM	Long Short-Term Memory
MDP	Markov Decision Process
ML	Machine Learning
MPC	Model Predictive Control
MRR	Material Removal Rate
MSBE	Mean Squared Bellman Error
MSE	Mean Squared Error
NN	Neural Network
PMF	Probability Mass Function
REDQ	Randomized Ensemble Double Q-learning

RL Reinforcement Learning
RNN Recurrent Neural Network

SAC Soft Actor-Critic
SAE Spatial Autoencoder
SGA Stochastic Gradient Ascent
SGD Stochastic Gradient Descent

TD3 Twin Delayed DDPG

UTD Update-to-Data

VAE Variational Autoencoder
VIC Variable Impedance Control

List of Figures

2.1. The agent-environment interaction in an MDP, adapted from Sutton and Barto [21]. In each time step, the agent observes a state s_t and a reward r_t and chooses an action a_t. This action leads to the next state state s_{t+1}, and the agent receives reward r_{t+1} for the transition. 21

2.2. Structure of a typical autoencoder for image data, consisting of an encoder $E_{\psi_{\text{enc}}}$ to map the image input I to a latent representation $z = E_{\psi_{\text{enc}}}(I)$ and a decoder reconstructing the input approximatively from the latent representation as $\hat{I} = D_{\psi_{\text{dec}}}(z)$. 28

2.3. The spatial soft-argmax layer takes the activation maps M_n of the preceding convolutional layer as input. It then applies a 2D softmax-normalisation to each map to obtain a PMF S_n before taking its expectation, resulting in a 2D weighted mean pixel coordinate [34]. 29

3.1. SAE extracts 2D positions from images via a spatial bottleneck (Figure a). The SAE encoder is then integrated into an RL framework to obtain a state representation for immeasurable objects (Figure b). 38

3.2. In this Figure, we show the complete learning framework. First, a video dataset is collected using a random policy (Figure a). An SAE is then trained on this dataset (Figure b), learning to capture 2D positions of moving objects in its latent space. Finally, the SAE's encoder is used to extract a state representation from live images in the RL process (Figure c). 41

3.3. Figure a shows an exemplary 3D offset from the CM and Figure b illustrates the resulting time-varying 2D offset for a moving object. If any fixed point of an object is tracked, the feature point trajectory $z_{\tau,n}$ that coincides with $\hat{x}_{\tau,k}$ (red) is considered to be equally informative as the ground-truth point trajectory $x_{\tau,k}$ of the CM (white). However, this constant 3D offset $r_{\hat{o}_k o_k}$ results in a time-variant 2D offset $r_{\hat{x}_k x_k}$, yielding a large tracking error. 45

3.4. Figure a shows the PandaPush-v3 task with the positions x_k of the three objects of interest marked. We show selected ground truth, keypoint, and transformed keypoint trajectories in white, red, and blue for the end effector (Figure b) and cube (Figure c). 48

3.5. Basic-kp32 and KeyNet-vel-std-bg tracking errors $e_{n_k^*,k}$ for $K = 3$ objects over epochs. 50

3.6. Box plots of the tracking error $e_{n_k^*,k}$ for $K = 3$ ground truth objects. . . . 51

3.7. Tracking capabilities $\overline{\mathrm{TC}_k}$ for $k = 3$ ground truth objects of the baseline architectures. 51

3.8. Success rate for different states: keypoints only in Figure a and extended state in Figure b. 53

4.1. We show a standard RL setting in Figure a, hybrid RL settings from prior work in Figure b and our contextualized hybrid RL setting based on RL action a_t^{RL} and weighting factor λ_t in Figure c. 61

4.2. We illustrate different hybrid RL formulations on a cart pole system with a biased control prior, pushing to the left. The environment is shown in Figure a. Figure b shows the return of different hybrid agents in the case where λ_t^{RL} is fixed, and Figure c shows the return with variable λ_t^{RL}. Only the contextualized hybrid RL agent observing dynamics $\hat{p}$ can cope with both scenarios. 63

4.3. Racing task and the training track. 67

4.4. Performance of all trained RL approaches in terms of evaluation return and training failures on the training track. We compare the training performance of CHEQ with fixed-weight hybrid RL and standard RL in Figure a. Figure b shows a comparison of our hybrid CHEQ-UTD20 agent to a standalone REDQ agent using the same UTD ratio. Lastly, we compare CHEQ to prior work in AHRL in Figure c. 70

4.5. Figures a and b show a comparison of the return and the failures of ten trained models per algorithm on ten transfer tracks. Figure c illustrates the development of λ^{RL} of the CHEQ-UTD20 agent for an exemplary transfer track. 71

4.6. The 10 transfer tracks and the corresponding zero-shot transfer of one seed of CHEQ-UTD20. 71

5.1. Figures a and b show the robot setup in simulation and on the real hardware. The robot's task is to polish along a path r^{path} defined via seven distinct via-points. The polishing path runs along the y-axis. For our later analysis of fixed-gain controllers, we split the task into five segments shown in Figure c. 77

5.2. Figure a shows the complete control pipeline with higher-level reference controller RC and lower-level impedance controller IC. Figure b gives a detailed view of the RC, an adaptive hybrid RL agent based on CHEQ. . 79

5.3. Generation of the next desired control point. The next desired control point $p^{\text{prior,ref}}$ is chosen as the farthest point along the task direction out of the set of close control points. 81

5.4. Comparison of force and velocity over one episode in Figure a and boxplots for the five sections in Figures b and c. The fixed gain controller tuned with BO shows increased performance, and tuning per section further improves behaviour. 83

5.5. Figures a-c show a comparison of return, fails and y-space coverage of CHEQ and the SAC variants. CHEQ achieves the highest performance with only 56 fails. Over training, the distribution of the weighting factor λ^{RL} rises to 1 as shown in Figure d. Figures e and f show the trained agent after 2.5×10^6 steps. The agent achieves good polishing behaviour by adapting the gains. 85

5.6. We show the profiles of the zero-shot transfer objects in orange and the original bridge object in blue. Figures b, d, f and h show the force, velocity and weighting behaviour of the trained policy on the respective zero-shot transfer objects. CHEQ detects whenever the object differs from the training object and reacts with a decrease in the weighting factor λ^{RL}. 88

5.7. We show a close look at the initial training return in Figure a. Longer training does not further improve the return (cf. Figure b) but increases the weighting factor (cf. Figure c). Figure f shows the force and velocity behaviour of an early evaluation run after $\sim 90 \times 10^3$ training steps and a later evaluation run after $\sim 930 \times 10^3$ training steps. In both cases, the agent achieves good polishing behaviour. Further, the accumulated training fails are shown in Figure d. 90

A.1. We show the reconstruction loss over training epochs for the Basic-kp32 architecture in Figure a and for the KeyNet architecture in Figure b. . . . 115

A.2. Trajectories from sample episodes for end-effector tracking in Figure a and cube tracking in Figure b. Ground truth points (white), untransformed keypoints obtained with KeyNet-vel-std-bg (red), and transformed keypoints (blue) are plotted over time. 116
A.3. Model ablations: tracking capability $\overline{\mathrm{TC}_k}$ for $k = 3$ ground truth objects of the adapted architectures. 117
A.4. KeyNet-vel-std-bg tracking errors $\overline{\mathrm{TC}_k}$ for $k = 3$ ground truth objects over epochs. 117
A.5. Box plots of tracking errors over 24 runs of the considered architecture modifications. 117

B.1. Comparison of CHEQ-UTD1 with different $u_{\max}$ thresholds. Plotting return (a) and number of fails (b) for the racing environment. 122
B.2. Distributions of λ^{RL} over training steps shown for all hybrid agents. . . . 122
B.3. Comparison of return (a) and number of fails (b) CORE and C-CORE runs with different C parameters. 123
B.4. Comparison of return (a) and number of fails (b) of BCF and C-BCF runs with different σ_{prior} parameters. 124
B.5. Scatter plot of the return and number of fails for different hyperparams for BCF, CORE, C-BCF, C-CORE. CHEQ-UTD20, CHEQ-UTD1 + Prior in Comparison 124

C.1. Positional safety limitations of the environment and the corresponding task space coordinate system. 126
C.2. We define two functions to map an error to a reward. The left graph shows the linear transformation function $l(e)$. The right graph shows the transformation function $q(e)$ mapping outer values quadratically and values close to targets linearly. 129

List of Tables

1.1. Overview and link between individual gaps and the chapters of this thesis. 9

A.1. Hyperparameters for SAE training. 111
A.2. Hyperparameters for RL training. 114
A.3. Specifications of state representation vector configurations used for RL experiments. 114

B.1. Shared Hyperparameters for SAC and CHEQ. 119

C.1. Reward weightings for the polishing environment in simulation and on hardware. 130
C.2. Reward transformation bounds for the polishing environment in simulation and on hardware. 131
C.3. Hyperparameters for the SAC agents. 131
C.4. Hyperparameters of CHEQ used for the weight adaption function λ^{RL} for the hybrid agent. 132
C.5. Hyperparameters for the nominal controller in the simulation and on hardware. In addition, we define the suboptimal controller that is used for CHEQ. As optimal gains for the real process are unknown, we label the parameters for the real process as suboptimal as well. 132
C.6. Definition of impedance gains for the fixed gain nominal controllers. . . . 133
C.7. Impedance profile for the *Nominal (tuned section-wise)* controller. 133