Sample-Efficient Reinforcement Learning of Robot Control Policies in the Real World

by

Kevin Sebastian Luck

A Dissertation Presented in Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

Approved October 2019 by the
Graduate Supervisory Committee:

Hani Ben Amor, Chair
Daniel Aukes
Georgios Fainekos
Jonathan Scholz
Yezhou Yang

ARIZONA STATE UNIVERSITY

December 2019

ABSTRACT

The goal of reinforcement learning is to enable systems to autonomously solve tasks in the real world, even in the absence of prior data. To succeed in such situations, reinforcement learning algorithms collect new experience through interactions with the environment to further the learning process. The behaviour is optimized by maximizing a reward function, which assigns high numerical values to desired behaviours. Especially in robotics, such interactions with the environment are expensive in terms of the required execution time, human involvement, and mechanical degradation of the system itself. Therefore, this thesis aims to introduce sample-efficient reinforcement learning methods which are applicable to real-world settings and control tasks such as bimanual manipulation and locomotion. Sample efficiency is achieved through directed exploration, either by using dimensionality reduction or trajectory optimization methods. Finally, it is demonstrated how data-efficient reinforcement learning methods can be used to optimize the behaviour and morphology of robots at the same time.

DEDICATION

This dissertation is dedicated to the memory of my dear friend Hong Linh Thai, who was taken the chance to complete his own dissertation.

ACKNOWLEDGMENTS

I am thankful for the support of quite a number of people who helped me during the typical ups and downs of a PhD. First and foremost I would like to thank my fiancée Tamara Jacqueline Blätte, who always believes in me and brings a smile to my face every single day. Especially on these days when a robot would not do what it was supposed to do.

I thank my advsior Prof. Heni Ben Amor for giving me the opportunity to collaborate with him at the Interactive Robotics Lab, which brought me from the cold winter in Finland to the desert climate of Phoenix. He gave me the academic freedom to follow my ideas and evaluate them on real world tasks, both at ASU and in collaboration with other research groups. I further thank my committee members Daniel Aukes, Georgios Fainekos and Yezhou Yang for their support, comments and providing input on my research.

I thank my collaborator and committee member Jonathan Scholz for his mentorship, guidance and ideas when I visited DeepMind in London, UK. Fumin Wang and Mel Vecerk made sure I always felt welcome in London and helped me to answer numerous technical questions. The idea of using a cylinder suspended from a string for a real world insertion task can be attributed to a conversation with Oleg Sushkov, probably over lunch or during a coffee break on the rooftop overlooking King's Cross. A special thanks goes to Thomas Rothörl who not only introduced me to Scottish whiskey during many a Friday night but with whom I also had many interesting, important and insightful conversations during my time in London.

Many project ideas and their realizations can be attributed to me meeting a sombrero-wearing PhD student one morning, 15 minutes before our class would start. Michael Andrew Jansen introduced me to the fascinating world of insects and I was lucky to meet such an enthusiastic person eager to work with me on projects in robotics.

Subsequently, I did not only end up doing robotic experiments in the desert of Arizona but also discovered my research interests in the adaptation of robot morphologies using reinforcement learning.

I am thankful for the opportunity Roberto Calandra gave me to work with him and the team at Facebook AI in Menlo Park, and believing in my ideas for the adaptation of design and behaviour of robots. I thank Akshara Rai, Sarah Bechtle and Behnoosh Parsa not only for their companionship during my time at Facebook AI, but also for the orange juice on my desk in the morning. And the coffee breaks, of course. Special thanks goes here to Ge Yang, who helped me to run my experiments when we needed additional data for a conference deadline.

I am thankful for the members of my family - and my new family in-law - who supported me over the years and during my research endeavors, which lead me repeatedly to new countries and overseas.

I would also like to thank every member of the Interactive Robotics Lab who supported me in my research and my academic advisor Arzuhan Kavak who answered uncountable questions of mine. Other people who I owe thanks to are (in no particular order): Laura Koponen, Teresa Weßels, Claudius Kleemann, Rebecca Alef, Johannes Alef, Meike Trillmann, Annemarie Arnold, Thomas Hesse, Georg Lutz, Veronika Lutz, Antony Hatton, Mehrdad Zaker, Paulina Davison, Joni Pajarinen, Simon Stepputtis, Joe Campbell, Erik Berger, Gerhard Neumann, Jan Peters, and Ville Kyrki.

TABLE OF CONTENTS

Page

LIST OF TABLES ix

LIST OF FIGURES x

CHAPTER

1 INTRODUCTION 1

1.1 Problem Statement 4

1.2 Contributions 8

1.2.1 Methodical Contributions 9

1.2.2 Contributions towards Real-World Applications 10

1.2.3 Algorithmic Properties 11

1.3 Thesis Outline 12

2 SPARSE LATENT SPACE POLICY SEARCH 15

2.1 Introduction 16

2.2 Problem Statement 18

2.3 Group Factor Policy Search 19

2.3.1 Variational Inference for Policy Search 20

2.3.2 Formulation of Group Factor Policy Search 22

2.3.3 Derivation of Update Equations 26

2.3.4 Algorithm 29

2.4 Evaluation 29

2.4.1 Setup of the Evaluation 31

2.4.2 Results 32

2.4.3 Importance of the Choice of Groups 33

2.5 Experiment: Lifting a Leg 35

2.6 Experiment: Learning Locomotion on Granular Media 36

CHAPTER Page

2.7 Conclusion 37
3 EXTRACTING BIMANUAL SYNERGIES WITH SAMPLE-EFFICIENT REINFORCEMENT LEARNING 39
3.1 Introduction 40
3.2 Related Work 42
3.3 Extracting Synergies with Policy Search 44
3.3.1 Group Factor Analysis for Synergies 44
3.3.2 Group Factor Policy Search 46
3.3.3 Transfer Learning with GrouPS 49
3.4 Experiments 50
3.4.1 Experimental Setup 51
3.4.2 Groups 52
3.4.3 Used Basis Functions 52
3.4.4 Reward Function 53
3.4.5 Objects 54
3.4.6 Time and Sample Size 55
3.4.7 Reproducibility 55
3.4.8 Experiments 55
3.5 Results 57
3.6 Discussion 57
3.7 Conclusions 62
4 IMPROVED EXPLORATION THROUGH LATENT TRAJECTORY OPTIMIZATION IN DEEP DETERMINISTIC POLICY GRADIENT . . 63
4.1 Introduction 64

CHAPTER Page

4.2 Related Work 66
4.3 Method 68
4.3.1 Image Embedding 69
4.3.2 Latent Dynamics 70
4.3.3 Deep Reinforcement Learning 71
4.3.4 Optimized Exploration 74
4.4 Experiments 75
4.4.1 Evaluation in Simulation on the Cheetah Task 77
Comparison between Ornstein-Uhlenbeck and optimized exploration 77
Comparison between different planning horizons 79
Comparison between different objectives 79
4.4.2 Insertion in the real world 82
4.5 Discussion 83
4.6 Conclusion 85
5 DATA-EFFICIENT CO-ADAPTATION OF MORPHOLOGY AND BEHAVIOUR WITH DEEP REINFORCEMENT LEARNING 87
5.1 Introduction 88
5.2 Related Work 89
5.3 Problem Statement 91
5.4 Optimization of Morphology and Behaviour 92
5.4.1 Using the Q-Function for Design Optimization 93
5.4.2 Design Generalization and Specialization of Actor and Critic 94
5.4.3 Exploration and Exploitation of Designs 95
5.4.4 Fast Evolution through Actor-Critic Reinforcement Learning 96

CHAPTER Page

5.5 Experimental Evaluation 98
5.5.1 Experimental Setting 98
5.5.2 Co-adaptation Performance 102
5.5.3 Visualization of the Latent Design Space 109
5.5.4 Evolution of Walker 110
5.6 Conclusion 110
6 FUTURE WORK 115
6.1 Bridging the Gap from Simulation to Reality for the Co-Adaptation of Morphology and Behaviour 115
6.1.1 Assumptions Required for Successful Transfer Learning between Environments 115
6.1.2 Proposed Approach for Transfer Learning between two Environments 118
6.1.3 Proposed Experiment for Evaluating the use of Transfer Learning for the Co-Adaptation of Morphology and Design . 118
6.2 Using latent Design-dependent Action Spaces 120
7 CONCLUSION 123
REFERENCES 125
APPENDIX
A COAUTHOR APPROVAL FOR CHAPTER 2 133
B COAUTHOR APPROVAL FOR CHAPTER 3 135

LIST OF TABLES

Table Page

3.1 The Groups for the Joints of the Baxter Robot Chosen for the Experiments Presented in this Chapter. 60

3.2 Comparison of GrouPS with the Proposed Modification for Reusing Synergies from Different Tasks 60

4.1 The Average Success Rate of Insertion for Policies Trained by DDPG with Standard Ornstein-Uhlenbeck Exploration or Trajectory Optimization with Varying Planning Horizons. 80

LIST OF FIGURES

Figure		Page
1.1	The Basic Reinforcement Learning Framework.	2
1.2	The Sim-to-Reality gap and the Lab-to-Reality gap	7
1.3	Dimensionality Reduction Methods are Able to Map High-dimensional State or Action Spaces to a Low-dimensional Manifold.	8
1.4	The performance of the Deterministic Policy of a Sample-Efficient Reinforcement Learning Algorithm.	10
1.5	Thesis Overview	13
2.1	The Main Idea of Group Factor Policy Search.	17
2.2	Graphical Model in Plate Notation for Group Factor Policy Search.	27
2.3	Two Simulated Arms with Six Degrees-Of-Freedom and the Same Base in their Initial Position.	30
2.4	Comparison between PePPEr, PoWER, Natural Actor-Critic and Three Instances of the GrouPS Algorithm on the Presented Simulated Task.	30
2.5	Comparison between the Original Chosen Four Groups and Three Permutations of the Groups.	33
2.6	Comparison between the Original Grouping and Two Other Variants with a Different Splitting Point.	34
2.7	Final Policy Found by the GrouPS Algorithm After 100 Iterations.	35
2.8	A Policy Trained by GrouPS in the Desert of Arizona.	37
2.9	The Evaluation of GrouPS on a Bio-Inspired Robot Manufactured from a Paper-Laminate.	37
3.1	A Bimanual Robot Learns to Lift an Object while Simultaneously Identifying Synergies Among the Control Variables.	41

Figure Page

3.2 In Each Iteration a Sample, i.e. Trajectory, is First Simulated and to be Accepted by the Process (or a Human Operator) for Execution. 52
3.3 The Current Height of an Object is Approximated by Color Filtering. . 53
3.4 The Four Different Objects Lifted Next to each other with the ICRA Duck as Size Reference. 54
3.5 The Final Sequence of Actions for Lifting a Cardboard Box Found by GrouPS. 54
3.6 One Final Sequence of Actions for Lifting the Orange Ball Found by GrouPS. 55
3.7 A Comparison Between the GrouPS and PoWER Algorithm on a Lifting Task with the Orange Ball. 56
3.8 The Final Pose of Trajectories for Lifting an Object. 58
3.9 A Sequence of Transformation Matrices $\mathbf{W}$ Computed in each Iteration t. 59
3.10 Two Synergies and their Combinations Found During the Execution of the GrouPS Algorithm for Learning to Lift the Orange Ball. 59
4.1 A Baxter Robot Learning a Visuo-Motor Policy for an Insertion Task using Efficient Exploration in Latent Spaces. The Peg is Suspended from a String. 65
4.2 The Experimental Setup in which a Baxter Robot has to Insert a Blue Cylinder into a White Tube. 66
4.3 The Original DDPG Algorithm can be Reformulated such that a Value Function is Used 72

Figure Page

4.4 The Proposed Exploration Strategy Unrolls the Trajectory in the Latent Space and Uses the Value/Q-Value to Optimize the Actions of the Trajectory. 73
4.5 Comparison between DDPG Using Exploration with Optimization and Classical Exploration using an Ornstein-Uhlenbeck Process on the Simulated Cheetah Task. 78
4.6 Comparison between DDPG Using Exploration with Optimization (Orange) and Classical Exploration Using an Ornstein-Uhlenbeck Process (Blue) on the Simulated Half-Cheetah Task while Using a Value Function as Critic. 78
4.7 Exploration through Optimization Evaluated with Different Horizons for the Planning Trajectory on the Simulated Half-Cheetah Task. 79
4.8 Comparison between Three Different Objective Functions for Optimized Exploration on the Simulated Half-Cheetah Task. 80
4.9 Comparison between Exploration with an Ornstein-Uhlenbeck and Exploration through Optimization on the Insertion Task in the Real World. 81
5.1 It is Proposed to Use an Actor and Critic for Design Exploration Instead of Creating Design Prototypes and Evaluating their Performance in Simulation or the Real World. 90
5.2 The Four Simulated Robots used in our Experiments. 99
5.3 Comparison of the Proposed Approach and Using Trials to Evaluate the Optimality of Candidate Designs. 103

Figure Page

5.4 Comparison Between the Proposed Approach of Using the Actor and Critic to Optimize the Design Parameters and Using Trials to Evaluate the Optimality of Candidate Designs. 103
5.5 Evaluation on the Half-Cheetah Task of Two Different Global Optimization Algorithms when Optimizing the Design via Rollouts. 104
5.6 Comparison between the Proposed Method and Population-based Approaches from [1] Using the Simulator to Evaluate Design Candidates. . 106
5.7 Evaluation of the use of Population and Individual Networks. 107
5.8 Evaluation of using Batches of Start States in the Objective Function in Eq. 5.6. 107
5.9 First Two Principal Components of the Six Dimensional Design Space of Half-Cheetah as Computed with PCA 108
5.10 The Visualized Cost Landscape of the Design Space of Half Cheetah. . . 109
5.11 The Visualized Cost Landscape of the Design Space of Walker. 110
5.12 The Visualized Cost Landscape of the Design Space of Hopper. 111
5.13 Design Space of the Daisy Hexapod. 112
5.14 First Two Principal Components of each Design Space as Computed with PCA. 113
5.15 A Selection of Designs for Half-Cheetah, Generated from the Principal Components. 113
5.16 Designs ξ_{Opt} Selected by the Proposed Method for the Half-Cheetah Task. 114
6.1 Latent Action Space Extracted with a Neural Autoencoder Network. . 122
6.2 Latent Action Space Sampled in the High-Dimensional Action Space . . 122

CHAPTER 1

Introduction

"The nice thing about artificial intelligence is
that at least it's better than artificial stupidity."

Terry Pratchett & Stephen Baxter

The Long War

The general goal of reinforcement learning (RL) [2, 3], a sub-field of machine learning, is to alleviate the requirement for a human to define control policies for autonomous agents. Defining these policies often requires significant effort, time and analysis. Instead, the idea in reinforcement learning is to allow an agent to select the actions to be performed on its own and assigning rewards to the resulting state transitions [4] (Fig. 1.1). Agents using reinforcement learning algorithms strive to choose actions in virtual or real environments such that a reward function, which assigns a numerical value to state transitions with higher values for desired outcomes, is maximized over a sequence of steps [2, 3, 5]. This process enables autonomous systems which require only an objective function to master a given task. The true power of reinforcement learning is that the reward function does not have to be fully known by the algorithm itself. Instead, the reward function is a black-box [6], which can be queried by the algorithm for values but whose exact nature is not necessarily known. This property allows for the use of a wide range of reward functions, which can even be non-differentiable, non-smooth or whose values can even be generated by querying a human operator for a performance extimate.

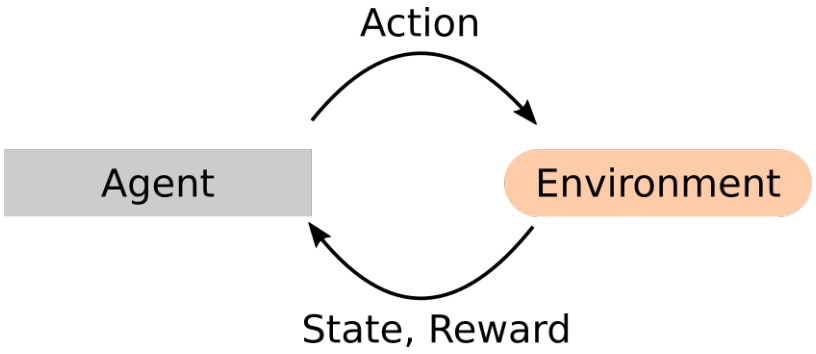


Figure 1.1: An agent interacts with the environment via actions and receives the state the environment is in and a reward. The reward is assigning a numerical value to state-action pairs, which is higher if the outcome is desired.

Developments in reinforcement learning have led to a series of breakthroughs in the recent past, such as beating the world champion in the Japanese board game Go [7] or the card game Poker [8, 9]; achievements comparable to the breakthroughs in Chess with Deep Blue in 1997 [10]. Similar achievements in the multiplayer computer games Starcraft [11, 12] and Dota 2 are on the horizon. All these results became possible through the collection and processing of large amounts of training data prior or during deployment. However, the implicit assumption made in many of these scenarios is that the environment and its variables are static and use deterministic transitions and dynamics. Reinforcement learning in the real world remains a challenging problem, especially if environmental variables are unknown, difficult to model in simulation or non-stationary. The collection of large training data sets is often impossible or costly, especially in robotics, and only sensible if the morphology and dynamics of the (robotic) agent and respective environmental variables remain constant. Thus, the use of sample- and data-efficient reinforcement learning algorithms is required. Sample-efficient reinforcement learning algorithms aim to minimize the number of samples, i.e. the number of episodes or executions in the environment, and not necessarily the

computational effort. This is under the assumption, that acquiring experience (in the real world) requires significantly more time and effort than processing this data.

Robotics is one of the engineering sciences which benefits considerably from research advances made in the field of reinforcement learning. Using reinforcement learning to learn motor skills enables robots to solve tasks in manipulation [13–17], locomotion [18–20] and even flying[21, 22]. In the area of manipulation tasks, reinforcement learning algorithms help to find motor skills for grasping or manipulating objects in the real world for complex robots with multiple degrees-of-freedom. Creating such motor skills by hand is often undesirable due to the complexity of possible motions, periodic changes in the environment, noisy or unreliable sensors, or inaccuracies when recording movements on robots. While manipulation tasks can be trained in simulation, such data created by simulators becomes increasingly unreliable as the robot interacts physically with the environment due to the complex nature of forces generated through contacts. This problem is also referred to as the simulation-to-reality gap [15]. Slow progress is made to close the simulation-to-reality gap for certain tasks and robots [15].

I will present in this thesis a number of reinforcement learning algorithms suitable for learning motor skills in the real world in the (complete) absence of simulators and utilizing latent spaces. This approach circumvents the problem of the simulation-to-reality gap but requires strategies to decrease the amount of time and samples required for learning motor skills in the real world. Thus, all algorithms presented in this work make either use of dimensionality reduction methods (Fig. 1.3) or repurpose value functions as objective functions for a faster training process. By utilizing low-dimensional latent spaces [23–27] during the training process, the number of samples and data required can be reduced by exploiting the latent structures of a given task, enabling a fast training process. This approach is based on the idea,

that the action and state space of many tasks can be represented by low-dimensional manifolds, especially in manipulation and locomotion tasks [28–32].

1.1 Problem Statement

The work presented in this thesis aims to develop and provide algorithms suitable for learning complex control strategies for physical systems in the real world. Two main application areas are considered: Manipulation and locomotion. In the presented manipulation tasks, a uni- or bi-manual robot has to handle objects with its end-effectors to achieve a defined goal such as lifting an object or inserting a cylinder into a tube. The underlying hypothesis is that the learning algorithms are able to uncover movement strategies quickly and can be adapted to new tasks when provided with an objective/reward function. In contrast to classical control algorithms or learning algorithms requiring large datasets, sample- and data-efficient reinforcement learning algorithms aim at avoiding three common problems:

Curse of Dimensionality The curse of dimensionality [33] describes the phenomenon that a search space with an increasing number of dimensions increases its volume exponentially and, thus, reinforcement learning and other optimization algorithms require more data during the training process. In reinforcement learning, the curse of dimensionality can affect two spaces: the state space and the action space. The state space describes the current state the environment and robot are in and can encode such information as images, joint accelerations and positions. The action space describes the possible actions the reinforcement learning algorithm can choose from and are usually joint positions, velocities or accelerations. It holds for both spaces, that the more degrees-of-freedom a robot has, the higher is the dimensionality of state and action space. In turn, the amount of samples required by learning algorithms for finding successful

policies to solve the given task increases. However, for many robots and tasks, low dimensional sub-spaces can be found which, for example, encode task-related motor skills. Learning along these low dimensional manifolds can decrease and counter the effect which the curse of dimensionality has on the learning process. This makes the use or integration of dimensionality reduction methods into reinforcement learning algorithms highly desirable.

System Identification Simulating robotic hardware requires usually a process called system identification [34, 35], that is identifying and fine-tuning the (mathematical) model of a robot based on its behaviour in the real world. System identification helps to reduce the sim-to-reality-gap by identifying all properties which are relevant or influence the dynamics of the overall-system. The process itself, however is lengthy and time-consuming and ideally done only once. Yet, most hardware changes its dynamical properties over its life time due to wear-and-tear. Additionally, manufacturing new robots or altering their designs introduces small manufacturing errors making each robot unique in its dynamical properties. This requires periodic re-calibrations of the simulation models, which is not practical in reality.

Simulation- and Lab-to-Reality-Gap A large body of work assumes that simulations are available and model the real world accurately enough. However, while this assumption might hold for a certain subset of non/low-contact tasks, in which the robot or its end-effector have minimal interaction with the environment, the gap between simulation and the real world widens in cases of contact-rich tasks or when handling deformable objects and granular media. This is mostly due to the computational requirement to make simplifications at contact points between rigid objects in simulation. Additionally, the accuracy of simulations

might have to be decreased for reasonable upper limits of computation times for reinforcement learning experiments. In other words, if one simulates a task with simplified models, the overall run-time of the simulation and learning process might be much smaller. This is comes at the price of increasing inaccuracies between real-world and simulation (Fig. 1.2). This often leads to situations, where movement strategies learned in simulation do not achieve the same performance in the real world or do not succeed at meeting the objectives. Increasing the simulation accuracy in order to close the sim-to-reality-gap, on the other hand, will lead to the situation where simulation takes considerably more time than training directly in the real world and becomes generally unpractical.

Another problem occurs when experiments are brought from the lab to the real environment. Most experimental systems are designed and used under controlled laboratory environments. This usually means that environmental variables such as temperature, humidity and light intensity are being held constant over the course of the experiment or when learning control policies (Fig. 1.2). However, the reality outside of the laboratories is that the human has little to no control over environmental variables. Thus, a changing environment requires periodically adapting robot behaviour to the changing conditions. Human oversight and intervention is simply not practical on a large scale and thus autonomous systems are required which are able to adapt themselves automatically. Moreover, while classical control algorithms can provide a limited adaptability with respect to environmental variables known to influence robot performance in a number of cases, it is generally a priori not clear which variable or combination thereof should be considered and how these affect the optimal policy. For example, in a series of experiments in which a robot was trained both in the lab and the

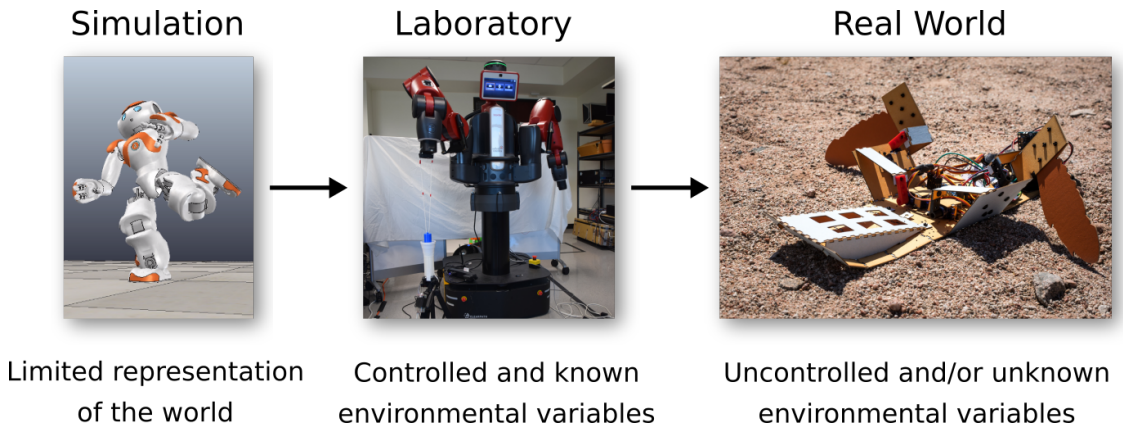


Figure 1.2: Two elementary gaps exist in robot learning: The simulation-to-reality and the lab-to-reality gap. The simulation-to-reality gap describes the often encountered issue that, due to simplifications in modeling, behavioural strategies learned in simulation do not transfer successfully to the real world. On the other hand, the lab-to-reality gap occurs when behaviour is learned in a controlled lab environment, with known and controlled environmental variables, and then transferred to an environment in the real world. The environmental variables outside of laboratories cannot be controlled and it is therefore often unknown which variables could contribute to success or failure of motor skills and have to be taken into account by the behavioural policy.

desert of Arizona, it became apparent that the humidity of sand is a major environmental variable to consider when learning locomotion strategies.

Considered together, these problems provide critical motivation for algorithms with the ability to adapt quickly to previously unencountered real-world scenarios. This need becomes even more apparent when considering the task of optimizing locomotion strategies: Assume a legged robot is tasked with the autonomous exploration of an unknown environment. More likely than not, this robot will be powered by solar energy if its mission is long-term. Thus, this mobile robot has a specific contingent of (electric) energy per day. In order to be able to cover as much terrain as possible, the robot needs to use locomotion strategies which are energy-efficient, in order to fulfill its task. Operating in unknown environments brings the problem of not knowing the composition of the ground. From experiments on granular media we know that the optimal locomotion strategies and their efficiency vary greatly given

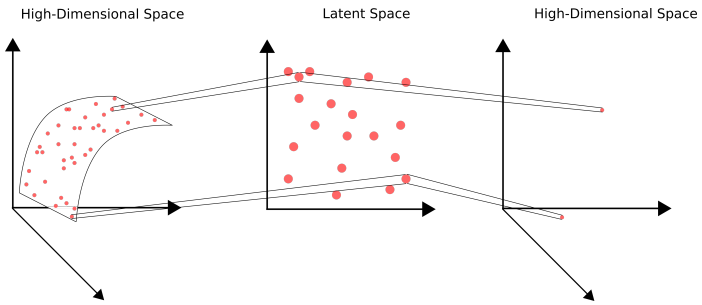


Figure 1.3: Dimensionality reduction methods are able to map high-dimensional state or action spaces to a low-dimensional manifold.

the ground composition. Therefore, this mobile robot would ideally be be able to adapt its locomotion strategy quickly when encountering new terrain, which requires data-efficiency.

1.2 Contributions

This dissertation will investigate the following hypothesis:

Exploration along low-dimensional manifolds allows for sample-efficient reinforcement learning of motor skills in the real world.

Consequently, this dissertation will introduce two main frameworks for sample-efficient reinforcement learning. The first two chapters will introduce sample-efficient reinforcement learning algorithms with an improved exploration strategy using a low-dimensional representation of the action space. The main idea is to integrate a dimensionality reduction method, here Group Factor Analysis, directly into a reinforcement learning framework based on variational inference. Following up on this idea of using low-dimensional spaces in reinforcement learning, a sample-efficient deep reinforcement learning algorithm using trajectory optimization for exploration is introduced in Chapter 4. Finally, in Chapter 5, it is demonstrated how sample-efficient

reinforcement learning algorithms can solve challenging problems such as the joint optimization of morphology and behaviour which required previously large amount of data. I will now summarize the methodical contributions.

1.2.1 Methodical Contributions

- A model-free and sample-efficient reinforcement learning algorithm using variational inference to find a latent action space for sampling.
- An extension of the aforementioned algorithm using previously learned synergies for a faster adaptation to changed tasks.
- A model-based algorithmic framework for deep reinforcement learning in a latent image embedding using trajectory optimization algorithms for fast exploration.
- Extension of the standard reinforcement learning framework and development of a new objective function for the joint adaptation of morphology and design of a robot using sample-efficient reinforcement learning algorithms.

All algorithms and methods introduced throughout the dissertation are evaluated on tasks in simulation and the real world. Manipulation and locomotion tasks were developed and solved, which are challenging and hard to solve for reinforcement learning algorithms as these would most benefit from such sample-efficient strategies. For example, the sample-efficient reinforcement learning algorithm presented in Chapter 4 requires 100 episodes less than the baseline using random noise on an insertion task in the real world (Fig. 1.4). Additionally, the thesis ends with a contribution towards the open problem of the data-efficient co-adaptation of morphology and design to highlight the impact of sample-efficient reinforcement learning algorithms. These contributions towards applications of sample-efficient reinforcement learning can be summarized as follows:

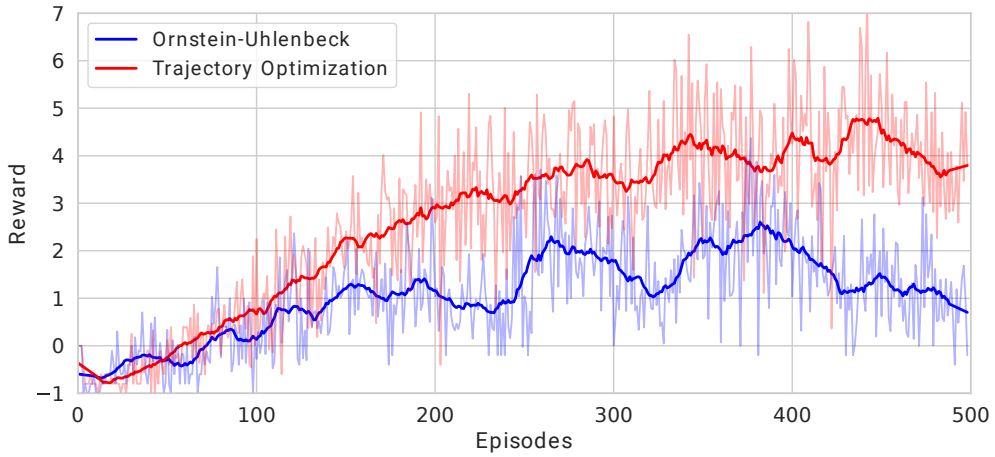


Figure 1.4: The figure shows that the sample-efficient method (red) proposed in chapter 4 achieves an average reward of two already after 150 episodes/sampled trajectories while the reinforcement learning algorithm using random (Ornstein-Uhlenbeck) noise for exploration requires 250 episodes. The plot shows the average reward over five experiments on an insertion task in the real world with sparse rewards.

1.2.2 Contributions towards Real-World Applications

- A framework to quickly learn manipulation synergies in the real world in less than one hour on dense reward tasks.

- A framework to solve a sparse reward task in which a robot arm has to insert a cylinder into a tube using only images. No additional state information, such as joint states, is used. Rewards are only provided when the cylinder is successfully inserted into the tube, otherwise rewards are zero. This task is entirely learned and executed in the real world without using simulations.

- A framework to evolve the design and behaviour of a robot in a data-efficient way and to reduce the amount of physical prototypes required.

1.2.3 Algorithmic Properties

I show for the algorithms introduced in chapter 2 and 3 that due to the use of mean-field Variational Bayes [36, 37], an approximate inference approach similar to Expectation Maximization (EM), the formulated lower bound of the expected reward is maximized under the assumption of a factorized distribution. Assuming a fixed dataset, the approximate inference approach is guaranteed to converge due to the convexity of the lower bound with respect to each of the factors [2, 38]. However, as for all approximate inference approaches, no global convergence guarantees for the original distribution are possible, with the exception of the approximate distribution being the same as the original. This holds especially true for the presented tasks in the real world for which the actual distributions of states and dynamics are generally unknown. The same problem exists for approaches introduced in chapter 4 and 5 which are using deep neural networks, for which no convergence guarantees can be given. However, all aproaches presented in this thesis are empirically evaluated in simulation and the real world on complex tasks in manipulation, locomotion and balancing.

1.3 Thesis Outline

This thesis presents sample-efficient reinforcement learning algorithms suitable for deployment in the real world in absence of any prior data, including the dynamics of agents or transition probabilities of the environment. Each chapter of the thesis document (Fig. 1.5) is either a published manuscript, currently in press and to be published, or under review. Therefore, each chapter will include its own introduction, discussion of related work, methods, results and conclusion. **Chapter 2** introduces the mathematical foundation of Sparse Latent Space Policy Search, a method which extracts a latent action space for improved sample-efficiency. **Chapter 3** follows up with a detailed investigation of the discovered latent action spaces on a bi-manual lifting task with a robot in the real world. Building up on the idea to use latent spaces and developing dedicated sampling methods for actions, **Chapter 4** introduces an improved version of Deep Deterministic Policy Search which uses trajectory optimization in an image embedding to find successful actions. This is evaluated in a sparse reward task in the real world, in which a robot arm has to insert a cylinder into a tube relying only on camera images. Finally, **Chapter 5** presents a method which enables the adaptation of both robot behaviour and morphology in the real world through the use of deep reinforcement learning algorithms. Together, these works present a coherent line of research starting with the development of novel and sample-efficient reinforcement learning algorithms, their evaluation in real-world scenarios and, finally, the application of data-efficient reinforcement learning to the joint optimization of behaviour and morphology, highlighting their potential for future research.

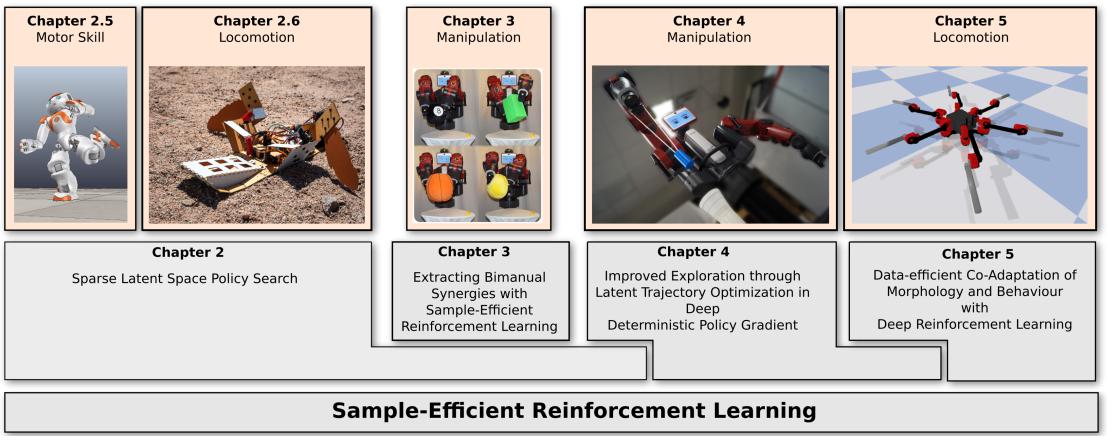


Figure 1.5: An overview of the different chapters and the presented tasks on which the algorithms were evaluated on. The chapters start to the left side with probabilistic, linear methods and end, to the right side, with deep reinforcement learning algorithms.

"Halt! - Noch eine Frage," - rief ich -"bevor wir weitergehen: Tun ihre Menschen denken?" - "Nein!" - rief er sofort mit dem Ton absolutester Sicherheit und nicht ohne den Ausdruck freudiger Erregung, als habe er die Frage erwartet oder sei froh, sie verneinen zu können. - "Nein!" - rief er - "das haben wir glücklick abgeschafft!"

Oskar Panizza, 1890

Die Menschenfabrik (engl.: The Factory of Men), Dämmerungsstücke

CHAPTER 2

Sparse Latent Space Policy Search

(Previously published as Luck, Pajarinen, Berger, Kyrki and Ben Amor 2016)

https://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/viewPaper/12275

ABSTRACT

Computational agents often need to learn policies that involve many control variables, e.g., a robot needs to control several joints simultaneously. Learning a policy with a high number of parameters, however, usually requires a large number of training samples. We introduce a reinforcement learning method for sample-efficient policy search that exploits correlations between control variables. Such correlations are particularly frequent in motor skill learning tasks. The introduced method uses Variational Inference to estimate policy parameters, while at the same time uncovering a low-dimensional latent space of controls. Prior knowledge about the task and the structure of the learning agent can be provided by specifying groups of potentially correlated parameters. This information is then used to impose sparsity constraints on the mapping between the high-dimensional space of controls and a lower-dimensional latent space. In experiments with a simulated bi-manual manipulator, the new approach effectively identifies synergies between joints, performs efficient low-dimensional policy search, and outperforms state-of-the-art policy search methods.

2.1 Introduction

Reinforcement learning (RL) is a promising approach to automated motor skill acquisition [39]. Instead of a human hand-coding specific controllers, an agent autonomously explores the task at hand through trial-and-error and learns necessary movements. Yet, reinforcement learning of motor skills is also considered to be a challenging problem, since it requires sample-efficient learning in high-dimensional state and action spaces. A possible strategy to address this challenge can be found in the human motor control literature [40]. Research on human motor control provides evidence for *motor synergies*; joint co-activations of a set of muscles from a smaller number of neural commands. The reduction in involved parameters results in a lower-dimensional latent space for control which, in turn, reduces cognitive effort and training time during skill acquisition. The existence of synergies has been reported in a variety of human motor tasks, e.g., grasping [30], walking [41], or balancing [42].

Recently, various synergy-inspired strategies have been put forward to improve the efficiency of RL for motor skill acquisition [31, 43]. Typically, these approaches use dimensionality reduction as a pre-processing step in order to extract a lower-dimensional latent space of control variables. However, extracting the latent space using standard dimensionality reduction techniques requires a significantly large training set of (approximate) solutions, prior simulations, or human demonstrations. Even if such data exists, it may drastically bias the search by limiting it to the subspace of initially provided solutions. In our previous work, we introduced an alternative approach called *latent space policy search* that tightly integrates RL and dimensionality reduction [44]. Using an expectation-maximization (EM) framework [45] we presented a latent space policy search algorithm that iteratively refines both

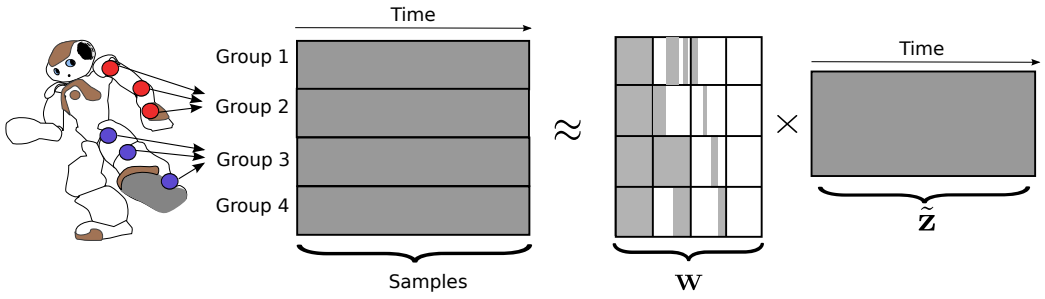


Figure 2.1: The main idea of Group Factor Policy Search: A number of variables, for example the joints of an arm or leg of a NAO robot, form one group. Given several of such groups for the action vector (left matrix) the transformation matrix $\mathbf{W}$ can be divided in several submatrices corresponding to those groups. Subsequently each factor, given by a column in $\mathbf{W}$, encodes information for all groups, e.g. four in the example given above. Factors may be non-zero for all groups, for a subset of groups, for exactly one group or zero for all groups. In the figure, grey areas correspond to non-zero values and white areas to zero values in the sparse transformation matrix. The transformation matrix is multiplied by the latent variables given by $\tilde{\mathbf{Z}} = (\tilde{\mathbf{z}}_1, \cdots, \tilde{\mathbf{z}}_t, \cdots, \tilde{\mathbf{z}}_T)$ distributed by $\tilde{\mathbf{z}}_t \sim \mathcal{N}(\mathbf{0}, \mathrm{trace}(\boldsymbol{\phi}(\mathbf{s}_t, t)\boldsymbol{\phi}(\mathbf{s}_t, t)^{\mathrm{T}})\mathbf{I})$.

the estimates of the low-dimensional latent space, as well as the policy parameters. Only samples produced during the search process were used.

In this chapter, we propose a different kind of latent space policy search approach, which similarly to our previous work combines RL and dimensionality reduction, but which also allows for prior structural knowledge to be included. Our method is based on the Variational Bayes (VB) [46, 47] framework. Variational Bayes is a Bayesian generalization of the expectation-maximization algorithm, which returns a distribution over optimal parameters instead of a single point estimate. It is a powerful framework for approximating integrals that would otherwise be intractable. Our RL algorithm exploits these properties in order to (1) perform efficient policy search, (2) infer the low-dimensional latent space of the task, and (3) incorporate prior structural information. Prior knowledge about locality of synergies can be included by specifying distinct groups of correlated sub-components. Often such prior knowledge about *groups* of variables, e.g. co-activated joints and limbs, is readily available from

the mechanical structure of a system. Structural prior knowledge is also common in other application domains. For example, in a wireless network the network topology defines receiver groups [48].

Our approach draws inspiration and incorporates ideas from Factor Analysis, in particular Group Factor Analysis [23], as can be seen in Fig. 2.1. Groups of variables, e.g., robot joints grouped into arms and legs, are provided as prior structural knowledge by a user. A factorized control policy is then learned through RL, which includes a transformation matrix $\mathbf{W}$. The transformation matrix holds factors that describe dependencies between either all of the groups or a subset of them. The individual factors can be regarded as synergies among the joints of the robot.

We will show that the resulting algorithm effectively ties together prior structural knowledge, latent space identification, and policy search in a coherent way.

2.2 Problem Statement

Policy search methods try to find an optimal policy for an agent which acts in an uncertain world with an unknown world model. At each time step t the agent executes an action $\mathbf{a}_t$ in state $\mathbf{s}_t$ and moves to the next state $\mathbf{s}_{t+1}$ with probability $p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$. After executing a certain number of actions, the agent receives a reward feedback given by an unknown reward function based on the performed execution trace (or trajectory/history) $\boldsymbol{\tau} = (\mathbf{s}_1, \mathbf{a}_1, \ldots, \mathbf{s}_T, \mathbf{a}_T, \mathbf{s}_{T+1})$. The overall objective in policy search is to maximize the *expected reward* over trajectories and policy parameters $\boldsymbol{\theta}$. For bounded rewards, maximizing expected reward is equivalent to maximizing the probability of a binary reward r [49]:

$$\mathbb{E}_{\boldsymbol{\tau}}\left[r = 1\right] = \iint p(\boldsymbol{\tau}, \boldsymbol{\theta}) p(r = 1|\boldsymbol{\tau}) \mathrm{d}\boldsymbol{\theta} \mathrm{d}\boldsymbol{\tau}, \tag{2.1}$$

where the probability of the trajectory $p(\boldsymbol{\tau}, \boldsymbol{\theta})$ contains the (stochastic) policy, r is a binary variable indicating maximum reward, and $p(r = 1|\boldsymbol{\tau}) \propto \exp\{-c(\boldsymbol{\tau})\}$ [50] is the conditional probability of receiving maximum expected reward given a cost function.

Assuming the Markov property and the independence of actions, the probability of a trajectory can be written as

$$p(\boldsymbol{\tau}, \boldsymbol{\theta}) = p(\boldsymbol{\theta})p(\mathbf{s}_1)\prod_{t=1}^{T} p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)\pi(\mathbf{a}_t|\mathbf{s}_t, \boldsymbol{\theta}). \tag{2.2}$$

The stochastic policy $\pi(\mathbf{a}_t|\mathbf{s}_t, \boldsymbol{\theta})$ depends on the parameters $\boldsymbol{\theta}$ for which we additionally introduce prior distributions $p(\boldsymbol{\theta})$. This formulation will subsequently be used for structuring the policy model. The prior distributions may also depend on hyperparameters – for reasons of clarity, however, we will omit any such parameters below. Furthermore, we assume that the initial state distribution $p(\mathbf{s}_1)$ and transition dynamics $p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$ are unknown but fixed. Thus, they will cancel out as constant values.

2.3 Group Factor Policy Search

We will now introduce a new policy search method, called Group Factor Policy Search (GrouPS), that uncovers the latent space on-the-fly based on prior structural information. In this section, we discuss how to incrementally improve the policy and the actual form of the new policy model. We parameterize the policy using Group Factor Analysis [23] in order to utilize prior information about the parameters and their correlations. Since our policy is a linear stochastic model with prior distributions, we first present a novel general Variational Inference framework for policy search that takes priors into account. Subsequently, we discuss how the policy is parameterized, and finally show the policy model update equations for Group Factor Policy Search which we derive using the introduced Variational Inference method.

2.3.1 *Variational Inference for Policy Search*

In each iteration our new policy search method samples a distribution over trajectories $p_{\text{old}}(\boldsymbol{\tau})$ using the current policy, and based on $p_{\text{old}}(\boldsymbol{\tau})$ finds a new policy which maximizes a lower bound on the expected reward. This is repeated until convergence.

In order to find a new policy based on samples from the old one, we introduce the sampling distribution $p_{\text{old}}(\boldsymbol{\tau})$ and the approximated parameter distribution $q(\boldsymbol{\theta})$ (defined later) into Equation 2.1. By applying the log-function and using Jensen's inequality [3, 51, Eq. (1.115)] we derive the lower bound

$$\begin{aligned} &\log \iint p_{\text{old}}(\boldsymbol{\tau}) q(\boldsymbol{\theta}) \frac{p(\boldsymbol{\tau}, \boldsymbol{\theta})}{p_{\text{old}}(\boldsymbol{\tau}) q(\boldsymbol{\theta})} p(r=1|\boldsymbol{\tau}) \mathrm{d}\boldsymbol{\theta} \mathrm{d}\boldsymbol{\tau} \\ &\geq \iint p_{\text{old}}(\boldsymbol{\tau}) q(\boldsymbol{\theta}) \log \left(\frac{p(\boldsymbol{\tau}, \boldsymbol{\theta})}{p_{\text{old}}(\boldsymbol{\tau}) q(\boldsymbol{\theta})} \right) p(r=1|\boldsymbol{\tau}) \mathrm{d}\boldsymbol{\theta} \mathrm{d}\boldsymbol{\tau}. \end{aligned} \tag{2.3}$$

Since $p_{\text{old}}(\boldsymbol{\tau})$ was generated using the old policy it does not depend on $\boldsymbol{\theta}$ and, assuming the Markov property holds, we can simplify the lower bound to

$$\begin{aligned} &\iint p_{\text{old}}(\boldsymbol{\tau}) q(\boldsymbol{\theta}) \log \left(\frac{p(\boldsymbol{\tau}, \boldsymbol{\theta})}{p_{\text{old}}(\boldsymbol{\tau}) q(\boldsymbol{\theta})} \right) p(r=1|\boldsymbol{\tau}) \mathrm{d}\boldsymbol{\theta} \mathrm{d}\boldsymbol{\tau} \\ &= \iint p_{\text{old}}(\boldsymbol{\tau}) q(\boldsymbol{\theta}) \log \left(\frac{p(\boldsymbol{\theta}) \prod_{t=1}^{T} \pi(\mathbf{a}_t|\boldsymbol{\theta}, \mathbf{s}_t)}{q(\boldsymbol{\theta})} \right) p(r=1|\boldsymbol{\tau}) \mathrm{d}\boldsymbol{\theta} \mathrm{d}\boldsymbol{\tau} \\ &+ \iint p_{\text{old}}(\boldsymbol{\tau}) q(\boldsymbol{\theta}) \log \left(\frac{p(\mathbf{s}_1) \prod_{t=1}^{T} p(\mathbf{s}_{t+1}|\mathbf{a}_t, \mathbf{s}_i)}{p(\mathbf{s}_1) \prod_{t=1}^{T} p(\mathbf{s}_{t+1}|\mathbf{a}_t, \mathbf{s}_t) \pi'(\mathbf{a}_t|\mathbf{s}_t)} \right) p(r=1|\boldsymbol{\tau}) \mathrm{d}\boldsymbol{\theta} \mathrm{d}\boldsymbol{\tau}, \end{aligned} \tag{2.4}$$

where $\pi'(\mathbf{a}_t|\mathbf{s}_t)$ is the old policy. Simplifications and the fixed nature of $\pi'(\mathbf{a}_t|\mathbf{s}_t)$ leads then directly to the proposed lower bound with

$$\begin{aligned}
&= \iint p_{\text{old}}(\boldsymbol{\tau}) q(\boldsymbol{\theta}) \log \left(\frac{p(\boldsymbol{\theta}) \prod_{t=1}^{T} \pi(\mathbf{a}_t|\boldsymbol{\theta}, \mathbf{s}_t)}{q(\boldsymbol{\theta})} \right) p(r=1|\boldsymbol{\tau}) \mathrm{d}\boldsymbol{\theta} \mathrm{d}\boldsymbol{\tau} \\
&\quad - \iint p_{\text{old}}(\boldsymbol{\tau}) q(\boldsymbol{\theta}) \log \left(\prod_{t=1}^{T} \pi'(\mathbf{a}_t|\mathbf{s}_t) \right) p(r=1|\boldsymbol{\tau}) \mathrm{d}\boldsymbol{\theta} \mathrm{d}\boldsymbol{\tau} \\
&= \iint p_{\text{old}}(\boldsymbol{\tau}) q(\boldsymbol{\theta}) \log \left(\frac{p(\boldsymbol{\theta}) \prod_{t=1}^{T} \pi(\mathbf{a}_t|\boldsymbol{\theta}, \mathbf{s}_t)}{q(\boldsymbol{\theta})} \right) p(r=1|\boldsymbol{\tau}) \mathrm{d}\boldsymbol{\theta} \mathrm{d}\boldsymbol{\tau} \\
&\quad + const.
\end{aligned} \tag{2.5}$$

By assuming the factorization $q(\boldsymbol{\theta}) = \prod q_i(\boldsymbol{\theta}_i)$ for the parameters and applying the Variational Bayes approach, we get the approximated distributions of the parameters:

$$\log q_j(\theta_j) = const + \int_{\boldsymbol{\theta}_{-j}} \prod_{i \neq j} q_i(\theta_i) \int p_{\text{old}}(\boldsymbol{\tau}) \log \prod_{t=1}^{T} \pi(\mathbf{a}_t, \boldsymbol{\theta}|\mathbf{s}_t) \frac{p(r=1|\boldsymbol{\tau})}{\widehat{R}} \mathrm{d}\boldsymbol{\tau} \mathrm{d}\theta_{-j}, \tag{2.6}$$

where the parameter vector $\boldsymbol{\theta}_{-j}$ contains all parameters except $\boldsymbol{\theta}_j$. The normalization constant $\widehat{R}$ is given by the integral

$$\widehat{R} = \left(\int p_{\text{old}}(\boldsymbol{\tau}) p(r=1|\boldsymbol{\tau}) \mathrm{d}\boldsymbol{\tau} \right)^{-1}. \tag{2.7}$$

This result is derived from the lower bound with

$$
\begin{aligned}
&\int\int p_{\text{old}}(\boldsymbol{\tau})q(\boldsymbol{\theta})\log\left(\frac{p(\boldsymbol{\theta})\prod_{t=1}^{T}\pi(\mathbf{a}_t|\boldsymbol{\theta},\mathbf{s}_t)}{q(\boldsymbol{\theta})}\right)p(r=1|\boldsymbol{\tau})\mathrm{d}\boldsymbol{\theta}\mathrm{d}\boldsymbol{\tau}\\
&=\int\prod_i q_i(\theta_i)\int p_{\text{old}}(\boldsymbol{\tau})\left(\log\left(p(\boldsymbol{\theta})\prod_{t=1}^{T}\pi(\mathbf{a}_t|\boldsymbol{\theta},\mathbf{s}_t)\right)-\sum_k\log q_k(\theta_k)\right)p(r=1|\boldsymbol{\tau})\mathrm{d}\boldsymbol{\tau}\mathrm{d}\boldsymbol{\theta}\\
&=\int_{\theta_j}\int_{\boldsymbol{\theta}_{-j}} q_j(\theta_j)\prod_{i\neq j}q_i(\theta_i)\int p_{\text{old}}(\boldsymbol{\tau})\left(\log\left(p(\boldsymbol{\theta})\prod_{t=1}^{T}\pi(\mathbf{a}_t|\boldsymbol{\theta},\mathbf{s}_t)\right)-\sum_k\log q_k(\theta_k)\right)p(r=1|\boldsymbol{\tau})\mathrm{d}\boldsymbol{\tau}\mathrm{d}\boldsymbol{\theta}_{-j}\mathrm{d}\boldsymbol{\theta}_j\\
&=\int_{\theta_j}\int_{\boldsymbol{\theta}_{-j}} q_j(\theta_j)\prod_{i\neq j}q_i(\theta_i)\int p_{\text{old}}(\boldsymbol{\tau})\log\left(p(\boldsymbol{\theta})\prod_{t=1}^{T}\pi(\mathbf{a}_t|\boldsymbol{\theta},\mathbf{s}_t)\right)p(r=1|\boldsymbol{\tau})\mathrm{d}\boldsymbol{\tau}\mathrm{d}\boldsymbol{\theta}_{-j}\mathrm{d}\boldsymbol{\theta}_j\\
&\quad-\int_{\theta_j}\int_{\boldsymbol{\theta}_{-j}} q_j(\theta_j)\prod_{i\neq j}q_i(\theta_i)\int p_{\text{old}}(\boldsymbol{\tau})\left(\sum_{k\neq j}\log q_k(\theta_k)+\log q_j(\theta_j)\right)p(r=1|\boldsymbol{\tau})\mathrm{d}\boldsymbol{\tau}\mathrm{d}\boldsymbol{\theta}_{-j}\mathrm{d}\boldsymbol{\theta}_j\\
&=\int_{\theta_j} q_j(\theta_j)\int_{\boldsymbol{\theta}_{-j}}\prod_{i\neq j}q_i(\theta_i)\int p_{\text{old}}(\boldsymbol{\tau})\log\left(p(\boldsymbol{\theta})\prod_{t=1}^{T}\pi(\mathbf{a}_t|\boldsymbol{\theta},\mathbf{s}_t)\right)p(r=1|\boldsymbol{\tau})\mathrm{d}\boldsymbol{\tau}\mathrm{d}\boldsymbol{\theta}_{-j}\mathrm{d}\boldsymbol{\theta}_j\\
&\quad-\int_{\theta_j} q_j(\theta_j)\int p_{\text{old}}(\boldsymbol{\tau})\log q_j(\theta_j)p(r=1|\boldsymbol{\tau})\mathrm{d}\boldsymbol{\tau}\mathrm{d}\boldsymbol{\theta}_j+const.
\end{aligned}
\tag{2.8}
$$

Omitting the constant term we find that the maximization of this formula is given when the inner parts of the two separated integrals over θ_j are equal. Thus, we can find that the maximization is given by

$$
\begin{aligned}
&\int p_{\text{old}}(\boldsymbol{\tau})\log q_j(\theta_j)p(r=1|\boldsymbol{\tau})\mathrm{d}\boldsymbol{\tau}=\int_{\boldsymbol{\theta}_{-j}}\prod_{i\neq j}q_i(\theta_i)\int p_{\text{old}}(\boldsymbol{\tau})\log\prod_{t=1}^{T}\pi(\mathbf{a}_t,\boldsymbol{\theta}|\mathbf{s}_t)p(r=1|\boldsymbol{\tau})\mathrm{d}\boldsymbol{\tau}\mathrm{d}\boldsymbol{\theta}_{-j}\\
&\Leftrightarrow\log q_j(\theta_j)=\frac{\int_{\boldsymbol{\theta}_{-j}}\prod_{i\neq j}q_i(\theta_i)\int p_{\text{old}}(\boldsymbol{\tau})\log\prod_{t=1}^{T}\pi(\mathbf{a}_t,\boldsymbol{\theta}|\mathbf{s}_t)p(r=1|\boldsymbol{\tau})\mathrm{d}\boldsymbol{\tau}\mathrm{d}\boldsymbol{\theta}_{-j}}{\left(\int p_{\text{old}}(\boldsymbol{\tau})p(r=1|\boldsymbol{\tau})\mathrm{d}\boldsymbol{\tau}\right)},
\end{aligned}
\tag{2.9}
$$

where the last line is the final approximation.

2.3.2 Formulation of Group Factor Policy Search

In order to identify sets of correlated variables during policy search, we use a linear stochastic policy of a form similar to the model used in Group Factor Analysis

Input: Reward function $R(\cdot)$ and initializations of parameters. Choose number of latent dimension n. Set fixed hyper-parameters $a^{\tilde{\tau}}, b^{\tilde{\tau}}, a^{\alpha}, b^{\alpha}$, σ^2 and define groupings.

while *reward not converged* **do**

for *h=1:H* **do** # Sample H rollouts

for *t=1:T* **do**

$\mathbf{a}_t = \mathbf{W}_i \mathbf{Z} \boldsymbol{\phi} + \mathbf{M} \boldsymbol{\phi} + \mathbf{E} \boldsymbol{\phi}$

with $\mathbf{Z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and $\mathbf{E} \sim \mathcal{N}(\mathbf{0}, \tilde{\boldsymbol{\tau}})$, where $\tilde{\boldsymbol{\tau}}^{(m)} = \tilde{\tau}_m \mathbf{I}$

Execute action $\mathbf{a}_t$

Observe and store reward $R(\boldsymbol{\tau})$

Initialization of q-distribution

while *not converged* **do**

Update $q(\mathbf{M})$ with Eq. (2.19)

Update $q(\mathbf{W})$ with Eq. (2.22)

Update $q\left(\tilde{\mathbf{Z}}\right)$ with Eq. (2.25)

Update $q(\boldsymbol{\alpha})$ with Eq. (2.15)

Update $q(\tilde{\boldsymbol{\tau}})$ with Eq. (2.28)

$\mathbf{M} = \mathbb{E}_{q(\mathbf{M})}[\mathbf{M}]$

$\mathbf{W} = \mathbb{E}_{q(\mathbf{W})}[\mathbf{W}]$

$\boldsymbol{\alpha} = \mathbb{E}_{q(\boldsymbol{\alpha})}[\boldsymbol{\alpha}]$

$\tilde{\boldsymbol{\tau}} = \mathbb{E}_{q(\tilde{\boldsymbol{\tau}})}[\tilde{\boldsymbol{\tau}}]$

Result: Linear weights $\mathbf{M}$ for the feature vector $\boldsymbol{\phi}$, representing the final policy. The columns of $\mathbf{W}$ represents the factors of the latent space.

Algorithm 1: Outline of the Group Factor Policy Search (GrouPS) algorithm.

(GFA) [23]. The main idea of GFA is to introduce prior distributions for the parameters, in particular a prior for a structured transformation matrix $\mathbf{W}$. The transformation matrix, responsible for mapping between a low-dimensional subspace and the original high-dimensional space, is forced to be sparse and constructed using prior knowledge about grouping of the dimensions, that is, $\mathbf{W}$ is a concatenation of transform matrices $\mathbf{W}^{(m)}$ for each group m. For example, if the dimensions of a vector represent the joints of a legged robot, we can group joints belonging to the same leg into the same group (see e.g. Fig. 2.1).

Applying the idea of Group Factor Analysis for directed sampling leads to a linear model, i.e. a stochastic policy

$$\mathbf{a}_t^{(m)} = \left(\mathbf{W}^{(m)}\mathbf{Z}_t + \mathbf{M}^{(m)} + \mathbf{E}_t^{(m)}\right) \boldsymbol{\phi}\left(\mathbf{s}_t, t\right) \ , \tag{2.10}$$

where, for group m, the action $\mathbf{a}_t^{(m)} \in \mathbb{R}^{D_m \times 1}$ is a linear projection of a feature vector $\boldsymbol{\phi}\left(\mathbf{s}_t, t\right) \in \mathbb{R}^{p \times 1}$. Each dimension of the feature vector is given by a basis function, which may depend on the current state and/or time. In the remainder of the chapter, we will write $\boldsymbol{\phi}$ instead of $\boldsymbol{\phi}\left(\mathbf{s}, t\right)$ for simplicity, even though there is an implicit dependency of $\boldsymbol{\phi}$ on the current state of a trajectory. $\mathbf{W}^{(m)} \in \mathbb{R}^{D_m \times l}$ is a transformation matrix mapping from the l-dimensional subspace to the original space. Each entry of the latent matrix $\mathbf{Z}_t \in \mathbb{R}^{l \times p}$ is distributed according to a standard normal distribution where $\mathcal{N}\left(0, 1\right)$, $\mathbf{M}^{(m)} \in \mathbb{R}^{D_m \times p}$ is the mean matrix, and the entries of the noise matrix $\mathbf{E}_t^{(m)} \in \mathbb{R}^{D_m \times p}$ are distributed by $\mathcal{N}(0, \tilde{\tau}_m^{-1})$.

We can derive a stochastic policy from the model defined in Equation 2.10. Since

$$\mathbf{Z}\boldsymbol{\phi} \sim \mathcal{N}(\mathbf{0}, \mathrm{trace}(\boldsymbol{\phi}\boldsymbol{\phi}^{\mathrm{T}})\mathbf{I}) \tag{2.11}$$

holds (see e.g. [44]), we can substitute $\mathbf{Z}\boldsymbol{\phi}$ by the random variable $\tilde{\mathbf{z}}_t \in \mathbb{R}^{l\times 1}$ resulting in the policy

$$\pi(\mathbf{a}_t|\boldsymbol{\theta}, \mathbf{s}_t) = \prod_{m=1}^{M} \mathcal{N}\left(\mathbf{a}_t^{(m)} \middle| \mathbf{W}^{(m)}\tilde{\mathbf{z}}_t + \mathbf{M}^{(m)}\boldsymbol{\phi}, \frac{\mathrm{Tr}\left(\boldsymbol{\phi}\boldsymbol{\phi}^{\mathrm{T}}\right)}{\tilde{\tau}_m}\mathbf{I}\right). \tag{2.12}$$

If we take a closer look at the latent space given by $\mathbf{W}\tilde{\mathbf{z}}$ we first find that the length of each factor is determined by $\|\boldsymbol{\phi}(\mathbf{s}_t, t)\|_2^2$. Secondly, a factor may be non-zero only for one or a subset of groups as can be seen in Fig. 2.1. This leads to a sparse transformation matrix and specialized factors.

As mentioned before, the form of our linear model in Equation 2.10 above is based on Group Factor Analysis. While GFA typically maps a vector from the latent space to the high-dimensional space, we map here a matrix from the latent space to the original space and then use this matrix as a linear policy on the feature vectors. GFA does not apply factor analysis (see e.g. [25]) on each group of variables separately, but instead introduces a sparsity prior on the transformation matrix $\mathbf{W}$ thereby forcing correlations between groups:

$$p\left(\mathbf{W}|\boldsymbol{\alpha}\right) = \prod_{m=1}^{M}\prod_{k=1}^{K}\prod_{d=1}^{D_m}\mathcal{N}\left(\mathbf{w}_{d,k}^{(m)}\middle|\mathbf{0}, \alpha_{m,k}^{-1}\right), \tag{2.13}$$

with M being number of groups, D_m the number of dimensions of the m-th group and K the number of factors, i.e. number of columns of $\mathbf{W}$. The precision $\boldsymbol{\alpha}$ is given by a log-linear model with

$$\log\boldsymbol{\alpha} = \mathbf{U}\mathbf{V}^{\mathrm{T}} + \boldsymbol{\mu}_u\mathbf{1}^{\mathrm{T}} + \mathbf{1}\boldsymbol{\mu}_v^{\mathrm{T}}, \tag{2.14}$$

where $\mathbf{U} \in \mathbb{R}^{M\times\mathcal{R}}$, $\mathbf{V} \in \mathbb{R}^{K\times\mathcal{R}}$ and $\boldsymbol{\mu}_u \in \mathbb{R}^M$ as well as $\boldsymbol{\mu}_v \in \mathbb{R}^K$ model the mean profile. $\mathcal{R}$ defines the rank of the linear model and is chosen $\mathcal{R} \ll \min(M, K)$. However, for the special case of $\mathcal{R} = \min(M, K)$ the precision is given by a simple gamma distribution [23]

$$q\left(\boldsymbol{\alpha}_{m,k}\right) = \mathcal{G}\left(a_m^{\alpha}, b_{m,k}^{\alpha}\right) \tag{2.15}$$

with parameters

$$a_m^\alpha = a^\alpha + \frac{D_m}{2}, \tag{2.16}$$

$$b_{m,k}^\alpha = b^\alpha + \frac{1}{2}\mathbb{E}_{q(\mathbf{W})}\left[\mathbf{w}_k^{(m)\mathrm{T}}\mathbf{w}_k^{(m)}\right]. \tag{2.17}$$

The hyper-parameters a^α and b^α are fixed and set to a small positive value. The prior distributions given above will lead to three kind of factors: (1) factors which are nonzero for only one group, (2) factors which are nonzero for several groups or (3) factors which are zero. In addition to the standard GFA prior distributions above, we introduce further prior distributions for $\mathbf{M}$ and $\tilde{\mathbf{z}}$ such that all prior distributions are given with

$$\mathbf{M} \sim \mathcal{N}\left(\mathbf{M}_{\text{old}}, \sigma^2\mathbf{I}\right), \qquad \tilde{\mathbf{z}} \sim \mathcal{N}\left(\mathbf{0}, \mathrm{Tr}\left(\boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathrm{T}}\right)\mathbf{I}\right),$$
$$\alpha_{m,k} \sim \mathcal{G}\left(a^\alpha, b^\alpha\right), \qquad \tilde{\tau}_m \sim \mathcal{G}\left(a^{\tilde{\tau}}, b^{\tilde{\tau}}\right).$$

Fig. 2.2 shows a graphical model of Group Factor Policy Search, given by the distributions stated above. Instead of $\mathbf{Z}$ the latent variable $\tilde{\mathbf{z}}_t$ is used, which depends on $\boldsymbol{\Phi}(\mathbf{s}_t, t)$ given a state and a point in time.

2.3.3 Derivation of Update Equations

We assume fixed hyper-parameters a^α, b^α, $a^{\tilde{\tau}}$ and $b^{\tilde{\tau}}$ for the distributions which we determine using the Variational Inference method presented earlier, assuming a factorization of the q-distributions

$$q\left(\boldsymbol{\theta}\right) = q(\tilde{\mathbf{Z}})q\left(\mathbf{W}\right)q\left(\tilde{\boldsymbol{\tau}}\right)q\left(\mathbf{M}\right)q\left(\boldsymbol{\alpha}\right) \tag{2.18}$$

and additionally the assumption $q(\tilde{\mathbf{Z}}) = \prod^T q(\tilde{\mathbf{z}}_t)$ with $\tilde{\mathbf{Z}}_{:,t} = \tilde{\mathbf{z}}_t$.

By using the factorization given above and the Variational Inference rule for deriving the parameter distribution in Equation (2.6), we can derive the approximated parameter distributions, which maximize the expected reward.

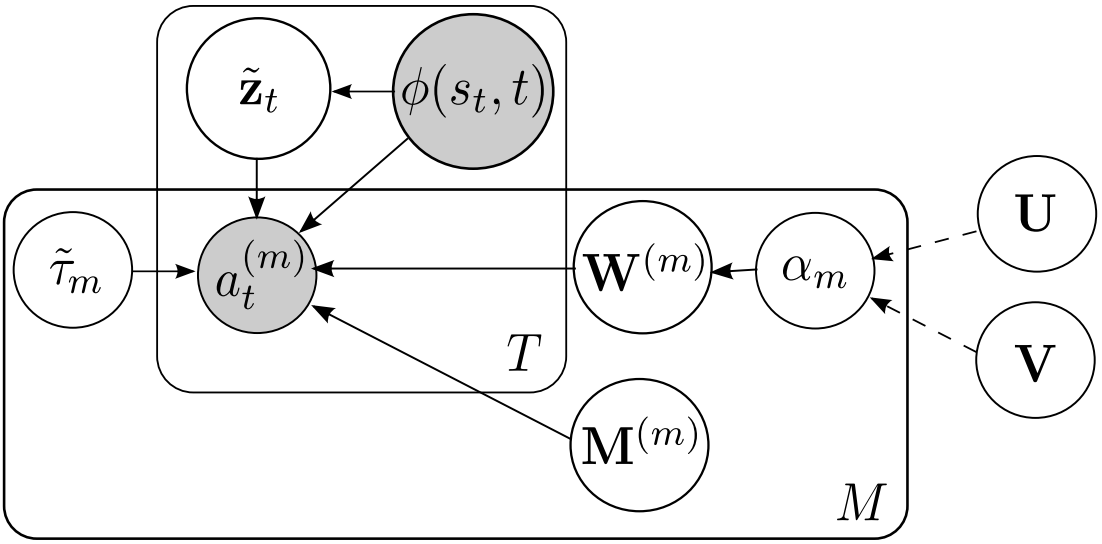


Figure 2.2: Graphical model in Plate notation for Group Factor Policy Search. The basis functions $\boldsymbol{\phi}(\mathbf{s}_t, t)$ as well as the action vector $\mathbf{a}^{(m)}$ are observed. Equation 2.12 shows the dependencies for $\mathbf{a}^{(m)}$. The latent variables $\tilde{\mathbf{z}}$ depend on the feature vector as stated in Equation (2.11) . The parameter α_m might either be given by a Gamma distribution as stated in Equation (2.15) or by a log-linear model with dependencies on parameters $\mathbf{U}$ and $\mathbf{V}$.

The approximated distribution for the mean matrix is given by a multiplicative normal distribution

$$q_{\mathbf{M}}(\mathbf{M}) = \prod_{m=1}^{M} \prod_{j=1}^{D_m} \mathcal{N}\left(\mathbf{m}_{j,:}^{(m)\mathrm{T}} \middle| \boldsymbol{\mu}_{mj}^{\mathbf{M}}, \boldsymbol{\Sigma}_j^{\mathbf{M}}\right) \tag{2.19}$$

where the mean and covariance parameters in dependency of the group and dimension are given by

$$\boldsymbol{\Sigma}_j^{\mathbf{M}} = \left(\sigma^{-2}\mathbf{I} + \mathbb{E}_{p(\boldsymbol{\tau})}\left[\frac{p(r=1|\boldsymbol{\tau})}{\widehat{R}}\left(\sum_{t=1}^{T} \frac{\boldsymbol{\phi}\boldsymbol{\phi}^{\mathrm{T}}}{\mathrm{Tr}\left(\boldsymbol{\phi}\boldsymbol{\phi}^{\mathrm{T}}\right)} \mathbb{E}_{\tilde{\tau}}\left[\tilde{\tau}_m\right]\right)\right]\right)^{-1} \tag{2.20}$$

and

$$\begin{aligned} \boldsymbol{\mu}_{mj}^{\mathbf{M}} &= \boldsymbol{\Sigma}_j^{\mathbf{M}} \cdot \frac{\mathbf{m}_{\mathrm{old}_{j,:}}{}^{\mathrm{T}}}{\sigma^2} + \boldsymbol{\Sigma}_j^{\mathbf{M}} \cdot \\ &\mathbb{E}_{p(\boldsymbol{\tau})}\left[\frac{p(r=1|\boldsymbol{\tau})}{\widehat{R}} \sum_{t=1}^{T} \frac{\boldsymbol{\phi}\left(a_{t,j}^{(m)} - \mathbb{E}_{\mathbf{w}}\left[\mathbf{w}_{j,:}^{(m)}\right] \mathbb{E}_{\tilde{\mathbf{z}}}\left[\tilde{\mathbf{z}}_t\right]\right)}{\mathrm{Tr}\left(\boldsymbol{\phi}\boldsymbol{\phi}^{\mathrm{T}}\right)\mathbb{E}_{\tilde{\tau}}\left[\tilde{\tau}_m\right]^{-1}}\right] \end{aligned} \tag{2.21}$$

The q-distribution for the transformation matrix is similarly given by

$$q_{\mathbf{W}}\left(\mathbf{W}\right)=\prod_{m=1}^{M}\prod_{j=1}^{D_m}\mathcal{N}\left(\mathbf{w}_{j,:}^{(m)^{\mathrm{T}}}|\boldsymbol{\mu}_{mj}^{\mathbf{W}},\boldsymbol{\Sigma}_{m}^{\mathbf{W}}\right) \tag{2.22}$$

with the mean and covariance parameters

$$\boldsymbol{\Sigma}_{m}^{\mathbf{W}}=\left(\mathbb{E}_{p(\boldsymbol{\tau})}\left[\frac{p(r=1|\boldsymbol{\tau})}{\widehat{R}}\left(\sum_{t=1}^{T}\frac{\mathbb{E}_{\tilde{\mathbf{z}}}\left[\tilde{\mathbf{z}}_t\tilde{\mathbf{z}}_t^{\mathrm{T}}\right]}{\mathrm{Tr}\left(\boldsymbol{\phi}\boldsymbol{\phi}^{\mathrm{T}}\right)\mathbb{E}_{\tilde{\tau}}\left[\tilde{\tau}_m\right]^{-1}}+\bar{\bar{\boldsymbol{\alpha}}}_{m,K}\right)\right]\right)^{-1}, \tag{2.23}$$

and

$$\boldsymbol{\mu}_{mj}^{\mathbf{W}}=\boldsymbol{\Sigma}_{m}^{\mathbf{W}}\cdot\mathbb{E}_{p(\boldsymbol{\tau})}\left[\frac{p(r=1|\boldsymbol{\tau})}{\widehat{R}}\sum_{t=1}^{T}\frac{\left(a_{t,j}^{(m)}-\mathbb{E}_{\mathbf{M}}\left[\mathbf{m}_{j,:}^{(m)}\right]\boldsymbol{\phi}\right)\mathbb{E}_{\tilde{\mathbf{z}}}\left[\tilde{\mathbf{z}}_t\right]^{\mathrm{T}}}{\mathrm{Tr}\left(\boldsymbol{\phi}\boldsymbol{\phi}^{\mathrm{T}}\right)\mathbb{E}_{\tilde{\tau}}\left[\tilde{\tau}_m^{-1}\right]}\right]. \tag{2.24}$$

The diagonal matrix $\bar{\bar{\boldsymbol{\alpha}}}_{m,K}$ is given by $\mathrm{diag}\left(\bar{\bar{\boldsymbol{\alpha}}}_{m,K}\right)=\left(\alpha_{m,1},\alpha_{m,2},\cdots,\alpha_{m,K}\right)$. The distribution for the latent variables $\tilde{\mathbf{Z}}$ depends on the trajectory and time. Hence the reward can be seen as a probabilistic weight $\tilde{R}$ of a multiplicative normal distribution. However, since we assume independent latent variables $\tilde{\mathbf{z}}_t^r$ we can ignore the reward and get

$$q_{\tilde{\mathbf{Z}}}\left(\tilde{\mathbf{Z}}\right)=\prod^{H}\tilde{R}\prod_{t=1}^{T}\mathcal{N}\left(\tilde{\mathbf{z}}_t^r|\boldsymbol{\mu}_t^{\tilde{\mathbf{Z}}},\boldsymbol{\Sigma}_t^{\tilde{\mathbf{Z}}}\right), \tag{2.25}$$

with time-dependent parameters

$$\boldsymbol{\Sigma}_{t}^{\tilde{\mathbf{Z}}}=\left(\mathrm{Tr}\left(\boldsymbol{\phi}\boldsymbol{\phi}^{\mathrm{T}}\right)^{-1}\mathbf{I}+\sum_{m=1}^{M}\frac{\mathbb{E}_{\mathbf{W}}\left[\mathbf{W}^{(m)^{\mathrm{T}}}\mathbf{W}^{(m)}\right]}{\mathrm{Tr}\left(\boldsymbol{\phi}\boldsymbol{\phi}^{\mathrm{T}}\right)\mathbb{E}_{\tilde{\tau}_m}\left[\tilde{\tau}_m^{-1}\right]}\right)^{-1}, \tag{2.26}$$

and

$$\boldsymbol{\mu}_{t}^{\tilde{\mathbf{Z}}}=\boldsymbol{\Sigma}_{t}^{\tilde{\mathbf{Z}}}\cdot\left(\sum_{m=1}^{M}\frac{\mathbb{E}_{\mathbf{W}}\left[\mathbf{W}^{(m)}\right]^{\mathrm{T}}\left(\mathbf{a}_t^{(m)}-\mathbf{M}^{(m)}\boldsymbol{\phi}\right)}{\mathrm{Tr}\left(\boldsymbol{\phi}\boldsymbol{\phi}^{\mathrm{T}}\right)\mathbb{E}_{\tilde{\tau}}\left[\tilde{\tau}_m\right]^{-1}}\right). \tag{2.27}$$

Unlike the other distributions, the precision is given by a multiplicative gamma distribution

$$q_{\tilde{\tau}}\left(\tilde{\tau}\right)=\prod_{m=1}^{M}\mathcal{G}\left(\tilde{\tau}_m|a^{\tilde{\tau}}+\frac{1}{2}D_mT,b^{\tilde{\tau}}+\frac{1}{2}b_m^{\tilde{\tau}\,\prime}\right) \tag{2.28}$$

with one fixed parameter and one variable parameter. Estimation of the parameter $b_m^{\tilde{\tau}\,\prime}$ is the most complex and computationally expensive operation given by

$$\begin{aligned} b_m^{\tilde{\tau}\,\prime} = \mathbb{E}_{p(\boldsymbol{\tau})} \Bigg[& \frac{p(r=1|\boldsymbol{\tau})}{\widehat{R}} \sum_{t=1}^{T} \mathrm{Tr}\left(\boldsymbol{\phi}\boldsymbol{\phi}^{\mathrm{T}}\right)^{-1} \Big(\mathbf{a}_t^{(m)\mathrm{T}} \mathbf{a}_t^{(m)} \\ & - 2\mathbf{a}_t^{(m)\mathrm{T}} \mathbb{E}_{\mathbf{M}} \left[\mathbf{M}^{(m)}\right] \boldsymbol{\phi} \\ & + 2\mathbb{E}_{\tilde{\mathbf{z}}} \left[\tilde{\mathbf{z}}_t\right]^{\mathrm{T}} \mathbb{E}_{\mathbf{W}} \left[\mathbf{W}^{(m)}\right]^{\mathrm{T}} \mathbb{E}_{\mathbf{M}} \left[\mathbf{M}^{(m)}\right] \boldsymbol{\phi} \\ & - 2\mathbf{a}_t^{(m)\mathrm{T}} \mathbb{E}_{\mathbf{W}} \left[\mathbf{W}^{(m)}\right] \mathbb{E}_{\tilde{\mathbf{z}}} \left[\tilde{\mathbf{z}}_t\right] \\ & + \boldsymbol{\phi}^{\mathrm{T}} \mathbb{E}_{\mathbf{M}} \left[\mathbf{M}^{(m)\mathrm{T}} \mathbf{M}^{(m)}\right] \boldsymbol{\phi} \\ & + \mathrm{Tr}\left(\mathbb{E}_{\mathbf{W}} \left[\mathbf{W}^{(m)\mathrm{T}} \mathbf{W}^{(m)}\right] \mathrm{Cov}_{\tilde{\mathbf{z}}} \left[\tilde{\mathbf{z}}_t\right] \right) \\ & + \mathbb{E}_{\tilde{\mathbf{z}}} \left[\tilde{\mathbf{z}}_t\right]^{\mathrm{T}} \mathbb{E}_{\mathbf{W}} \left[\mathbf{W}^{(m)\mathrm{T}} \mathbf{W}^{(m)}\right] \mathbb{E}_{\tilde{\mathbf{z}}} \left[\tilde{\mathbf{z}}_t\right] \Big) \Bigg] . \end{aligned} \tag{2.29}$$

2.3.4 Algorithm

Algorithm 1 summarizes all update steps for performing Group Factor Policy Search. The reward function $R(\cdot)$, number n of latent dimensions, and a set of hyperparameters need to be provided by the user.

2.4 Evaluation

For evaluations and experiments the expectation $\mathbb{E}_{p(\boldsymbol{\tau})}[\cdot]$ used above in Eq.(16-20,25) was approximated by a sample mean,

$$\mathbb{E}_{p(\boldsymbol{\tau})}[f(\boldsymbol{\tau})] \approx \frac{1}{H} \sum_{i=1}^{H} f(\boldsymbol{\tau}_i) \tag{2.30}$$

as proposed in [51], where $\boldsymbol{\tau}_i$ is the i-th of the H realized trajectories and $f(\boldsymbol{\tau})$ a function value, vector or matrix for $\boldsymbol{\tau}_i$ and will be replaced by the parameter approximations given above.

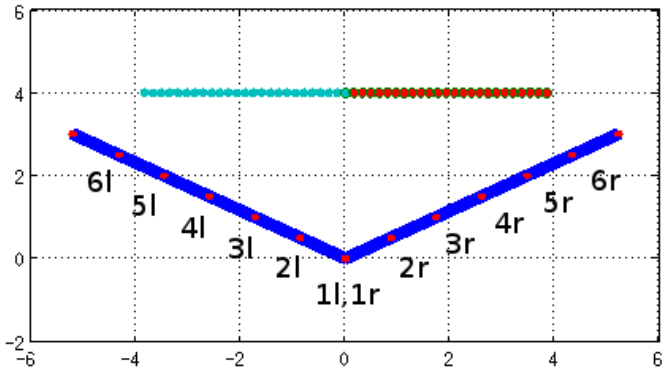


Figure 2.3: Two simulated arms with six degrees-of-freedom and the same base in their initial position. Each end effector has a desired position for each time step, s shown by the green and red dots. The final position at time step 25 is given by the coordinate $(0, 4)$. The numbers represent the joints with l for left and r for right.

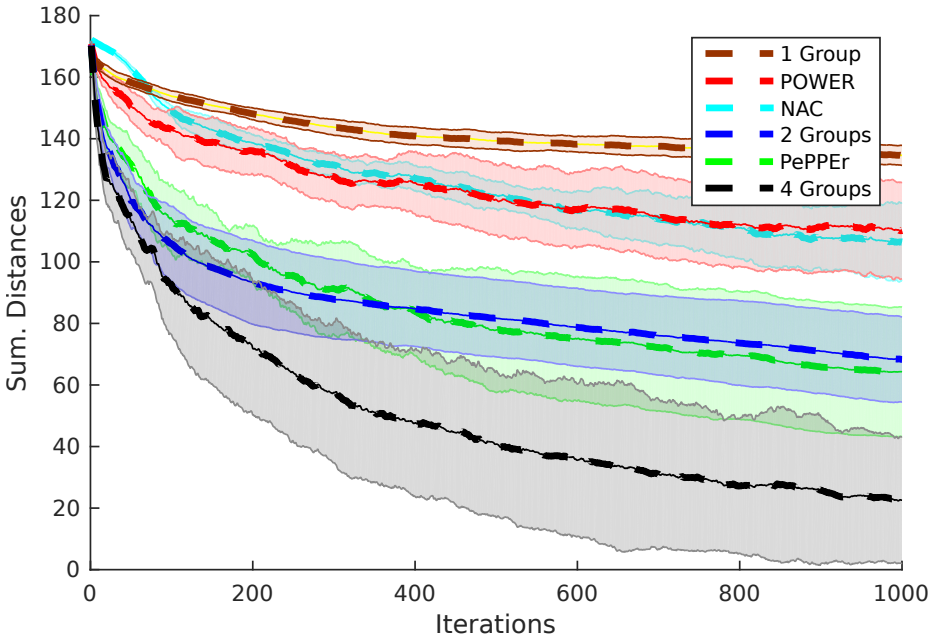


Figure 2.4: Comparison between PePPEr, PoWER, Natural Actor-Critic and three instances of the GrouPS algorithm on the presented simulated task. Values correspond to the summarized distances between each end effector and its desired position given the current policy for the iteration. The mean value as well as the standard deviations are shown.

2.4.1 Setup of the Evaluation

For the comparison between the above presented GrouPS algorithm and previous policy search algorithms, a simulated task of a bi-manual robot operating in a planar task space was used. Each of the two arms (see Fig. 2.3) has six degrees-of-freedom and the same base for the first joint. The initial configuration of the arms is presented in Fig. 2.3 as well as the desired positions for each end effector (tip of an arm). At each of the 25 time steps we give a different goal position for each arm's end effector, starting from the left for the left arm and starting from the right for the right arm, with the same final position at $(0, 4)$ for both arms. In this task, the 12 dimensions of the action vector $\mathbf{a}$ represent the joint angles for each arm. For the basis functions eleven isotropic Gaussian distributions were used with $\boldsymbol{\phi}_i(t) = \mathcal{N}(t|\mu_i^{\boldsymbol{\phi}}, 3)$ for $t \in \{1, 2, \ldots, 24, 25\}$. In total, 132 parameters have to be estimated given $\mathbf{M} \in \mathbb{R}^{12\times 11}$.

As reference algorithms PoWER [51], Natural Actor-Critic (NAC) [52] and PePPEr [44] were chosen: NAC is a policy gradient method while PoWER is an efficient policy search method based on expectation maximization (EM). PoWER has been experimentally validated in both simulated and physical robotic experiments [16]. PePPEr is also based on EM and incorporates policy search and dimensionality reduction, but without priors and thus without a structured transformation matrix. For comparison with PePPEr and PoWER the GrouPS algorithm was evaluated in three different configurations: (1) One group which contains all joints of both arms, (2) two groups, where each group contains the joints of one arm and (3) four groups with two groups per arm and joints 1-4 in one and joints 5-6 in the second group. The hyper-parameters of GrouPS were set to $a^{\tilde{\tau}} = b^{\tilde{\tau}} = 1000$, $a^{\alpha} = b^{\alpha} = 1$ and $\sigma^2 = 100$. No optimizations of the hyper-parameters were performed. Furthermore, to prevent early convergence and collapsing of the distributions due to small sample sizes the

parameter $\mathbf{W}$ and $\tilde{\tau}$ are resized after each iteration by a factor of 1.5. The same is done after each iteration for PePPEr. However, the factor was set to $\sqrt{1.09}$ since higher numbers lead to divergence in the parameters of the algorithm with unstable and divergent results. PePPEr was implemented as presented in [44] and in each iteration 20 inner iterations for the optimizations of the parameters were used. The same setup was used for GrouPS and for both algorithms the number of latent dimensions were set to six. The static variance parameter for PoWER as presented in [51] and the initial variance of the other algorithms were all set to $10^{1.5}$, also for NAC with learning parameter set to 0.5. In each iteration, we sampled 30 trajectories and evaluated the trajectories based on the reward function

$$\begin{aligned} r(\mathbf{a}_t, t) = \sum_{t=1}^{25} & \exp\left(- \parallel \mathrm{eff}_l(\mathbf{a}_t) - \mathrm{pos}_l(t) \parallel_2\right) \\ & + \exp\left(- \parallel \mathrm{eff}_r(\mathbf{a}_t) - \mathrm{pos}_r(t) \parallel_2\right), \end{aligned} \tag{2.31}$$

where the function $\mathrm{eff}_l(\mathbf{a}_t)$ returns the position of the left end effector given the action vector and $\mathrm{pos}_l(t)$ the corresponding desired goal position for time point t. $\mathrm{eff}_r(\mathbf{a}_t)$ and $\mathrm{pos}_r(t)$ return the actual and desired positions, respectively, for the right end effector. Then the 15 best trajectories are chosen for the computation of the parameters for each algorithm as described in [51].

2.4.2 Results

Fig. 2.4 depicts the results of the explained experiment. For each algorithm ten different runs were executed and both mean and standard deviation computed. As can be seen in the figure, PePPEr outperforms both PoWER and NAC, as well as our method in case only one group spanning all variables is used. However, using two groups (one for each arm) already leads to comparable performance. Finally, the

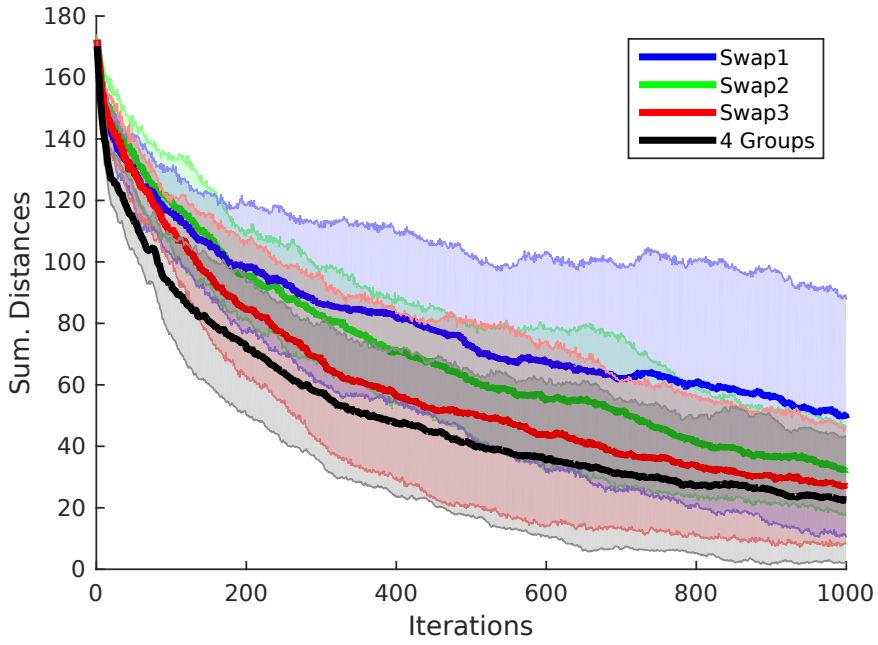


Figure 2.5: Comparison between the original chosen four groups and three permutations of the Groups. Values correspond to the summarized distance between each end effector and its desired position for each time step given the current policy for the iteration.

GrouPS algorithm with 4 different groups significantly outperforms the comparison methods.

2.4.3 Importance of the Choice of Groups

In order to investigate the effect of choosing joint groups we conducted an additional experiment. Our working hypothesis throughout the chapter is that structural information about inherent groups of correlated variables will improve the search. Conversely, if we provide wrong information about groupings the performance of the algorithm should deteriorate. To evaluate this hypothesis, we took the original partitioning of the joints into four groups and swapped two, later three pairs of joints randomly. As described above, the original group partitioning is

$$\{(1l, 2l, 3l, 4l), (5l, 6l), (1r, 2r, 3r, 4r), (5r, 6r)\}. \tag{2.32}$$

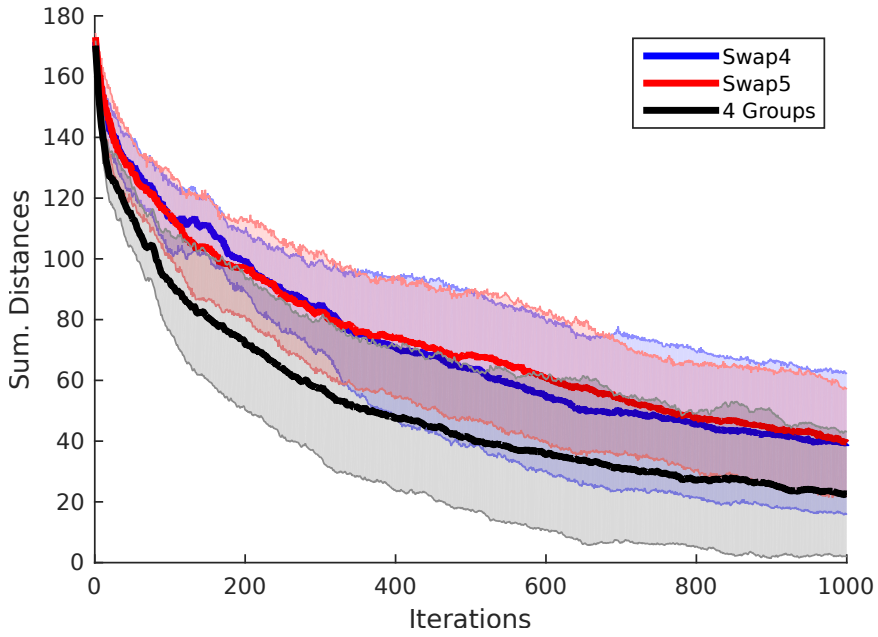


Figure 2.6: Comparison between the original grouping and two other variants with a different splitting point. Again, the values represent the summarized distances and shaded ares corresponds to the standard deviation given ten executions.

Performing two random swaps between the left and right side (Fig. 2.6, Swap4) results in

$$\{(1l, 2l, 2r, 4l), (5l, 5r), (1r, 3l, 3r, 4r), (6l, 6r)\}. \tag{2.33}$$

For three swaps the resulting partition (Fig. 2.6, Swap5) is

$$\{(1l, 6r, 2r, 4l), (3r, 6l), (1r, 3l, 5l, 4r), (5r, 2l)\}. \tag{2.34}$$

Furthermore, three other groupings with different splitting points were evaluated:

$$\{(1l, 2l), (3l, 4l, 5l, 6l), (1r, 2r), (3r, 4r, 5r, 6r)\} \text{ (Fig. 2.5, Swap1)}, \tag{2.35}$$

$$\{(1l, 2l), (3l, 4l), (5l, 6l), (1r, 2r), (3r, 4r), (5r, 6r)\} \text{ (Fig. 2.5, Swap2)}, \tag{2.36}$$

$$\text{and } \{(1l, 2l, 3l), (4l, 5l, 6l), (1r, 2r, 3r), (4r, 5r, 6r)\} \text{ (Fig. 2.5, Swap3)}. \tag{2.37}$$

The result of executing GrouPS with these groupings can be seen in Fig. 2.5 and Fig. 2.6. All new groupings (resulting from above swaps) are clearly outperformed by

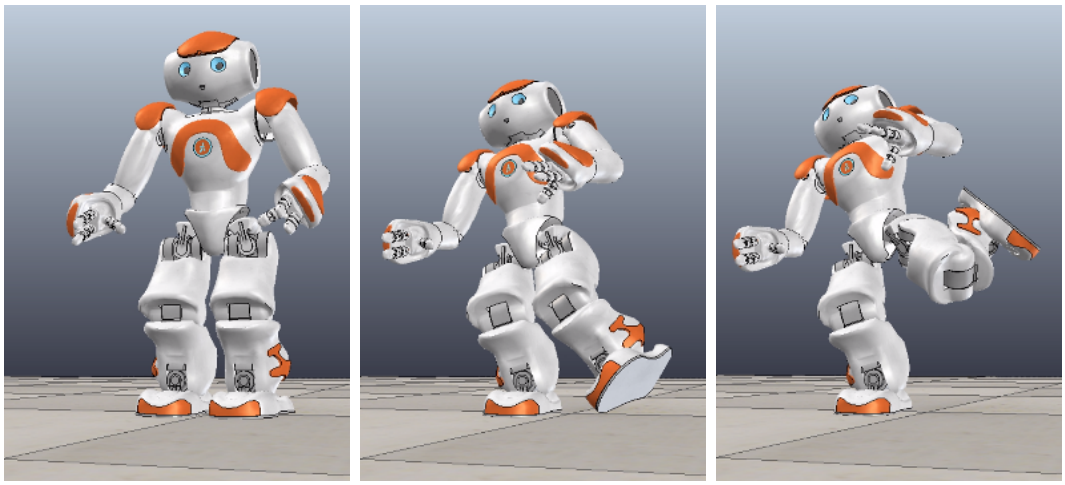


Figure 2.7: Final policy found by the GrouPS algorithm after 100 iterations. A high reward is given if the head as well as the left foot of the robot are high above the ground.

the original partition. This result corroborates our assumption that a proper selection of groups can ameliorate the performance of the policy search algorithm.

2.5 Experiment: Lifting a Leg

To test the GrouPS algorithm in experiments following the real world closely, we reproduced the experiment stated in [44]: We simulate a NAO robot [53] using the V-REP framework [54] in the task of lifting its left leg without falling. The same reward function was used as presented in [44, Eq. (22)] with parameters $\alpha = 5$, $\beta = 10$, $\gamma = 10$ and $\lambda_{max} = 6$. The V-REP framework [54] allows for simulations with high physical accuracy by utilizing the bullet physics library. In this experiment, the actions represent the 26 joint velocities for each of the 15 points in time. Again, for feature functions Gaussian distributions were used and the same parameters for GrouPS were chosen like given in the evaluation above.

We ran GrouPS for 100 iterations. In each iteration, we used a set of 20 samples, of which ten were randomly selected from the set of 20 in the previous iteration and

ten generated by the current policy. We used ten best samples out of this set of 20 for computing the new policy parameters. The groups were created in such a manner that the joints of each arm or leg form a single group as well as the joints of the head. The results are given in Fig. 2.7, where we find that the GrouPS algorithm is able to find a satisfactory solution even with a relatively small number of samples: the head and left leg of the NAO robot are at high positions corresponding to a high reward.

2.6 Experiment: Learning Locomotion on Granular Media

To highlight the sample-efficiency of GrouPS further, an experiment was conducted in which a robot manufactured from paper has to learn an effective locomotion strategy on granular media[55]. Due to its properties, granular media such as sand is difficult to simulate and thus policies trained in simulation are not effective in the real world. Therefore, the training was conducted in the real world in absence of any prior data or simulations. The robot has four degrees-of-freedom in total, two per fin. Three experiments were conducted: In a lab-environment using poppy seeds, the execution of the policy trained on poppy seeds in a desert environment, and the training and evaluation of GrouPS in the desert of Arizona (Fig. 2.8). In all experiments, the reinforcement learning algorithm has to learn policies which are effective movement strategies, covering as much distance from the start position as possible. Ten iterations were executed, with a total of 210 policy executions. The results (Fig. 2.9) show that training policies in the real environment outperform policies trained on a substitute, in this case poppy seeds. This shows the effect of the lab-to-reality gap, here between experiments conducted in the laboratory and in the desert of Arizona. Furthermore, the results show that GrouPS is able to find policies with only a small number of samples in a difficult environment.

Figure 2.8: A policy trained by GrouPS in the desert of Arizona. The white line shows the initial position of the robot. [55]

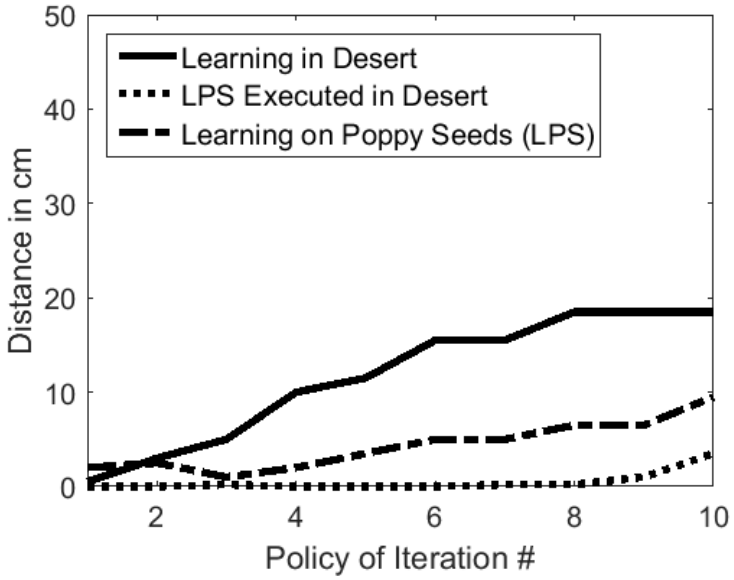


Figure 2.9: The evaluation of GrouPS on a bio-inspired robot manufactured from a paper-laminate. Experiments were conducted in the lab and the desert of Arizona. It can be seend that learning a locomotion policy in the real environment leads to significantly better movements than using a pre-trained strategy, which was trained on poppy seeds in the lab. [55]

2.7 Conclusion

In this chapter, we introduced a novel algorithm for reinforcement learning in low-dimensional latent spaces. To this end, we derived a Variational Inference framework for policy search that takes prior structural information into account. The resulting policy search algorithm can efficiently learn new policy parameters, while also uncovering the underlying latent space of solutions, and incorporating prior knowledge about groups of correlated parameters. In experiments using motor skill learning tasks,

we showed that the introduced GrouPS algorithm efficiently learns new motor skills. It significantly outperformed state-of-the-art policy search methods, whenever prior information about structural groups was provided.

So far, the dimensionality of the latent space needs to be provided as a parameter to the reinforcement learning algorithm. We plan to investigate automatic adjustments of the dimensionality using current rewards. In this chapter, we focused on intra-group correlations. In future work, we plan to investigate correlations among extracted group factors, e.g., correlations between arms and legs.

CHAPTER 3

Extracting Bimanual Synergies with Sample-Efficient Reinforcement Learning

(Previously published as Luck, and Ben Amor 2017)

https://doi.org/10.1109/IROS.2017.8206356

ABSTRACT

Motor synergies are an important concept in human motor control. Through the co-activation of multiple muscles, complex motion involving many degrees-of-freedom can be generated. However, leveraging this concept in robotics typically entails using human data that may be incompatible for the kinematics of the robot. In this chapter, the goal is to enable a robot to identify synergies for low-dimensional control using trial-and-error only. I discuss how synergies can be learned through latent space policy search and introduce an extension of the algorithm for the re-use of previously learned synergies for exploration. The application of the algorithm on a bimanual manipulation task for the Baxter robot shows that performance can be increased by reusing learned synergies intra-task when learning to lift objects. But the reuse of synergies between two tasks with different objects did not lead to a significant improvement.

3.1 Introduction

The ability to manipulate objects in the environment is an important skill for humans and robots and has attracted a large body of research. Motor skills for reaching and grasping, for example, allow robots to physically interact with their immediate surroundings. Yet, generating control policies for these tasks is highly challenging due to the high number of involved degrees-of-freedom (DOF) and the inherent uncertainty.

An important concept in motor control that has helped to overcome these challenges, is the concept of motor synergies [40]; joint co-activations of a set of muscles from a smaller number of neural commands. A combination of a small number of synergies, leads to a large range of different possible movements. In humans, such synergies reduce the dimensionality of the control task and, in turn, reduce the cognitive effort during learning and execution [30, 41, 42]. Similarly, in robotics, synergies have been shown to improve grasp planning performance, while at the same time reducing the computational complexity [28, 57]. However, it is unclear how to identify and extract such synergies for different robot types and morphologies. Existing approaches typically rely on human demonstrations recorded through motion capture systems. Yet, synergies highly depend on the underlying kinematics and mechanics of the system and may not be easily transferred between a human and a robot.

Going beyond uni-manual manipulation, there are also many tasks that require the coordinated use of multiple limbs. Research in human motor control has presented evidence for the existence of bimanual synergies during object manipulation [58]. Different motor synergies may span both arms at the same time, or each arm separately. The ability to identify synergies that affect one or multiple groups of variables at the same time, would allow robots to efficiently learn bimanual manipulation tasks,

Figure 3.1: A bimanual robot learns to lift an object while simultaneously identifying synergies among the control variables.

e.g., pouring, turning a valve, or picking up a box. In this chapter, we present a reinforcement learning method that jointly learns motor synergies, as well as control policies for bimanual robot manipulation. Based on our previous discussion of *Group Factor Policy Search* (GrouPS) [59], we will show how sample-efficient reinforcement learning can be performed on a physical robot, without the need for potentially inaccurate simulations. In particular, we will show how GrouPS can autonomously learn dual-arm lifting of objects (see Fig. 3.1), without relying on prior human demonstrations. The algorithm extracts custom-made synergies that best fit the current robot and task. This is achieved through a combination of dimensionality reduction and policy search. Both, synergies and control policies, are updated while learning, thus exhausting the information provided by the sampling set executed in

each iteration. The result is a fast motor skill learning method for tasks that involve the coordination of multiple limbs. In our experiments, the robot autonomously learned object lifting strategies within a relatively small number of trials, i.e., about 1 hour of training time.

The presented method also allows visualizing the extracted bimanual synergies. Visualizing motor synergies enables users to better understand the couplings between control variables. Additionally, extracted synergies typically form basic movement "building blocks" which can be superimposed to generate a large variety of different behaviors.

In the remainder of this chapter, we will first introduce our policy search method and subsequently present its application to bimanual manipulation tasks.

3.2 Related Work

Human beings and animals are capable of producing a wide variety complex behaviors and motions that involve the coordinated activation of a large number of muscles. This ability poses the question of whether each muscle, or degree-of-freedom is independently controlled by the brain and the central nervous system. Research in neuroscience indicates that groups of muscles may be organized in a modular fashion to form *muscle synergies*. Activating a synergy will jointly co-activate all involved muscles and related joints. The result is a significant reduction in the number of controlled DOFs. In the case of grasping, Santello et al. [30] showed that $\approx 90\%$ of the variance during human grasping can be explained using only three synergies. To this end, human demonstration of grasps were first collected and, then, processed using Principal Component Analysis (PCA). Dimensionality reduction techniques, such as PCA, can uncover the lower-dimensional manifold in which the recorded data points are embedded. Using a similar strategy, other researcher teams have found

evidence for synergies in walking [41], running [60], or balancing [42]. Safavynia et al. [61] showed that the acquisition of new motor skills can enable the formation of new synergies. Hence, the composition of synergies is *not static* but can change as a result of repeated practice and reinforcement learning. This also means, that synergies across different subjects may converge towards the same optimal composition that is induced by the task constraints. Hug et al. [62] reported evidence for the formation of similar muscles synergies across expert cyclists over time.

In robotics, motor synergies such as Eigengrasps [28] are typically generated by applying PCA on a set of training data. In the field of grasping and manipulation, this methodology has found wide spread application such as in [29, 57, 63, 64] since it significantly reduces the number of control parameters, while at the same time generating interpretable principal components. Besides grasping, dimensionality reduction was also used to extract synergies for various other robotics tasks. In [65], linear and non-linear manifold learning techniques are used to extract postural synergies for walking and standing-up. The majority of these approaches relies on a large training set of (approximate) solutions, prior simulations, or human demonstrations to perform dimensionality reduction. Even if such data exists, it may drastically bias the search by limiting it to the subspace of initially provided solutions. Especially human demonstrations may be ill-suited for identifying robot synergies. In [66], an approach is present in which robot manipulator can learn synergies from random movements. However, synergies are often required for a specific task at hand. Hence, methods are needed that can generate a set of synergies from a task specification. In [44], we presented a first approach in which synergies can be learned through reinforcement. However, the approach did not allow for the specification of groups of variables within a synergy. In contrast to that, the work presented in the remainder of this chapter allows for users to identify specific connected groups of variables, e.g., left arm vs.

right arm. Providing this structural information, the algorithm generates synergies that can both model inter-group, as well as intra-group correlations [59]. This is particularly useful for tasks that involve multiple limbs. Synergies can be used to seed the learning algorithm with information about the structure of the manifold to explore. In the case of Group Factor Analysis, a first approach for transfer learning was introduced in [67]. Although, we build upon the same general idea of reusing previously learned factors, we focus in this chapter on a reinforcement learning setup rather than a supervised learning setup. Furthermore, we investigate the use of learned synergies for exploration in similar tasks.

3.3 Extracting Synergies with Policy Search

Synergies for robot motions are typically generated through the application of dimensionality reduction methods on existing data, e.g., joint angles recorded from a human subject. Our approach uses Group Factor Analysis (GFA) as introduced by Klami et al. [23]. However, in contrast to other work that relies on training data, we derive a reinforcement learning method that inherently performs factor analysis. In this section, we will introduce Group Factor Analysis briefly, describe its properties, and then proceed to show how Group Factor Analysis and Policy Search can be combined to yield the Group Factor Policy Search (GrouPS) algorithm. We close this section with the introduction of a new prior distribution for the transformation matrix in GrouPS, which enables the re-use of learned synergies for exploration.

3.3.1 Group Factor Analysis for Synergies

Based upon Factor Analysis, GFA assumes that the dimensions of a dataset can be split into groups of variables. The approach inherently assumes the existence of a strong correlation between variables of the same group, e.g., because they form a

logical unit such as the leg of a robot, a regions of the brain, or a gene set [23]. The model equation of GFA for M groups reads

$$\mathbf{a}^{(m)} = \mathbf{W}^{(m)}\mathbf{z} + \boldsymbol{\mu}^{(m)} + \boldsymbol{\epsilon}^{(m)}, \tag{3.1}$$

where $\mathbf{W}^{(m)}$ is the transformation matrix, $\boldsymbol{\mu}^{(m)}$ the mean vector, and isotropic noise

$$\boldsymbol{\epsilon}^{(m)} \sim \mathcal{N}\left(\mathbf{0}, \tilde{\tau}_{(m)}^{-1}\mathbf{I}\right) \tag{3.2}$$

defined by the precision $\tilde{\tau}_{(m)}$. The random vector $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ is the same for all groups while the action $\mathbf{a}^{(m)}$ contains the dimensions of the m-th group. Klami and colleagues introduced prior distributions over the parameters of the model equation given above. The most important prior distribution is

$$p(\mathbf{W}|\boldsymbol{\alpha}) = \prod_{m=1}^{M}\prod_{k=1}^{K}\prod_{d=1}^{D_m}\mathcal{N}\left(w_{d,k}^{(m)}\middle|0, \alpha_{m,k}^{-1}\right), \tag{3.3}$$

which defines a normal distribution over each entry of the transformation matrix $\mathbf{W}$, such that the matrix becomes structurally sparse. The parameter $\alpha_{m,k}$ is specific to each group m and component k and is given by the log-linear model

$$\log \boldsymbol{\alpha} = \mathbf{U}\mathbf{V}^{\mathrm{T}} + \boldsymbol{\mu}_u\mathbf{1}^{\mathrm{T}} + \mathbf{1}\boldsymbol{\mu}_v^{\mathrm{T}}. \tag{3.4}$$

The two matrices $\mathbf{U} \in \mathbb{R}^{M\times R}$ and $\mathbf{V} \in \mathbb{R}^{K\times R}$ are distributed according to a normal distribution with zero mean and λ precision. The rank factor $R \leq \min(M, K)$ influences how sensitive the components are to inter-group correlations. For $R = \min(M, K)$ the log-linear model is equal to a gamma distributed model assuming independent groups [24]. In order to compute the parameters of GFA given a data set, Variational Inference is used while assuming a factorization of parameters according to

$$p(\theta) = q(\mathbf{W})q(\boldsymbol{\tau})q(\mathbf{U})q(\mathbf{V})\prod_{t}^{T} q(\mathbf{z}_t) \tag{3.5}$$

with q being the approximated distributions. For $q(\mathbf{U})$ and $q(\mathbf{V})$, point estimators are chosen in order to compute the parameters with an optimization method such as L-BFGS [68].

3.3.2 Group Factor Policy Search

In its traditional form, GFA requires a dataset of examples in order to extract a low-dimensional manifold. However, in our case we would like to uncover the low-dimensional manifold without prior access to any such dataset. Instead, our goal is to enable a robot to identify synergies for low-dimensional control using trial-and-error only. To this end, we derive a reinforcement algorithm that jointly estimates parameters for dimensionality reduction, as well as a control policy [59].

In our framework, a trajectory consisting of actions $\mathbf{a}$ and states $\mathbf{s}$ is defined by

$$\boldsymbol{\tau} = (\mathbf{s}_1, \mathbf{a}_1, \ldots, \mathbf{s}_T, \mathbf{a}_T, \mathbf{s}_{T+1}) \tag{3.6}$$

where T is the number of time steps, and $\mathbf{s}_{T+1}$ is the final state. The objective of policy search is to maximize the expected reward over all possible trajectories given by

$$\mathbb{E}_{p(\boldsymbol{\tau})}\left[r = 1\right] = \iint p(\boldsymbol{\tau}, \theta) p(r = 1|\boldsymbol{\tau}) \mathrm{d}\theta \mathrm{d}\boldsymbol{\tau}, \tag{3.7}$$

where the reward r is defined as a binary variable with probability

$$p(r = 1|\boldsymbol{\tau}) \propto \exp(-c(\boldsymbol{\tau})) \tag{3.8}$$

and the cost function $c(\cdot)$ [50]. The parameters of the policy are defined by θ. Assuming the Markov property, the probability $p(\boldsymbol{\tau}, \theta)$ of the trajectory can be written as

$$p(\boldsymbol{\tau}, \theta) = p(\theta) p(\mathbf{s}_1) \prod_{t=1}^{T} p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t) \pi(\mathbf{a}_t|\mathbf{s}_t, \theta), \tag{3.9}$$

with a prior distribution $p(\theta)$ over the parameters θ, state probabilities $p(\mathbf{s}_1)$ and $p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$ and the stochastic policy $\pi(\mathbf{a}_t|\mathbf{s}_t, \theta)$.

Input: Reward function $R(\cdot)$ and initializations of parameters. Choose number of latent dimension n and rank r. Set hyper-parameter and define groupings of actions. Set $\hat{\mathbf{W}}$ either to a previously learned synergy or to zero.

---

while *reward not converged* **do**

  for *h=1:H* **do** # Sample H rollouts

    for *t=1:T* **do**

      $\mathbf{a}_t = \mathbf{W}_i\mathbf{Z}\boldsymbol{\phi} + \mathbf{M}\boldsymbol{\phi} + \mathbf{E}\boldsymbol{\phi}$

      with $\mathbf{Z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and $\mathbf{E} \sim \mathcal{N}(\mathbf{0}, \tilde{\boldsymbol{\tau}})$, where $\tilde{\boldsymbol{\tau}}^{(m)} = \tilde{\tau}_m^{-1}\mathbf{I}$

      Execute action $\mathbf{a}_t$

    Observe and store reward $R(\boldsymbol{\tau})$

  Initialization of q-distribution

  while *not converged* **do**

    Update $q(\mathbf{M})$, $q(\mathbf{W})$, $q\left(\tilde{\mathbf{Z}}\right)$, $q(\boldsymbol{\alpha})$ and $q(\tilde{\boldsymbol{\tau}})$

  $\mathbf{M} = \mathbb{E}_{q(\mathbf{M})}[\mathbf{M}]$

  $\mathbf{W} = \mathbb{E}_{q(\mathbf{W})}[\mathbf{W}]$ with Eq. (9)

  $\boldsymbol{\alpha} = \mathbb{E}_{q(\boldsymbol{\alpha})}[\boldsymbol{\alpha}]$ with Eq. (11-14)

  $\tilde{\boldsymbol{\tau}} = \mathbb{E}_{q(\tilde{\boldsymbol{\tau}})}[\tilde{\boldsymbol{\tau}}]$

Result: Linear weights $\mathbf{M}$ for the feature vector $\boldsymbol{\phi}$, representing the final policy. The columns of $\mathbf{W}$ represent the factors of the latent space.

Algorithm 2: Outline of the Group Factor Policy Search (GrouPS) algorithm.

The algorithm is compatible to the previous version presented in [59] since setting $\hat{\mathbf{W}}$ to zero results in the original update equations.

The model equation for our algorithm Group Factor Policy Search (GrouPS) reads similar to Group Factor Analysis with

$$\mathbf{a}_t^{(m)} = \left(\mathbf{W}^{(m)}\mathbf{Z}_t + \mathbf{M}^{(m)} + \mathbf{E}_t^{(m)}\right)\boldsymbol{\phi}(\mathbf{s}_t, t), \tag{3.10}$$

where $\mathbf{a}_t^{(m)} \in \mathbb{R}^{D_m}$ represents the joint or action vector and $\boldsymbol{\phi}(\mathbf{s}_t, t) \in \mathbb{R}^p$ the feature vector containing the basis functions in its entries. The actual values of the basis functions depend on the current state or time step. For notational clarity, we are going to omit states and actions for the feature vector $\boldsymbol{\phi}$ in the remainder of the chapter. The matrices $\mathbf{Z}_t$ and $\mathbf{E}_t^{(m)}$ are distributed according to matrix-variate normal distributions: each entry of the latent matrix $\mathbf{Z}_t$ is sampled from a standard normal distribution, whereas the entries of $\mathbf{E}_t^{(m)}$ model the isotropic noise with $\mathcal{N}(0, \tilde{\tau}_m^{-1})$. The mean policy is given by matrix $\mathbf{M}^{(m)} \in \mathbb{R}^{D_m \times p}$ whose parameters, i.e. entries, have to be estimated. The transformation matrix is given by $\mathbf{W}^{(m)}$ and contains the extracted synergies in its columns. As shown in [44], the term $\mathbf{Z}_t\boldsymbol{\phi}$ can be rewritten as

$$\tilde{\mathbf{z}}_t = \mathbf{Z}_t\boldsymbol{\phi} \sim \mathcal{N}\left(\mathbf{0}, \boldsymbol{\phi}^{\mathrm{T}}\boldsymbol{\phi}\mathbf{I}\right), \tag{3.11}$$

indicating that the noise depends on the values of the basis functions. In the case of normalized basis functions, i.e. $\| \boldsymbol{\phi} \|_2 = 1$, this term is distributed according to a standard normal distribution. Finally, given above distributions, the stochastic policy $\pi(\mathbf{a}_t|\mathbf{s}_t, \theta)$ (Eq. 3.9) of Group Factor Analysis can be found with

$$\prod_{m=1}^{M} \mathcal{N}\left(\mathbf{a}_t^{(m)} \middle| \mathbf{W}^{(m)}\tilde{\mathbf{z}}_t + \mathbf{M}^{(m)}\boldsymbol{\phi}, (\boldsymbol{\phi}^{\mathrm{T}}\boldsymbol{\phi})\tilde{\tau}_m^{-1}\mathbf{I}\right). \tag{3.12}$$

As stated in [59], Group Factor Policy Search does not perform Factor Analysis for each group separately. Rather, Eq. 3.12 in combination with the prior distribution of $\mathbf{W}$ from Eq. 3.3 allows us to uncover components, i.e. columns, in $\mathbf{W}$ with a strong correlation among the groups. For the computation of the parameters we utilize a

Variational Inference approach and determine approximated distributions given by the factorization

$$q(\boldsymbol{\theta}) = q(\tilde{\mathbf{Z}})q(\mathbf{W})q(\tilde{\boldsymbol{\tau}})q(\mathbf{M})q(\boldsymbol{\alpha}). \tag{3.13}$$

The final update equations of the algorithm (Alg. 2) and more details regarding Group Factor Policy Search can be found in [59].

3.3.3 Transfer Learning with GrouPS

One possibility to incorporate the idea of transfer learning into Group Factor Policy Search is to use the latent space found in one experiment as prior information for a subsequent experiment. The prior of the latent space is given by the normal distribution

$$p(w_{d,k}^{(m)}|\alpha_{m,k}) = \mathcal{N}(w_{d,k}^{(m)}|0, \alpha_{m,k}^{-1}). \tag{3.14}$$

for each entry $w_{d,k}^{(m)}$ of the transformation matrix $W = ({W^1}^{\mathrm{T}}, {W^2}^{\mathrm{T}}, \ldots)^{\mathrm{T}}$. The parameter $\alpha_{m,k}$ is the variance parameter controlling the inter-group flexibility of each dimension, i.e. column of $\mathbf{W}$. We can now incorporate a previously learned latent space by changing the prior of $\mathbf{W}$ to

$$p\left(w_{d,k}^{(m)}|\alpha_{m,k}\right) = \mathcal{N}\left(w_{d,k}^{(m)}\middle|\hat{w}_{d,k}^{(m)}, \alpha_{m,k}^{-1}\right) \tag{3.15}$$

with $\hat{w}_{d,k}^{(m)}$ being the entries of $\hat{\mathbf{W}}$ learned in a previous experiment. This new prior changes the update equations for both α and $\mathbf{W}$ for the Variational Bayes update. For the transformation matrix $\mathbf{W}$ only the update of the mean [59, Eq. 21] has to be changed to

$$\boldsymbol{\mu}_{m,j}^{W} = \boldsymbol{\Sigma}_{m}^{W} \cdot \mathbb{E}_{p(\boldsymbol{\tau})}\left[\frac{p(r=1|\boldsymbol{\tau})}{\hat{R}} \sum_{t=1}^{T}\left(\frac{\left(a_{t,j}^{(m)} - \mathbb{E}_{\mathbf{M}}\left[\mathbf{m}_{j,:}^{(m)}\right]\boldsymbol{\phi}\right)\mathbb{E}_{\tilde{\mathbf{z}}}\left[\tilde{\mathbf{z}}_{t}^{\mathrm{T}}\right]}{\boldsymbol{\phi}^{\mathrm{T}}\boldsymbol{\phi}\mathbb{E}_{\tilde{\tau}_m}\left[\tilde{\tau}_m\right]^{-1}} - \hat{\mathbf{w}}_{j,:}^{(m)}\overline{\overline{\boldsymbol{\alpha}}}_{m,K}\right)\right] \tag{3.16}$$

while for the log-linear model of $\boldsymbol{\alpha}$ the derivatives change: Since the prior distribution of the transformation matrix $\mathbf{W}$ depends on $\boldsymbol{\alpha}$, the update rule for the log-linear model has also to be updated. Changing the prior of $\mathbf{W}$ leads to a slightly different log-likelihood for the optimization of the parameters as stated in [23] with $\boldsymbol{\Gamma}$ being

$$\boldsymbol{\Gamma} = \mathbb{E}_{\mathbf{W}}\left[(\mathbf{w}_{k,:}^{(m)} - \mathbf{w}_{k,:}^{(m)\prime})(\mathbf{w}_{k,:}^{(m)} - \mathbf{w}_{k,:}^{(m)\prime})^{\mathrm{T}}\right]. \tag{3.17}$$

The final gradients are then given with

$$\frac{\partial L_{\mathbf{U},\mathbf{V}}(\theta)}{\partial \mathbf{U}_{m,:}} = 2\lambda \mathbf{U}_{m,:} + \sum_{k=1}^{K} D_m \mathbf{V}_{k,:} - \sum_{K=1}^{K} \boldsymbol{\Gamma} \exp\left(\mathbf{U}_{m,:}\mathbf{V}_{k,:}^{\mathrm{T}} + \boldsymbol{\mu}_{\mathbf{U}_m} + \boldsymbol{\mu}_{\mathbf{V}_k}\right) \mathbf{V}_{k,:}, \tag{3.18}$$

$$\frac{\partial L_{\mathbf{U},\mathbf{V}}(\theta)}{\partial \mathbf{V}_{m,:}} = 2\lambda \mathbf{V}_{k,:} + \sum_{m=1}^{M} D_m \mathbf{U}_{m,:} - \sum_{m=1}^{M} \boldsymbol{\Gamma} \exp\left(\mathbf{U}_{m,:}\mathbf{V}_{k,:}^{\mathrm{T}} + \boldsymbol{\mu}_{\mathbf{U}_m} + \boldsymbol{\mu}_{\mathbf{V}_k}\right) \mathbf{U}_{m,:}, \tag{3.19}$$

$$\frac{\partial L_{U,V}(\theta)}{\partial \boldsymbol{\mu}_{\mathbf{U}_m}} = D_m K - \sum_{K=1}^{K} \boldsymbol{\Gamma} \exp\left(\mathbf{U}_{m,:}\mathbf{V}_{k,:}^{\mathrm{T}} + \boldsymbol{\mu}_{\mathbf{U}_m} + \boldsymbol{\mu}_{\mathbf{V}_k}\right), \tag{3.20}$$

$$\frac{\partial L_{U,V}(\theta)}{\partial \boldsymbol{\mu}_{\mathbf{V}_k}} = D_m M - \sum_{m=1}^{M} \boldsymbol{\Gamma} \exp\left(\mathbf{U}_{m,:}\mathbf{V}_{k,:}^{\mathrm{T}} + \boldsymbol{\mu}_{\mathbf{U}_m} + \boldsymbol{\mu}_{\mathbf{V}_k}\right). \tag{3.21}$$

Introducing this new prior offers the possibility to infuse the algorithm with transformation matrices from different runs or tasks (Alg. 1). The intuition behind sparsity for the transformation matrix $\mathbf{W}$ is now slightly different: instead of driving the entries of the transformation matrix to zero the sparsity prior is trying to maintain the mean and $\boldsymbol{\alpha}$ controls if deviations are added to one group or several groups per column of $\mathbf{W}$.

3.4 Experiments

In order to evaluate the ability of the GrouPS algorithm to extract meaningful synergies during reinforcement learning, we performed experiments with bimanual

manipulation tasks on the Baxter robot. In the following experiments, the robot was tasked to learn how to lift an object with both arms. The goal is to lift the object as high as possible while retaining stability. The initial policy consist of a zeroed **M**-matrix, i.e., the robot performs no action.

3.4.1 Experimental Setup

Each Object was placed on a table of height 77cm in front of a Baxter robot with the same initial position of the arms. Then the GrouPS algorithm was executed for five latent dimensions and rank three over ten iterations. In each iteration but the first, ten samples were newly generated and executed. Each sample constitutes a full lifting trajectory. The samples were included in a sample set of size twenty, from which only the ten best samples were selected for processing while the others were discarded. This is similar to the importance sampling process used in [44, 59, 69]. In the very first iteration, twenty samples were generated. The motivation behind this approach is to allow failure during learning in the real world and compensate for any noise in the reward function. The actions $\mathbf{a}_t$ represent velocities in joint angles of the Baxter robot, thus the action space has 14 dimensions. The whole learning process was performed with real executions on the robot only, and using only rewards generated from the real world (Fig. 3.2). A kinematic validation process was only used for the purpose of detecting hazardous trajectories before execution (Validation). If a trajectory is deemed dangerous, the process can deny its execution and assign a reward of zero to the trajectory, thus effectively removing the trajectory from the set used for the estimation of the parameters.

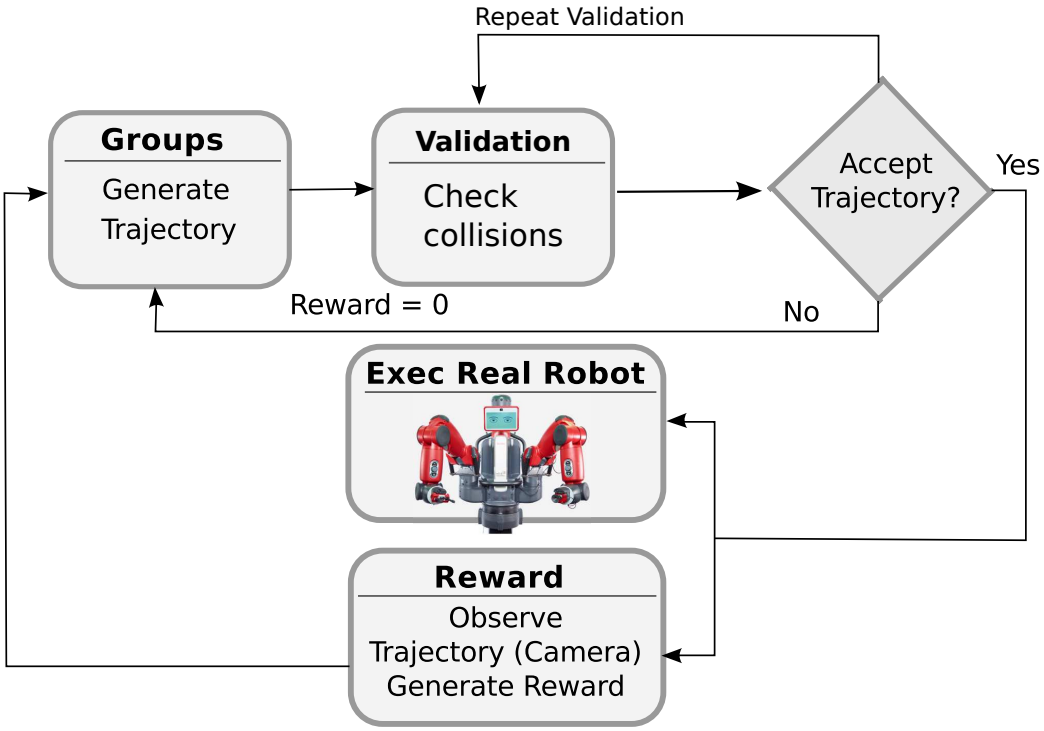


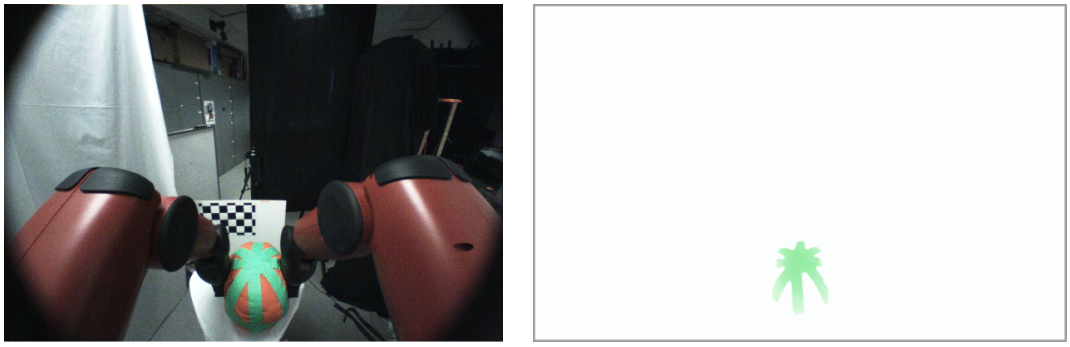
Figure 3.2: In each iteration a sample, i.e. trajectory, is first simulated and to be accepted by the process (or a human operator) for execution. Then the trajectory is executed on the real robot and a reward is generated, which will be used by GrouPS to compute the updates of the policy parameters.

3.4.2 Groups

For the experiments on the Baxter robot, four groups were chosen in total, two for each arm. The first group for each arm contains four joints with all rotational and one twisting joint, while the second group consists of three twisting joints (see Table 3.1).

3.4.3 Used Basis Functions

As basis functions ϕ, eight radial basis functions, i.e. Gaussian distributions, with a variance of three were used. The mean values were equidistant distributed over the 15 time steps of the trajectory starting with time step -3 and ending with time step 18.



(a) The 800x1280 image delivered by Baxter's head camera.

(b) The image after using color and median filtering.

Figure 3.3: The current height of an object is approximated by color filtering.

3.4.4 Reward Function

As input for the reward function, we chose the height of the object in the picture delivered by the integrated head camera of the Baxter robot. In order to detect the object during lifting, we used basic color filtering and reflective green tape on the objects (Fig. 3.3).

For each time step in the trajectory, we create and process one image and detect the maximal height of green pixels in the image. Thus, our reward function does not use the actual height of the object, but the pixel height h in the projected 2D image. Since we employed episodic rewards in our experiments, we used the sum of the exponential cost function resulting in equation $\sum_{t=1}^{T} \exp(-(1 - \frac{h_t}{800}))$ for the reward. The height h_t is here normalized with the image height of 800 pixel. Due to the angle of the camera, the reward function is sensitive to horizontal movements of the objects, thus leading to noticeable noise in the generated reward values.

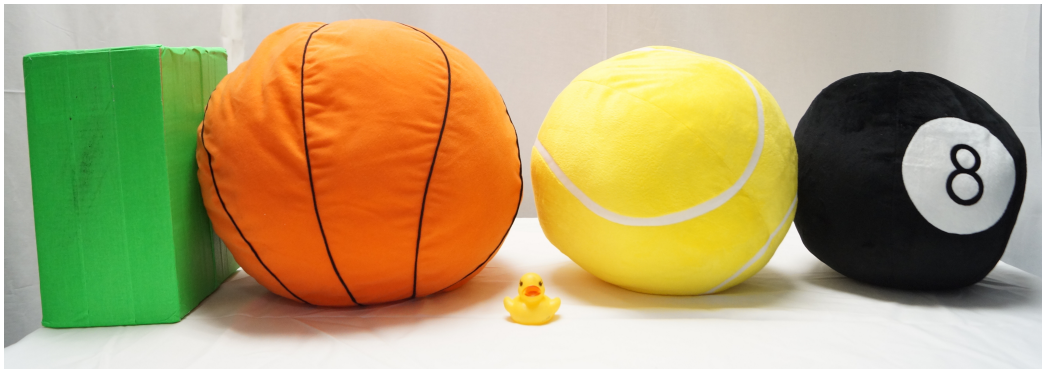


Figure 3.4: The four different objects lifted next to each other with the ICRA duck as size reference.

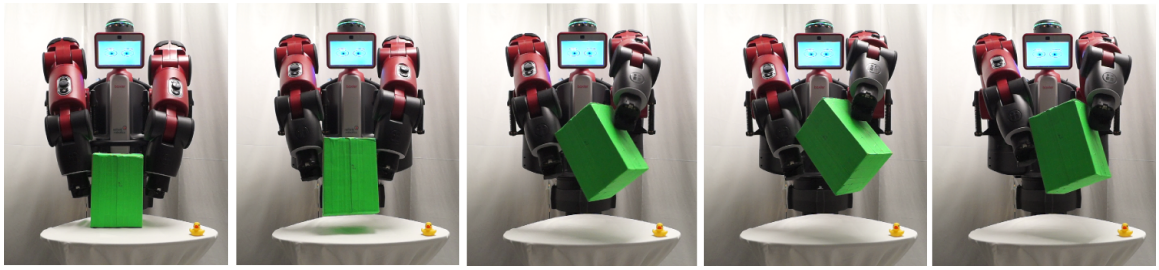


Figure 3.5: The final sequence of actions for lifting a cardboard box found by GrouPS.

3.4.5 Objects

Experiments were performed with four different objects: An orange ball with diameter 33cm, a yellow ball with diameter 27cm, a black ball with diameter 22cm, and a cardboard box with dimensions 26cm × 13.5cm × 18cm (Fig. 3.4). While the cardboard box is a rigid object, all balls are soft, non-rigid and deformable. This property makes them particularly challenging to handle for the robot. While the evaluations concentrate on comparisons between the orange ball and box, final results of the remaining two balls will be shown to demonstrate the general capability of GrouPS to solve the task.

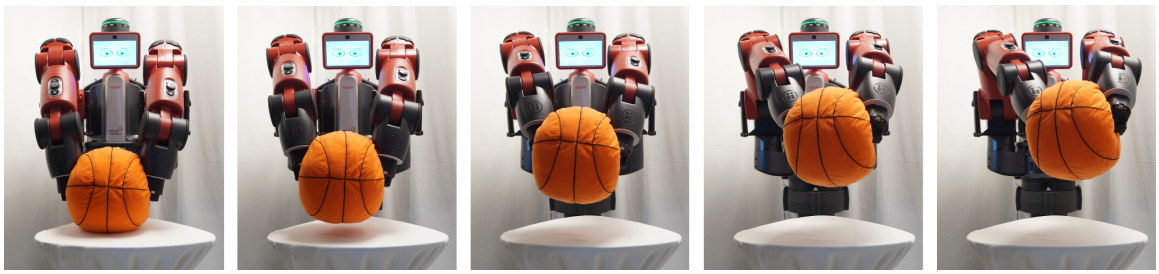


Figure 3.6: One final sequence of actions for lifting the orange ball found by GrouPS.

3.4.6 Time and Sample Size

In all but the first iterations, ten samples were generated while in the very first iteration twenty samples were produced. Thus, the total sample size is 110 samples for one experiment. Each execution of a sample requires about 25 seconds and one complete iteration about four minutes. Accordingly, one experiment requires approximately one hour.

3.4.7 Reproducibility

All involved items are internationally available through the company IKEA®. The cardboard box is a standard parcel size with dimensions 26cm × 13.5cm × 18cm and is used by several international logistics companies.

3.4.8 Experiments

While not the main scope of this chapter, a comparison was performed between GrouPS and the *Policy Learning by Weighting Exploration with the Returns* algorithm (PoWER) [69] on the task of lifting the orange ball (Fig. 3.6). PoWER was used in a configuration with a full covariance matrix over the number of basis functions. PoWER and GrouPS are naturally two very similar algorithms based on stochastic search. While PoWER makes use of an Expectation Maximization framework, the

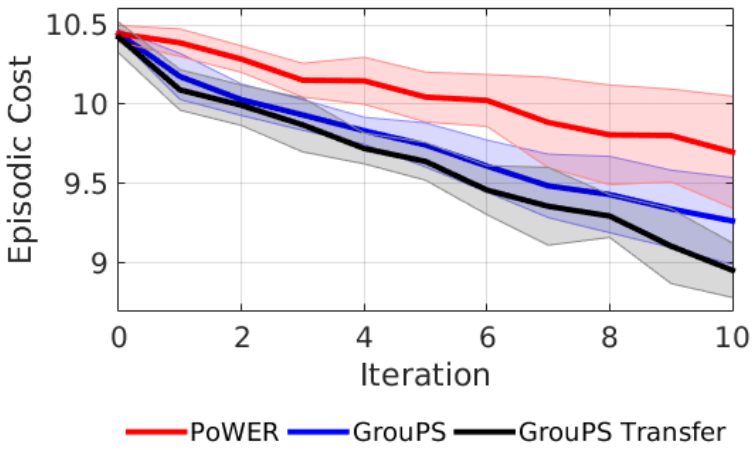


Figure 3.7: A comparison between the GrouPS and PoWER algorithm on a lifting task with the orange ball. The presented variant of GrouPS using a synergy matrix *from the same task* outperforms both GrouPS and PoWER. Each algorithm was executed four times and the mean and standard variance were calculated. The vertical axis shows the total cost over 15 time steps and is based on the height of the ball.

GrouPS algorithm is based on Variational Inference. Also, exploration in PoWER is solely in the high dimensional space without exploiting latent structures for directed exploration while GrouPS incorporates this feature due to its more complex model. Both algorithms made use of the same number of samples over ten iterations.

In order to evaluate the introduced modification of initializing GrouPS with previously learned synergies, three experiments were conducted: First, GrouPS was initialized with synergies found while learning to lift the orange ball (Fig. 3.9) and then executed four times on the same task. Then, the same initialization was used to learn to lift the box with the Baxter robot. Finally, Groups with random initialization and without pre-initialized mean was applied on the box lifting task and the learned synergies used to initialize GrouPS for lifting the orange ball. All above described experiments were performed four times each.

3.5 Results

The GrouPS algorithm (without extension) was able to find trajectories for lifting non-rigid objects of different sizes such as the orange ball (Fig. 3.6), as well as for a rigid cardboard box (Fig. 3.5). All of the trajectories resulted in a stable final position holding the object in a higher position (Fig. 3.8). One sequence of synergy matrices $\mathbf{W}$ is shown in Figure 3.9, where the color of the squares indicates whether the values are negative (gray) or positive (black) and their size correspond to their absolute value. The depicted transformation matrices were computed during an experiment aiming to learn how to lift the orange ball. The extracted synergies can now be found in the columns of $\mathbf{W}$ and replayed directly on the robot for evaluation. Figure 3.10 shows two synergies found by GrouPS during the learning process which encodes movements for both arms. The first synergy is an opening and closing movement of both arms, while the second synergy showcases a movement to up or down. Both synergies can be combined to generate more complex movements like an upward, closing movement.

Fig. 3.7 depicts a comparison of results between GrouPS, GrouPS using synergies, and PoWER on the task of lifting the orange ball. While GrouPS outperforms PoWER, pre-initializing GrouPS with learned synergies from the same task leads to another increase in performance. The comparison between using synergies learned from different tasks and GrouPS without modification is presented in Table 3.2 which shows that the differences between both variants are not significant.

3.6 Discussion

The purpose of the experiments presented above was to evaluate the ability of the Group Factor Policy Search algorithm to extract not only a successful policy [59], but also uncover latent synergies specific to the task and robot during the learning process.

(a) Orange Ball (b) Yellow Ball

(c) Box (d) Black Ball

Figure 3.8: The final pose of trajectories for lifting an object.

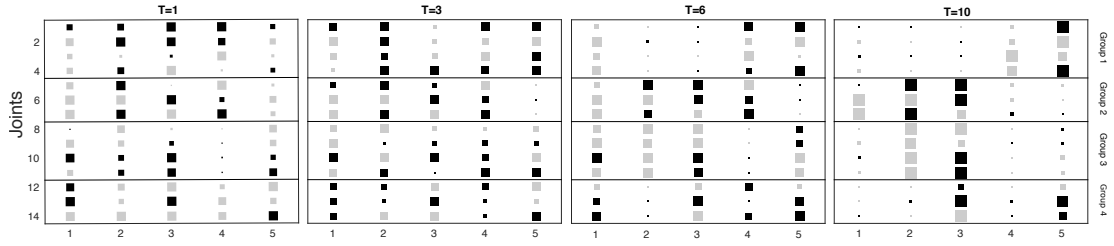


Figure 3.9: A sequence of transformation matrices $\mathbf{W}$ computed in each iteration t. The transformation matrix contains the uncovered synergies and is forced to be sparse by the prior distribution on $\mathbf{W}$. The color of the squares represent the sign of the value for this entry: Gray color means negative values and black positive values. The size of the squares are corresponding with the absolute value in such a way that a square is small for small values and the opposite for bigger values.

Figure 3.10: Two synergies and their combinations found during the execution of the GrouPS algorithm for learning to lift the orange ball. The horizontal axis is showing a synergy of closing and opening motions whereas the vertical axis represents a up- and downwards movement of both arms. Different combinations of those synergies are shown in the four corners of this figure.

Table 3.1: The groups for the joints of the Baxter robot chosen for the experiments presented in this chapter. The joint names correspond to the technical documentation by Rethink Robotics

Group	Left Arm	Group	Right Arm
1	W1	3	W1
1	S1	3	S1
1	E0	3	E0
1	E1	3	E1
2	W0	4	W0
2	S0	4	S0
2	W2	4	W2

Table 3.2: Comparison of GrouPS with the proposed modification for reusing synergies *from different tasks*. For the task of lifting the orange ball one set of synergies, the transformation matrix $\mathbf{W}$, learned in the eight iteration on the box lifting task were used and vice-versa for the box-lifting task. The table shows the final mean cost and standard deviation.

Algorithm	**Cost (Orange Ball)**
GrouPS	9.55 ± 0.19
GrouPS initialized with Synergies	9.47 ± 0.22
Algorithm	**Cost (Box)**
GrouPS	9.26 ± 0.27
GrouPS initialized with Synergies	9.28 ± 0.25

It was found that Group Factor Policy Search is in fact able to uncover synergies (Fig. 3.9 and Fig. 3.10) while developing a successful policy for a bi-manual lifting task. Figure 3.10 also demonstrates, that these synergies can be combined in a meaningful way to produce new or adapted motions which may be used for similar tasks or initializations of other learning algorithms. Interestingly, the algorithm uncovers two synergies which resemble the nature of the task very well (Fig. 3.10): While different values of one synergy lead to an opening or closing movement, the second synergy controls the vertical movement of both hands.

An analysis of the transformation matrices in Figure 3.9 shows that weak correlations between groups disappear over time and strong ones reinforce. However, it can be noted that the transformation matrix is thinning out towards the end of the learning process, very likely due to the convergence to an optimal policy. Thus, it is more likely to find useful synergies in earlier iterations. The most usable synergies were be found in iterations seven and eight.

An variant of the GrouPS algorithm was presented which can make use of uncovered synergies for directed exploration. While it was found that GrouPS initialized with synergies *learned from the same task* indeed leads to an increase in performance (Fig. 3.7) it is surprising to find that is not the case when using synergies *learned from another task*. This applies for both directions: using synergies from lifting the orange ball to learn to lift the box, and using synergies learned from lifting the box to learn to lift the orange ball. Both objects, box and ball, are naturally different in shape and so is the optimal strategy for lifting them. Especially the box posed challenges for the robot, since the endeffectors can slide easily along the sides of the box. However, the robot learned to exploit this property over time in order to change the orientation of the box such that one corner of the box points upwards.

3.7 Conclusions

In this chapter, I presented a methodology and algorithm for extracting synergies for motor skill learning in robots and using them to accelerate learning. The approach does not require any prior data from human demonstrations or other sources. Instead, we presented a reinforcement learning method that naturally combines dimensionality reduction and policy search. We have shown in experiments with a real-world robot that this combination leads to sample-efficient reinforcement learning. In addition, we have discussed how the generated synergies can be visualized in order to introspect the learning process and better understand the generated coupling of joints.

The potential for speeding up learning in inter- or intra-task transfer using synergies was evaluated. It was found that the presented variant of GrouPS can outperform the base algorithm when reusing synergies from the same task. However, using synergies from a different task did not lead to an increased performance. In future work we will investigate if this insight applies to the general case of transfer learning with GrouPS.

CHAPTER 4

Improved Exploration through Latent Trajectory Optimization in Deep Deterministic Policy Gradient

ABSTRACT

Model-free reinforcement learning algorithms such as Deep Deterministic Policy Gradient (DDPG) often require additional exploration strategies, especially if the actor is of deterministic nature. This work evaluates the use of model-based trajectory optimization methods used for exploration in Deep Deterministic Policy Gradient when trained on a latent image embedding. In addition, an extension of DDPG is derived using a value function as critic, making use of a learned deep dynamics model to compute the policy gradient. This approach leads to a symbiotic relationship between the deep reinforcement learning algorithm and the latent trajectory optimizer. The trajectory optimizer benefits from the critic learned by the RL algorithm and the latter from the enhanced exploration generated by the planner. The developed methods are evaluated on two continuous control tasks, one in simulation and one in the real world. In particular, a Baxter robot is trained to perform an insertion task, while only receiving sparse rewards and images as observations from the environment.

4.1 Introduction

Reinforcement learning (RL) methods enabled the development of autonomous systems that can autonomously learn and master a task when provided with an objective function. RL has been successfully applied to a wide range of tasks including flying [21, 22], manipulation [13–15, 17, 56], locomotion [18, 55], and even autonomous driving [70, 71]. The vast majority of RL algorithms can be classified into the two categories of (a) inherently stochastic or (b) deterministic methods. While inherently stochastic methods have their exploration typically built-in [72, 73], their deterministic counterparts require an, often independent, exploration strategy for the acquisition of new experiences within the task domain [51, 74]. In deep reinforcement learning, simple exploration strategies such as Gaussian noise or Ornstein-Uhlenbeck (OU) processes [75], which model Brownian motion, are standard practice and have been found to be effective [74]. However, research has shown that advanced exploration strategies can lead to a higher sample-efficiency and performance of the underlying RL algorithm [44]. In practice, there are two ways to incorporate advanced exploration strategies into deterministic policy search methods. Where possible, one can reformulate the deterministic approach within a stochastic framework, such as by modeling the actions to be sampled as a distribution. Parameters of the distribution can then be trained and are tightly interconnected with the learning framework. One example for this methodology, is the transformation of Policy Search with Weighted Returns (PoWER) [51] into Policy Search with Probabilistic Principal Component Exploration (PePPEr) [44]. Instead of using a fixed Gaussian distribution for exploration, the noise generating process in PePPEr is based on Probabilistic Principal Component Analysis (PPCA) and generates samples along the latent space of high-reward actions. Generating explorative noise from PPCA and sampling along the latent space was

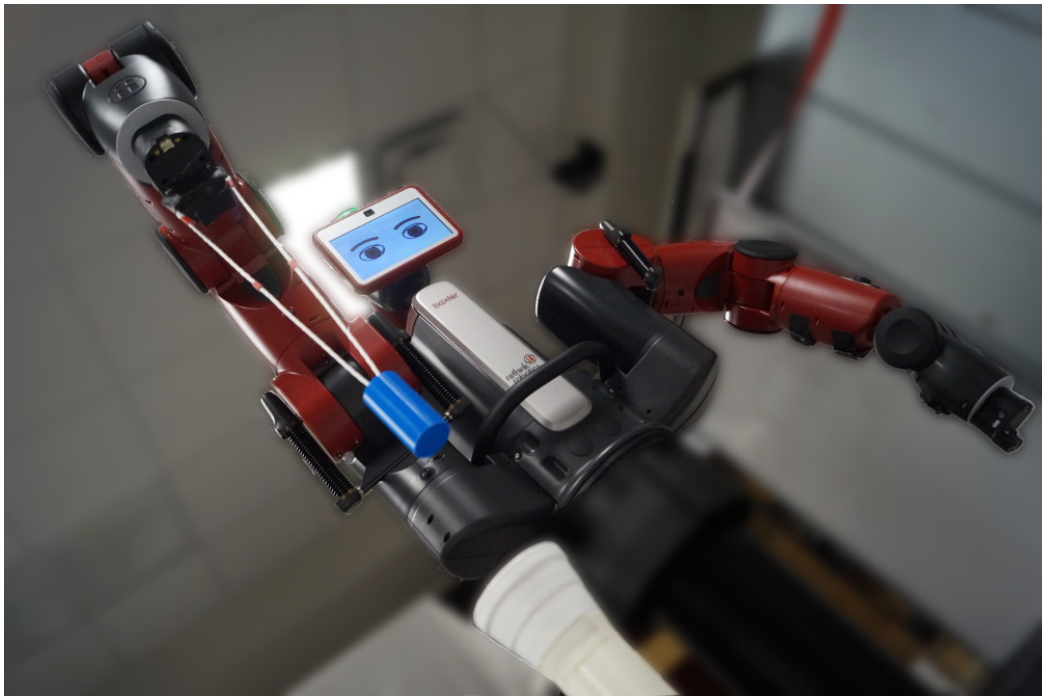
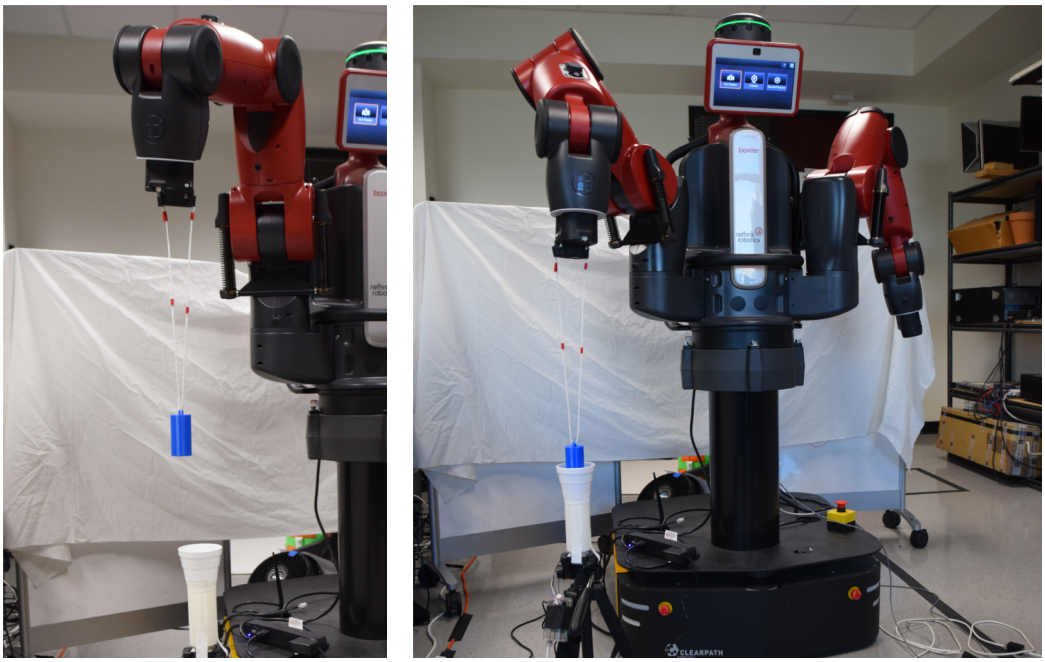


Figure 4.1: A Baxter robot learning a visuo-motor policy for an insertion task using efficient exploration in latent spaces. The peg is suspended from a string.

shown to outperform the previously fixed Gaussian exploration. Alternatively, one can choose to optimize the exploration strategy itself. Examples of this methodology are count-based exploration strategies [76], novelty search [77] or curiosity-driven approaches [78] which can be transferred with ease to other algorithms or frameworks. Typically, when incorporating these techniques into reinforcement learning, they are limited to local exploration cues based on the current state. This chapter aims to combine the model-free deep deterministic policy gradient method with a model-based exploration technique for increased sample-efficiency in real world task domains. The proposed method generates exploratory noise by optimizing a (latent) trajectory from the current state to ideal future states, based on value functions learned by an RL algorithm. This experience is, in turn, used by the RL algorithm to optimize policy and value functions in an off-policy fashion, providing an improved objective function



(a) Rand. initial position (b) Insertion started

Figure 4.2: The experimental setup in which a Baxter robot has to insert a blue cylinder into a white tube (b). The cylinder is with a string attached to the end-effector of the robot. Camera images are recorded with the integrated end-effector camera. The sensor detecting the state of insertion is integrated into the white tube. Experiments on this platform were run fully autonomously without human intervention or simulations.

for the trajectory optimizer. We investigate whether this strategy of formulating exploration as a latent trajectory optimization problem leads to an improved learning process both in simulation, as well as in a robotic insertion task executed solely in the real world. In particular, we apply our approach to a challenging, flexible insertion task as seen in Fig. 4.1.

4.2 Related Work

The advancement of deep reinforcement learning in recent years has lead to the development of a number of methods combining model-free and model-based learning techniques, in particular to improve the sample complexity of deep reinforcement

learning methods. Nagabandi et al. [79] present a model-based deep reinforcement learning approach which learns a deep dynamic function mapping a state and action pair $(\mathbf{s}_t, \mathbf{a}_t)$ to the next state $\mathbf{s}_{t+1}$. The dynamics function is used to unroll a trajectory and to create an objective function based on the cumulative reward along the trajectory. This objective function is, then, used to optimize the actions along the trajectory and thereafter the first action is executed. The procedure is repeated whenever the next state is reached. After a dataset of executed trajectories is collected by the planning process, the policy of a model-free reinforcement learning algorithm is initialized in a supervised fashion by training it to match the actions produced by the planner. This technique is different to our approach in that we do not force the actor to match the executed action, but rather see it as an exploration from which we generate off-policy updates. Furthermore, it is implicitly assumed in [79] that a reward function is available for each state during the planning process. This can be a rather strong assumption, especially when learning in the real world without access to a simulation of the task and only providing minimal human supervision. Using executions in the environment during the planning process would be too costly since each change in state would require a re-execution of the whole trajectory. Since our insertion task provides only sparse rewards during execution, the trajectory planning algorithm would fail when relying only on rewards due to flat regions with zero reward and require additional reward engineering. This leaves a large and mostly flat region in the state space with a reward of zero.

In [80], Chua et al. introduce the model-based *probabilistic ensembles with trajectory sampling* method. This work builds upon [79], but also makes use of a reward function. It makes use of a probabilistic formulation of the deep dynamics function by using an ensemble of bootstrapped models encoding distributions to improve the sample

complexity and improves the properties of the trajectory planner. Both approaches do not explicitly train an actor or a critic network.

Similarly to us, Universal planning networks [81] introduced by Srinivas et al. use a latent, gradient-based trajectory optimization method. However, the planner requires a goal state for the trajectory optimization. In certain tasks such as walking or running, it might be hard to acquire such a goal state to use in place of a velocity-based reward function. It is mentioned in [81] that to achieve walking, it was necessary to re-render images or reformulate the objective function by including an accessible dense reward function.

In contrast to previous work, we focus explicitly on the impact of using trajectory optimization as an additional technique for exploration and its impact on the learning process when used by a deep reinforcement learning algorithm such as Deep Deterministic Policy Gradient. Furthermore, using an actor-critic architecture is a key element in our work to allow off-policy updates in a fast manner during the training process and to inform the trajectory optimization process initially.

4.3 Method

The following sections introduce the different components used to generate explorative actions via trajectory optimization. We first describe the image embedding used, then the training process of the dynamics function and Deep Deterministic Policy Gradient (DDPG) [74], as well as its extension for the use of a value function. The section ends with a description of our trajectory optimization based exploration for DDPG.

4.3.1 Image Embedding

All tasks used throughout this chapter are setup such that they use only images as observations, which have to be projected into a latent image embedding. This serves two main purposes: First, the number of parameters is greatly reduced since the actor, critic, and the dynamics network can be trained directly in the low dimensional latent space. Second, it is desirable to enforce temporal constraints within the latent image embedding, namely that subsequent images are close to each other after being projected into the latent space. Therefore, we make use of the recently introduced approach of time-contrastive networks [26]: the loss function enforces that the distance between latent representations of two subsequent images are small but the distance between two randomly chosen images is above a chosen threshold α. Enforcing a temporal constraint in the latent space improves the learning process of a consistent deep dynamics function in the latent space [26]. Time-contrastive networks make use of two losses. The first is defined on the output of the decoder network and the input image as found in most autoencoder implementations. The second loss, the triplet loss, takes the latent representation $\mathbf{z}_t$ and $\mathbf{z}_{t+1}$ of two temporally close images and the latent representation $\mathbf{z}_r$ of a randomly chosen image.

Thus, given two temporal images Im_t and Im_{t+1} and a randomly chosen image Im_r, the loss functions for each element in the batch is given by

$$L(\mathrm{Im}_t, \mathrm{Im}_{t+1}, \mathrm{Im}_r) = L_{\mathrm{ae}}(\mathrm{Im}_t) + L_{\mathrm{contr}}(\mathrm{Im}_t, \mathrm{Im}_{t+1}, \mathrm{Im}_r). \tag{4.1}$$

The classical autoencoder loss L_{ae} and the contrastive loss L_{contr} are here defined as

$$\begin{aligned} L_{\mathrm{ae}} =& \parallel \mathrm{Im}_t - \mathrm{D}(E(\mathrm{Im}_t)) \parallel, \\ L_{\mathrm{contr}}\,(\mathrm{Im}_t, \mathrm{Im}_{t+1}, \mathrm{Im}_r) =& \parallel E(\mathrm{Im}_t) - E(\mathrm{Im}_{t+1}) \parallel \\ &+ \max(\alpha - \parallel E(\mathrm{Im}_t) - E(\mathrm{Im}_r) \parallel, 0), \end{aligned} \tag{4.2}$$

with E being the encoder and D being the decoder network. The scalar value α defines the desired minimum distance between two random images in the latent embedding. Thus the classic autoencoder loss L_{ae} trains both the encoder and decoder network to learn a reconstructable image embedding. The contrastive loss L_{contr}, on the other hand, generates only a learning signal for the encoder network and places a temporal constraint on the image embedding. The encoder and decoder consist of three convolutional networks with a kernel shape of $(3, 3)$ and a stride of $(2, 2)$, followed by a linear layer of size 20 and an l2-normalized embedding which projects the states on a unit sphere [26]. All activation functions are rectified linear units (ReLU).

4.3.2 Latent Dynamics

Using a trajectory optimization algorithm in latent space requires a dynamics function which maps a latent state $\mathbf{z}_t$ and an action $\mathbf{a}_t$ to a subsequent latent state $\mathbf{z}_{t+1}$. This allows us to unroll trajectories into the future. In the case of a single image with $\mathbf{z}_t = E(\text{Im}_t)$, we learn a dynamics mapping of $\Psi(\mathbf{z}_t, \mathbf{a}_t) = \tilde{\mathbf{z}}_{t+1}$. In the other case, when our latent state is derived from several stacked images, then we project each image into the latent space, for example by

$$\begin{bmatrix} E(\text{Im}_{t-2}) \\ E(\text{Im}_{t-1}) \\ E(\text{Im}_t) \end{bmatrix} = \begin{bmatrix} \mathbf{z}_t^{t-2} \\ \mathbf{z}_t^{t-1} \\ \mathbf{z}_t^t \end{bmatrix} = \mathbf{z}_t. \tag{4.3}$$

To predict the next latent state, the dynamics function simply has to rotate the state and only predict the third latent sub-state. This function can be described with

$$\mathbf{z}_t = \begin{bmatrix} \mathbf{z}_t^{t-2} \\ \mathbf{z}_t^{t-1} \\ \mathbf{z}_t^t \end{bmatrix} \mapsto \begin{bmatrix} \mathbf{z}_t^{t-1} \\ \mathbf{z}_t^t \\ \overline{\Psi}(\mathbf{z}_t, \mathbf{a}_t) \end{bmatrix} = \tilde{\mathbf{z}}_{t+1}, \tag{4.4}$$

where $\overline{\Psi}$ is the output of the neural network while we will use the notation $\Psi(\mathbf{z}_t, \mathbf{a}_t) = \tilde{\mathbf{z}}_{t+1}$ for the whole operation, and $\tilde{\mathbf{z}}_{i+1}$ is the predicted next latent state. The loss function for the dynamics network is then simply the difference between the predicted latent state and the actual latent state. Therefore, the loss is given as

$$L_{\text{dyn}}(\text{Im}_{t-2:t}, \mathbf{a}_t, \text{Im}_{t+1}) = \quad \|\overline{\Psi}(\mathbf{z}_t, \mathbf{a}_t) - E(\text{Im}_{t+1}) \|, \tag{4.5}$$

for each state-action-state triple $(\text{Im}_{t-2:t}, \mathbf{a}_t, \text{Im}_{t-1:t+1})$ observed during execution. The dynamics networks is constructed out of 3 fully connected layers of size 400, 400 and 20 with ReLUs as nonlinear activation functions.

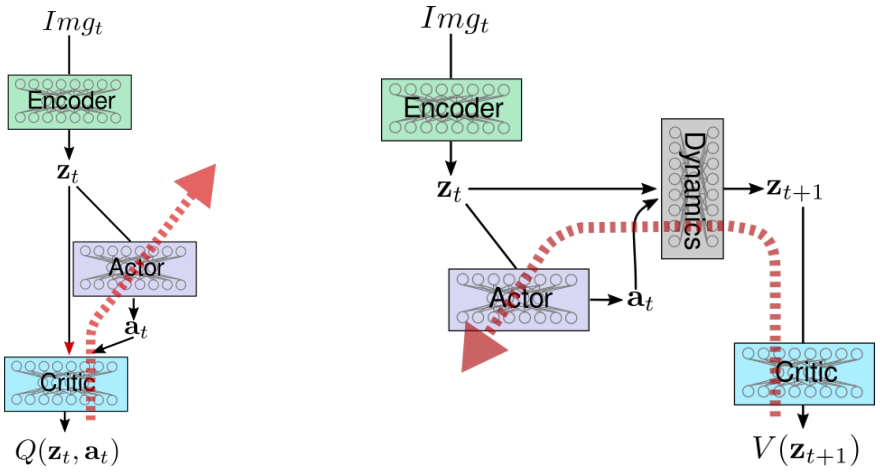
4.3.3 Deep Reinforcement Learning

We make use of the Deep Deterministic Policy Gradient (DDPG) algorithm since action and state/latent space are continuous. DDPG is based on the actor-critic model which is characterized by the idea to generate a training signal for the actor (network) from the critic (network). In turn, the critic utilizes the actor to achieve an off-policy update and models usually a Q-value function. In DDPG, the actor is a network mapping (latent) states to an action with the goal of choosing optimal actions under a reward function. Hence, the loss function for the actor is given by

$$L_{\text{actor}}(\mathbf{z}_t) = -\text{Q}(\mathbf{z}_t, \pi(\mathbf{z}_t)), \tag{4.6}$$

where only the parameters of the actor $\pi(\mathbf{z}_t)$ are optimized (see Eq. 6 in [74]). In the case of classical DDPG, the critic is a Q-function network, which maps state and action pairs to a Q-value: $Q(\mathbf{z}_t, \mathbf{a}_t) = r(\mathbf{z}_t, \mathbf{a}_t) + \gamma Q(\mathbf{z}_{t+1}, \pi(\mathbf{z}_{t+1}))$. The scalar *gamma* is a discount factor and $r(\mathbf{z}_t, \mathbf{a}_t)$ is the reward. The loss function of the critic network is based on the Bellman equation:

$$L_{\text{critic}}(\mathbf{z}_t, \mathbf{a}_t, r_{t+1}, \mathbf{z}_{t+1}) = \| \text{Q}(\mathbf{z}_t, \mathbf{a}_t) - (r_{t+1} + \gamma \text{Q}'(\mathbf{z}_{t+1}, \pi'(\mathbf{z}_{t+1}))) \|, \tag{4.7}$$



(a) Q-Value based actor update

(b) Value based actor update

Figure 4.3: The original DDPG algorithm (a) can be reformulated such that a value function (b) is used. In the case of a value function the policy gradient (red arrow) is computed via a neural dynamics function.

where Q′ and π' are target networks. For more details on DDPG we refer the interested reader to [74]. It is worth noting that DDPG can be reformulated such that the critic resembles a value function instead of a Q-value function (Fig. 4.3, see also [82]). A naive reformulation of the loss function given above is

$$L_{\text{critic}}(\mathbf{z}_t, \mathbf{a}_t, r_t, \mathbf{z}_{t+1}) = \parallel \mathrm{V}(\mathbf{z}_t) - (r_{t+1} + \gamma \mathrm{V}'(\mathbf{z}_{t+1})) \parallel, \tag{4.8}$$

given an experience $(\mathbf{z}_t, \mathbf{a}_t, r_{t+1}, \mathbf{z}_{t+1})$. But this reformulation updates only on-policy and lacks the off-policy update ability of classical DDPG. Even worse, we would fail to use such a critic to update the actor since no action gradient can be computed due to the sole dependency on the state. However, since we have access to a dynamics function we reformulate for our extension of DDPG the loss function and incorporate off-policy updates with

$$L_{\text{critic}}(\mathbf{z}_t, \mathbf{a}_t, r_t, \mathbf{z}_{t+1}) = \parallel \mathrm{V}(\mathbf{z}_t) - (r_t + \gamma \mathrm{V}'(\Psi(\mathbf{z}_t, \pi'(\mathbf{z}_t))) \parallel . \tag{4.9}$$

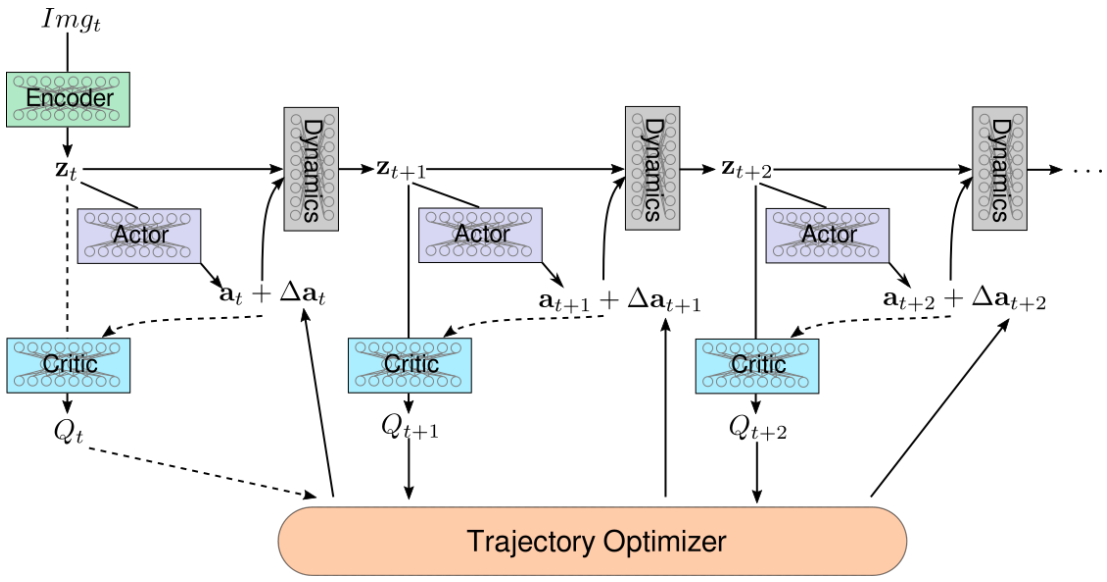


Figure 4.4: The proposed exploration strategy unrolls the trajectory in the latent space and uses the Value/Q-Value to optimize the actions of the trajectory. Dotted connections might not be used when using a Value function as critic.

This formulation allows for off-policy updates given the experience $(\mathbf{z}_t, \mathbf{a}_t, r_t, \mathbf{z}_{t+1})$, for which we assume that the reward $r(\mathbf{z})$ is only state-dependent. While this might appear to be a strong assumption at first, it holds true for most tasks in robotics. The insertion task presented in the remainder of this chapter is such a case in which the reward is fully described by the current position of both end-effector and the object to be inserted.

The loss function for the actor is then given with

$$L_{\text{actor}}(\mathbf{z}_t) = -\text{V}(\Psi(\mathbf{z}_t, \pi(\mathbf{z}_t))), \tag{4.10}$$

which is fully differentiable and, again, only used to optimize the parameters of the actor network. We use for both actor and critic two fully connected hidden layers of size 400 and 300 with ReLUs as nonlinear activation functions.

4.3.4 Optimized Exploration

Due to the deterministic nature of the actor network in DDPG and similar algorithms, the standard approach for exploration is to add random noise to actions. Random noise is usually generated from an Ornstein-Uhlenbeck process or a Gaussian distribution with fixed parameters. Such parameters, like the variance for a Gaussian distribution, are usually chosen by intuition or have to be optimized as hyper-parameter, for example with grid-search. In preliminary experiments we found Ornstein-Uhlenbeck processes with $\sigma = 0.5$ and $\theta = 0.15$ most effective on the chosen simulated task. In the presented approach we make use of the fact that we can access a dynamics function and therefore unroll trajectories throughout the latent space. The basic idea is to first unroll a trajectory using the actor network a number of steps into the future from the current point in time. We then optimize the actions $\mathbf{a}_t, \cdots, \mathbf{a}_{t+n}$ such that we maximize the Q-values/rewards along the latent trajectory. We characterize a latent trajectory, given a start state $\mathbf{z}_t = E(\mathrm{Im}_t)$, as a sequence of state-action pairs $(\mathbf{z}_t, \mathbf{a}_t, \cdots, \mathbf{z}_{t+H}, \mathbf{a}_{t+H}, \mathbf{z}_{t+H+1})$. We can then formulate a scalar function to be maximized by the trajectory optimizer based on the Q-value or reward-functions available. This process is visualized in Fig. 4.4. The Q-function in the following equations can be substituted with a learned value function. An intuitive objective function to optimize is to simply sum up all Q-values for each state-action pair of the trajectory

$$f_Q(\mathbf{a}_{t:t+H}, \mathbf{z}_t) = w_0 Q(\mathbf{z}_t, \mathbf{a}_t) + \sum_{j=1}^{H} w_j Q(\mathbf{z}_{t+j}, \mathbf{a}_{t+j}), \tag{4.11}$$

with $\mathbf{z}_{t+j} = \Psi(\mathbf{z}_{t+j-1}, \mathbf{a}_{t+j-1})$ and $\mathbf{z}_t$ being the current state from which we start unrolling the trajectory. The time-dependent weight w_i determines how much actions are going to be impacted by future states and their values and can be uniform, linearly increasing or exponential. We consider in our experiments the special case of $w_i = \frac{1}{H}$.

Alternatively, if one has access to a rewards function or learns a state-to-reward mapping simultaneously, then an objective function can be used which accumulates all rewards along the latent trajectory and adds only the final q-value:

$$f_{r+Q}(\mathbf{a}_{t:t+H}, \mathbf{z}_t) = \sum_{j=1}^{H-1} w_j r(\mathbf{z}_{t+j}) + w_H Q(\mathbf{z}_{t+H}, \mathbf{a}_{t+H}). \tag{4.12}$$

Clearly, this objective function is especially useful in the context of tasks with dense rewards. Both objective functions will be evaluated on the simulated cheetah task, which provides such dense rewards. While executing policies in the real world, we unroll a planning trajectory from the current state for n steps into the future. Then, the actions $\mathbf{a}_{t:t+H}$ are optimized under one of the introduced objectives from above with a gradient-based optimization method such as L-BFGS [83]. After a number of iterations of trajectory optimization, here 20, the first action of the trajectory, namely $\mathbf{a}_t$, is executed in the real world (Alg. 3).

4.4 Experiments

We compare in our experiments the classical approach of exploration in DDPG with an optimized Ornstein-Uhlenbeck process against the introduced approach of exploration through optimization. First, an experiment in simulation was conducted using the DeepMind Control Suite [84]. The cheetah task, in which a two-dimensional bipedal agent has to learn to walk, is especially interesting because it involves contacts with the environment that makes the dynamics hard to model. In the second experiment, we evaluate the algorithms directly on a robot and aim to solve an insertion task in the real world.

Require: Horizon H, Encoder network

for number of episodes **do**

while end of episode not reached **do**

Compute latent state $\mathbf{z}_t$ from images with encoder

Initialize action with $\mathbf{a}_t = \pi(\mathbf{z}_t)$

if training **then**

for $k = t + 1 : t + H$ **do**

Initialize action with $\mathbf{a}_k = \pi(\mathbf{z}_k)$

Predict latent state $\mathbf{z}_{k+1} = \Psi(\mathbf{z}_k, \mathbf{a}_k)$

end for

Optimize $\max_{\mathbf{a}_{t:t+H}} f(\mathbf{a}_{t:t+H}, \mathbf{z}_t)$

end if

Execute step in environment with action $\mathbf{a}_t$

Store $(\mathbf{z}_t, \mathbf{a}_t, r_t, \mathbf{z}_{t+1})$ in replay buffer

end while

Optimize dynamics network

Optimize actor network

Optimize critic network

Update target networks

end for

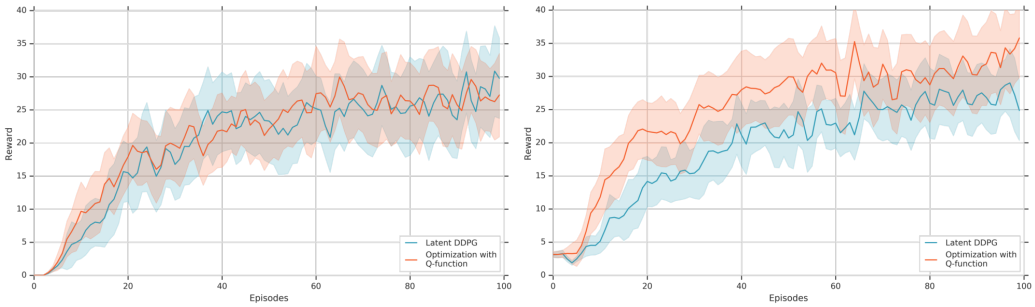
Algorithm 3: Exploration through trajectory optimization in DDPG

4.4.1 Evaluation in Simulation on the Cheetah Task

The cheetah environment of the DeepMind control suite [84] has six degrees-of-freedom in its joints and we only use camera images as state information. The actions are limited to the range of $[-1, 1]$ and camera images are of the size 320×240 px in RGB and were resized to 64×64 px. Each episode consists of 420 time steps and actions are repeated two times per time step. First, a dataset of 50 representative episodes was collected through the use of DDPG on the original state space of joint positions, joint velocities, relative body pose and body velocity of cheetah. This dataset was used to train the time-contrastive autoencoder as described above. The same parameters for the neural encoder were use for all exploration strategies. This was done to allow the sole evaluation of the exploration strategies independently of the used embedding. Since cheetah is a quite dynamic task and rewards depend on the forward velocity, this velocity must be inferable from each state. Hence, we project three subsequent images $(\mathrm{Im}_{t-2}, \mathrm{Im}_{t-1}, \mathrm{Im}_t)$ down by using the encoder network and define the current state $\mathbf{z}_t$ as the three stacked latent states $\mathbf{z}_t = [\mathbf{z}_{t-2}, \mathbf{z}_{t-1}, \mathbf{z}_t]^T$. For each of the presented evaluations 25 experiments were executed and the mean and standard deviations of the episodic cumulative rewards are shown in Figures 4.5-4.8.

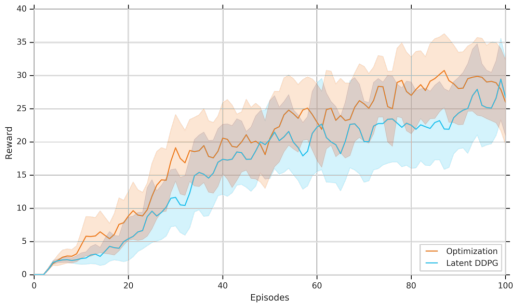
Comparison between Ornstein-Uhlenbeck and optimized exploration

As a first step we optimized the hyperparameter σ of DDPG and found that an Ornstein-Uhlenbeck process with $\sigma = 0.5$ and $\theta = 0.15$ achieve a better result for DDPG on this task than the variance of $\sigma = 0.2$ proposed in [74], especially in the early stages of the training process. A planning horizon of ten steps was used to generate the optimized noise. We make comparisons between the training process, in which we use the exploration strategies, and the test case, in which we execute the

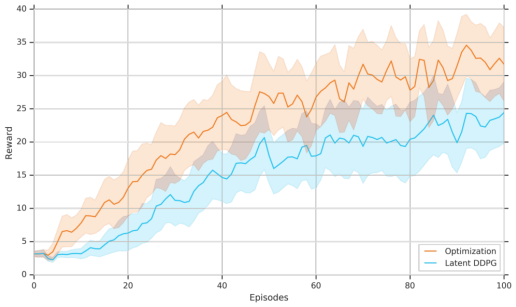


(a) Deterministic Policy (Q-Value)

(b) Exploration (Q-Value)



(c) Deterministic Policy (Value)



(d) Exploration (Value)

Figure 4.5: Comparison between DDPG using exploration with optimization (orange) and classical exploration using an Ornstein-Uhlenbeck process (blue) on the simulated cheetah task. The exploitation graph shows the evaluation of actions produced by the deterministic actor while exploration strategies are applied during training.

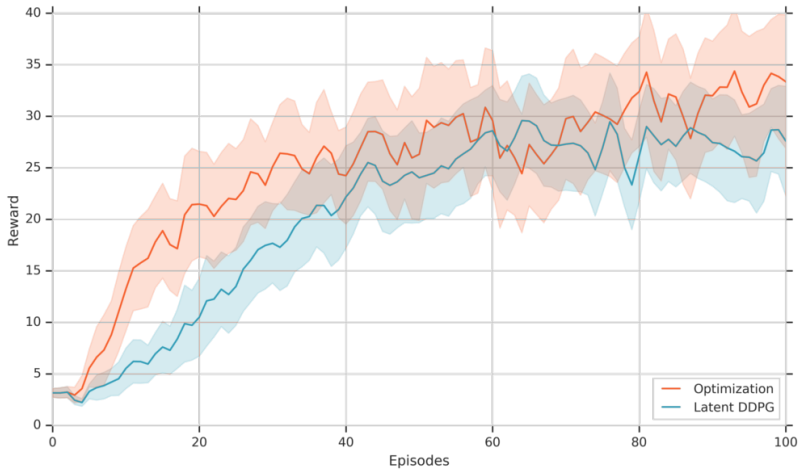


Figure 4.6: Comparison between DDPG using exploration with optimization (orange) and classical exploration using an Ornstein-Uhlenbeck process (blue) on the simulated cheetah task while using a value function as critic. The number of training iterations per episode were raised from 1000 (Fig. 4.5-d) to 3000 for this evaluation.

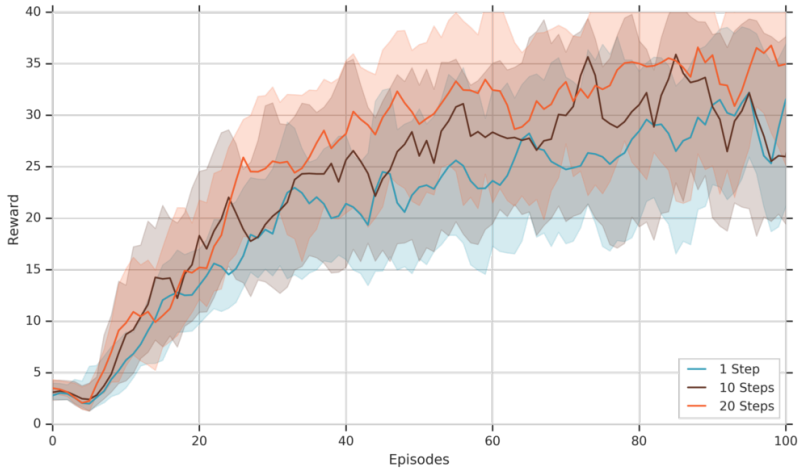


Figure 4.7: Exploration through optimization evaluated with different horizons for the planning trajectory on the simulated cheetah task.

deterministic actions produced by the actor without noise. Throughout the training process we evaluate the current policy of the actor after each episode. The results are presented in Fig. 4.5.

Comparison between different planning horizons

The main hyperparameter for optimized noise is the length of the planning horizon. If it is too short, actions are optimized greedily for immediate or apparent short-term success; if it is too long, the planning error becomes too large. Figure 4.7 shows the optimized exploration strategy with three different step-sizes: one step, ten steps and 20 steps into the future from the current state.

Comparison between different objectives

We introduced two potential objective functions, based on Q-values (Eq. 4.11) and a mix of reward- and Q-function (Eq. 4.12). We compare both of these against another objective where we only optimize for the q-value of the very last state-action pair of the unrolled trajectory (Fig. 4.8).

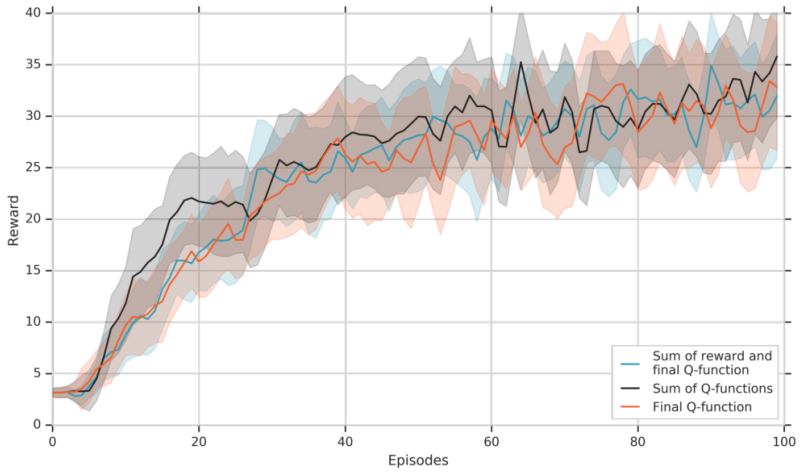
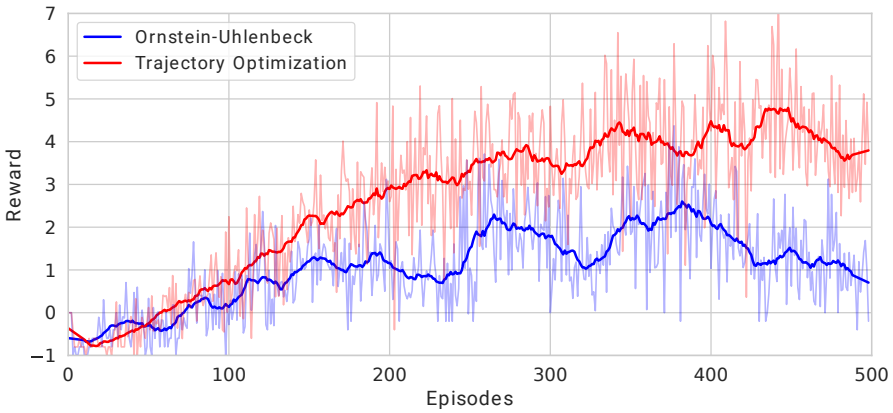


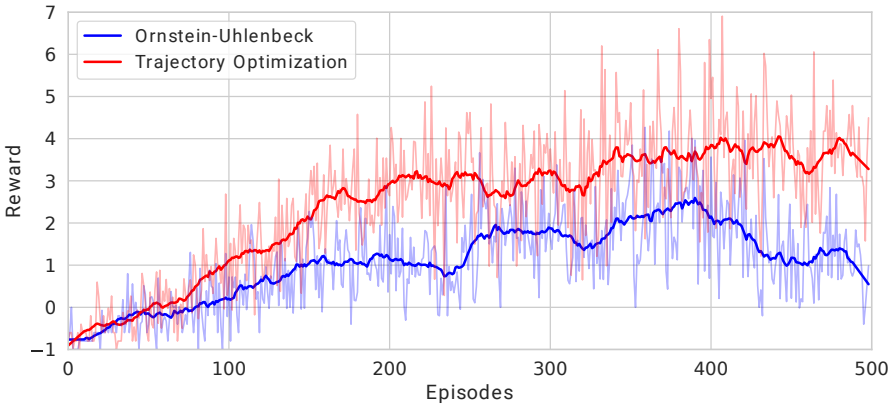
Figure 4.8: Comparison between three different objective functions for optimized exploration on the simulated cheetah task.

Table 4.1: The average success rate of insertion for policies trained by DDPG with standard Ornstein-Uhlenbeck exploration or trajectory optimization with varying planning horizons. The individual success rates for each experiment were computed over a window of 50 subsequent episodes of 500 executions total. The average success rates and standard deviations were then computed with the highest success rate achieved in each experiment. A total of five experiments were executed for each method.

Method	Avg. Success rate ($\pm std$)
Ornstein-Uhlenbeck Exploration	75.2% ($\pm$11.7%)
1 Step Planning Horizon	**93.2**% ($\pm$5.2%)
3 Steps Planning Horizon	91.6% ($\pm$**1.5**%)
5 Steps Planning Horizon	84.0% ($\pm$14.1%)
15 Steps Planning Horizon	84.4% ($\pm$9%)



(a) Deterministic Policy



(b) Exploration

Figure 4.9: Comparison between exploration with an Ornstein-Uhlenbeck (blue) and exploration through optimization (red) on the insertion task in the real world. The planning horizon is three steps. The figures show the cumulative rewards averaged over five experiments in light colours and in bold colours, for better interpretability due to the sparse reward, the mean smoothed with a Savitzky-Golay filter with window size 21 and 1st order polynomials.

4.4.2 Insertion in the real world

Fast exploration is especially important when tasks have to be solved in a real world environment and training needs to be executed on the real robot. An insertion task was set up in which a Baxter robot had to insert a cylinder into a tube where both training and testing were performed in the real world environment, without the use of simulation (Fig. 4.2). Cylinder and tube were 3D-printed. The cylinder was attached to the right end-effector of the robot with a string. The position control mode was used because there is a variable delay in the observations. Image observation were acquired from the end-effector camera of the Baxter robot via ethernet. The six dimensional actions are in the range of $[-0.05, 0.05]$ radians and represent the deviation for each joint of the arm at a point in time. This restriction ensures a strong correlation between subsequent camera images throughout the execution and allows the task to be solved in 20 steps. The initial position (radians) of the robot arm was randomized by sampling from a normal distribution with mean $\mu_{1:6} = (0.48, -1.23, -0.15, 1.42, 0.025, 1.35)$ and variances $\sigma_{1:6} = (0.05, 0.05, 0.05, 0.05, 0.05, 0.1)$, ensuring that the tube is in the image. As a simplification of the task, we excluded the last rotational wrist joint of the robot arm. Because of the adynamic nature of this task and the necessity to use position control mode it is sufficient to use the latent representation of the current image versus a stack of images as in simulation. Larger movements of the cylinder appear as blur in the images. Each episode consists of 20 time steps and a sparse reward is used: For safety reasons, if the end-effector left the designated workspace area, the episode ended and a reward of -1 is assigned. When the cylinder is inserted into the tube, the extent of insertion is transformed into a reward from $[0, 1.0]$ and an episode stops if a reward of 0.9 or higher is assigned. The state of insertion is measured with a laser-based time-of-flight sensor (VL6180). The reward for all other possible states is

zero. Five experiments were conducted on the robot: DDPG with a value function as critic and Ornstein-Uhlenbeck exploration, DDPG with exploration using trajectory optimization and a varying planning horizon (1, 3, 5 and 15 steps). We use a reduced planning horizon in this task due to the low number of time steps per episode. The comparison between Ornstein-Uhlenbeck exploration and optimized exploration with a horizon of three is shown in Fig. 4.9. Every episode which ends with a negative cumulative reward violated the workspace boundaries and episodes reaching a reward of 0.9 or more were successful insertions. Table 4.1 shows the comparison between exploration with Ornstein-Uhlenbeck noise and using planning horizons of different lengths in terms of successful insertions. Each experiment was repeated five times and the cumulative reward for each episode is used to compute the mean shown in Figure 4.9. For better interpretability, the figures show, in bold lines, additionally a smoothed version of the mean where a Savitzky-Golay filter was applied with a window size of 21 and polynomials of order one. The autoencoder network as well as the dynamics network were trained with a demonstration dataset of 50 trajectories. Of these, 19 were positive demonstrations, in which the cylinder was successfully inserted. At the beginning of each training process, 5 of these 19 trajectories were added to the replay buffer to ensure convergence of the training process due to the difficulty of the task caused by using sparse reward.

4.5 Discussion

We start with a discussion of the results from the simulated bipedal cheetah task which uses a dense reward function: The first insight is that both actors seem to perform equally well after 20 episodes, with the actor trained with optimized noise outperforming classical DDPG throughout the first 20 episodes (Fig. 4.5 (a)). However, during training the optimized exploration does not only perform better

than exploration with an Ornstein-Uhlenbeck process (Fig. 4.5 (b)) but also performs better than the actions produced by both actors during test time (Fig. 4.5 (a)).

We found that using a critic network modelling the Q-function (Fig. 4.5 (b)) outperformed the formulation of DDPG using a value network when using optimized exploration (Fig. 4.5 (d)), while DDPG with Ornstein-Uhlenbeck noise performs slightly better with a Value network (Fig. 4.5 (a,d)). One could argue, that the effects of using optimized noise could vanish when increasing the number of trainings per episode, giving DDPG more time to find an optimal actor given the current training set. Following this line of thought we increased the number of training iterations per episode three times to 3000 (Fig. 4.6). The evaluation shows that while DDPG with OU noise improves in the later stages of the learning process, the trajectory optimization uncovers valuable training experience now much faster early on. This strongly indicates that the data distribution generated by the exploration strategy has an impact on the performance of DDPG. Evaluating the step-lengths we could find that trajectory optimization improved up to a planning horizon of 20 steps, although we opted for our experiments with a conservative planning horizon of 10 steps to reduce the overall training time. The evaluation of the three introduced objective functions show that the summation of Q-values along the planning trajectory yields better performance in the early training stages, up to episode 25, for the dense reward task (Fig. 4.8). This is an interesting result given that many other trajectory optimization approaches use a Bellman-inspired sum of weighted rewards [79, 80]. It is also worth to notice that the Q-Value is the more suitable objective function for optimizing actions in the presented real-world insertion task due to the reward function being zero for the majority of time steps.

The results showing the learning progress on the insertion task in the real world draw a clearer picture of the benefit of exploration through optimization (Fig. 4.9,

Table 4.1). Generally, after roughly 50 training episodes, the networks trained with optimized exploration outperformed DDPG with OU and also achieved higher rewards in later stages of the learning process (Fig. 4.9). An evaluation of the length of the planning horizon shows, as expected, that longer planning horizons lead to a decreases performance (Table 4.1). This is very likely due to the accumulating error of predicted future states from the dynamics network. However, even with longer planning horizons the presented approach outperformed exploration using OU noise.

4.6 Conclusion

This work investigated the possibility of combining an actor-critic reinforcement learning method with a model-based trajectory optimization method for exploration. By using trajectory optimization only to gain new experience, the ability of DDPG to learn an optimal policy is not affected and we can furthermore make use of DDPG's off-policy training ability. We were able to show that by using this strategy, a performance gain can be achieved, especially in the presented real world insertion task learned from images. It is worth noting that this performance gain can be mainly attributed to the change in exploration strategy since a fixed image embedding was used, reducing the possibility of performance differences caused by using different image embeddings. This work only considered using reward, Q-Value or value functions as objective functions for optimizing the latent trajectory. In future work we plan to investigate the possibility of using additional cost terms, eg. safety and state-novelty. Furthermore, another natural next step would be to use probabilistic dynamics networks and advanced trajectory optimization algorithms to evaluate their impact on deep reinforcement learning algorithms when used for exploration in this setup.

CHAPTER 5

Data-efficient Co-Adaptation of Morphology and Behaviour with Deep Reinforcement Learning

ABSTRACT

Humans and animals are capable of quickly learning new behaviours to solve new tasks. Yet, we often forget that they also rely on a highly specialized morphology that co-adapted with motor control throughout thousands of years. Although compelling, the idea of co-adapting morphology and behaviours in robots is often unfeasible because of the long manufacturing times, and the need to re-design an appropriate controller for each morphology. In this chapter, I propose a novel approach to automatically and efficiently co-adapt a robot morphology and its controller. Our approach is based on recent advances in deep reinforcement learning, and specifically the soft actor critic algorithm. Key to our approach is the possibility of leveraging previously tested morphologies and behaviors to estimate the performance of new candidate morphologies. As such, we can make full use of the information available for making more informed decisions, with the ultimate goal of achieving a more data-efficient co-adaptation (i.e., reducing the number of morphologies and behaviors tested). Simulated experiments show that our approach requires drastically less design prototypes to find good morphology-behaviour combinations, making this method particularly suitable for future co-adaptation of robot designs in the real world.

5.1 Introduction

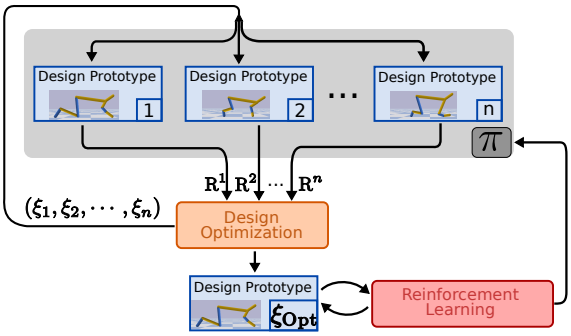
In nature, both morphology and behaviour of a species crucially shape its physical interactions with the environment [85]. For example, the diversity in animal locomotion styles is an immediate result of the interplay between different body structures, e.g., different numbers, compositions and shapes of limbs, as well as as different neuromuscular controls, e.g., different sensory-motor loops and neural periodic patterns. Adaptation of a species to new ecological opportunities often comes with changes to both body shape and control signals – *morphology and behaviour are co-adapted*. Building upon this insight, we investigate in this chapter a methodology for co-adaptation of the morphology and behaviour for computational agents using deep reinforcement learning. Without loss of generality, we focus in particular on legged locomotion. The goal of legged robots in such locomotion tasks is to transform as much electric energy as possible into directional movement [86–88]. To this end, two approaches exist: 1) optimization of the behavioural policy, and 2) optimization of the robot design, which affects the achievable locomotion efficiency [86, 89, 90]. Policy optimization is, especially in novel or changing environments, often performed using reinforcement learning [4]. Design optimization is frequently based on evolutionary algorithms or evolution-inspired and use a population of design prototypes for this process (Fig. 5.1a) [86, 89, 91]. However, manufacturing and evaluating a large quantity of design candidates is often infeasible in the real world due to cost and time constraints, especially for larger robots. Therefore, the evaluation of designs is often restricted to simulation, which is feasible but suffers from the simulation-to-reality-gap [92, 93]. Designs and control policies optimized in simulation are often not the best possible choice for the real world, especially if the robotics system is complex and the environmental parameters hard to model. For example, in the work

of Lipson and Pollack [94] designs were first optimized in simulation in an evolutionary manner and then manufactured in the real world. However, the performances of the manufactured designs in the real world were significant lower than in simulation in all but one case (see Table 1 in [94]), even though efforts were undertaken to close the simulation-to-reality gap for the described robot.

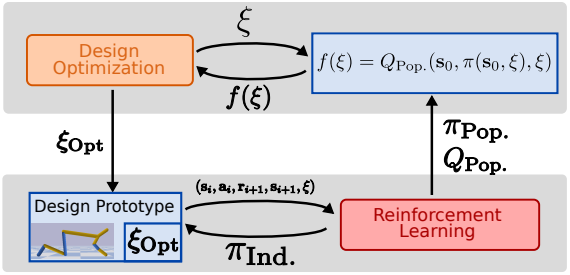
The method proposed in this work caters towards the need of roboticists for data-efficiency in respect to the number of prototypes required to achieve an optimal design. We are combining design optimization and reinforcement learning in such a way that the reinforcement learning process provides us with an objective function for the design optimization process (Fig. 5.1b). Thus, eliminating the need for a population of prototypes and requiring only one functioning prototype at a time.

5.2 Related Work

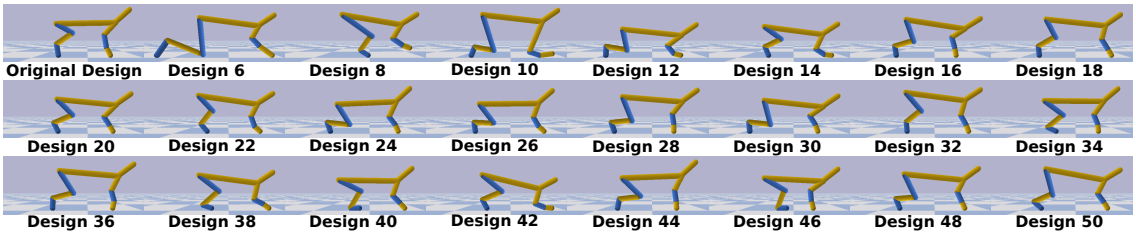
The work of Schaff et al. [90] is a relatively recent approach to combine reinforcement learning and design optimization into one framework. The common idea is to consider the design parameter ξ as an additional input to the policy $\pi(\mathrm{s}, \xi)$ and to optimize the expected reward $\mathbb{E}[R]$ given the policy and design. The policy is trained such that it is able to generalize over many designs and is iteratively updated with experience collected from a population of n prototypes. The algorithm maintains a distribution over designs, whose parameters are optimized to maximize the expected reward. However, this approach [90] requires the maintenance of a population of designs, which is updated every t timesteps and relies on the simulator to compute the fitness of designs. Similarly, the work of David Ha [1] uses the design parameters ξ as input to the policy $\pi(\mathrm{s}, \xi)$ but uses REINFORCE [95] to update the design parameters. Again, this approach requires a population of design prototypes to compute the introduced population-based policy gradient for the design as well as rewards



(a) Traditional approach in which designs are evaluated in simulation during the design optimization process.



(b) Proposed method which evaluates designs with the learned Q- and policy network and thus reduces the amount of simulations or physical prototypes required during the design optimization process.



(c) Designs ξ_{Opt} selected by the proposed method for the Half-Cheetah task.

Figure 5.1: We propose to (b) use an actor and critic for design exploration instead of (a) creating design prototypes and evaluating their performance in simulation or the real world. Our goal is to reduce the amount of time needed to (c) evolve a robotic prototype in the real world.

collected from the simulator. The recent method introduced by Liao et al. [96] employs Batch Bayesian Optimization to improve morphology and policies. The expected performance of designs is here learned and inferred by Gaussian Processes (GP), a second GP is also used to optimize the parameters of central pattern generators representing movement policies. The chapter demonstrates the design optimization of a simulated micro-robot with three parameters defining the morphology. While the presented results are using a prototype population of 5 designs, the authors mention that the proposed method can handle a single prototype as well. One drawback of [96] is, however, that the GP predicting the fitness of designs is trained only with a single value per design: the single highest reward achieved for a design. Since the

maximum reward is potentially affected by the initial state a robot is in, this approach has a reduced applicability to tasks with noisy or random start states. In [86], the leg lengths and controller of a quadruped robot were optimized in the real world. The controller was here based on the inverse kinematics of the robot and defined by tuning eight parameters. All leg segment lengths were described by a two-dimensional design vector. Two different evolutionary algorithms were used to optimize these parameters over eight generations with a population size of eight and based on the reward received. While this experiment is an impressive demonstration of the potential of adapting behaviour and morphology in the real world, the task was simplified through the use of a re-configurable robot which is able to adapt its leg-lengths automatically. This decreases the setup-time required between experiments because manufacturing of leg-segments or other body parts are not necessary. All four of these approaches rely on a population of design prototypes whose performance must be evaluated in simulation or the real world, or rely on a single reward.

5.3 Problem Statement

We formalize the problem of co-adapting morphology and behavior as the optimization

$$\theta^* = \arg\max_\theta R|_\theta \,, \tag{5.1}$$

of the reward R w.r.t. the variables $\theta = [\xi, \pi]$ were ξ are the morphological properties of the agent, and π the behavior. There are multiple ways to tackle this problem. One commonly used way is to decompose it as bi-level optimization, where we iteratively optimize the morphology first ξ, and after fixing it, we optimize the behavior π. One advantage of this formulation is that by decoupling the two optimization, we can take into consideration the fact that evaluating different morphologies has an associated

cost (e.g., manufacturing a physical robot) which can be substantially higher than evaluating different behaviors (e.g., running multiple controllers). In this chapter, we frame the learning of the behaviors as an extension of the standard Markov decision process (MDP) [97] given the additional design variable ξ (i.e., the context). In this model, the transition probability to reach a state s_t after performing action a_t is given by $p(\mathrm{s}_{t+1}|\mathrm{s}_t, \mathrm{a}, \xi)$ and depends on design properties ξ of the agent. The reward function $\mathrm{r}(\mathrm{s}, \mathrm{a}, \xi)$ can be dependent on the design as well. For notational clarity, we will generally use $r(\mathrm{s})$ in the remainder of the chapter. The actions are generated from the policy $\pi(\mathrm{s}, \xi)$ and the goal is to maximize the expected future reward given by

$$\mathbb{E}_\pi \left[\sum_{i=0}^{\infty} \gamma^i \mathrm{r}(\mathrm{s}_{t+i+1}, \mathrm{a}_{t+i+1}, \xi) \middle| \mathrm{s}_t = \mathrm{s}, \mathrm{a}_i = \pi(\mathrm{s}_i), \xi \right], \tag{5.2}$$

with $\gamma \in [0, 1]$ being a discount factor and future states s_{t+i+1} produced by the transition function. Our goal is hence to maximize this objective function for both the policy π and the design ξ using deep reinforcement learning.

5.4 Optimization of Morphology and Behaviour

We now introduce our proposed framework for sample-efficient optimization of behaviour and design for robotic prototypes. We first describe our novel objective function based on an actor and critic to remove the dependency on prototypes and simulations during design optimization. Thereafter, a method is described for fast behaviour adaptation by training a copy of actor and critic primarily on experience collected with the current design prototype. We continue with an explanation of two different design exploration mechanisms, random selection and novelty search. The chapter closes with a description of the reinforcement learning algorithms and optimization routines used.

5.4.1 *Using the Q-Function for Design Optimization*

Optimizing the behaviour of an agent usually requires learning a value or Q-value function and a policy π by the means of reinforcement learning. The rationale of our approach is to extend this methodology to the evaluation of the space of designs, thereby reducing the need for large numbers of simulations or manufactured robot prototypes.

The goal of *design optimization* is to increase the efficiency of the agent given an optimal policy for each design. The objective function for this case can be the sum of rewards collected by evaluating the behaviour of the agent with this design, given by

$$\max_{\xi} \mathbb{E}_{\pi} \left[\sum_{i=0}^{T} \mathrm{r}_t \right], \tag{5.3}$$

where the rewards are collected through the execution of a policy π on the agent with design ξ in the real world or in simulation.

To alleviate the aforementioned problems with the evaluation through executions in simulation or real world, we instead propose to reuse the Q-function learned by a deep reinforcement learning algorithm and re-formulate our objective as

$$\max_{\xi} \mathbb{E}_{\pi} \left[Q(\mathrm{s}, \mathrm{a}, \xi) | \mathrm{a} = \pi(\mathrm{s}, \xi) \right], \tag{5.4}$$

where the action a is given by the policy $\pi(\mathrm{s}, \xi)$. This creates a strong coupling between the design optimization and reinforcement learning loop: We effectively reduce the problem of finding optimal designs to the problem of training a critic which is able to generate an estimated performance of a design given state and action. This means, while optimizing a policy for a design, we also train the objective function given above at the same time. We hypothesize that, during the training process, the critic learns to distinguish and interpolate between designs due to the influence of the design on the reward of transitions. We further reformulate Eq. 5.4 to optimize over the distribution

of start states encountered in trajectories $(\mathrm{s}_0, \mathrm{a}_0, \mathrm{s}_1, \cdots, \mathrm{s}_T)$. The objective function becomes then the expected future reward given a design choice ξ. This could be, for example, the case if the leg lengths of a robot are optimized and the initial position is a standing one. Here, the initial height of the robot would vary with the design choice. Thus, we reformulate the objective function in Eq. 5.4 such that we optimize over the distribution of start states with

$$\max_{\xi} \mathbb{E}_{\mathrm{s}_0 \sim p(\mathrm{s}_0|\xi)} \left[\mathbb{E}_{\pi} \left[Q(\mathrm{s}_0, \mathrm{a}_0, \xi) | \mathrm{a}_0 = \pi(\mathrm{s}_0, \xi) \right] \right]. \tag{5.5}$$

The motivation to optimize this function over the distribution of start states is to take potential randomness in the initial positions, or even inaccuracies when resetting the initial position of a robot, into account. Since the distribution of start states might be unknown or even depend on the design, we approximate the expectation by drawing a random batch of start states s_0 from a replay buffer, which contains exclusively all start states seen so far. If we use a deterministic deep neural network for policy π, Eq. 5.5 reduces to

$$\max_{\xi} \frac{1}{n} \sum_{\mathrm{s} \in \mathrm{s}_{\mathrm{batch}}} Q(\mathrm{s}, \pi(\mathrm{s}, \xi), \xi), \tag{5.6}$$

with $\mathrm{s}_{\mathrm{batch}} = (\mathrm{s}_0^1, \mathrm{s}_0^2, \cdots, \mathrm{s}_0^n)$ containing n randomly chosen start states. This objective function can be optimized with classical global optimization methods such as Particle Swarm Optimization (PSO) [98, 99] or Covariance Matrix Adaptation - Evolution Strategy (CMA-ES) [100].

5.4.2 Design Generalization and Specialization of Actor and Critic

A naive solution to input the design variable into the actor and critic network would be to append the design vector to the state and train a single set of networks using the experience of all designs. A more promising approach is to have two sets of networks: One *population* actor and critic network which is trained on the training experience

from all designs, and *individual* networks which are initialized with the *population* network but use primarily training experience from the current design. In practice, we found it helpful to allocate 10% of the training batch for samples from the *population* replay buffer when training the *individual* networks. Essentially, this approach allows the *individual* networks $Q_{\text{Indvd.}}$ and $\pi_{\text{Indvd.}}$ to specialize in a fast manner to the current design and its nuances to quickly achieve maximum performance. In parallel, we are training the *population* networks $Q_{\text{Pop.}}$ and $\pi_{\text{Pop.}}$ with experience from all designs seen so far by selecting samples with equal probability from the *population* replay buffer $\text{Replay}_{\text{Pop.}}$. These *population* networks are then able to better generalize across different designs and provide initial weights for the *individual* networks. Hence, policies do not have to be learned from scratch for each new prototype. Instead, previously collected training data is used so that different designs can inform each other and make efficient use of all the experiences collected thus far.

5.4.3 Exploration and Exploitation of Designs

We alternate between design exploration and exploitation to increase the diversity of explored designs, improve generalization capabilities of the critic and avoid an early convergence to regions of the design space. Therefore, every time we find an optimal design during the design optimization process with the objective function (Eq. 5.6) and conclude the subsequent reinforcement learning process, we next choose one design using the exploration strategy. To this end, we implemented two different approaches: sampling new designs 1) randomly, and 2) using Novelty search [101]. Novelty search is an exploration strategy in which the objective maximizes distance to the closest neighbours. The objective function is given by

$$\max_{\xi} \frac{1}{m} \sum_{\tilde{\xi} \in \text{NN}(\xi, \bar{\Xi})} \| \xi - \tilde{\xi} \|_2, \tag{5.7}$$

where the function $\text{NN}(\xi, \bar{\Xi})$ returns the m nearest neighbors of a design ξ from the set $\bar{\Xi}$ of chosen designs so far. This set includes only designs which were selected for evaluation in the real world or simulation, i.e., were handed over to the reinforcement learning algorithm as ξ_{Opt} (Fig. 5.1b).

5.4.4 Fast Evolution through Actor-Critic Reinforcement Learning

The proposed algorithm, Fast Evolution through Actor-Critic Reinforcement Learning, is presented in Algorithm 4. We will now discuss the specifics of the used reinforcement learning algorithm and global optimization method. However, it is worth noting that our methodology is agnostic to the specific algorithms used for design and behaviour optimization.

Reinforcement Learning Algorithm While in principal every reinforcement learning method can be employed to train the Q and policy functions necessary to optimize the designs, we use a deep reinforcement learning method due to the continuous state and action domains of our tasks. Specifically, we employed the Soft-Actor-Critic (SAC) algorithm [72], a state-of-the-art deep reinforcement learning method based on the actor-critic architecture. All neural networks had three hidden layers with a layer size of 200. Per episode we train the *individual* networks $\pi_{\text{Indvd.}}$ and $Q_{\text{Indvd.}}$ 1000 times while the *population* networks $\pi_{\text{Pop.}}$ and $Q_{\text{Pop.}}$ are trained 250 times. The motivation was to assign more processing power to the *individual* networks to adapt quickly to a design and specialize. A batch size of 256 was used for each training updated.

Optimization Algorithm To optimize the objective function given in Eq. 5.6, the global optimization method Particle Swarm Optimization (PSO) [98, 99] was

Initialize replay buffers: $\text{Replay}_{\text{Pop.}}$, $\text{Replay}_{\text{Indvd.}}$ and $\text{Replay}_{\text{s}_0}$

Initialize first design ξ

for $i \in (1, 2, \cdots, M)$ **do**

$\pi_{\text{Indvd.}} = \pi_{\text{Pop.}}$

$Q_{\text{Indvd.}} = Q_{\text{Pop.}}$

Initialize and empty $\text{Replay}_{\text{Indvd.}}$

while not finished optimizing local policy **do**

Collect training experience $(\text{s}_0, \text{a}_0, \text{r}_1, \text{s}_1, \cdots, \text{s}_\text{T}, \text{r}_\text{T})$ for

current design ξ with policy network $\pi_{\text{Indvd.}}$

Add quadruples $(\text{s}_i, \text{a}_i, \text{r}_{i+1}, \text{s}_{i+1})$ to $\text{Replay}_{\text{Indvd.}}$

Add quintuples $(\text{s}_i, \text{a}_i, \text{r}_{i+1}, \text{s}_{i+1}, \xi)$ to $\text{Replay}_{\text{Pop.}}$

Add start state s_0 to $\text{Replay}_{\text{s}_0}$

Train networks $\pi_{\text{Indvd.}}$ and $Q_{\text{Indvd.}}$ with random batches from $\text{Replay}_{\text{Indvd.}}$

Train networks $\pi_{\text{Pop.}}$ and $Q_{\text{Pop.}}$ with random batches from $\text{Replay}_{\text{Pop.}}$

end while

if i is even **then**

Sample batch of start states $\text{s}_{\text{batch}} = (\text{s}_0^1, \text{s}_0^2, \cdots, \text{s}_0^n)$ from $\text{Replay}_{\text{s}_0}$

Exploitation: Compute optimal design ξ with objective function

$\max_\xi \frac{1}{n} \sum_{\text{s} \in \text{s}_{\text{batch}}} Q_{\text{Pop.}}(\text{s}, \pi_{\text{Pop.}}(\text{s}, \xi), \xi)$

else

Exploration: Sample design ξ with exploration strategy

end if

end for

Algorithm 4: Fast Evolution through Actor-Critic Reinforcement Learning (FEAR)

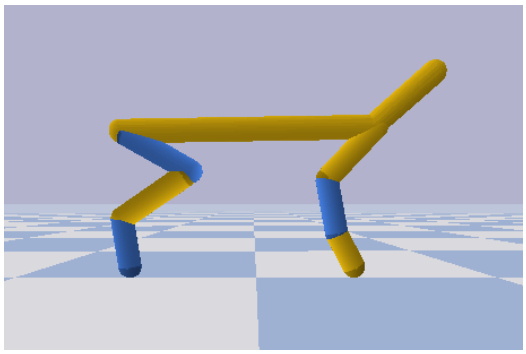
used. We chose PSO primarily because of its ability to search the design space exhaustively using a large number of particles. The objective function (Eq. 5.6) was optimized using about 700 particles, each representing a candidate design, and updated over 250 iterations. Accordingly, PSO used a total contingent of 175.000 objective function evaluations to find an optimal design. To optimize the design using rollouts in simulation, we had to reduce this number to about 1050 design candidates, i.e. 35 particles updated over 30 iterations. Although this contingent is only about 0.6% of the size of the Q-function contingent, it takes about two times longer to evaluate this number of designs in simulation. For example, on a system with an Intel Xeon CPU E5-2630 v4 CPU equipped with an NVIDIA Quadro P6000, the design optimization via simulation takes approximately 30 minutes while the optimization routine using the critic requires only 15 minutes. To put this into perspective, the reinforcement learning process on a single design requires approximately 60 minutes for 100 episodes.

5.5 Experimental Evaluation

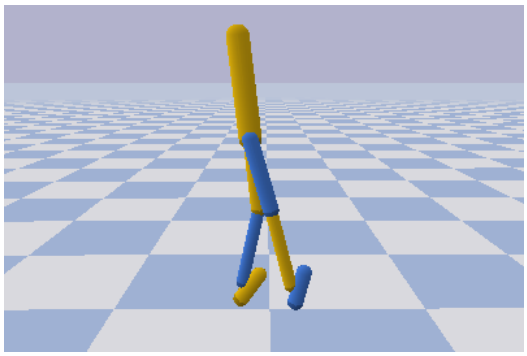
I now experimentally evaluate the proposed approach, with the aim of answering the following questions: 1) Can we obtain with our algorithm comparable task performance as optimizing the design by performing extensive trials, by instead relying on the learned model? 2) If so, how much can the proposed approach reduce the number of trials? 3) Can the proposed approach help us to get insight into the design space that we are trying to optimize for a specific task?

5.5.1 Experimental Setting

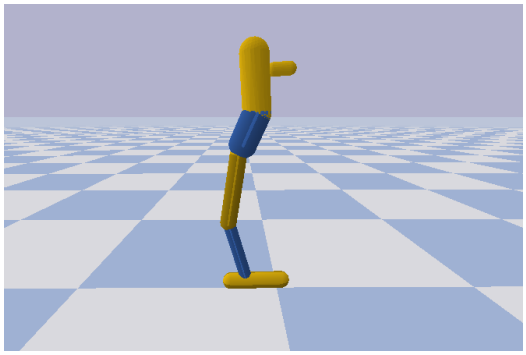
To evaluate our algorithm, we considered the four control tasks simulated using PyBullet [103] shown in Fig. 5.2. The design of agents for each task is described as a



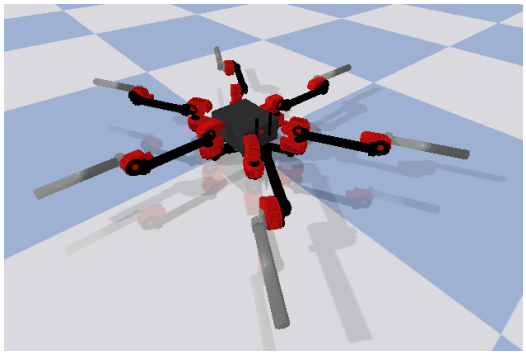
(a) Half Cheetah



(b) Walker



(c) Hopper



(d) Daisy Hexapod

Figure 5.2: An overview of locomotion tasks used for evaluation. Three are standard tasks and part of PyBullet which were adapted to allow changes in morphology. Image 5.2d shows HEBI's Daisy robot [102] for which a simulation in PyBullet was created.

continuous design vector $\xi \in \mathbb{R}^d$ The initial five designs for each task were pre-selected with the original design and four randomly chosen designs which were consistent over all experiments. All experiments were repeated five times. For the standard PyBullet tasks (Figures 5.2a to 5.2c) we executed 300 episodes for the initial five designs and 100 episodes thereafter. The latter was increased to 200 episodes for the more complex Daisy Hexapod task (Fig. 5.2d) [102]. I will give a short description of the simulated locomotion tasks and state for each task the number of states, actions and design parameters as a vector $(\mathrm{s}, \mathrm{a}, \xi)$.

Half-Cheetah (17, 6, 6) The half cheetah task has an 17 dimensional state space consisting of joint positions, joint velocities, horizontal speed, angular velocity, vertical

speed and relative height. Actions have six dimensions and are accelerations of joints. The original reward function used in PyBullet was adapted to be design independent and is given by

$$r(\mathrm{s}) = \max(\frac{\Delta x}{10}, 0), \tag{5.8}$$

where Δx is the horizontal speed to encourage forward motion. This function is inspired by the reward used in the Deepmind Control Suite. The continuous design vector is a scaling factor of the original leg lengths of Half-Cheetah:

$$(\xi_1 \cdot 0.29, \xi_2 \cdot 0.3, \xi_3 \cdot 0.188, \xi_4 \cdot 0.29, \xi_5 \cdot 0.3, \xi_6 \cdot 0.188). \tag{5.9}$$

The dimensions of the design vector are in the interval $\xi_i \in [0.8, 2.0]$.

Walker (17, 6, 6) Similar to the Half-Cheetah task, the state space of the Walker task is given by joint positions, joint velocities, horizontal speed, angular velocity, vertical speed and relative height and has 17 dimensions. The two legs of Walker are controlled through acceleration with a six dimensional action. Again, the original reward was adapted to be design agnostic. The term encouraging maximum height of the torso of walker was replaced by two terms favouring vertical orientation y_{rot} of the torso and reaching a minimal height h_{torso} of 0.8. The full reward function is given by

$$\mathrm{r(s)} = \frac{1}{10}\left((h_{\mathrm{torso}} > 0.8) \cdot (\max(\Delta x, 0) + 1) - \parallel y_{\mathrm{rot}} \parallel_2 0.1\right). \tag{5.10}$$

The design vector is a scaling factor of the leg and foot lengths of the Walker agent:

$$(\xi_1 \cdot 0.45, \xi_2 \cdot 0.5, \xi_3 \cdot 0.2, \xi_4 \cdot 0.45, \xi_5 \cdot 0.5, \xi_6 \cdot 0.2) \tag{5.11}$$

Each design dimension lies in the interval $\xi_i \in [0.5, 1.5]$.

Hopper (13, 4, 5) In the planar Hopper task a one-legged agent has to learn jumping motions in order to move forward. The state space of this task has thirteen

dimensions and four dimensions in the action space. We use the same reward function as for the Walker task with

$$\mathrm{r(s)} = \frac{1}{10}\left((h_{\mathrm{torso}} > 0.8) \cdot (\max(\Delta x, 0) + 1) - \parallel y_{\mathrm{rot}} \parallel_2 0.1\right). \tag{5.12}$$

In addition to the length of the four movable leg segments, the length of the nose-like feature of walker is an additional design parameter, here ξ_1. The full design vector is given by

$$\xi = (\xi_1 \cdot 0.7, \xi_2 \cdot 0.15, \xi_3 \cdot 0.33, \xi_4 \cdot 0.32, \xi_5 \cdot 0.25) \tag{5.13}$$

with $\xi_{2:5}$ being the length of each movable segment from pelvis to foot. The design parameters were bounded with $\xi_1 \in [0.5, 4.0]$ for the length of the nose and $\xi_{2:5} \in [0.5, 2.0]$ for all leg lengths.

Daisy Hexapod (43, 18, 9) For a preliminary study and to evaluate whether the proposed method is suitable for real world applications, a simulation of the six-legged Daisy robot by HEBI Robotics [102] was created in PyBullet. Each leg of the robot has three motors and hence the action space has 18 dimensions. The state space has 60 dimensions and consists of joint positions, joint velocities, joint accelerations, the velocity of the robot in x/y/z directions and the orientation of the robot in Euler angles. The task of the robot is to learn to walk forward while keeping its orientation and thus the reward function is given by

$$r(s) = \frac{\max(\Delta y, 0)}{0.066} - 0.25 \cdot \mathrm{diff}(e_{\mathrm{original}}, e_{\mathrm{current}}), \tag{5.14}$$

with Δy being the dislocation along the y-axis, the direction the robot faces at initialization, and $\mathrm{diff}(e_{\mathrm{original}}, e_{\mathrm{current}})$ representing the angle between the original and current orientation in quaternions. This function encourages the robot to keep its original orientation and move forward. The design vector consists of two parts: leg

lengths, and movement range of the motors at the base of the legs. All parameters are symmetric between the left and right side of the robot. The leg lengths are in $\xi_{1:6} \in [0.12, 0.5]$ for the two leg segments of each leg. Additionally, we allowed the algorithm to optimize the movement range of the first out of three motors on each leg. The base motors are restricted in movement between $(-0.35 + \xi_{7:9}, 0.35 + \xi_{7:9})$ radians with the design parameters $\xi_{7:9} \in [-0.2, 0.2]$.

5.5.2 Co-adaptation Performance

I compared the proposed framework, using actor-critic networks for design evaluation, and the classical approach, optimizing the design through candidate evaluations in simulation, on all four locomotion tasks (Fig. 5.3). We can see that, especially in the Half-Cheetah task, using actor-critic networks might perform worse over the first few designs but quickly reaches a comparable performance and even surpasses the baseline. It is hypothesized that the better performance in later episodes is due to the ability of the critic to interpolate between designs while the evaluations of designs in simulation suffers from noise during execution. Interestingly, using simulations to optimize the design does not seem to lead to much improvement in the case of the Walker task. This could be due to the randomized start state, which often leads to the agent being in an initial state of falling backwards or forwards, which would have an immediate effect on the episodic reward. Additionally, we compared the proposed method using the introduced objective function for evaluating design candidates against the method used for design optimization in [1]. Fig. 5.6 shows that the evolution strategy OpenAI-ES [104], using the simulator to evaluate design candidates with a population size of 256, is outperformed by our proposed method. Moreover, we verified that for all experiments, designs selected randomly, with a uniform distribution, performed worse than designs selected through optimization (see Fig. 5.5).

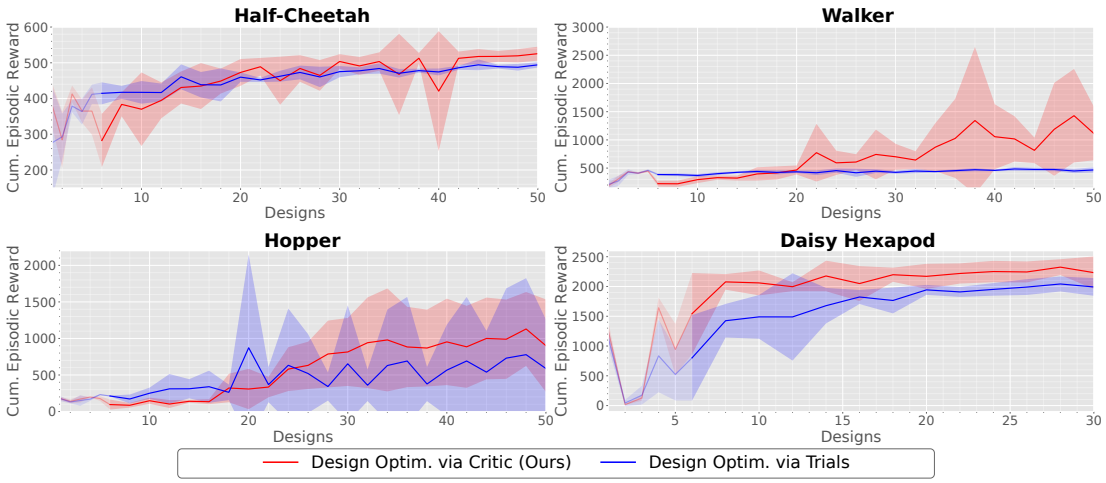


Figure 5.3: Comparison of the proposed approach (red) and using trials to evaluate the optimality of candidate designs (blue). The plots each show the mean and standard deviation of the highest reward achieved over five experiments for optimal designs ξ_{Opt}. We can see that the proposed method has a comparable or even better performance than optimization via simulation.

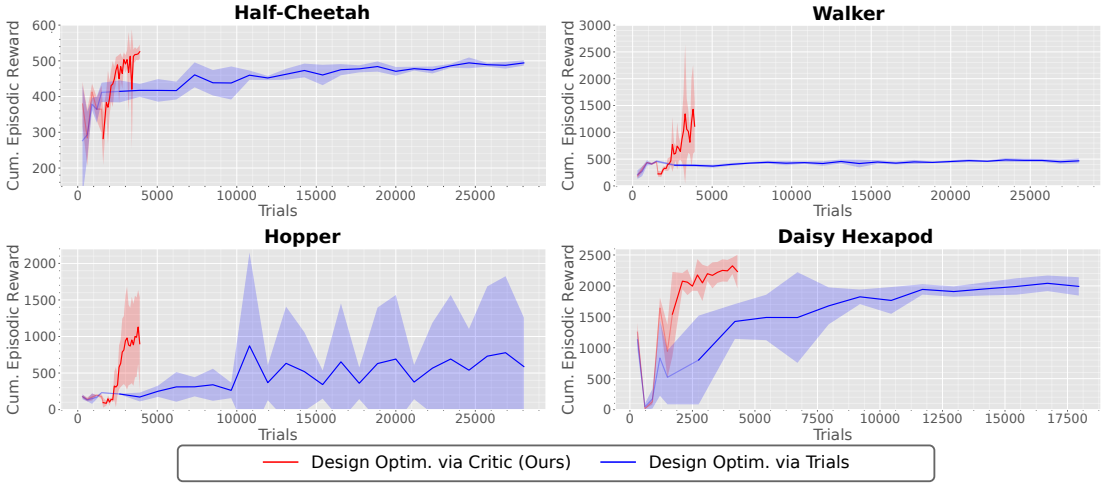
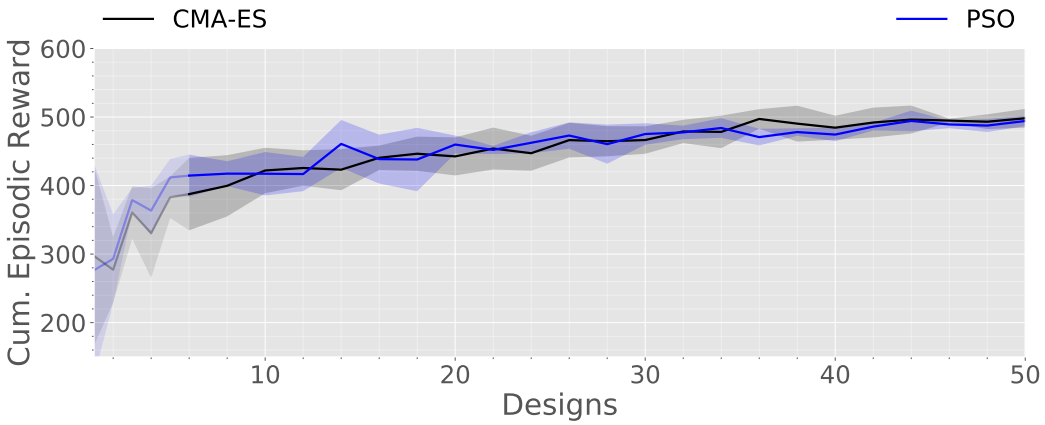
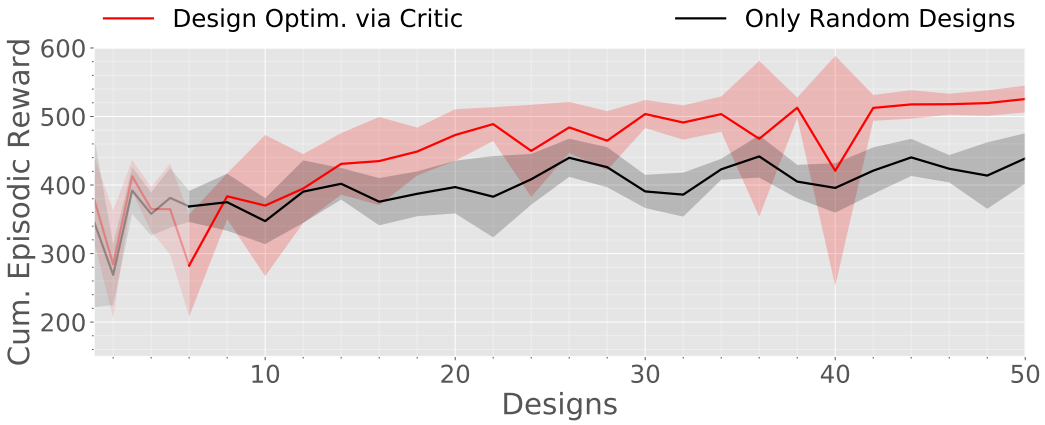


Figure 5.4: Comparison between the our proposed approach of using the actor and critic to optimize the design parameters (red) and using trials to evaluate the optimality of candidate designs (blue). The plots show the mean and standard deviation of the highest reward achieved over five experiments. The x-axis shows the number of episodes executed in simulation. We can see that removing the need to simulate design candidates leads to comparable performance in a much shorter time.



(a) Comparison of CMA-ES and PSO when using rollouts from the simulator to optimize designs.



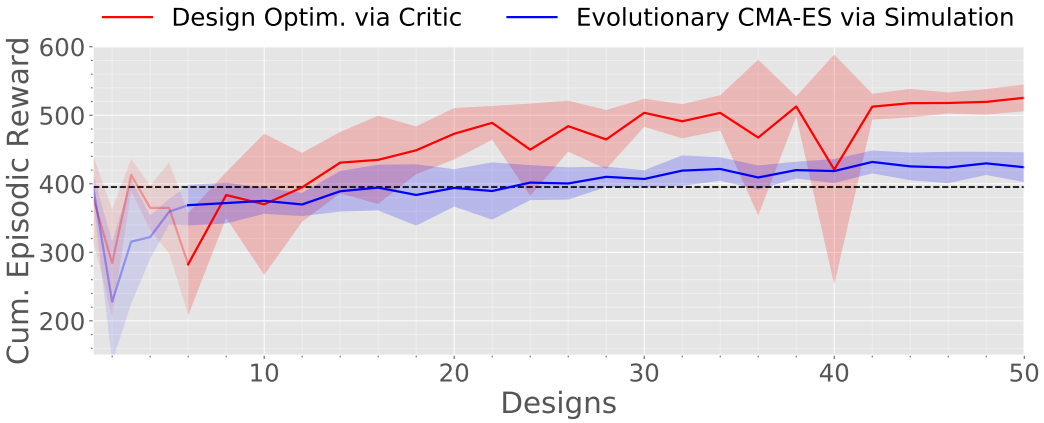
(b) Comparison of selecting designs randomly instead of optimizing the proposed objective.

Figure 5.5: (a) Evaluation on the Half-Cheetah task of two different global optimization algorithms when optimizing the design via rollouts. PSO and CMA-ES lead to the same performance for optimization. (b) Comparison of the proposed method and sampling **only** random designs instead of optimizing the objective function. The plots show the mean and standard deviation of the highest reward achieved over five experiments. The proposed approach outperformes the random baseline.

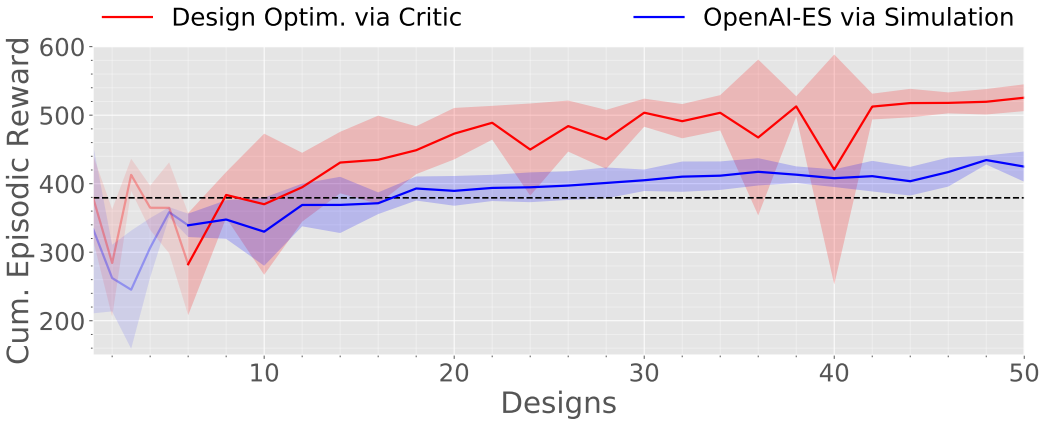
Comparison to Population-Based Approaches The proposed framework was evaluated against two previously used approaches, CMA-ES and OpenAI-ES, as proposed in [1]. In these methods, a population of design candidates is generated and evaluated in simulation. Then, a single iterative update is performed and the best design seen is trained on with reinforcement learning. This is different from the proposes method and evaluations above, where multiple iterations of PSO and CMA-ES can be performed between two learning phases. Figure 5.6 shows that the proposed algorith shows a better performance than the previously used approaches.

Efficiency of Using Population and Individual Networks As mentioned above, the proposed method uses *individual* networks which a primarily trained on experience collected from the current design and *population* networks, which are trained on all experience collected thus far. The *individual* networks are initialized from the *population* networks, and are able to adapt and specialize quickly to the current design. Additionally, the *population* networks are used in the proposed objective function. An evaluation between the proposed approach and using only a single set of *population* networks (Fig. 5.7) highlights the importance of being able to quickly adapt to the current design.

Using Batches of Start States The use of batches of start states in the proposed objective function in Eq. 5.6 is evaluated in Fig. 5.8. The figure shows that larger batches of start states lead indeed to an increase in performance. This is because values of states might depend on the design of an agent and, thus, the objective function should be evaluated over several start states from different designs. An example for this is the current height of Half-Cheetah, which is part of the state space. Using a single start state might biase the objective function towards a specific design from



(a) Comparison between the proposed method (red) and a population-based version of CMA-ES (blue) in which a population of 9 design candidates is evauated in simulation, a single iteration executed and the best design seen selected.



(b) Comparison between the proposed method (red) and the OpenAI Evolution Strategy (blue) in which a population of 256 design candidates is evauated in simulation, a single iteration executed and the best design seen selected.

Figure 5.6: Comparison between the proposed method and population-based approaches from [1] using the simulator to evaluate design candidates. Both graphs show the performance on the Half-Cheetah task with the mean and standard deviation over five experiments. The dotted line shows the best performance achieved on the standard design of Half-Cheetah.

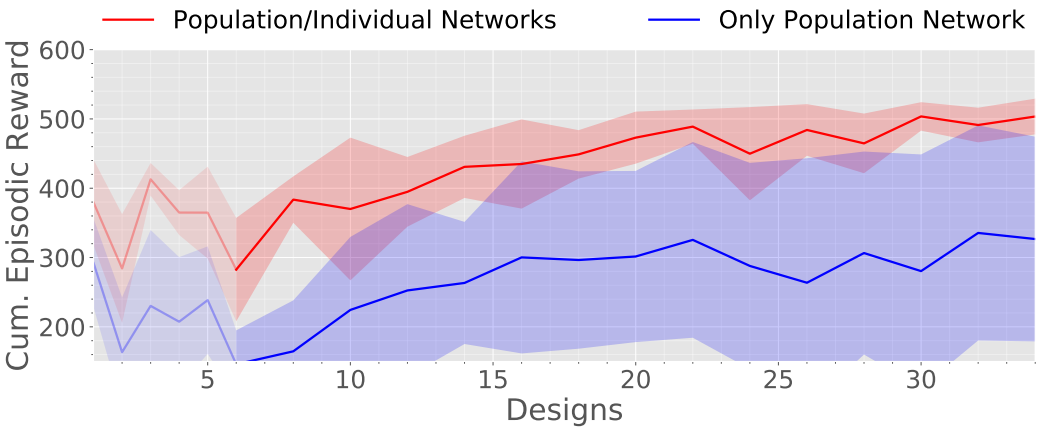


Figure 5.7: Using *individual* networks which are solely trained on experience collected from the current design in addition to *population* networks, which are trained on all experience, shows an increased performance than unsing a single set of neural networks trained on all experience (*Population*). The local networks are initialized from the *population* networks initially. Five experiments were performed on the Half-Cheetah task and the means and standard deviations are reported.

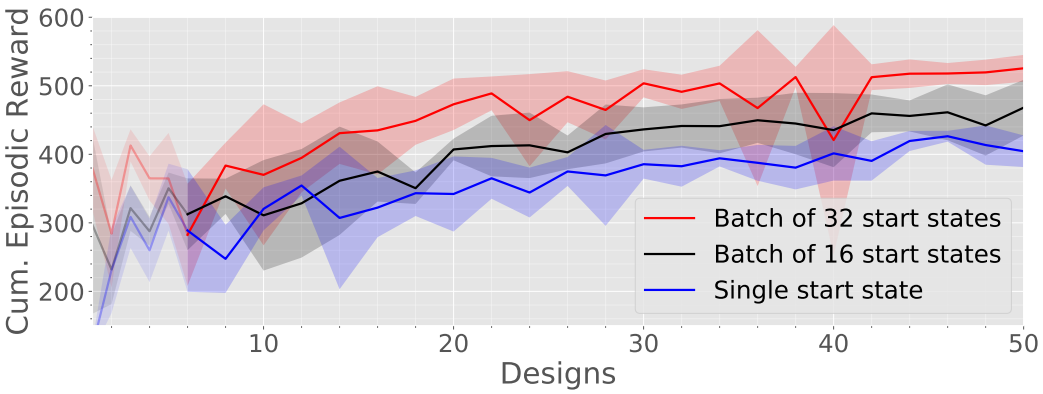


Figure 5.8: Evaluation of using batches of start states in the objective function in Eq. 5.6. Batches of 32, 16 and a single start state are compared against each other. It can be seen that the performance increases with the number of start states used.

which the state originates. This can be seen in Figure 5.8 by the poor performance when using a single start state.

Simulation Efficiency To evaluate the suitability of the proposed method for deployment in the real world, we compared the methods based on the number of simulations required. As we can see in Fig. 5.4, the actor-critic approach quickly

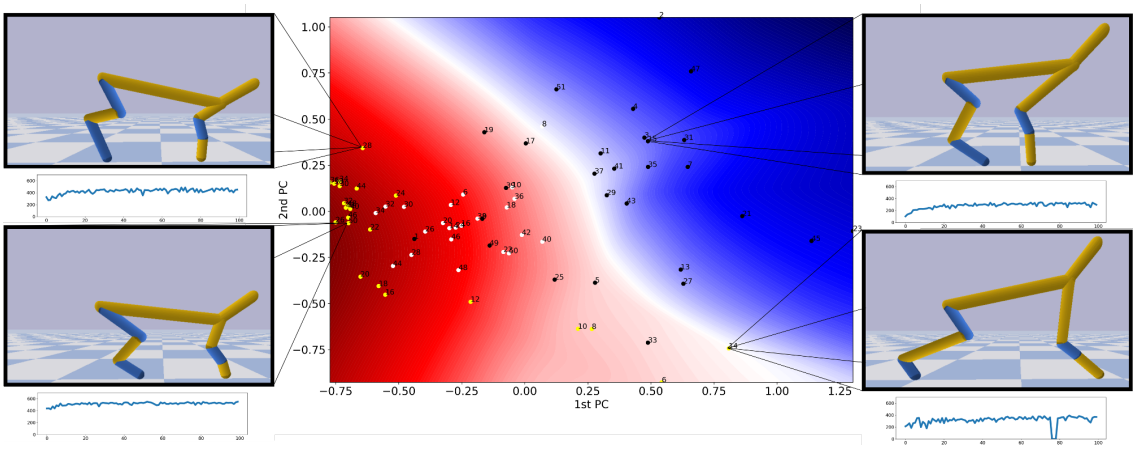


Figure 5.9: First two principal components of the six dimensional design space of Half-Cheetah as computed with PCA. Colours indicate the Q-value given by the critic on a batch of 256 start states after 50 evaluated designs, with red indicating regions of higher expected reward, and blue the regions of low expected reward. The designs chosen by our approach are depicted as yellow dots, the white dots are the designs selected when optimizing via simulation, and the black shows randomly selected design. Numbers indicate the order in which the designs were chosen for reinforcement learning.

reaches a high performance quickly with a low number of simulations. As explained above, this is due to the design optimization via simulation requiring 1050 simulations to find an optimal design while the proposed method requires none.

Visualization of Reward Landscapes for Designs A major advantage of the proposed method is the possibility to visualize the expected reward for designs. Instead of selecting a number of designs to evaluate, which would take a significant effort in the real world as well as computationally, we are able to query the introduced objective function (Eq. 5.6) in a fast manner. This allows us to visually inspect the reward landscape of designs and take a closer look at what makes designs perform better or worse. In Fig. 5.9, the first two principal components were computed based on the designs selected for learning in the Half-Cheetah task. We can see, for example, that a shorter second segment of the back leg and as well as a shorter first segment of the front leg seems to be desirable.

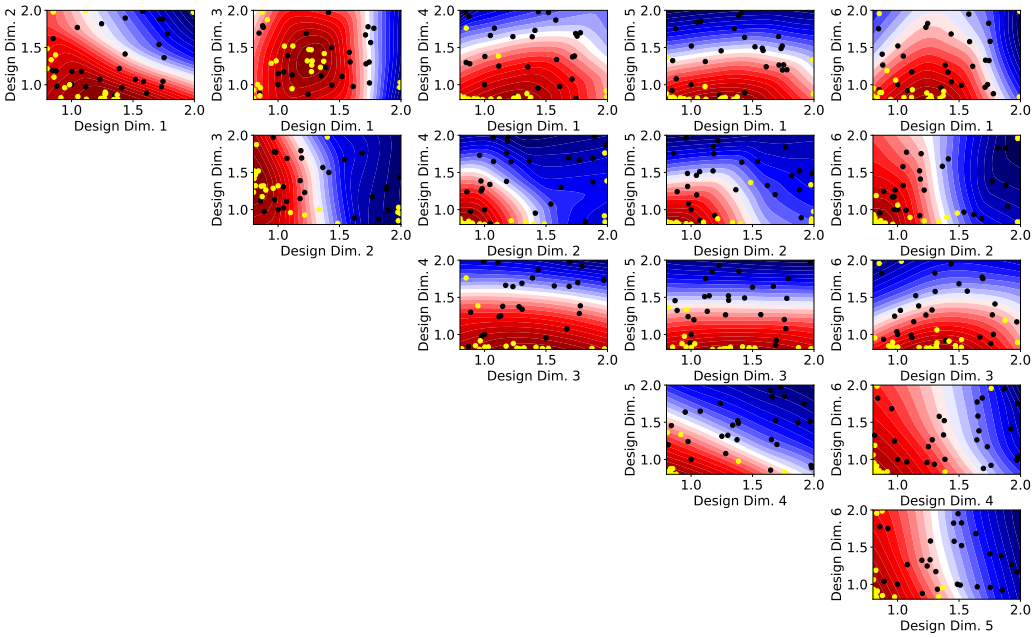


Figure 5.10: The visualized cost landscape of the design space of Half Cheetah for $\xi = (1.27, 0.87, 1.17, 0.84, 0.80, 0.82)$. A batch of 256 start states was used. The designs chosen by our approach are depicted as yellow dots, the white dots are the designs selected when optimizing via simulation, and the black shows randomly selected design.

5.5.3 Visualization of the Latent Design Space

For a better understanding of the cost landscape a low dimensional design space was computed with principal component analysis. Figures 5.10, 5.11, 5.12 shows the low-dimensional projection of the design space as well as the designs ξ_{Opt} chosen by the proposed method (yellow) and randomly selected designs for exploration (black) In white designs chosen by the optimization via simulation method are shown. We can see that the convergence rate of *optimization via simulation* appears to be slower than our method. To see what properties of the design lead to a better performance we visualized the design along the two principal components (Fig. 5.15). We can see that just longer leg do not appear to lead automatically to better performance but shorter front legs and slightly longer back legs do.

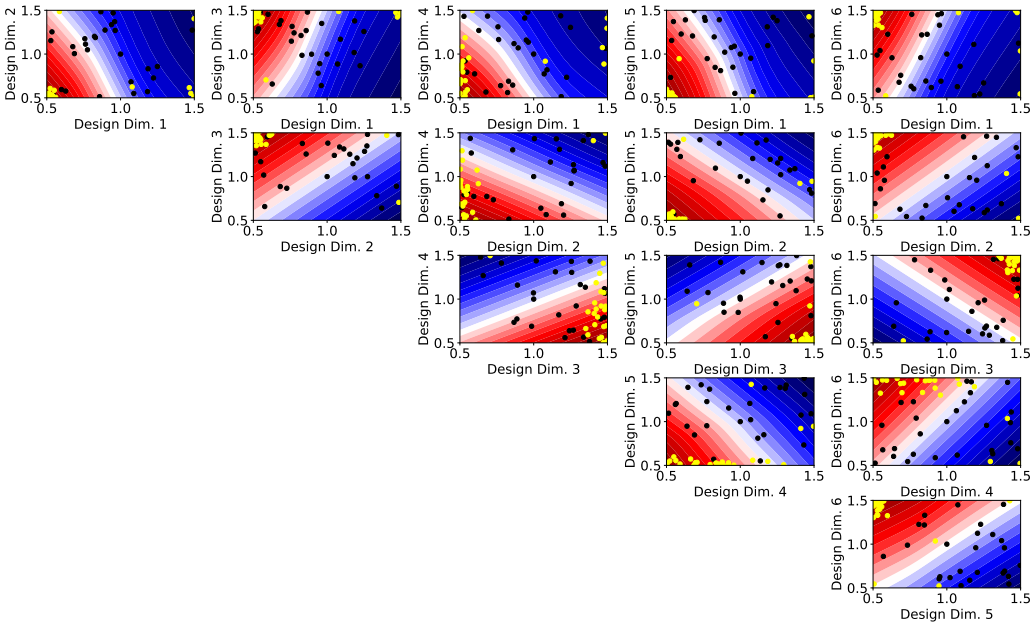


Figure 5.11: The visualized cost landscape of the design space of Walker for $\xi = (0.50, 0.52, 1.35, 0.66, 0.57, 1.49)$. A batch of 256 start states was used. The designs chosen by our approach are depicted as yellow dots, the white dots are the designs selected when optimizing via simulation, and the black shows randomly selected design.

5.5.4 Evolution of Walker

Figure 5.16 shows the evolution of designs with the proposed objective function. We can see that the start states are random and lead to different poses of Walker, sometimes falling for- or backwards. It can be seen that while shorter legs seem desirable, the larger the foot length the better the performance.

5.6 Conclusion

In this chapter, I studied the problem of data-efficiently co-adapting morphologies and behaviors of robots. My contribution is a novel algorithm, based on recent advances in deep reinforcement learning, which can better exploit previous trials to estimate the performance of morphologies and behaviors before testing them. As a result, the approach can drastically reduce the number of morphology designs tested

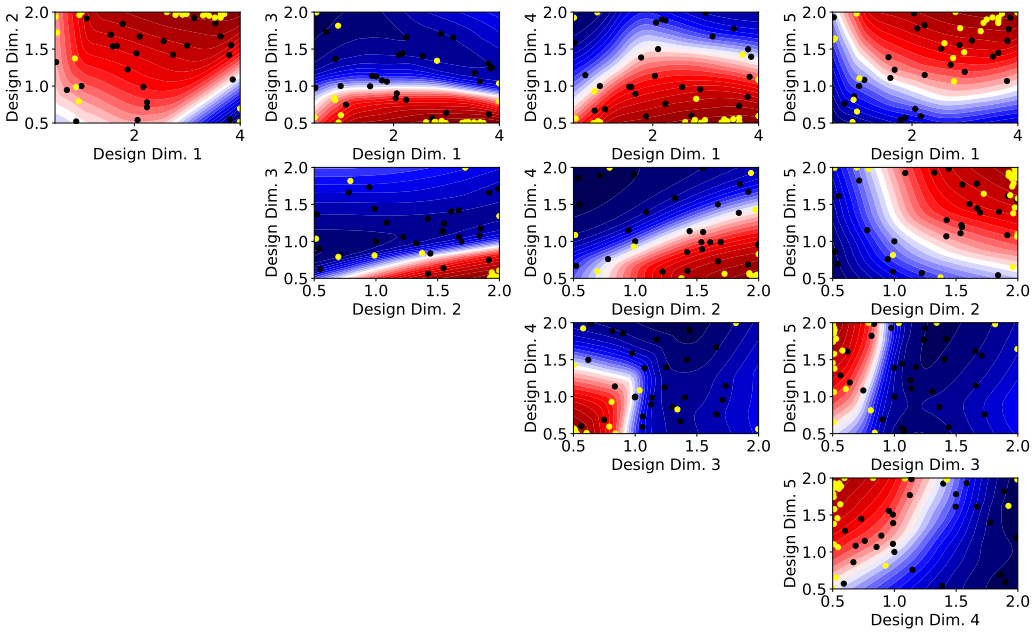


Figure 5.12: The visualized cost landscape of the design space of Hopper for $\xi = (3.41, 1.95, 0.52, 0.52, 1.90)$. A batch of 256 start states was used. The designs chosen by our approach are depicted as yellow dots, the white dots are the designs selected when optimizing via simulation, and the black shows randomly selected design.

(and their eventual manufacturing time/cost). Experimental results on 4 simulated robots show strong performance and a drastically reduced number of design prototypes, with one robot requiring merely 50 designs compared to the 24177 of the baseline – that is about 3 orders of magnitudes less data. It highlights the ability of the method to optimize the behaviour and morphology of robots and opens an exciting avenue and first stepping stone for future experiments in the real world. The unparalleled data-efficiency of our approach opens exciting venues towards the use in the real world of robots that can co-adapt both their morphologies and their behaviors to more efficiently learning to perform the desired tasks with minimal expert knowledge. In future work, I aim to demonstrate the capabilities of this algorithm on a robot in the real world.

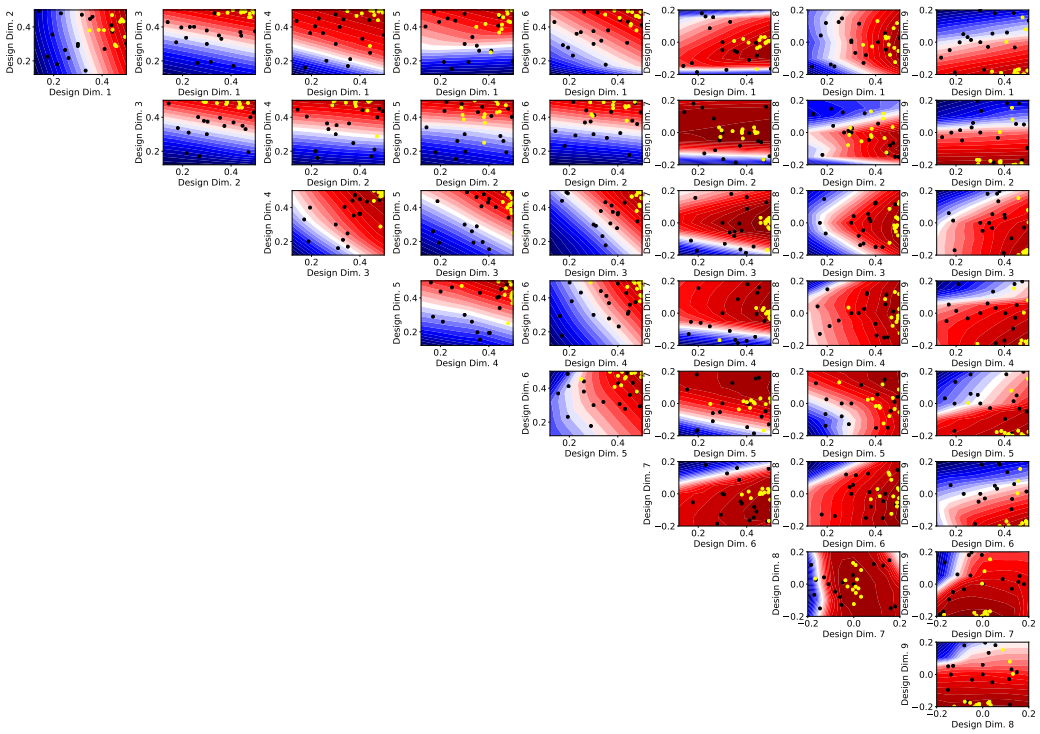
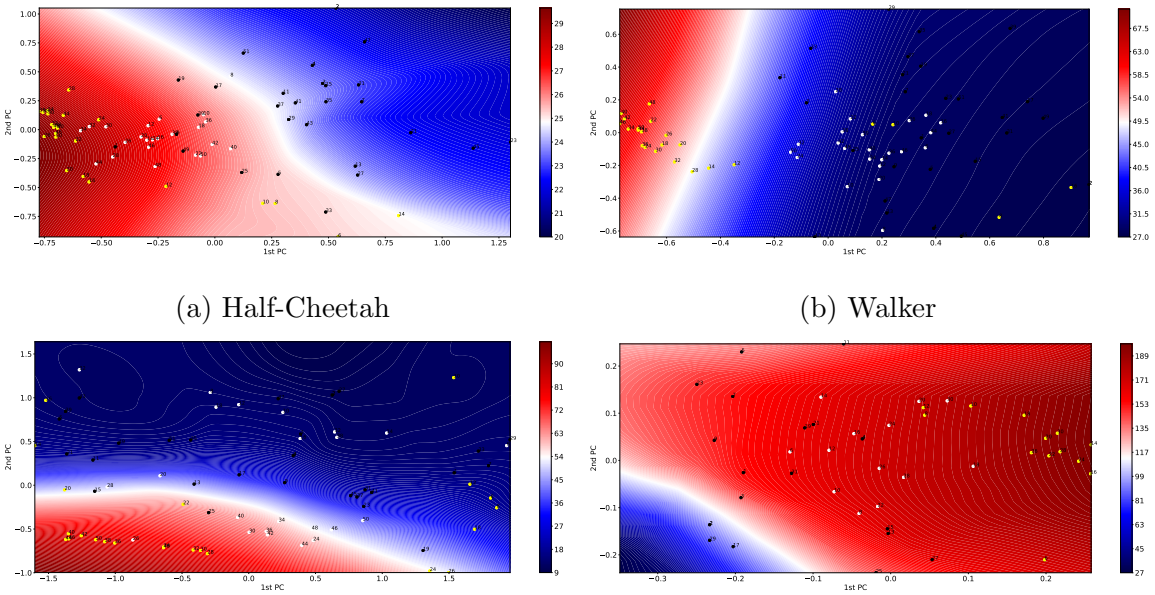


Figure 5.13: Design space of the Daisy Hexapod for $\xi = (1.27, 0.87, 1.18, 0.84, 0.80, 0.83)$. A batch of 256 start states was used. The designs chosen by our approach are depicted as yellow dots, the white dots are the designs selected when optimizing via simulation, and the black shows randomly selected design.



(a) Half-Cheetah

(b) Walker

(c) Hopper

(d) Daisy Hexapod

Figure 5.14: First two principal components of each design space as computed with PCA. Colours indicate the Q-value given by the critic on a batch of 256 start states after 50 (30 for the Daisy Hexapod) evaluated designs, with red indicating regions of higher expected reward, and blue the regions of low expected reward. The designs chosen by our approach are depicted as yellow dots, the white dots are the designs selected when optimizing via simulation, and the black shows randomly selected design. Numbers indicate the order in which the designs were chosen for reinforcement learning.

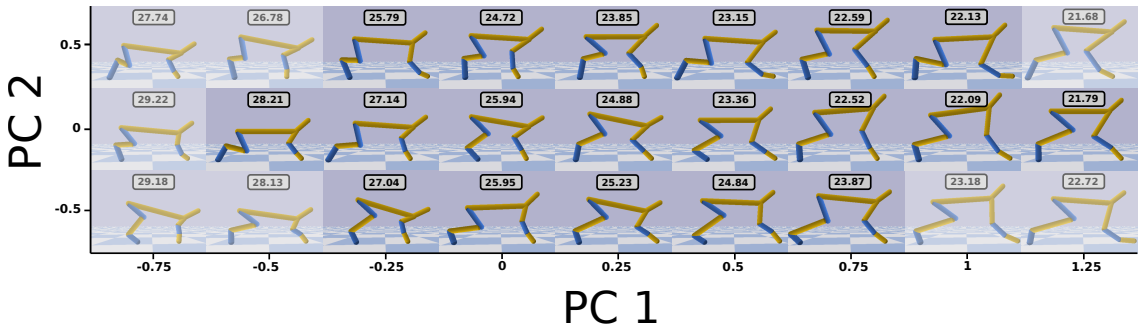


Figure 5.15: A selection of designs for Half-Cheetah, generated from the principal components in Fig. 5.14a. Designs which are outside of the bounds set for the design space are reduced in opacity. Each design is evaluated with the objective function stated in Eq. 5.6.

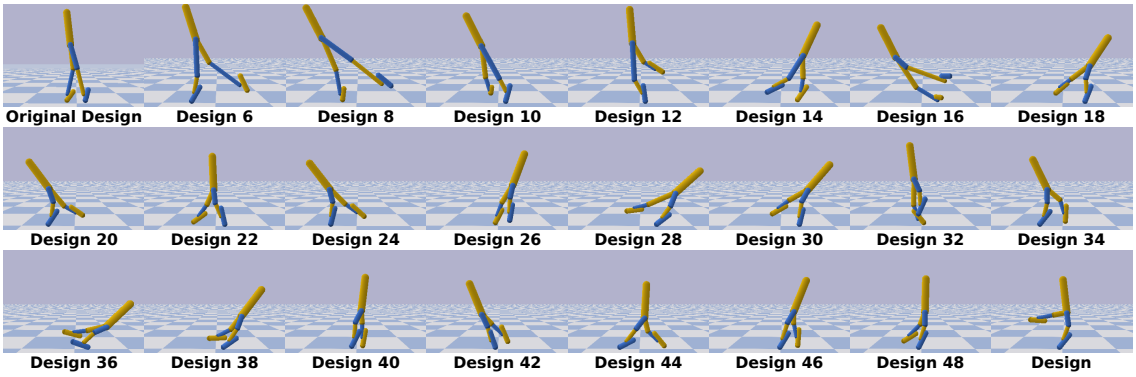


Figure 5.16: Designs ξ_{Opt} selected by the proposed method for the Half-Cheetah task.

CHAPTER 6

Future Work

6.1 Bridging the Gap from Simulation to Reality for the Co-Adaptation of Morphology and Behaviour

The question arises about how the cost of design- and behaviour adaptation can be further reduced. One possible approach is to make use of transfer learning approaches in order to transfer motor skills learned in simulation or a previous environment to the new environment the robot is currently interacting with. This section will outline a possible approach and discuss the underlying assumptions necessary for a successful transfer of skills *between environments*. This deviates slightly from the classical idea of transfer learning, namely to adapt to new tasks quickly. In this section, we will instead discuss the idea of exploiting prior knowledge about previous environments in new scenarios.

6.1.1 Assumptions Required for Successful Transfer Learning between Environments

This section describes the underlying assumptions for transfer learning between environments in the case of reinforcement learning. Assume we have two environments, let them be E_1 and E_2, each described by a quintuple $(\mathbf{S}_i, \mathbf{A}_i, \mathbf{\Xi}_i, p^i_{\text{transition}}, p^i_{\text{initial}}, r^i)$. S describes here the set of possible states and A the set of possible actions the agent can take [1] . The set of Ξ describes the set of all possible designs ξ which our agent, respectively the robot, can have in the environment. This set is is specific to the

[1] Without loss of generality and for simplicity, we assume here that every action can be taken in any state. This is usually the case in most robotic tasks.

described scenario of co-adaptation in this thesis and represents an extension of the standard reinforcement learning framework. The probability distribution $p_{\text{transition}}$ describes the transition probability of reaching a state $\mathbf{s}_{t+1}$ after being in state $\mathbf{s}_t$ with design ξ and taking the action $\mathbf{a}$, in other words $p_{\text{transition}}(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t, \xi)$ [2] . The distribution p_{initial} represents the distribution of the initial start states $\mathbf{s}_0$ and might also depend on the design: $p_{\text{initial}}(\mathbf{s}|\xi)$. A simple example for this dependency would be if the state vector encodes the current height of a robot and the design parameters vary the lengths of its legs. Generally, we will assume that these distributions are unknown but fixed in nature. The function $r : \mathbf{S} \times \mathbf{A} \to \mathbb{R}$ maps state-action tuples to a scalar values and describes the reward function. It assigns values to transitions, with higher values to desired transitions which bring us closer to our goal/solving the task.

We can make the following observations and assumptions for successful transfers of motor skills between environments requiring only minimal re-adaptation:

- The sets of states, actions and designs have to be equal between environments. Thus $\mathbf{S_0} = \mathbf{S_1}, \mathbf{A_0} = \mathbf{A_1}$ and $\mathbf{\Xi}_0 = \mathbf{\Xi}_1$.

- The inherent assumption is that the transition probability function $p_{\text{transition}}$ is not equal between environments, thus $p^1_{\text{transition}} \neq p^2_{\text{transition}}$ for two environments E_1 and E_2. That this must hold is intuitive: Assume E_1 is a simulation of environment E_2. As described in the introduction of this thesis, the simulation-to-reality gap describes the effect that simulations are, by definition, only a limited representation of the real world. For example, the modeling of contact forces are routinely simplified in simulation. Thus, discrepancies must exist between the transition functions although they might only occur in a limited

[2] This formulation makes use of the standard Markov assumption, which was introduced earlier.

number of states. However, as I stated above, we are interested in cases in which only minimal re-adaptation through reinforcement learning is required. This implies that the differences between the two transition probabilities $p^1_{\text{transition}}$ and $p^2_{\text{transition}}$ are sufficiently small. In other words, for all possible state-action-design triples $(\mathbf{s}, \mathbf{a}, \xi)$ the term $\text{KL}\left(p^1_{\text{transition}}(\mathbf{S}|\mathbf{s}, \mathbf{a}, \xi)||p^2_{\text{transition}}(\mathbf{S}|\mathbf{s}, \mathbf{a}, \xi)\right)$ should be as small as possible, with $\text{KL}(\cdot||\cdot)$ describing the Kullback-Leibler divergence. A similar statement can be made about the initial state distribution p^i_{initial}.

- As discussed above we assume that the task remains the same between environments. In order to exploit a previously trained Q-function it is proposed to ensure that the reward function of the new environment, r_2, is proportional to r_1 with a positive, constant scaling factor λ such that $r_1(\mathbf{s}, \mathbf{a}) \propto \lambda r_2(\mathbf{s}, \mathbf{a})$ for all possible state-action pairs. This assumption implies that the theoretical Q-values are proportional as well between the two environments, which means that the action maximizing $\max_{\mathbf{a}} Q_1(\mathbf{s}, \mathbf{a})$ maximizes $\max_{\mathbf{a}} Q_2(\mathbf{s}, \mathbf{a})$ as well. However, note that this implication would only hold true if $p^1_{\text{transition}} = p^2_{\text{transition}}$. By assuming $p^1_{\text{transition}} \neq p^2_{\text{transition}}$, but with only a small values in the Kullback-Leibler divergence, the action $\mathbf{a}^{(1)}$ maximizing $Q_1(\mathbf{s}, \mathbf{a}^{(1)})$ will be close to the action $\mathbf{a}^{(2)}$ maximizing $Q_2(\mathbf{s}, \mathbf{a}^{(2)})$. In other words, it is expected that $\mathbf{a}^{(2)} = \delta + \mathbf{a}^{(1)}$ holds with a sufficiently small δ for continous actions. This means, that a policy π^1 which is optimal for Q_1 will be a good intitial policy when starting the training process in environment E_2. Relaxing this assumptions will lead to a considerably decrease of sample-efficiency when training on the new environment E_2 due to the necessary corrections to the policy.

It is hypothesized that, if the assumptions given above hold, we require only a limited amount of re-adaptation between two environments, thus leading to sample-efficient transfer learning between environments.

6.1.2 Proposed Approach for Transfer Learning between two Environments

I will describe in this section how the algorithm introduced in chapter 5 can adapt quickly to a new environment E_2, when its network were previously trained on environment E_1. I will assume that the assumptions stated in the previous section hold, especially that the nature of the task remains the same. This means, we expect to see only deviations between the transition probabilities of both environments but only proportional changes in the reward functions. In this case it is sufficient to execute Algorithm 4 on environment E_1 until convergence in the design parameters is reached and transfer the two global networks Q_{Global} and π_{Global} to the new environment E_2. The global networks of Algorithm 4 are now initialized with the previously trained networks Q_{Global} and π_{Global}. It is expected that this approach would decrease the amount of design prototypes which have to be manufactured/created in environment E_2 considerably. It is hypothesized that the amount of designs/samples required to reach optimality in the new environment will be proportional with some unknown factor to the Kullback-Leibler divergence between the two transition probabilities $p^1_{\text{transition}}$ and $p^2_{\text{transition}}$.

6.1.3 Proposed Experiment for Evaluating the use of Transfer Learning for the Co-Adaptation of Morphology and Design

A set of three experiments is proposed in order to evaluate the hypothesis described in the sections above. The goal of these experiments is to learn the design and

behaviour of agents tasked with learning movement policies for running fast. The first experiment will be on a simulated and artificial agent introduced previously, namely the Half-Cheetah agent. The subsequent tasks will be on a four-legged robot, for example the ANYmal C robot, the Minitaur robot or the Spot Mini produced by Boston Dynamics. The motivation for the selection of these robots are the small contact surfaces of their legs and thus it is anticipated that the differences between the transition probabilities from different environments will be small. Thus, it is hypothesized that such a task would be ideal for evaluating the possibility to use transfer networks between environments and reducing the sample-complexity of the algorithm.

- It is proposed that the first experiment is a simulated locomotion task such as Half-Cheetah. In this experiment, Algorithm 4 is first executed in a simulation with a lateral friction coefficient of x. After the algorithm converged and produced a design adapted to the environment, the lateral friction coefficient is changed to y such that $x \neq y$. This causes the agent to either slip more or less on the ground and thus alters the transition probability slightly. Subsequently, Algorithm 4 is executed again with pre-initialized global networks for the Q- and policy networks. This experiment will help to validate the approach before moving to real-world experiments which will require considerably more time and effort than evaluations in simulation.

- The second experiment has the goal to pre-train the global networks on a simulation of a four-legged robot, for example as the *ANYmal C* robot. After finding an optimal design parameter (describing, for example, the lengths of the legs) the experiment is repeated in the real world on a concrete floor. The assumption is that the differences in the transition probabilities, caused by

different properties of the simulation ground and the concrete floor in the real world, will be sufficiently small.

- If the previous experiment succeeds and an increased sample-efficiency can be shown, the natural next step would be to vary the real world environment itself. For example, one could vary the type of ground the robot has to move on by using glass, sand, wood and different types of soil. This exchange of material is expected to cause only small differences in the Kullback-Leibler divergence between the transition probabilities.

The proposed experiments will prove to be important stepping stones for the increase of sample-efficiency of the proposed algorithm for the co-adaptation of morphology and behaviour. The sample-efficiency will be increased by transferring policies and value functions between environments in which agents have to achieve the same goal.

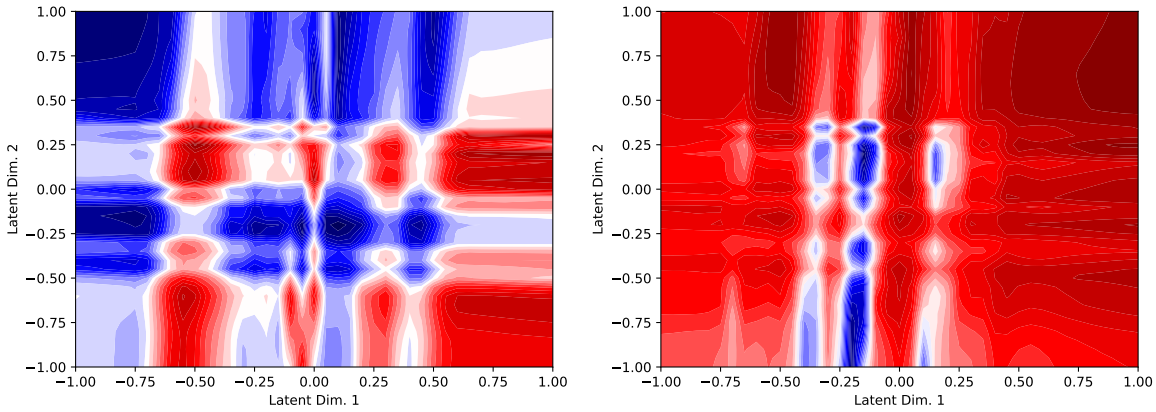
6.2 Using latent Design-dependent Action Spaces

Another potential avenue for future wok related to the co-adaptation of morphology and behaviour is to improve the sample-complexity of the reinforcement learning method used. One could exploit latent movement synergies which are design-dependent in the same vein as methodologies introduced in previous sections of this thesis. For example, one could train an autoencoder network to extract latent representations of actions given the current state $\mathbf{s}$ and design ξ. This means two neural networks are trained: An encoding network $e(\mathbf{a}, \mathbf{s}, \xi) = \mathbf{z}$ which maps an action-state-design triple to a low-dimensional latent space. The latent representation $\mathbf{z}$ can then be re-projected by the decoding network back into the original action space by $d(\mathbf{z}, \mathbf{s}, \xi) = \tilde{\mathbf{a}}$. The goal is to find encoding and decoding networks such that $\mathbf{a} \approx d(e(\mathbf{a}, \mathbf{s}, \xi), \mathbf{s}, \xi)$. In

order to find a latent space which models latent movement synergies, it is proposed to use a weighted loss using the Q-Value with

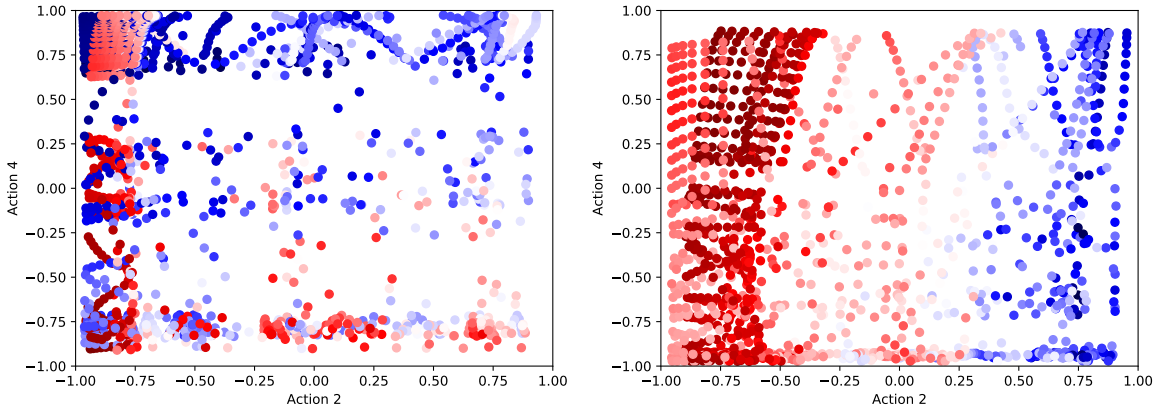
$$L = \sum_{(\mathbf{s},\mathbf{a},\xi)\in D} \| \mathbf{a} - \ d(e(\mathbf{a},\mathbf{s},\xi),\mathbf{s},\xi) \| \max(Q(\mathbf{s},\mathbf{a},\xi),0), \tag{6.1}$$

with D being the data set containing the experience collected from several designs. A latent action space uncovered in this manner can be used to improve the exploration abilities of a reinforcement learning algorithm, for example, by adding a correlated noise term to the actions with $\tilde{\mathbf{a}} = \pi(\mathbf{s}) + d(\mathbf{z},\mathbf{s},\xi)$, with the latent variable $\mathbf{z}$ being distributed with a standard normal distribution. In an initial experiment an autoencoder was trained using Eq. 6.1 to uncover a latent action space for the Half-Cheetah task introduced in chapter 5 over 50 designs. A visualization of the latent action space for two different designs ξ can be seen in Fig. 6.1. The figure shows the regions of latent actions which achieve high Q-values (red). If this latent space is projected back to the six-dimensional action space of the Half-Cheetah task the ability of the decoder to adapt the latent space to the design can be seen (Fig. 6.2). It can be seen how latent trajectories are contracted or expand for longer and shorter legs. It is hypothesized that exploiting this latent space for exploration leads to an increase in sample-efficiency for deep reinforcement learning algorithms during the design adaptation process. Evaluating this potential could proof to be a fruitful avenue.



(a) Design $\xi_i = 1$ (b) Design $\xi_i = 0.3$

Figure 6.1: Latent action space extracted with a neural autoencoder network trained on the Half-Cheetah task given a start state $\mathbf{s}_0$. Colours indicated the value of the Q-function when taking a latent action with blue being the lowest possible value and red the highest.



(a) Design $\xi_i = 1$ (b) Design $\xi_i = 0.3$

Figure 6.2: The 2-dimensional action space found by an autoencoder network sampled in the original 6-dimensional action space for an initial start state $\mathbf{s}_0$. Colours indicated the value of the Q-function when taking a latent action with blue being the lowest possible value and red the highest. The pots show action 2 and 4 of the Half-Cheetah task given two different designs and start states.

CHAPTER 7

Conclusion

"But the reason I call myself by my childhood name is to remind myself that a scientist must also be absolutely like a child. If he sees a thing, he must say that he sees it, whether it was what he thought he was going to see or not. See first, think later, then test. But always see first. Otherwise you will only see what you were expecting. Most scientists forget that."

by Douglas Adams

in **So Long, and Thanks for All the Fish**

This thesis introduced a number of methodologies for the exploitation of latent spaces in dynamical real world tasks such as locomotion and manipulation. First, the algorithm Group Factor Policy Search was introduced which uncovers latent movement synergies for improved exploration by combining Group Factor Analysis with reinforcement learning. The method was evaluated on a number of simulated and real-world tasks such as balancing and lifting (non-) deformable objects with a bimanual robot. The ability of GrouPS to adapt quickly to new environments was shown in an experiment in which a robot had to learn how to crawl over granular media both in the laboratory and the desert of Arizona. Following the idea of exploiting latent spaces for improved exploration, a methodology was introduced which extracts a latent image embedding from camera images acquired from the endeffector-camera of a robotic arm. By training a dynamics network in this latent space it became possible to use trajectory optimization methods in the latent space for solving an

insertion task. The initial trajectory would be unrolled by exploiting the current policy and subsequently optimized using optimization methods and the learned Q-function. This leads to deviations from the current policy which are optimal given the current Q-function. This method was evaluated in a real-world experiment with sparse rewards and found to outperform random noise, which is the standard approach used for exploration in (deep) policy search. Finally, it was demonstrated how we can use sample-efficient reinforcement learning algorithms for the co-adaptation of design and behaviour of robots. This new paradigm opens exciting avenues for future developments of sample-efficient algorithms and the extraction of latent movement synergies which take the current design of an agent into account. Taken together, the methods introduced in this dissertation enable robots to quickly adapt to their environments both in behaviour and morphology.

REFERENCES

[1] D. Ha, "Reinforcement learning for improving agent design," *arXiv preprint arXiv:1810.03779*, 2018.

[2] K. P. Murphy, *Machine learning: a probabilistic perspective*. MIT press, 2012.

[3] C. M. Bishop, *Pattern recognition and machine learning*. Springer, 2006.

[4] M. P. Deisenroth, G. Neumann, J. Peters, *et al.*, "A survey on policy search for robotics," *Foundations and Trends® in Robotics*, vol. 2, no. 1–2, pp. 1–142, 2013.

[5] K. C. Berridge, "Reward learning: Reinforcement, incentives, and expectations," in *Psychology of learning and motivation*, vol. 40, pp. 223–278, Elsevier, 2000.

[6] D. Golovin, B. Solnik, S. Moitra, G. Kochanski, J. Karro, and D. Sculley, "Google vizier: A service for black-box optimization," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1487–1495, ACM, 2017.

[7] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, p. 484, 2016.

[8] M. Moravčík, M. Schmid, N. Burch, V. Lisỳ, D. Morrill, N. Bard, T. Davis, K. Waugh, M. Johanson, and M. Bowling, "Deepstack: Expert-level artificial intelligence in heads-up no-limit poker," *Science*, vol. 356, no. 6337, pp. 508–513, 2017.

[9] N. Brown and T. Sandholm, "Superhuman ai for multiplayer poker," *Science*, vol. 365, no. 6456, pp. 885–890, 2019.

[10] M. Campbell, A. J. Hoane Jr, and F.-h. Hsu, "Deep blue," *Artificial intelligence*, vol. 134, no. 1-2, pp. 57–83, 2002.

[11] O. Vinyals, T. Ewalds, S. Bartunov, P. Georgiev, A. S. Vezhnevets, M. Yeo, A. Makhzani, H. Küttler, J. Agapiou, J. Schrittwieser, *et al.*, "Starcraft ii: A new challenge for reinforcement learning," *arXiv preprint arXiv:1708.04782*, 2017.

[12] O. Vinyals, I. Babuschkin, J. Chung, M. Mathieu, M. Jaderberg, W. Czarnecki, A. Dudzik, A. Huang, P. Georgiev, R. Powell, T. Ewalds, D. Horgan, M. Kroiss, I. Danihelka, J. Agapiou, J. Oh, V. Dalibard, D. Choi, L. Sifre, Y. Sulsky, S. Vezhnevets, J. Molloy, T. Cai, D. Budden, T. Paine, C. Gulcehre, Z. Wang, T. Pfaff, T. Pohlen, D. Yogatama, J. Cohen, K. McKinney, O. Smith, T. Schaul, T. Lillicrap, C. Apps, K. Kavukcuoglu, D. Hassabis, and D. Silver, "AlphaStar: Mastering the Real-Time Strategy Game StarCraft II." `https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/`, 2019.

[13] F. Li, Q. Jiang, S. Zhang, M. Wei, and R. Song, "Robot skill acquisition in assembly process using deep reinforcement learning," *Neurocomputing*, 2019.

[14] M. Večerík, T. Hester, J. Scholz, F. Wang, O. Pietquin, B. Piot, N. Heess, T. Rothörl, T. Lampe, and M. Riedmiller, "Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards," *arXiv preprint arXiv:1707.08817*, 2017.

[15] Y. Chebotar, A. Handa, V. Makoviychuk, M. Macklin, J. Issac, N. Ratliff, and D. Fox, "Closing the sim-to-real loop: Adapting simulation randomization with real world experience," *arXiv preprint arXiv:1810.05687*, 2018.

[16] J. Kober and J. Peters, "Policy search for motor primitives in robotics," *Machine Learning*, vol. 84, no. 1, pp. 171–203, 2011.

[17] A. Colome, F. Amadio, and C. Torras, "Exploiting symmetries in reinforcement learning of bimanual robotic tasks," *IEEE Robotics and Automation Letters*, 2019.

[18] T. Li, A. Rai, H. Geyer, and C. G. Atkeson, "Using deep reinforcement learning to learn high-level policies on the atrias biped," *arXiv preprint arXiv:1809.10811*, 2018.

[19] X. B. Peng, G. Berseth, K. Yin, and M. Van De Panne, "Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning," *ACM Transactions on Graphics (TOG)*, vol. 36, no. 4, p. 41, 2017.

[20] M. Zhang, X. Geng, J. Bruce, K. Caluwaerts, M. Vespignani, V. SunSpiral, P. Abbeel, and S. Levine, "Deep reinforcement learning for tensegrity robot locomotion," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 634–641, IEEE, 2017.

[21] G. Reddy, J. Wong-Ng, A. Celani, T. J. Sejnowski, and M. Vergassola, "Glider soaring via reinforcement learning in the field," *Nature*, vol. 562, no. 7726, p. 236, 2018.

[22] R. Tedrake, Z. Jackowski, R. Cory, J. W. Roberts, and W. Hoburg, "Learning to fly like a bird," in *14th International Symposium on Robotics Research. Lucerne, Switzerland*, Citeseer, 2009.

[23] A. Klami, S. Virtanen, E. Leppaaho, and S. Kaski, "Group Factor Analysis," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 26, no. 9, pp. 2136–2147, 2014.

[24] S. Virtanen, A. Klami, S. A. Khan, and S. Kaski, "Bayesian group factor analysis.," in *AISTATS*, pp. 1269–1277, 2012.

[25] H. H. Harman, *Modern factor analysis*. University of Chicago Press, 1976.

[26] P. Sermanet, C. Lynch, Y. Chebotar, J. Hsu, E. Jang, S. Schaal, S. Levine, and G. Brain, "Time-contrastive networks: Self-supervised learning from video," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1134–1141, IEEE, 2018.

[27] M. E. Tipping and C. M. Bishop, "Probabilistic principal component analysis," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 61, no. 3, pp. 611–622, 1999.

[28] M. T. Ciocarlie and P. K. Allen, "Hand posture subspaces for dexterous robotic grasping," *The International Journal of Robotics Research*, vol. 28, no. 7, pp. 851–867, 2009.

[29] R. Deimel and O. Brock, "A novel type of compliant and underactuated robotic hand for dexterous grasping," *Int. J. Rob. Res.*, vol. 35, pp. 161–185, Jan. 2016.

[30] M. Santello, M. Flanders, and J. Soechting, "Postural hand synergies for tool use," *The Journal of Neuroscience*, vol. 18, no. 23, 1998.

[31] S. Bitzer, M. Howard, and S. Vijayakumar, "Using dimensionality reduction to exploit constraints in reinforcement learning," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3219–3225, IEEE, 2010.

[32] H. P. Shum, T. Komura, T. Shiratori, and S. Takagi, "Physically-based character control in low dimensional space," in *International Conference on Motion in Games*, pp. 23–34, Springer, 2010.

[33] R. E. Bellman, *Adaptive control processes: a guided tour*, vol. 2045. Princeton university press, 2015.

[34] R. Johansson, A. Robertsson, K. Nilsson, and M. Verhaegen, "State-space system identification of robot manipulator dynamics," *Mechatronics*, vol. 10, no. 3, pp. 403–418, 2000.

[35] J. Swevers, C. Ganseman, D. B. Tukel, J. De Schutter, and H. Van Brussel, "Optimal robot excitation and identification," *IEEE transactions on robotics and automation*, vol. 13, no. 5, pp. 730–740, 1997.

[36] M. I. Jordan, Z. Ghahramani, T. S. Jaakkola, and L. K. Saul, "An introduction to variational methods for graphical models," in *Learning in graphical models*, pp. 105–161, Springer, 1998.

[37] H. Attias, "Inferring parameters and structure of latent variable models by variational bayes," in *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*, pp. 21–30, Morgan Kaufmann Publishers Inc., 1999.

[38] S. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.

[39] J. Peters, K. Mülling, J. Kober, D. Nguyen-Tuong, and O. Krömer, "Towards motor skill learning for robotics," in *Robotics Research*, pp. 469–482, Springer, 2011.

[40] N. A. Bernstein, *The co-ordination and regulation of movements*. Pergamon Press, 1967.

[41] X. Wang, N. O'Dwyer, and M. Halaki, "A review on the coordinative structure of human walking and the application of principal component analysis," *Neural Regeneration Research*, vol. 8, no. 7, pp. 662–670, 2013.

[42] G. Torres-Oviedo and L. H. Ting, "Subject-specific muscle synergies in human balance control are consistent across different biomechanical contexts," *Journal of Neurophysiology*, vol. 103, no. 6, pp. 3084–3098, 2010.

[43] J. Z. Kolter and A. Y. Ng, "Learning omnidirectional path following using dimensionality reduction," in *Proceedings of the Robotics: Science and Systems (R:SS) conference*, The MIT Press, 2007.

[44] K. S. Luck, G. Neumann, E. Berger, J. Peters, and H. B. Amor, "Latent space policy search for robotics," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1434–1440, IEEE, 2014.

[45] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum Likelihood from Incomplete Data via the EM Algorithm," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 39, no. 1, pp. 1–38, 1977.

[46] G. Neumann, "Variational inference for policy search in changing situations," in *Proceedings of the 28th International Conference on Machine Learning (ICML)*, pp. 817–824, 2011.

[47] J.-W. van de Meent, D. Tolpin, B. Paige, and F. Wood, "Black-box policy search with probabilistic programs," *arXiv preprint arXiv:1507.04635*, 2015.

[48] Y. E. Sagduyu and A. Ephremides, "The problem of medium access control in wireless sensor networks," *IEEE Wireless Communications*, vol. 11, no. 6, pp. 44–53, 2004.

[49] M. Toussaint and A. Storkey, "Probabilistic inference for solving discrete and continuous state Markov Decision Processes," in *Proceedings of the 23rd International Conference on Machine Learning (ICML)*, pp. 945–952, 2006.

[50] M. Toussaint, "Robot trajectory optimization using approximate inference," in *Proceedings of the 26th annual international conference on machine learning*, pp. 1049–1056, ACM, 2009.

[51] J. Kober and J. R. Peters, "Policy search for motor primitives in robotics," in *Advances in neural information processing systems*, pp. 849–856, 2009.

[52] J. Peters and S. Schaal, "Natural actor-critic," *Neurocomputing*, vol. 71, no. 7, pp. 1180–1190, 2008.

[53] D. Gouaillier, V. Hugel, P. Blazevic, C. Kilner, J. Monceaux, P. Lafourcade, B. Marnier, J. Serre, and B. Maisonnier, "The nao humanoid: a combination of performance and affordability," *arXiv preprint arXiv:0807.3223*, 2008.

[54] E. Rohmer, S. P. Singh, and M. Freese, "V-REP: A versatile and scalable robot simulation framework," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1321–1326, IEEE, 2013.

[55] K. S. Luck, J. Campbell, M. A. Jansen, D. M. Aukes, and H. Ben Amor, "From the lab to the desert: Fast prototyping and learning of robot locomotion," in *Robotics: Science and Systems*, 2017.

[56] K. S. Luck and H. B. Amor, "Extracting bimanual synergies with reinforcement learning," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4805–4812, IEEE, 2017.

[57] H. Ben Amor, G. Heumer, B. Jung, and A. Vitzthum, "Grasp synthesis from low-dimensional probabilistic grasp models," *Computer Animation and Virtual Worlds*, vol. 19, no. 3-4, pp. 445–454, 2008.

[58] N. Kang and J. H. Cauraugh, "Force control improvements in chronic stroke: bimanual coordination and motor synergy evidence after coupled bimanual movement training," *Experimental Brain Research*, vol. 232, no. 2, pp. 503–513, 2014.

[59] K. S. Luck, J. Pajarinen, E. Berger, V. Kyrki, and H. B. Amor, "Sparse latent space policy search," in *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[60] S. Hagio, F. M., and K. M., "Identification of muscle synergies associated with gait transition in humans," *Frontiers in Human Neuroscience*, vol. 9, no. 48, 2015.

[61] S. A. Safavynia, G. Torres-Oviedo, and L. H. Ting, "Muscle synergies: implications for clinical evaluation and rehabilitation of movement," *Topics in spinal cord injury rehabilitation*, vol. 17, no. 1, p. 16, 2011.

[62] F. Hug, N. A. Turpin, A. Guével, and S. Dorel, "Is interindividual variability of emg patterns in trained cyclists related to different muscle synergies?," *Journal of Applied Physiology*, vol. 108, no. 6, pp. 1727–1736, 2010.

[63] J. Bohg, A. Morales, T. Asfour, and D. Kragic, "Data-driven grasp synthesis-a survey," *Trans. Rob.*, vol. 30, pp. 289–309, Apr. 2014.

[64] S. Andrews and P. G. Kry, "Technical section: Goal directed multi-finger manipulation: Control policies and analysis," *Comput. Graph.*, vol. 37, pp. 830–839, Nov. 2013.

[65] H. B. Amor, E. Berger, D. Vogt, and B. Jung, "Kinesthetic bootstrapping: Teaching motor skills to humanoid robots through physical interaction," in *Annual conference on artificial intelligence*, pp. 492–499, Springer, 2009.

[66] K. C. D. Fu, F. D. Libera, and H. Ishiguro, "Extracting motor synergies from random movements for low-dimensional task-space control of musculoskeletal robots," *Bioinspiration and Biomimetics*, vol. 10, no. 5, p. 056016, 2015.

[67] E. Leppäaho, "Transfer Learning with Group Factor Analysis," Master's thesis, Aalto University, Finland, 2013.

[68] D. C. Liu and J. Nocedal, "On the limited memory bfgs method for large scale optimization," *Mathematical programming*, vol. 45, no. 1, pp. 503–528, 1989.

[69] J. Kober and J. R. Peters, "Policy search for motor primitives in robotics," in *Advances in neural information processing systems*, pp. 849–856, 2009.

[70] M. Jaritz, R. De Charette, M. Toromanoff, E. Perot, and F. Nashashibi, "End-to-end race driving with deep reinforcement learning," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2070–2075, IEEE, 2018.

[71] A. Kendall, J. Hawke, D. Janz, P. Mazur, D. Reda, J.-M. Allen, V.-D. Lam, A. Bewley, and A. Shah, "Learning to drive in a day," *arXiv preprint arXiv:1807.00412*, 2018.

[72] 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*, pp. 1856–1865, 2018.

[73] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[74] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[75] G. E. Uhlenbeck and L. S. Ornstein, "On the theory of the brownian motion," *Physical review*, vol. 36, no. 5, p. 823, 1930.

[76] H. Tang, R. Houthooft, D. Foote, A. Stooke, O. X. Chen, Y. Duan, J. Schulman, F. DeTurck, and P. Abbeel, "# exploration: A study of count-based exploration for deep reinforcement learning," in *Advances in neural information processing systems*, pp. 2753–2762, 2017.

[77] B. C. Stadie, S. Levine, and P. Abbeel, "Incentivizing exploration in reinforcement learning with deep predictive models," *arXiv preprint arXiv:1507.00814*, 2015.

[78] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell, "Curiosity-driven exploration by self-supervised prediction," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 16–17, 2017.

[79] A. Nagabandi, G. Kahn, R. S. Fearing, and S. Levine, "Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 7559–7566, IEEE, 2018.

[80] 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*, pp. 4759–4770, 2018.

[81] A. Srinivas, A. Jabri, P. Abbeel, S. Levine, and C. Finn, "Universal planning networks," in *International Conference on Machine Learning (ICML)*, 2018.

[82] N. Heess, G. Wayne, D. Silver, T. Lillicrap, T. Erez, and Y. Tassa, "Learning continuous control policies by stochastic value gradients," in *Advances in Neural Information Processing Systems*, pp. 2944–2952, 2015.

[83] C. Zhu, R. H. Byrd, P. Lu, and J. Nocedal, "Algorithm 778: L-bfgs-b: Fortran subroutines for large-scale bound-constrained optimization," *ACM Transactions on Mathematical Software (TOMS)*, vol. 23, no. 4, pp. 550–560, 1997.

[84] Y. Tassa, Y. Doron, A. Muldal, T. Erez, Y. Li, D. d. L. Casas, D. Budden, A. Abdolmaleki, J. Merel, A. Lefrancq, *et al.*, "Deepmind control suite," *arXiv preprint arXiv:1801.00690*, 2018.

[85] R. C. Bertossa, "Morphology and behaviour: functional links in development and evolution introduction," *Philosophical transactions of the Royal Society of London. Series B, Biological sciences*, vol. 366, pp. 2056–68, 07 2011.

[86] T. F. Nygaard, C. P. Martin, E. Samuelsen, J. Torresen, and K. Glette, "Real-world evolution adapts robot morphology and control to hardware limitations," in *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 125–132, ACM, 2018.

[87] S. Seok, A. Wang, M. Y. M. Chuah, D. J. Hyun, J. Lee, D. M. Otten, J. H. Lang, and S. Kim, "Design principles for energy-efficient legged locomotion and implementation on the mit cheetah robot," *Ieee/asme transactions on mechatronics*, vol. 20, no. 3, pp. 1117–1129, 2014.

[88] R. M. Alexander, "Walking and running: Legs and leg movements are subtly adapted to minimize the energy costs of locomotion," *American Scientist*, vol. 72, no. 4, pp. 348–354, 1984.

[89] K. Sims, "Evolving virtual creatures," in *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pp. 15–22, ACM, 1994.

[90] C. Schaff, D. Yunis, A. Chakrabarti, and M. R. Walter, "Jointly learning to construct and control agents using deep reinforcement learning," *arXiv preprint arXiv:1801.01432*, 2018.

[91] F. Corucci, M. Calisti, H. Hauser, and C. Laschi, "Novelty-based evolutionary design of morphing underwater robots," in *Proceedings of the 2015 annual conference on Genetic and Evolutionary Computation*, pp. 145–152, ACM, 2015.

[92] J. C. Zagal, J. Ruiz-del Solar, and P. Vallejos, "Back to reality: Crossing the reality gap in evolutionary robotics," in *IAV 2004 the 5th IFAC Symposium on Intelligent Autonomous Vehicles, Lisbon, Portugal*, 2004.

[93] S. Koos, J.-B. Mouret, and S. Doncieux, "The transferability approach: Crossing the reality gap in evolutionary robotics," *IEEE Transactions on Evolutionary Computation*, vol. 17, no. 1, pp. 122–145, 2012.

[94] H. Lipson and J. B. Pollack, "Automatic design and manufacture of robotic lifeforms," *Nature*, vol. 406, no. 6799, p. 974, 2000.

[95] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, vol. 8, no. 3-4, pp. 229–256, 1992.

[96] T. Liao, G. Wang, B. Yang, R. Lee, K. Pister, S. Levine, and R. Calandra, "Data-efficient learning of morphology and controller for a microrobot," *arXiv preprint arXiv:1905.01334*, 2019.

[97] R. Bellman, "A Markovian decision process," *Journal of Mathematics and Mechanics*, pp. 679–684, 1957.

[98] M. R. Bonyadi and Z. Michalewicz, "Particle swarm optimization for single objective continuous space problems: a review," 2017.

[99] R. Eberhart and J. Kennedy, "A new optimizer using particle swarm theory," in *MHS'95. Proceedings of the Sixth International Symposium on Micro Machine and Human Science*, pp. 39–43, Oct 1995.

[100] N. Hansen and A. Ostermeier, "Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation," in *Proceedings of IEEE international conference on evolutionary computation*, pp. 312–317, IEEE, 1996.

[101] J. Lehman and K. O. Stanley, "Exploiting open-endedness to solve problems through the search for novelty," *Artificial Life*, vol. 11, p. 329, 2008.

[102] HEBI Robotics, "Hebi robotics x-series hexapod data sheet." `http://docs.hebi.us/resources/kits/assyInstructions/A-2049-01_Data_Sheet.pdf`, 2019. Accessed: 06.07.2019.

[103] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning." `http://pybullet.org`, 2016–2019.

[104] T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever, "Evolution strategies as a scalable alternative to reinforcement learning," *arXiv preprint arXiv:1703.03864*, 2017.

CHAPTER A

COAUTHOR APPROVAL FOR CHAPTER 2

The chapter titled "Sparse Latent Space Policy Search" was published earlier – 2016 – in the Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence. The paper had five contributing authors. Kevin Sebastian Luck was the first and corresponding author. The original publication can be found at: `https://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/viewPaper/12275`

Joni Pajarinen, Erik Berger, Ville Kyrki and Heni Ben Amor have all consented for the publication to be included in this dissertation by Kevin Sebastian Luck.

CHAPTER B

COAUTHOR APPROVAL FOR CHAPTER 3

The chapter titled "Extracting Bimanual Synergies with Sample-Efficient Reinforcement Learning" was published earlier – 2017 – in the Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). The paper had two contributing authors. Kevin Sebastian Luck was the first and corresponding author. The original publication can be found at: `https://doi.org/10.1109/IROS.2017.8206356`

Heni Ben Amor has consented for the publication to be included in this dissertation by Kevin Sebastian Luck.