

# Advancing Manipulation Skill Learning Towards Sample-Efficiency and Generalization

Von der Fakultät 5: Informatik, Elektrotechnik und Informationstechnik der  
Universität Stuttgart zur Erlangung der Würde eines Doktors der  
Naturwissenschaften (Dr. rer. nat.) genehmigte Abhandlung

Vorgelegt von  
Peter Englert  
aus Erlenbach am Main

Hauptberichter: Prof. Dr. Marc Toussaint  
Mitberichter: Prof. Dr. Jan Peters

Tag der mündlichen Prüfung: 17. Oktober 2018

Institut für Parallele und Verteilte Systeme der Universität Stuttgart

2018



# Abstract

The world is incredibly versatile and complex. It consists of objects and mechanisms which have different properties and follow certain rules. Humans have the ability to master the complexity for large parts of the world by making intelligent decisions that change the world in a desired way. It is challenging to transfer such intelligent behavior to systems like robots and enable them to perform skills in complex environments. One approach to address this decision-making problem is to design behavior based on a profound mathematical and physical understanding of how certain actions affect the state of the world. However, such prior knowledge is not fully available in many domains or cannot be directly incorporated due to uncertainty in estimating the environment's state. An alternative approach to address this problem is based on learning algorithms that extract concepts from observation data. The work in this thesis combines the two approaches, prior knowledge and learning, to create intelligent manipulation skills. Robot manipulations are an important field of study since manipulations are crucial for achieving many tasks in the world. It is also a very challenging field since it is set at the intersection where robots and environment interact with each other. Robotic systems are well-studied and their states are usually observable from internal sensor data. However, the large versatility of the world makes it a harder problem to estimate the state of the environment in a comparable manner. The main goal we want to achieve is to create skills with large generalization abilities, which allow robots to apply a skill in many different situations. Besides the high generalization abilities, we also aim for sample-efficient learning, safe exploration, high performance, and a low amount of human supervision. We propose novel algorithms, which can incorporate prior knowledge about manipulations to achieve these goals. Specifically, we propose the algorithm kinematic morphing networks (KMN) that allows robots to gain an understanding of their environment from sensor data, the algorithm Inverse KKT (IKKT) that generalizes a skill by imitating human behavior, and the algorithm combined optimization and reinforcement learning (CORL) that improves a skill by exploring different interaction strategies. The proposed algorithms are evaluated based on different synthetic and real robot experiments.

# Zusammenfassung

Die Welt ist unglaublich vielseitig und komplex. Sie besteht aus Objekten und Mechanismen, die verschiedene Eigenschaften besitzen und sich nach bestimmten Regeln verhalten. Menschen besitzen die Fähigkeit, diese Komplexität für weite Teile der Welt zu meistern und sind in der Lage, intelligente Entscheidungen zu treffen, um ihre Umgebung auf gewünschte Weise zu verändern. Es ist eine große Herausforderung, solch ein intelligentes Verhalten auf Roboter zu transferieren, um es ihnen zu ermöglichen, Fertigkeiten in komplexen Umgebungen durchzuführen. Ein Ansatz, um dieses Problem der Entscheidungsfindung anzugehen, ist es Verhalten basierend auf mathematischem und physikalischem Wissen zu erstellen. Allerdings ist ein solches Verständnis in vielen Bereichen nicht vollständig verfügbar oder lässt sich nicht direkt verwenden (z.B. aufgrund von Messunsicherheit). Ein alternativer Ansatz, dieses Problem zu lösen, basiert auf Lernalgorithmen, welche das Ziel verfolgen, nützliche Konzepte aus Daten zu extrahieren. Die vorliegende Doktorarbeit kombiniert diese beiden Ansätze miteinander, um Roboter mit intelligenten Manipulationsfertigkeiten auszustatten. Robotermanipulationen sind ein sehr komplexes Gebiet, da es an der Schnittstelle stattfindet, wo Roboter und Umgebung miteinander interagieren. Das vorhandene Wissen über Robotersysteme ist schon recht ausgereift und ihr Zustand lässt sich in der Regel mit internen Sensoren beobachten. Allerdings ist es um einiges schwieriger, den Zustand der Umgebung auf ähnliche Weise zu beobachten, da die Welt unglaublich vielseitig und Sensorsignale in der Regel sehr abstrakt sind. Das Hauptziel, welches wir erreichen wollen, ist dass Roboter Fertigkeiten mit hoher Generalisierbarkeit erlernen, so dass eine Fertigkeit auf vielen unterschiedlichen Situationsumgebungen anwendbar ist. Weitere Ziele sind ein dateneffizientes Lernen, eine sichere Exploration, eine hohe Performanz und ein möglichst autonomes Lernen. Es werden originelle Algorithmen präsentiert, die Vorwissen über Manipulationen verwenden, um diese Ziele zu erreichen. Wir schlagen den Algorithmus KMN (Kinematic Morphing Network) vor, der es Robotern erlaubt, aus Sensordaten ein Verständnis über ihre Umgebung zu erlangen. Des Weiteren präsentieren wir den Algorithmus IKKT (Inverse KKT), welcher eine Fertigkeit durch das Imitieren von menschlichem Verhalten generalisiert. Ein weiterer vorgestellter Algorithmus ist CORL (Combined Optimization and Reinforcement Learning), welcher die Fertigkeit eines Roboters durch das Explorieren verschiedener Interaktionsstrategien verbessert. Die vorgestellten Algorithmen werden basierend auf Experimenten mit simulierten und echten Robotern evaluiert.





# Acknowledgements

I would like to express my thankfulness to several people who have made this thesis possible.

First of all, I thank my advisor Marc Toussaint for his continuous support and supervision that he has given me over the years. I also would like to thank Jan Peters for agreeing to evaluate this thesis and for encouraging me to pursue a Ph.D. in robot learning.

It has been an immense pleasure to be a part of the Machine Learning and Robotics Lab for the last five years. I would like to thank my current and former colleagues: Andrea, Carola, Stefan, Robert, Johannes, Stanimir, Marion, Vien, Nathan, Dmitry, Ingo, Jens, Erwan, Robin, Hung, Tatsiana, Matt, Danny, Sabrina, Ruth, Duy, Janik, Camille, Jim, Daniel, Philipp, Heiko, and Yoojin.

A big thank goes to Ruth, Sabrina, Danny, and Janik for proofreading this thesis. I also want to thank the staff at IPVS for their organizational support and the German Science Foundation (DFG) for their financial support.

Finally, I wish to thank my family and friends for their continuous support and encouragement throughout my life.



# Contents

<b>1</b>	<b>Introduction</b>	<b>17</b>
1.1	Motivation for Manipulation Skill Learning . . . . .	18
1.2	Advancing Manipulation Skill Learning . . . . .	19
1.3	Contributions of this Thesis . . . . .	20
1.4	Overview of Thesis Structure . . . . .	23
<b>2</b>	<b>Background</b>	<b>27</b>
2.1	Motion Representation . . . . .	27
2.2	Gaussian Processes . . . . .	29
2.3	Bayesian Optimization . . . . .	30
2.4	Convolutional Neural Networks . . . . .	32
<b>3</b>	<b>Related Work</b>	<b>36</b>
3.1	Reinforcement Learning . . . . .	36
3.2	Inverse Optimal Control . . . . .	40
3.3	Environment Model Learning . . . . .	46
<b>4</b>	<b>Inverse KKT</b>	<b>51</b>
4.1	Inverse KKT Objective . . . . .	51
4.2	Regularization & Sparsity . . . . .	54
4.3	Weight Parametrizations . . . . .	55
4.4	Nonparametric Inverse KKT . . . . .	56
<b>5</b>	<b>Combined Optimization and Reinforcement Learning</b>	<b>60</b>
5.1	Optimal Control & Reinforcement Learning . . . . .	60
5.2	Problem Formulation . . . . .	61
5.3	Combining Motion Optimization and Reinforcement Learning . . . . .	63
<b>6</b>	<b>Prior Knowledge for Manipulation Skill Learning</b>	<b>69</b>
6.1	Projection Constraints for Sample-Efficient Manipulation Skill Learning	69
6.2	Cost Features for Representing Generalizable Manipulation Skills . . .	70
6.3	Learning Skills from a Single Demonstration . . . . .	73

<i>CONTENTS</i>	9
<b>7 Kinematic Morphing Networks</b>	<b>76</b>
7.1 Skill Transfer with Kinematic Morphing Networks . . . . .	76
7.2 Kinematic Model of the Manipulated Environment . . . . .	79
7.3 Kinematic Morphing Networks . . . . .	80
<b>8 Experiments</b>	<b>86</b>
8.1 Evaluation of Inverse KKT . . . . .	87
8.2 Evaluation of CORL . . . . .	97
8.3 Learning a Manipulation Skill from a Single Demonstration . . . . .	107
8.4 Evaluation of Kinematic Morphing Networks . . . . .	112
<b>9 Conclusion</b>	<b>121</b>
9.1 Summary . . . . .	121
9.2 Outlook . . . . .	123
<b>List of Symbols</b>	<b>130</b>
<b>Bibliography</b>	<b>131</b>



# Acronyms

<b>CIOC</b>	continuous inverse optimal control.
<b>CMA-ES</b>	covariance matrix adaptation evolution strategy.
<b>CNN</b>	convolutional neural network.
<b>CORL</b>	combined optimization and reinforcement learning.
<b>DMP</b>	dynamic movement primitive.
<b>DOF</b>	degrees of freedom.
<b>GP</b>	Gaussian process.
<b>ICP</b>	iterative closest point.
<b>IFR</b>	International Federation of Robotics.
<b>IKKT</b>	inverse Karush-Kuhn-Tucker.
<b>IOC</b>	inverse optimal control.
<b>KMN</b>	kinematic morphing network.
<b>KOMO</b>	k-order Markov optimization.
<b>PI</b>	probability of improvement.
<b>PI<sup>2</sup></b>	policy improvement with path integrals.
<b>PIBU</b>	probability of improvement + boundary uncertainty.
<b>PoWER</b>	policy learning by weighting exploration with the returns.
<b>QP</b>	quadratic program.
<b>ReLU</b>	rectified linear unit.
<b>RKHS</b>	reproducing kernel Hilbert space.
<b>RL</b>	reinforcement learning.
<b>SAL</b>	safe active learning.
<b>UCB</b>	upper confidence bound.

# List of Figures

1.1	Various manipulation domains used in the experiments . . . . .	24
2.1	Robot state and environment . . . . .	28
2.2	Example of Bayesian optimization . . . . .	31
2.3	Fully connected neural network . . . . .	33
2.4	Neural network convolution and max pooling operation . . . . .	34
4.1	Concept of skill learning from demonstration with IKKT and KOMO . . .	52
5.1	Visualization of a two-dimensional projection constraint . . . . .	62
5.2	Example of PIBU acquisition function. . . . .	65
6.1	Overview of cost and constraint features . . . . .	72
6.2	Concept of learning skills from a single demonstration . . . . .	73
7.1	Sketch of a kinematic door model . . . . .	77
7.2	Multi-step network predictions . . . . .	81
7.3	Network architecture . . . . .	84
8.1	Inverse KKT: 2D motion planning task — Problem visualization . . . . .	87
8.2	Inverse KKT: 2D motion planning task — Evaluation . . . . .	89
8.3	Inverse KKT: Weight parametrizations — Learned weights . . . . .	90
8.4	Inverse KKT: Noisy demonstrations — Evaluation . . . . .	91
8.5	Inverse KKT: Box sliding — Motion sequence . . . . .	92
8.6	Inverse KKT: Box sliding — Generalization abilities . . . . .	92
8.7	Inverse KKT: Box sliding — Learned cost weights . . . . .	93
8.8	Inverse KKT: Door experiment — Demonstration . . . . .	95
8.9	Inverse KKT: Door experiment — Generalization abilities . . . . .	96
8.10	CORL: Benchmark — Objective and constraint . . . . .	97
8.11	CORL: Door experiment — Different grasps executed during learning . .	101
8.12	CORL: Door experiment — Learned return function . . . . .	102
8.13	CORL: Door experiment — Analytic cost function . . . . .	103
8.14	CORL: Door experiment — Grasp point trajectories . . . . .	104
8.15	CORL: Lockbox experiment — Motion sequence . . . . .	105
8.16	CORL: Lockbox experiment — Best candidate and success rate . . . . .	106



8.17	Single demonstration: Cabinet experiment — Motion sequence . . . . .	108
8.18	Single demonstration: Cabinet experiment — Learning evaluation . . . . .	109
8.19	Single demonstration: Cabinet experiment — Classification boundary . . . . .	110
8.20	Single demonstration: Cabinet experiment — Generalization abilities . . . . .	111
8.21	KMN: Prediction accuracy — Depth images of inputs and predictions . . . . .	112
8.22	KMN: Prediction accuracy — Training and prediction error . . . . .	115
8.23	KMN: Predictions — Overlay of prediction and point clouds . . . . .	118
8.24	KMN: Skill transfer — Door opening sequence . . . . .	119

# List of Algorithms

1	Inverse KKT . . . . .	54
2	Combined Optimization and Reinforcement Learning . . . . .	63
3	KMN Multi-step Network Predictions . . . . .	82
4	KMN Data Generation and Network Training . . . . .	83
5	Black-box IOC . . . . .	94

# List of Tables

8.1	Overview of the experiments in this thesis. . . . .	86
8.2	Inverse KKT: 2D motion planning task — Task error . . . . .	89
8.3	Inverse KKT: Box sliding — Comparison to black-box IOC . . . . .	94
8.4	CORL: Benchmark — Evaluation results . . . . .	100
8.5	CORL: Door experiment — Comparison to CMA-ES . . . . .	104
8.6	KMN: Prediction accuracy — Evaluation results . . . . .	114



# Chapter 1

## Introduction

Robotics is an interdisciplinary research area that combines many disciplines such as mechanical engineering, electrical engineering, and computer science. Throughout the last centuries, researchers developed a solid foundation and understanding of robots (Siciliano and Khatib, 2016). The International Federation of Robotics (IFR) estimates that in 2016 around 1 828 000 industrial robots were used and predicts an increase to 3 053 000 by the end of 2020 (International Federation of Robotics, 2017a). These industrial robots are mainly used in manufacturing halls, where they perform different steps in the production pipeline (e.g., assembly, drilling). The reasons for this success are on the one hand economic factors to reduce production costs and on the other hand properties of the robots (e.g., precision, force) that exceed the capabilities of humans. Besides the large number of industrial robots, there is also an increasing number of service robots (International Federation of Robotics, 2017b) for professional (e.g., logistics, medicine, farming) and private usage (e.g., entertainment, household cleaning). The IFR predicts on the service robot market an average increase of 20–35% per year until 2020. These numbers indicate that the current abilities of robots already enable them to solve a wide range of problems. However, they also show the large potential of future applications that are not solvable yet. Many of these future applications are in domains where robots encounter an unstructured environment and have to make decisions on their own. Examples of such domains are earthquake scenarios or living rooms, which share an environment structure that is unknown in advance and that can occur in many different variations. The hardware capabilities of robots are already sufficient to perform many manipulation tasks in such domains. However, the difficulty arises in the decision-making part of the system. There, it is still an open challenge regarding how to perform complex manipulation tasks in unstructured environments.

In this thesis, we address this challenge by building on the robotics research of the last decades and contributing algorithms in the area of manipulation skill learning. These algorithms should overcome the limitations of current systems and give robots the ability to learn generalizable manipulation skills in a sample-efficient manner.

## 1.1 Motivation for Manipulation Skill Learning

Handling the large versatility of the world is an enormous challenge during the design of robot manipulation skills. A main issue lies in representing the desired behavior in such a way that it is usable for many different variations of an environment. For example, a door opening task has different properties like the shape of the handle or the unlocking mechanism of the door joint, which have an effect on the motion that opens a specific door. However, many manipulation tasks have an underlying concept that describes how to perform the task, which is consistent for all environment configurations. In the door manipulation case, a possible concept is to first establish contact with the door handle, then to rotate the handle until the door joint is unlocked, and finally to apply a force on the door until a desired state is reached. Such concepts can be exploited to define generalizable manipulation skills. In this context, a core challenge is that the model of the environment is not completely known, which results from sensors that observe the environment in the form of a high-dimensional signal (e.g., point cloud, image). The interpretation of such sensor data into a meaningful representation is still an open problem due to measurement uncertainty and the wide range of possible environments. The geometric shape of the environment can usually be obtained from different sensors. However, it is more difficult to estimate the precise kinematic structure of the environment and how to manipulate it through contact. Manufacturing halls are—in contrast to such environments—very structured, which is why a transfer of existing robotics techniques to uncertain and incomplete estimates of the environment is not straightforward. Furthermore, the design of efficient manipulation skills is hard to program by hand and requires expert knowledge. The field of machine learning consists of different data-driven strategies that have the potential to enhance the abilities of robot manipulation skills. Machine learning uses data in various forms to build models that capture underlying concepts. In this thesis, we follow this paradigm and combine it with classical robotic techniques to solve various problems in manipulation skill learning. Specifically, we perform imitation learning to extract underlying concepts from demonstrations that are used to generalize a skill to unobserved environments. The large versatility of the world is addressed by using deep neural networks to map high-dimensional sensor data to meaningful kinematic representations, which contain all the necessary information to perform a skill. Further, we design a reinforcement learning algorithm to let the robot explore and improve

a skill by interacting with the environment, which allows the learning of certain properties of the environment that are not visible from passive sensor data. In the next section, we describe the underlying objectives of this work.

## 1.2 Advancing Manipulation Skill Learning

We consider the scenario in which a robot observes an environment that should be manipulated into a desired state. Throughout this thesis, we make the assumption that the environment consists of rigid bodies that are connected with joints. The goal of this thesis is to find solutions that enhance the current capabilities of robots to solve such manipulation tasks in the real world. The proposed algorithms combine machine learning with planning and control techniques to achieve the following objectives:

- **Generalization:** The main objective of this work is to achieve skills with high generalization capacities. The learning process should result in skills that can be applied to many different environment configurations. Specifically, we aim for skills that are able to handle a wide range of different initial and final environment states. For example, the box sliding experiment in Section 8.1.4 shows how a robot can slide a box from various initial to various final states (see Figure 8.6). Further, the robot should be able to do the manipulation independent of its own initial state. Another kind of generalization that should be achieved is the transfer of a skill between different geometric properties of the manipulated objects.
- **Performance:** A further objective is that the skills should reach a high level of performance in specified objective criteria. The quality of manipulations can be measured in numerous ways. In many practical applications, there are multiple objectives that should be achieved at the same time, e.g. having a low duration, smooth motion, and reaching a certain precision for the final configuration of the environment. The combination of these objectives is non-trivial since the balancing between them is important. All the objectives have different properties and it is hard to optimize them with a single algorithm. The objectives should also be defined in an independent manner such that they are reusable for various manipulation tasks.

- **Sample Efficiency:** The learning process should be sample-efficient on the real system. This is one of the main goals that many reinforcement learning algorithms try to achieve. It is difficult to accomplish since the robot motions as well as the sensor inputs are in general high-dimensional and learning in high-dimensional spaces usually requires a large amount of samples until convergence. To achieve a reasonable sample-efficiency on the real robot it is necessary to make certain assumptions about the problem that reduce the search space by incorporating prior knowledge (e.g., demonstrations, simulators).
- **Safety:** The safety of the robot and its environment during learning should be ensured. In particular, manipulation tasks bring a certain risk that the robot or its environment gets harmed due to the interaction forces, which makes the integration of safety into the learning process unavoidable. On the one side, specific robot controllers and hardware are necessary to ensure safety during execution. On the other side, the exploration strategies should only select parameters that are safe to execute on the real robot. However, safety can be defined in several ways and often depends on the task. Safety also competes with the goal of being sample-efficient since a safe learning strategy usually requires more samples until convergence.

The proposed algorithms exploit the inherent problem structure to achieve these objectives for a wide range of different manipulation types. We present evaluations for different tasks in simulation and with a real robot (see Figure 1.1) that evaluate the proposed algorithms regarding these objectives. In the next section, we summarize the contributions of this thesis.

### 1.3 Contributions of this Thesis

The core contributions of this thesis are algorithms that allow robots to acquire manipulation skills in an autonomous way and aim to achieve the objectives described in the previous section. The design of these algorithms was inspired by the specific structure of manipulation tasks. A key aspect of this structure is that external degrees of freedom are manipulated through contacts between the robot and the en-



vironment. In the following sections, we briefly summarize the main contributions of this thesis.

### 1.3.1 Inverse KKT (Chapter 4)

The goal of this work is to learn a policy by imitating human behavior. We propose the inverse optimal control method Inverse KKT to learn a manipulation skill from expert demonstrations. Inverse KKT assumes that the demonstrations fulfill the Karush-Kuhn-Tucker (KKT) conditions of an unknown underlying constrained optimization problem and extracts parameters of this underlying problem. By using this, we can exploit the latter to extract the relevant task spaces and cost parameters from demonstrations of skills that involve contacts. For a typical linear parametrization of cost functions, this reduces to a quadratic program, ensuring guaranteed and very efficient convergence, but Inverse KKT can deal also with arbitrary non-linear parametrizations of cost functions. Further, we present a nonparametric variant of Inverse KKT that represents the cost function as a functional in reproducing kernel Hilbert spaces. The aim of this approach is to push learning from demonstration to more complex manipulation scenarios that include the interaction with objects and therefore the realization of contacts/constraints within the motion. The use of a constrained optimization problem as skill representation allows to describe tasks in an abstract manner that can generalize to various environments. This work has been published in:

- Englert, P., Vien, N. A., and Toussaint, M. (2017). Inverse KKT — Learning Cost Functions of Manipulation Tasks from Demonstrations. *International Journal of Robotics Research*, 36(13-14):1474-1488.
- Englert, P. and Toussaint, M. (2015). Inverse KKT — Learning Cost Functions of Manipulation Tasks from Demonstrations. In *Proceedings of the International Symposium of Robotics Research*.

### 1.3.2 Combined Optimization and Reinforcement Learning (Chapter 5)

We propose a novel learning framework to improve a skill by exploring different manipulation strategies. Combined optimization and reinforcement learning (CORL) is

an algorithm that combines analytic optimization and episodic reinforcement learning. As an alternative to the standard reinforcement learning formulation where all objectives are defined in a single reward function, we propose to decompose the problem into analytically known objectives, such as motion smoothness, and black-box objectives, such as trial success or reward depending on the interaction with the environment. While the overall policy optimization problem is high-dimensional, in typical robot manipulation problems we can assume that the black-box return and constraint only depend on a low-dimensional projection of the solution. With this formulation we can exploit the problem structure for a sample-efficient learning framework that iteratively improves the policy with respect to the objective functions under the success constraint. We employ efficient 2nd-order optimization methods to optimize the high-dimensional policy with respect to the analytic cost function while keeping the low-dimensional projection fixed. This is alternated with safe Bayesian optimization over the low-dimensional projection to address the black-box return and success constraint. During both improvement steps, the success constraint is used to keep the optimization in a secure region and to clearly distinguish between motions that lead to success or failure.

One objective of this thesis is to learn generalizable skills from only a small amount of human knowledge. We consider the scenario in which a robot is demonstrated a manipulation skill only once and should then require only few own trials to learn to reproduce, optimize, and generalize that same skill. We combine the two algorithms, Inverse KKT and CORL, to learn a skill from a single demonstration. This combination allows leveraging and combining (i) constrained optimization methods to address analytic objectives, (ii) constrained Bayesian optimization to explore black-box objectives, and (iii) inverse optimal control methods to eventually extract a generalizable skill representation. This work has been published in

- Englert, P. and Toussaint, M. (2018b). Learning Manipulation Skills from a Single Demonstration. *International Journal of Robotics Research*, 37(1):137-154.
- Englert, P. and Toussaint, M. (2016). Combined Optimization and Reinforcement Learning for Manipulations Skills. In *Proceedings of Robotics: Science and Systems*.

### 1.3.3 Kinematic Morphing Networks (Chapter 7)

In this work, we propose kinematic morphing networks to find the relation of different geometric environments and use this relation to transfer skills between the environments. We assume that the environment can be modeled as a kinematic structure and represented with a low-dimensional parametrization. This parametrization is defined relative to a prototype model, which is used as a reference for the morphing operation. A further assumption is that a simulator exists that is able to generate an environment for a given parametrization. This simulator is used to generate synthetic training data in form of depth images and point clouds. A deep neural network is trained on this dataset to map depth image observations of the environment to morphing parameters, which include transformations and configurations of the prototype model. The skill transfer is achieved by defining a policy on the prototype model and morphing observed environments to this prototype. A key element of this work is the usage of the concatenation property of affine transformations and the ability to convert point clouds to depth images, which allows to apply the network in an iterative manner. The same functionality is also used to create more data points by applying the network on the training data. The experimental results show that this functionality leads to a lower prediction error compared to single step predictions. This work was accepted for publication in

- Englert, P. and Toussaint, M. (2018). Kinematic Morphing Networks for Manipulation Skill Transfer. Accepted at the International Conference on Intelligent Robots and Systems.

## 1.4 Overview of Thesis Structure

This thesis is structured in the following chapters:

- **Background (Chapter 2):** We introduce fundamental techniques on motion optimization, Gaussian processes, Bayesian optimization, and neural networks, which are employed in the proposed algorithms.
- **Related Work (Chapter 3):** This chapter summarizes research in the areas of reinforcement learning, imitation learning, and environment model learning. Here, a special focus is put on related work in robotics and manipulation skill learning and their differences to the proposed algorithms.



Figure 1.1: The images show different manipulations performed by a PR2 robot. These tasks are used to evaluate the proposed algorithms (see Chapter 8).

- **Inverse KKT (Chapter 4):** We derive the Inverse KKT objective function by inverting the Karush-Kuhn-Tucker conditions. Further, we discuss various possibilities to regularize and parametrize the weights. We also introduce a nonparametric Inverse KKT variant that represents the objective as a functional in a reproducing kernel Hilbert space.
- **Combined Optimization and Reinforcement Learning (Chapter 5):** The problem structure of the combined optimization and reinforcement learning problem is introduced by describing the policy parametrization and the various objectives. Afterwards, we introduce an algorithm to solve this problem by combining two improvement strategies, Bayesian optimization and analytic motion optimization, with each other.
- **Prior Knowledge for Manipulation Skill Learning (Chapter 6):** We describe how the proposed learning algorithms can be configured for the manipulation domain. We present how to define the projection constraint in CORL and the feature set in Inverse KKT such that prior knowledge of manipulation tasks is exploited. Further, we describe a way to combine these two algorithms with the goal to learn skills from a single demonstration.
- **Kinematic Morphing Networks (Chapter 7):** In this chapter, we introduce kinematic morphing networks and how they can be employed to transfer manipulation skills. We describe the parametrization of the environment that consists of two different parameter types. Finally, we present the prediction and training algorithms for kinematic morphing networks.

- **Experiments (Chapter 8):** This chapter presents the experimental evaluations of the proposed algorithms on different synthetic and real robot experiments. Figure 1.1 shows three real robot manipulation tasks on which we benchmark the algorithms.
- **Conclusion (Chapter 9):** The thesis is concluded with a summary of the proposed methods and an outlook on potential future research directions.



# Chapter 2

## Background

We introduce fundamental techniques in robotics and machine learning that are relevant for the proposed algorithms in this thesis.

### 2.1 Motion Representation

In this thesis, we assume that the robot and the environment consist of rigid bodies that are connected with joints. The state of a robot at time  $t$  is described by its joint configuration  $\mathbf{x}_t \in \mathcal{X} \subset \mathbb{R}^n$  where  $n$  is the number of degrees of freedom (DOF). In Figure 2.1, a simple robot with two bodies and two rotational joints is visualized. The configuration of the environment is denoted with  $\mathbf{y} \in \mathcal{Y}$  and contains all the properties (e.g., shape, color, location) of all the objects in the scene. A kinematic map is defined as a differentiable function

$$\phi_t : \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}^m \quad (2.1)$$

that maps the state of the robot and the environment into a  $m$ -dimensional feature space. Such a feature space could be for example the endeffector position of a robot in world coordinates as visualized in Figure 2.1. The Jacobian of such a kinematic map is  $\mathbf{J}(\mathbf{x}_t, \mathbf{y}) = \frac{\partial}{\partial \mathbf{x}_t} \phi_t(\mathbf{x}_t, \mathbf{y})$ . We represent skills as a set of features that are used to describe robot behavior in form of costs and constraints of a motion optimization problem.

A trajectory  $\bar{\mathbf{x}}$  is defined as a sequence  $(\mathbf{x}_0, \mathbf{x}_1, \dots, \mathbf{x}_T)$  of  $T+1$  robot configurations  $\mathbf{x}_t$  with a time step  $\Delta t$  and duration  $d$ . This leads to a total amount of  $n(T+1)$  trajectory parameters. The goal of trajectory optimization is to find a trajectory  $\bar{\mathbf{x}}$  that, given an initial configuration  $\mathbf{x}_0$ , optimizes an objective. Toussaint (2017) introduced k-order Markov optimization (KOMO), in which the objective function is defined as

$$\begin{aligned} \mathcal{F}(\bar{\mathbf{x}}, \mathbf{y}, \mathbf{w}) &= \sum_{t=1}^T w_t \phi_t(\tilde{\mathbf{x}}_t, \mathbf{y})^\top \phi_t(\tilde{\mathbf{x}}_t, \mathbf{y}) \\ &= \Phi(\bar{\mathbf{x}}, \mathbf{y})^\top \text{diag}(\mathbf{w}) \Phi(\bar{\mathbf{x}}, \mathbf{y}) \end{aligned} \quad (2.2)$$

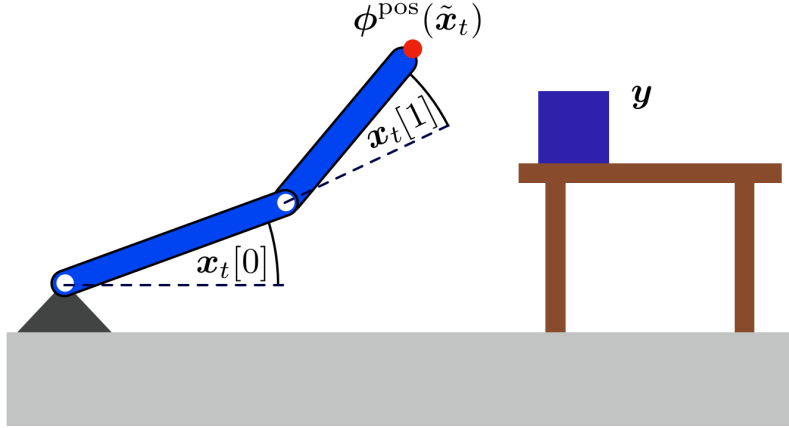


Figure 2.1: The sketch shows a robot with two rotational joints and two bodies as well as an environment  $\mathbf{y}$  that consists of a table and a box.

where  $\mathbf{w} \in \mathbb{R}^u$  are the feature weights and  $\Phi(\bar{\mathbf{x}}, \mathbf{y})$  is a vector function that contains the kinematic maps of all time steps. This objective is defined as the weighted sum of squared features over all time steps. Each cost term depends on a  $k$ -order tuple of consecutive configurations  $\tilde{\mathbf{x}}_t = (\mathbf{x}_{t-k}, \dots, \mathbf{x}_{t-1}, \mathbf{x}_t)$ , containing the current and  $k$  previous robot configurations. In addition to the task costs, we also consider inequality constraints

$$\forall_t \quad \mathbf{g}_t(\tilde{\mathbf{x}}_t, \mathbf{y}) \leq \mathbf{0} \quad (2.3)$$

and equality constraints

$$\forall_t \quad \mathbf{h}_t(\tilde{\mathbf{x}}_t, \mathbf{y}) = \mathbf{0} \quad (2.4)$$

which, as features  $\phi_t(\tilde{\mathbf{x}}_t, \mathbf{y})$ , can refer to arbitrary task spaces. The resulting KOMO optimization problem is

$$\begin{aligned} \bar{\mathbf{x}}^* &= \arg \min_{\bar{\mathbf{x}}} \mathcal{F}(\bar{\mathbf{x}}, \mathbf{y}, \mathbf{w}) \\ \text{s.t.} \quad &\mathbf{g}(\bar{\mathbf{x}}, \mathbf{y}) \leq \mathbf{0} \\ &\mathbf{h}(\bar{\mathbf{x}}, \mathbf{y}) = \mathbf{0} \end{aligned} \quad (2.5)$$

where  $\mathbf{g}$  and  $\mathbf{h}$  are vector functions that contain the inequality and equality constraints of all time steps. In this approach, the equality constraints are often defined to represent persistent contacts with the environment (e.g.,  $\mathbf{h}$  describes the distance between hand and object that should exactly be 0). The motivation for using equality constraints for contacts, instead of using cost terms in the objective function as



in Equation (2.2), is the fact that minimizing costs does not guarantee that they will become 0, which is essential for establishing a contact. We incorporate the constraints with the augmented Lagrangian method and solve the resulting problem with Gauss-Newton optimization (Wright and Nocedal, 1999). For this, the gradient is

$$\nabla_{\bar{\mathbf{x}}}\mathcal{F}(\bar{\mathbf{x}}, \mathbf{y}, \mathbf{w}) = 2\mathbf{J}(\bar{\mathbf{x}}, \mathbf{y})^\top \text{diag}(\mathbf{w})\Phi(\bar{\mathbf{x}}, \mathbf{y}) \quad (2.6)$$

and the Hessian is approximated as in Gauss-Newton as

$$\nabla_{\bar{\mathbf{x}}}^2\mathcal{F}(\bar{\mathbf{x}}, \mathbf{y}, \mathbf{w}) \approx 2\mathbf{J}(\bar{\mathbf{x}}, \mathbf{y})^\top \text{diag}(\mathbf{w})\mathbf{J}(\bar{\mathbf{x}}, \mathbf{y}) \quad (2.7)$$

with the Jacobian  $\mathbf{J}(\bar{\mathbf{x}}, \mathbf{y}) = \frac{\partial}{\partial \bar{\mathbf{x}}}\Phi(\bar{\mathbf{x}}, \mathbf{y})$ . The structure of the gradient and Hessian is exploited for efficient optimization (see (Toussaint, 2017) for more details). In addition to the solution  $\bar{\mathbf{x}}^*$  we also get the Lagrange parameters  $\boldsymbol{\lambda}^*$ , which provide information on when the constraints are active during the motion. This knowledge can be incorporated to make the control of interactions with the environment more robust (see (Toussaint et al., 2014)).

The constrained trajectory optimization problem in Equation (2.5) plays a central part in this thesis. Throughout this thesis, we represent skills in form of cost and constraint features  $\phi_t(\mathbf{x}_t, \mathbf{y})$  with the goal to generalize a skill to various environments  $\mathbf{y}$ . In Chapter 4, we propose an inverse optimal control formulation for this problem that learns the feature weights  $\mathbf{w}$  from demonstration data. KOMO is also used in Chapter 5 as a policy optimization strategy.

## 2.2 Gaussian Processes

A Gaussian process (GP) defines a probability distribution over functions (Rasmussen and Williams, 2006). We will employ GPs for function approximation since they can express a broad range of different functions and provide a probability distributions over predictions. We will first handle the regression and afterwards the classification case. A GP is fully specified by a mean function  $m(\mathbf{x})$  and a covariance function  $k(\mathbf{x}, \mathbf{x}')$ . In the regression case, we have data of the form  $\mathcal{D} = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^D$  with inputs  $\mathbf{x} \in \mathbb{R}^m$  and targets  $y \in \mathbb{R}$ . Predictions for a test input  $\mathbf{x}_*$  are given by mean

$$\mu(\mathbf{x}_*) = m(\mathbf{x}_*) + \kappa(\mathbf{x}_*)^\top (\mathbf{K} + \sigma_n^2 \mathbf{I})^{-1} \mathbf{y} \quad (2.8)$$

and variance

$$\mathbb{V}(\mathbf{x}_*) = k(\mathbf{x}_*, \mathbf{x}_*) - \kappa(\mathbf{x}_*)^\top (\mathbf{K} + \sigma_n^2 \mathbf{I})^{-1} \kappa(\mathbf{x}_*) \quad (2.9)$$

with  $\kappa_i(\mathbf{x}_*) = k(\mathbf{x}^{(i)}, \mathbf{x}_*)$ , the Gram matrix  $\mathbf{K}$  with  $K_{ij} = k(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$ , training target vector  $\mathbf{y} = [y^{(1)}, \dots, y^{(D)}]^\top$ , and noise variance  $\sigma_n^2$ .

In the binary classification case, we have data of the form  $\{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^D$  where the outputs are discrete labels  $y \in \{0, 1\}$ . Here, we cannot directly make use of a GP to model the output. Therefore, the GP models a discriminative function  $g(\mathbf{x})$ , which defines a class probability

$$P(y = 1 | \mathbf{x}) = \sigma(g(\mathbf{x})) . \quad (2.10)$$

with the sigmoid function  $\sigma(x) = \frac{1}{1 + \exp(-x)}$ . Since this likelihood is non-Gaussian the exact posterior over  $g$  is not a Gaussian process—one instead applies a Laplace approximation (Nickisch and Rasmussen, 2008). Throughout this thesis, we use the squared exponential kernel function

$$k(\mathbf{x}, \mathbf{x}') = \sigma_f^2 \exp\left(-\frac{1}{2}(\mathbf{x} - \mathbf{x}')^\top \boldsymbol{\Sigma}^{-1}(\mathbf{x} - \mathbf{x}')\right) , \quad (2.11)$$

where  $\boldsymbol{\Sigma} = \text{diag}([l_1^2, l_2^2, \dots, l_m^2])$  is a matrix with squared length scales and  $\sigma_f$  is the signal standard deviation. We mainly use GPs to do Bayesian optimization, which is described in the next section.

## 2.3 Bayesian Optimization

Bayesian optimization (Mockus et al., 1978) is a strategy to find the maximum of a function  $f(\mathbf{x})$ , where the function  $f(\mathbf{x})$  is not known in closed-form expression and only noisy observations  $y \in \mathbb{R}$  of the function value can be made at sampled values  $\mathbf{x} \in \mathbb{R}^m$ . These samples are collected in a dataset  $\mathcal{D} = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^d$  on which a GP model of  $f$  is built. The next sample point  $\mathbf{x}^{(d+1)}$  is chosen by maximizing an acquisition function  $a(\mathbf{x}; \mathcal{D})$  for  $\mathbf{x}$ . There are many different ways to define this acquisition function (see (Brochu et al., 2010)). Kushner (1964) introduced the common acquisition function *probability of improvement* (PI) that is defined as

$$a_{\text{PI}}(\mathbf{x}; \mathcal{D}) = P(f(\mathbf{x}) \geq f(\mathbf{x}^+)) = \Phi\left(\frac{\mu(\mathbf{x}) - f(\mathbf{x}^+)}{\sqrt{\mathbb{V}(\mathbf{x})}}\right) \quad (2.12)$$

$$\text{with } \mathbf{x}^+ = \arg \max_{\mathbf{x} \in \{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(d)}\}} f(\mathbf{x}) ,$$

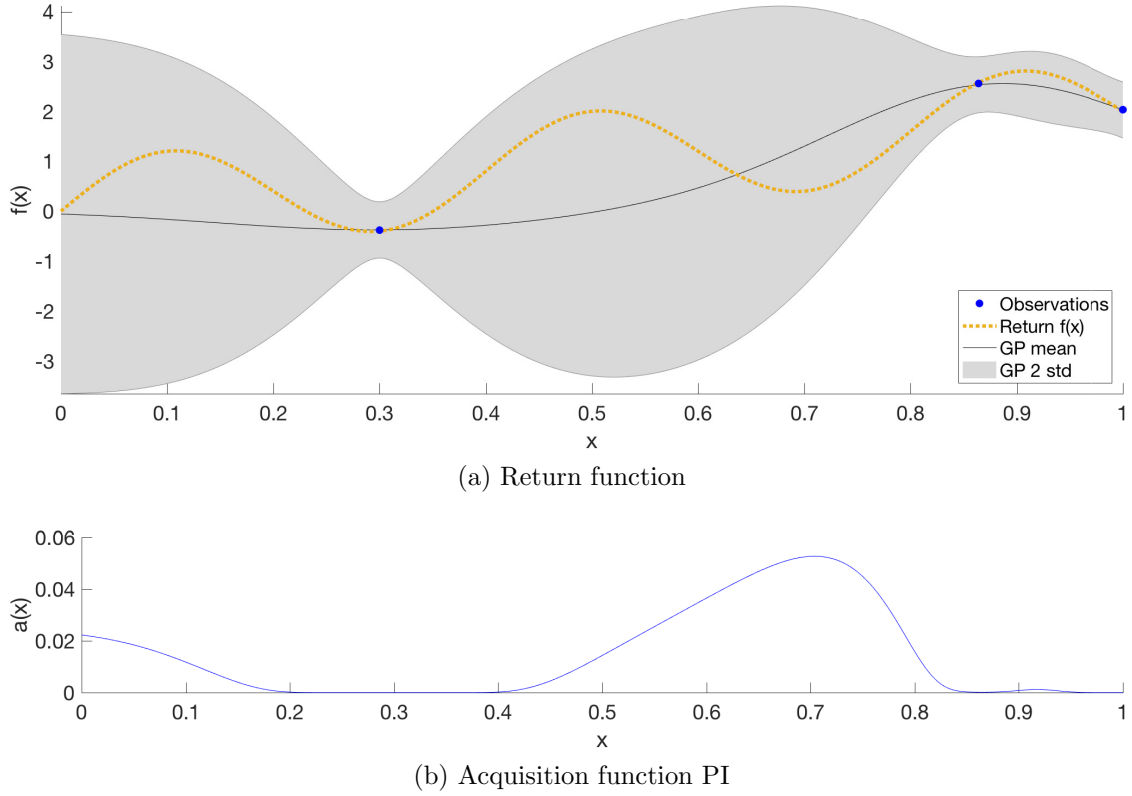


Figure 2.2: Example of Bayesian optimization.

with the normal cumulative distribution function  $\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^x e^{-t^2/2} dt$ . Another widely used acquisition function is based on upper confidence bounds (Srinivas et al., 2012). The acquisition function *Gaussian process upper confidence bound* (UCB) is defined as

$$a_{\text{UCB}}(\mathbf{x}; \mathcal{D}) = \mu(\mathbf{x}) + \sigma \sqrt{\mathbb{V}(\mathbf{x})}. \quad (2.13)$$

Figure 2.2 shows a one-dimensional example of Bayesian optimization where Figure 2.2a visualizes a GP model of the return function and Figure 2.2b visualizes the corresponding acquisition function  $a_{\text{PI}}(\mathbf{x}; \mathcal{D})$ . In Chapter 5, we will combine Bayesian optimization with the constrained motion optimization method presented in Section 2.1 to improve a policy with respect to various objectives. We extend the PI criteria with a boundary uncertainty function to improve a skill in a safe manner.

## 2.4 Convolutional Neural Networks

A neural network (McCulloch and Pitts, 1943; Widrow and Hoff, 1960; Rosenblatt, 1962; Rumelhart et al., 1986) implements a non-linear function  $\mathbf{y} = f(\mathbf{x}; \boldsymbol{\omega})$  that maps an input  $\mathbf{x} \in \mathbb{R}^n$  to an output  $\mathbf{y} \in \mathbb{R}^m$  and is parametrized by weights  $\boldsymbol{\omega} \in \mathbb{R}^q$ . Feed-forward neural networks consist of a layer-wise structure and are a common choice for function approximation with large amounts of data. The network  $f$  has an input layer that represents the raw inputs  $\mathbf{x}$  and  $L$  subsequent layers where each layer  $i$  consists of inputs  $\mathbf{a}^{(i-1)}$ , outputs  $\mathbf{a}^{(i)}$ , weight parameters  $\mathbf{W}^{(i)}$ , bias parameters  $\mathbf{b}^{(i)}$ , hidden states  $\mathbf{z}^{(i)}$ , and an activation function  $h^{(i)}(\mathbf{z}^{(i)})$ . The network output  $f(\mathbf{x}; \boldsymbol{\omega})$  is computed by propagating the inputs  $\mathbf{x}$  through the chain-based structure:

$$\begin{aligned}
 \mathbf{a}^{(1)} &= \mathbf{x} \\
 \mathbf{z}^{(2)} &= \mathbf{W}^{(1)}\mathbf{a}^{(1)} + \mathbf{b}^{(1)} \\
 \mathbf{a}^{(2)} &= h^{(2)}(\mathbf{z}^{(2)}) \\
 \mathbf{z}^{(3)} &= \mathbf{W}^{(2)}\mathbf{a}^{(2)} + \mathbf{b}^{(2)} \\
 \mathbf{a}^{(3)} &= h^{(3)}(\mathbf{z}^{(3)}) \\
 &\vdots \\
 f(\mathbf{x}; \boldsymbol{\omega}) &= \mathbf{a}^{(L+1)} .
 \end{aligned} \tag{2.14}$$

Figure 2.3 visualizes a fully connected neural network with two hidden layers and two outputs. The parameters  $\boldsymbol{\omega} = (\mathbf{W}^{(1)}, \mathbf{b}^{(1)}, \dots, \mathbf{W}^{(L)}, \mathbf{b}^{(L)})$  of the network are trained by stochastic gradient descent in which the gradient of the loss function with respect to the parameters is back-propagated through the network via the chain rule. Common loss functions to train such a network are the squared error for regression problems and the cross entropy for classification problems. In this thesis, we will mainly use the linear activation function  $h(\mathbf{z}) = \mathbf{z}$  and the rectified linear unit (ReLU; Nair and Hinton (2010)) activation function  $h(\mathbf{z}) = \max(0, \mathbf{z})$ .

A convolutional neural network (CNN; Le Cun et al. (1989)) is a special kind of network that implements the mathematical convolution operation. They are widely used when working with images, since the convolution enables an efficient weight sharing and feature extraction. A convolutional layer consists of a set of  $K$  learnable filters, where each filter is defined by a two-dimensional kernel  $g \in \mathbb{R}^{N \times N}$ .

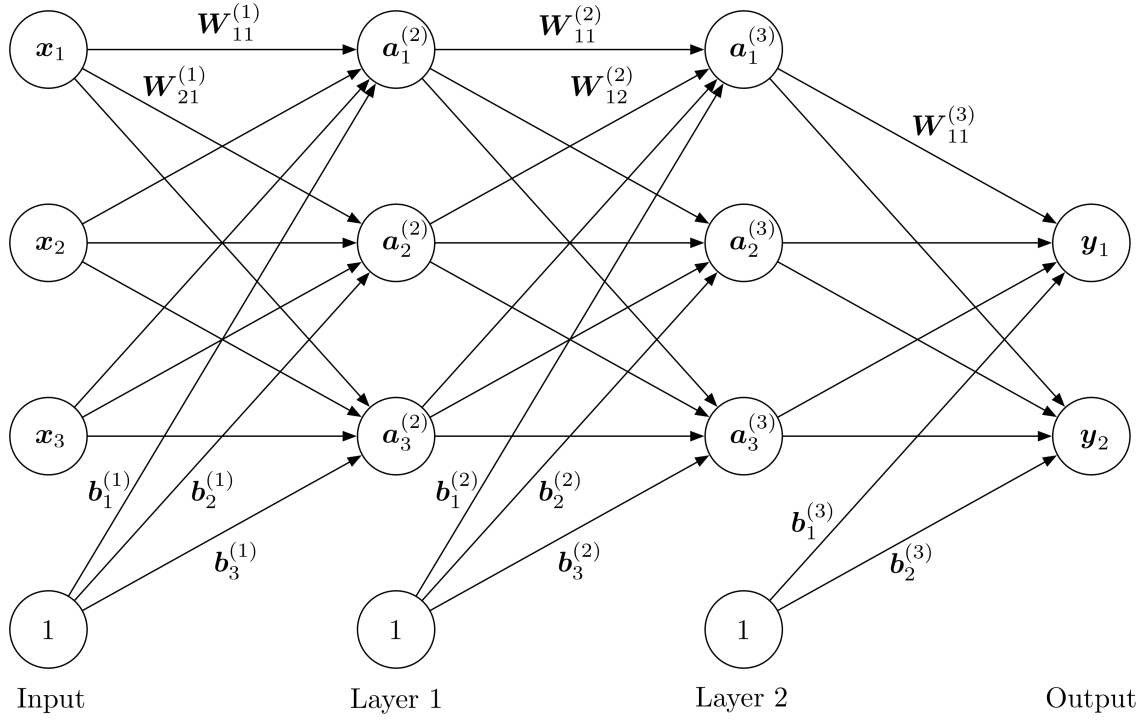


Figure 2.3: Fully connected neural network with 2 layers + 2 outputs.

The convolution operation to compute the hidden state of such a layer is defined as

$$z_{h',w'} = \sum_{h=1}^N \sum_{w=1}^N g_{h,w} x_{h'-\lfloor \frac{N}{2} \rfloor + h - 1, w' - \lfloor \frac{N}{2} \rfloor + w - 1} \quad (2.15)$$

for an input image  $x \in \mathbb{R}^{H \times W}$ . Successive convolutional layers are often followed by pooling operations since they lead to invariance regarding local input translations. Pooling also reduces the size of the representation, which leads to fewer network parameters. A common choice is max pooling (Zhou and Chellappa, 1988) with a  $2 \times 2$  filter and stride 2 that selects the maximum value over 4 numbers of the representation and reduces the width and height of a representation by a factor of 2. Figure 2.4 visualizes a  $3 \times 3$  convolution operation that is followed by a  $2 \times 2$  max pooling operation. A widely used network structure consists of concatenating multiple convolution and max pooling blocks (Goodfellow et al., 2016). In this thesis, we define such a CNN to map depth images to parameters of a 3D environment (see Chapter 7).

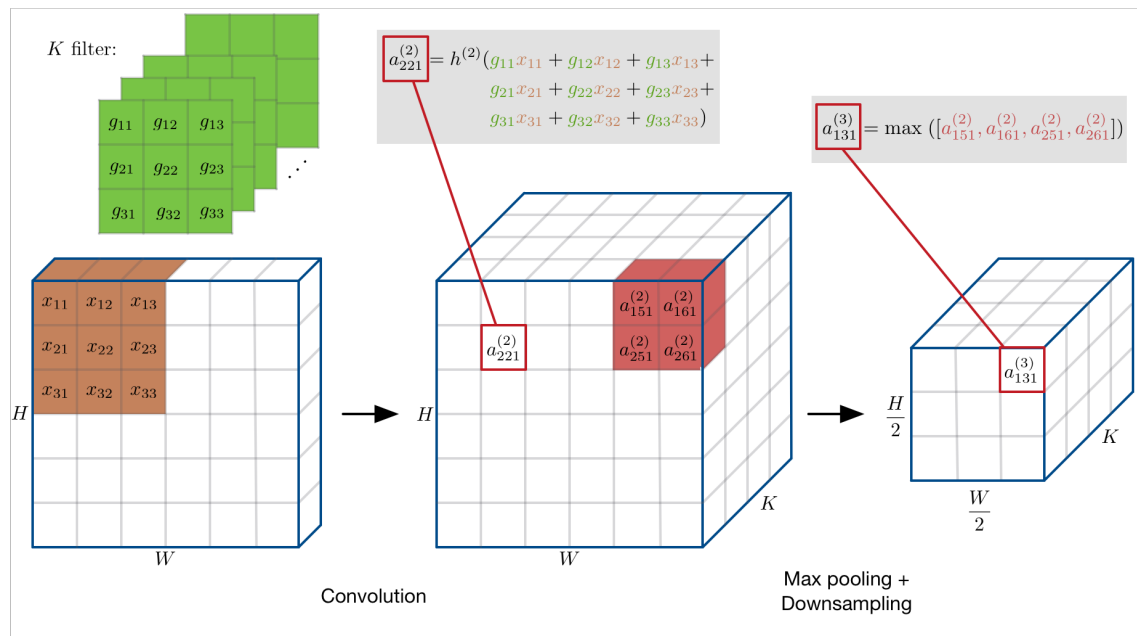


Figure 2.4: Visualization of neural network convolution and max pooling operation.



# Chapter 3

## Related Work

This chapter describes related work in the areas of reinforcement learning, inverse optimal control, and environment model learning.

### 3.1 Reinforcement Learning

In reinforcement learning (RL) an agent learns a policy  $\pi$  by interacting with its environment (Sutton and Barto, 1998). The system dynamics as well as rewards can only be observed by doing rollouts on the real system. In the following sections, we present work that is related to the combined optimization and reinforcement-learning algorithm (CORL) in Chapter 5.

#### 3.1.1 Policy Search in Robotics

Policy search is widely used to learn skills in robotics (Deisenroth et al., 2013) and aims at finding an optimal policy  $\pi^*$  for unknown stationary rewards  $r(\mathbf{x}_t, \mathbf{u}_t)$  by maximizing the return  $R(\pi) = \mathbb{E} \left[ \sum_{t=0}^T r(\mathbf{x}_t, \mathbf{u}_t) \mid \pi \right]$ . One approach introduced by Kober and Peters (2008) uses dynamic movement primitives (DMPs) as policy representation and the policy search method policy learning by weighting exploration with the returns (PoWER) to learn the shape and properties of the motion. An alternative approach introduced by Kalakrishnan et al. (2011) learns force control policies for manipulations. The policy is initialized with position control via imitation and afterwards augmented with a force profile that is learned with the reinforcement learning method policy improvement with path integrals (PI<sup>2</sup>; Theodorou et al. (2010)). Both approaches define a single reward function that combines different terms (e.g., smoothness, force, tracking errors). A main difference to the work in this thesis is that CORL performs learning on two policy parametrizations. This property allows to apply efficient Gauss-Newton optimization routines for the parts of a motion where an analytic cost function is available. Further, we demonstrate in Chapter 6 how it can be combined with an inverse optimal control method to extract



a cost function representation with high generalization capabilities. The experiment in Section 8.2.2 compares CORL to covariance matrix adaptation evolution strategy (Hansen and Ostermeier, 2001), which has been shown in (Stulp and Sigaud, 2013) to be closely related to PI<sup>2</sup>.

### 3.1.2 Episodic RL as Black-box Optimization

A restricted case of episodic reinforcement learning is where only the total return of an episode  $\sum_{t=0}^T r(\mathbf{x}_t, \mathbf{u}_t)$  is observed but not the individual rewards at each timestep (Stulp et al., 2013b). This property transforms the problem into a black-box optimization problem that allows one to utilize standard black-box optimization methods (e.g., covariance matrix adaptation evolution strategy (CMA-ES; Hansen and Ostermeier (2001)) or Bayesian optimization (Mockus et al., 1978)). These methods were previously applied to learn parameters in robotics. For example, Bayesian optimization was applied to learn gait parameters for locomotion skills (Lizotte et al., 2007; Calandra et al., 2015) and to learn parameters of a linear quadratic regulator for a balancing task (Marco et al., 2016). There exist different variants of additionally including constraints into the Bayesian optimization formulation (Schonlau et al., 1998; Gelbart et al., 2014; Gardner et al., 2014; Gramacy and Lee, 2011). The main concept of these approaches is to integrate the probability of a constraint being fulfilled into existing acquisition functions. In (Gardner et al., 2014), for example, it is assumed that the objective can also be evaluated in the unfeasible region and in (Gelbart et al., 2014) the case of a decoupled observation of objective and constraint is considered. CORL also employs such a constrained Bayesian optimization formulation to select the next policy parameter. However, the proposed acquisition function in CORL aims at achieving a secure and efficient learning process.

### 3.1.3 Safe Learning

An important aspect in learning manipulation skills is the safety that the robot does not damage itself or the environment. Schreiter et al. (2015) introduced a safe exploration strategy that optimizes a function in a safe manner where the feasible region is unknown. To do this, they assumed that the observations include a safety measure when they are close to the boundary. This information is integrated into

a differential entropy exploration criteria to select next candidates. Schreiter et al. also provided an upper bound for the probability of failure. However, it requires additional information about the distance to the decision boundary in critical regions, which is not always available. The experiment in Section 8.2.1 shows a comparison between this safe active learning approach and the strategy proposed in this thesis. SAFEOPT is another approach for safe exploration (Sui et al., 2015), which optimizes an unknown function with Bayesian optimization that is combined with a safety criterion of the form that the function value should exceed a certain threshold. SAFEOPT implements the concept of reachability to categorize the search space in different sets for safe exploration and exploitation. The next data point is selected by sampling the most uncertain decision. Berkenkamp and Schoellig (2015) applied this approach on a quadrotor vehicle stabilization task. Garcia Polo and Fernandez-Rebollo (2011) introduced a safe reinforcement learning approach that improves demonstrated behavior in a risk-sensitive manner. The behavior is represented with case-based reasoning techniques, which allows to use multiple trajectories as demonstration. The safety criterion is defined with the distance to the nearest neighbor that is limited with a threshold. The exploration is done by adding Gaussian noise to the current optimal actions. Achiam et al. (2017) proposed constrained policy optimization, which is based on constrained Markov decision processes to achieve a safe learning. They derived a bound on the difference between the rewards of two different policies, which is incorporated in the policy update while guaranteeing to improve the return and to satisfy a constraint. They showed that their method can be applied to train high-dimensional neural network policies for robotics tasks.

None of the above methods for safe or Bayesian exploration would be sample-efficient when directly applied on a high-dimensional non-stationary policy. However, they can be used within the proposed CORL framework (see Chapter 5), as demonstrated for UCB and SAL in the experiment in Section 8.2.1.

### 3.1.4 Combined Optimization and Learning

Various approaches exist that combine learning with optimization or planning methods. The advantage of this combination is that models can be incorporated in the optimization problems. This usually leads to a lower dimensional space for the learning part, which results in fewer rollouts until convergence. In (Rückert et al., 2013), a reinforcement-learning algorithm for planning movement primitives is in-

roduced that contains a two-layered learning formulation. In an outer loop the policy search method CMA-ES optimizes an extrinsic cost function that measures the task performance. The policy search is over parameters that are used in the inner loop to define a cost function for a trajectory optimization problem. This problem computes trajectories that are fed back as input to the extrinsic cost function. A core difference to the proposed approach is that they directly coupled the objective functions with each other in a hierarchical way and only optimized the extrinsic objective function. The intrinsic objective function is only used to perform rollouts.

Levine et al. (2016) proposed to learn a deep convolutional neural network that maps raw images directly to motor torques. This end-to-end training is done by iterating between reinforcement learning to generate rollouts and supervised learning to train the neural network. They also added a pretraining step for the initial policy and the vision system to reduce the amount of interaction time. CORL applies reinforcement learning on a low-dimensional projection of the full policy, which leads to a more sample-efficient learning. Instead of using an end-to-end approach, we propose in this thesis a skill learning pipeline that consists of different modules. In Chapter 7, we train deep neural networks for extracting parameters of the environment, which serve as input to skill policies defined as constrained motion optimization problems. This has the advantages that it is more interpretable rather than end-to-end approaches.

Kupcsik et al. (2013) presented a policy search method that combines model-free reinforcement learning with learned forward models (Deisenroth et al., 2015). Their approach learns probabilistic forward models of the robot and the skill, which are combined to generate artificial samples in simulation. These samples are combined with real robot rollouts to update the policy. The relative entropy policy search algorithm (Peters et al., 2010) maximizes the reward by balancing the exploration and experience loss while staying close to the observed data. A difference to CORL is that their approach learns a model of the task for internal simulations, whereas CORL directly learns a model that maps parameters to return values. Vuga et al. (2015) introduced an approach that combines the reinforcement learning method  $PI^2$  with the optimization algorithm iterative learning control (Bristow et al., 2006). The policy is represented with dynamic movement primitives. This approach applies iterative learning control as exploration strategy in the first part of the learning to

adapt the trajectory and speed profile. In the second part, a random exploration strategy fine-tunes the policy. The proposed algorithm CORL differs especially by dividing the problem in model-based motion optimization that improves the motion efficiently and reinforcement learning that improves the skill by exploring a low-dimension representation.

## 3.2 Inverse Optimal Control

Imitation learning aims at finding a policy  $\pi$  that reproduces the behavior demonstrated by an expert. Inverse optimal control (IOC), also known as inverse reinforcement learning, extracts a cost function from data (Kalman, 1964; Ng and Russell, 2000). In Chapter 4 of this thesis, the inverse optimal control method Inverse KKT (IKKT) is proposed, which learns a cost function for a constrained motion optimization problem from demonstrations. For a broader overview on IOC approaches we refer the reader to the survey paper by Zhifei and Joo (2012) and for an overview on imitation learning in robotics we recommend the paper by Argall et al. (2009).

### 3.2.1 Max-Entropy and Lagrangian-Based IOC

Maximum-margin planning (Ratliff et al., 2006), MaxEnt (Ziebart et al., 2008), and learning to search (Ratliff et al., 2009) are common inverse optimal control algorithms. These algorithms use forward solvers or policy optimization methods (e.g., value iteration,  $A^*$ ) in the inner loop, which requires perfect knowledge of the environment dynamics and is computational intensive. The work by Levine and Koltun (2012), which is similar to IKKT, does not need to solve the forward problem in an inner loop. They proposed a probabilistic formulation of inverse optimal control that approximates the MaxEnt model. In the framework of trajectory optimization (cf. Section 2.1) this translates to

$$\min_w \nabla_{\bar{x}} \mathcal{F}^\top (\nabla_{\bar{x}}^2 \mathcal{F})^{-1} \nabla_{\bar{x}} \mathcal{F} - \log |\nabla_{\bar{x}}^2 \mathcal{F}|. \quad (3.1)$$

The first term of this equation is similar to the Inverse KKT loss that we introduce in Equation (4.4), where the objective is to achieve small gradients. Additionally, they included the inverse Hessian in the first term as a weighting of the gradient. The second term ensures the positive definiteness of the Hessian and also acts as

a regularizer on the weights. The learning procedure is performed by maximizing the log-likelihood of the approximated reward function. Instead of enforcing a fully probabilistic formulation, we focus on finite-horizon constrained motion optimization formulation with the benefit that it can handle constraints and leads to a fast QP formulation. MaxEnt is like other Bayesian approaches (Ramachandran and Amir, 2007) very robust. However, it is proposed to the simple case of linear dynamics and quadratic rewards. This is hardly the case of arbitrary trajectory optimization. On the contrary, IKKT is based on constrained trajectory optimization, which learns a cost function that fits well with many trajectory optimization solvers and therefore can deal with a wider range of optimal control problems. In the experiment in Section 8.1.1, we compare the proposed algorithm Inverse KKT to the work by Levine and Koltun (2012) on a 2D planning task.

Puydupin-Jamin et al. (2012) introduced an approach to IOC that can handle linear constraints. Their approach learns the weight parameters  $\mathbf{w}$  and Lagrange parameters  $\boldsymbol{\lambda}$  by solving the least-squares optimization problem

$$\min_{\mathbf{w}, \boldsymbol{\lambda}} \left\| 2\mathbf{J}^\top \text{diag}(\boldsymbol{\Phi})\mathbf{w} + \mathbf{J}_c^\top \boldsymbol{\lambda} + \mathbf{J}^{/w} \right\|^2 \quad (3.2)$$

where  $^{/w}$  denotes the part in the cost function that is not weighted with  $\mathbf{w}$ . The method only addresses equality constraints (no complementarity condition for  $\boldsymbol{\lambda}$ ). A main concern with this formulation is that there are no constraints that ensure that the weight parameter vector  $\mathbf{w}$  do not become 0 or negative. If  $\mathbf{J}^{/w}$  is zero, as in Inverse KKT, the solution  $(\mathbf{w}^*, \boldsymbol{\lambda}^*)$  is identically zero. Starting with the KKT condition, they derived a linear residual function that is optimized analytically as unconstrained least square problem. In the experimental section they considered human locomotion with a unicycle model, where they learned one weight parameter of torques and multiple constraints (i.e., unicycle dynamics, initial state, final state). Their idea of using KKT conditions is similar to Inverse KKT. However, Inverse KKT allows for inequality constraints and leads to a QP with boundary constraints that ensures that the resulting parameters are feasible. Instead of optimizing for  $\boldsymbol{\lambda}$ , we eliminate  $\boldsymbol{\lambda}$  from the Inverse KKT optimization using Equation (4.7).

The work by Albrecht et al. (2011) learns cost functions for human reaching motions from demonstrations that are a linear combination of different transition types (e.g., jerk, torque). They transformed a bilevel optimization problem, similar

to (Mombaur et al., 2010), into a constrained optimization problem of the form

$$\begin{aligned}
& \min_{\bar{\mathbf{x}}, \mathbf{w}, \boldsymbol{\lambda}} \left\| \boldsymbol{\phi}^{\text{pos}}(\mathbf{x}_T) - \boldsymbol{\phi}^{\text{pos}}(\mathbf{x}_T^{(d)}) \right\|^2 \\
& \text{s.t. } \nabla_{\bar{\mathbf{x}}} L(\bar{\mathbf{x}}, \mathbf{y}, \boldsymbol{\lambda}, \mathbf{w}) = \mathbf{0} \\
& \quad \mathbf{h}(\bar{\mathbf{x}}, \mathbf{y}) = 0 \\
& \quad \sum_i w_i = 1 \\
& \quad \mathbf{w} \geq 0
\end{aligned} \tag{3.3}$$

The objective is the squared distance between optimal and demonstrated final hand position. They optimize this objective for the trajectory  $\bar{\mathbf{x}}$ , the weight parameters  $\mathbf{w}$ , and the Lagrange parameters  $\boldsymbol{\lambda}$  with the constraints that the KKT conditions of the trajectory  $\bar{\mathbf{x}}$  are fulfilled. To apply this approach, demonstrations are first preprocessed by extracting a characteristic movement with dynamic time warping and a clustering step. The results show that a combination of different transition costs represent human arm movements best and that they are able to generalize to new hand positions. The advantage of this approach is that it does not only return the weight parametrization  $\mathbf{w}$ , but also an optimal trajectory  $\bar{\mathbf{x}}^*$  out of the inverse problem in Equation (3.3). The integration of the KKT conditions differs to Inverse KKT in two ways. First, they put the KKT conditions in the constraint part of the formulation, whereas we use them directly as scalar product in the cost function. Second, they defined them on the optimization variables  $\bar{\mathbf{x}}$ , whereas we defined them on the demonstrations  $\bar{\mathbf{x}}^{(d)}$  (see Equation (4.4)). Instead of minimizing a function directly of the final endeffector position and only learning weights of transition costs, we present a more general solution to imitation learning that can learn transition and task costs in arbitrary feature spaces. Inverse KKT also handles multiple demonstrations directly without preprocessing them to a characteristic movement.

Kalakrishnan et al. (2013) introduced an inverse formulation of the path integral reinforcement learning method PI<sup>2</sup> (Theodorou et al., 2010) to learn objective functions for manipulation tasks. The cost function consists of a control cost and a general state dependent cost term at each time step. Their approach maximizes the trajectory likelihood of demonstrations  $P(\bar{\mathbf{x}}|\mathbf{w})$  for all demonstrations by sampling trajectories around the demonstrations. Further, the weights  $\mathbf{w}$  are L1 regularized so that only a subset is selected. The method is evaluated on grasping tasks. Finn

et al. (2016) proposed to learn a cost function in an inner loop of a policy search method. They formulated a sample-based approximation for non-linear maximum entropy IOC. The cost function is represented as a neural network and regularization is achieved by penalizing an acceleration term and preferring strict monotonically decrease in the costs of the demonstration. They evaluated the method on robot manipulation tasks that include autoencoder features from camera images.

### 3.2.2 Nonparametric Inverse Optimal Control

Marinho et al. (2016) proposed to represent trajectories as vectors in reproducing kernel Hilbert spaces (RKHS). Their approach incorporates a functional gradient motion planning algorithm where trajectories are represented as a linear combination of kernels. The motion planning objective is to minimize a cost functional that maps each trajectory in RKHS to a scalar cost. The cost functional consists of a smoothness term and an obstacle avoidance term. The optimization is done by computing the functional gradient. Their approach is similar to the nonparametric variant of Inverse KKT (see Section 4.4). Whereas they represent the trajectory in RKHS and define a cost over these trajectories, we represent the cost function at each time step as a functional in a RKHS. The usage of functional gradient techniques for imitation learning was introduced by Ratliff et al. (2009), which extended maximum margin planning methods to non-linear cost functions.

The works from Grubb and Bagnell (2010) and Bradley (2009) rely on deep modular systems to learn non-linear cost functions. Functional backpropagation combines functional gradient descent with backpropagation mechanics in Euclidean function space. It allows the use of a greater class of learning algorithms than standard backpropagation. A key aspect of their work is a modular system that separates the structural aspects of the network from the learning in individual modules. Their results show that the functional gradient variant is more robust to local minima than the parametrized gradient. The work by Levine et al. (2011) employs Gaussian processes to learn the reward as a non-linear function. In addition to learn the reward, their approach also learns the kernel hyperparameters to recover the structure of the reward function by maximizing the likelihood of the reward under the observed expert demonstrations.

Another research area focuses on using Bayesian nonparametric methods for inverse optimal control. Choi and Kim (2013) presented a Bayesian nonparametric

approach to construct features for the cost function using the Indian buffet process (Griffiths and Ghahramani, 2011). Michini and How (2012) proposed a Bayesian nonparametric inverse reinforcement learning approach that partitions the demonstrations into sets of smaller sub-demonstrations. For each sub-demonstration a simple reward function is learned. The partition process is automated by using a Chinese restaurant process (Aldous, 1985) prior as a generative model over partitions, which does not make it necessary to specify the number of partitions by hand. Both methods are well formulated and very powerful. However, these Bayesian nonparametric inference approaches suffer from the problems of expensive computation and local approximations.

### 3.2.3 Black-box Inverse Optimal Control

Black-box optimization approaches are another category of IOC algorithms. They usually consist of a hierarchical optimization procedure with two layers, where in the outer loop black-box optimization methods are employed to find suitable parameters of the inner motion problem. The outer loop does not require a gradient of the cost function, which makes it very flexible in the definition of costs. Mombaur et al. (2010) presented such a two-layered approach, where the outer loop consists of a derivative free trust region optimization technique and the inner loop of a direct multiple shooting technique. The fitness function of their outer loop is the squared distance between inner loop solution and demonstrations. They evaluated it on a human locomotion task where demonstrations from humans are recorded and transferred to a humanoid robot by learning a cost function. Doerr et al. (2015) proposed to do policy search on a reward function that measures similarity to the demonstrations. They apply covariance matrix adaptation evolution strategy to learn parameters of a trajectory optimization problem that is similar to KOMO. The advantage of their method is that it can handle a wide range of potential search parameters and black-box objectives. However, such methods usually have large computational costs for high-dimensional spaces since the black-box optimizer needs many evaluations. Their experimental evaluation for pointing tasks shows that the algorithm requires between 2000 and 4000 evaluations of the forward problem. One also needs to find a cost function for the outer loop that leads to reasonable behavior. In the Inverse KKT problem formulation, we do not require evaluations of the forward problem and the inverse cost function is given by the KKT conditions.



A hierarchical combination of analytic IOC and black-box IOC could also be worth studying, in which the analytic method optimizes the linear parameters and the black-box method optimizes the non-linear parameters of the cost function. We compare a two-layered imitation learning approach with Inverse KKT on a box sliding task in Section 8.1.4.

### 3.2.4 Task Space Extraction

Jetchev and Toussaint (2014) introduced a method that discovers task relevant features by training a specific kind of value function, assuming that demonstrations can be modeled as down-hill walks of this function. Similar to Inverse KKT, the function is modeled as linear in several potential task spaces, allowing to extract the one most consistent with demonstrations. In (Muhlig et al., 2009), the relevant task spaces are automatically selected from demonstrations. Therefore, the demonstrations are mapped on a set of predefined task spaces, which is then searched for the task spaces that best represent the movement. In contrast to these methods, Inverse KKT more rigorously extracts task features in the Inverse KKT motion optimization framework, including motions that involve contacts.

### 3.2.5 Direct Imitation Learning

A more direct approach of imitation learning is the idea of using movement primitives (Schaal et al., 2003; Paraschos et al., 2013; Pastor et al., 2011) to represent demonstrations. They do not try to estimate the cost function of the demonstration. Instead they represent the demonstrations in a parametrized form that should generalize to new situations (e.g., changing duration of motion, adapting the target). Many extensions with different parametrization exist that try to generalize to more complex scenarios (Calinon et al., 2013; Stulp et al., 2013a). This kind of movement representation are very efficient to learn from demonstrations and has previously been used for manipulation tasks. The advantage over IOC is that the learning algorithms are often simpler and faster. However, a cost function usually provides better generalization abilities than movement primitives. This is the case since cost functions are a more abstract representation of task knowledge. Examples of such generalization abilities are demonstrated in Section 8.1.4 on a box sliding task for which a cost function is learned from demonstrations. This cost function is

optimized to generate motions for various initial and final box locations.

### 3.3 Environment Model Learning

An important factor in skill learning is the transfer of a skill between different environments. Robots are equipped with different sensors that allow them to perceive their environment in various forms (e.g., image, point cloud). In Chapter 7, we propose kinematic morphing networks (KMNs) that map depth images to environment parameter. In this chapter, related work about extracting models of the environment from sensor data is presented.

#### 3.3.1 Extracting 3D Models from Data

Point set registration methods try to find the transformation between two observations of a model (Chen and Medioni, 1992; Gold et al., 1998). Iterative closest point (ICP) is a common algorithm to align two point clouds (Chen and Medioni, 1992). ICP iterates between the three steps: 1) Matching the points between the two point clouds by finding the closest pairs; 2) Computing a transformation that minimizes the distances between the point pairs; 3) Applying the transformation on one of the point clouds and continuing with step 1. The advantage of these kind of methods is that they do not require an expensive training procedures such as neural networks. However, they still have open parameters that influence the performance and the initial estimate of the transformation is important. The main difference to the work in this thesis is that KMN can also handle kinematic model parameters beyond affine transformations. We compare KMN to ICP in the experiment in Section 8.4.1.

An alternative strategy is to extract models from motions of the environment (Sturm et al., 2011; Martin-Martin and Brock, 2014). Martin-Martin and Brock (2014) do a feature based approach by tracking the motion of different feature points and extracting a kinematic model from them. The work by Sturm et al. (2011) follows a probabilistic approach to extract a kinematic model from pose trajectories of rigid bodies. The objective of these algorithms is similar to KMN. The main difference is that we do not require object motions, which are difficult to produce in an automated way when the environment is initially unknown.

Zhou et al. (2016) proposed to learn a deep neural network that predicts the configuration of a kinematic hand model from depth images. The forward kinematic function is integrated as final layer into the network and outputs the location of each joint. The loss function is defined on the location of these joints and an additional loss term ensures that the predicted parameters fulfill physical constraints. In KMN, we directly compute the error in the parameter space and also use the network to predict configuration parameters of the model structure.

### 3.3.2 Learning in Simulated Environments

The usage of simulation environments is a way to bypass the lack of large supervised datasets in robotics. However, bridging the gap between simulated and real sensor data (e.g., images, point clouds) is still an open research question. In (Bousmalis et al., 2017a), a robot grasping skill is improved by using synthetic data generated with a simulator. Their approach relies on domain adaptation techniques (Bousmalis et al., 2017b; Ganin et al., 2016) that map synthetic images to realistically looking images. They learn this mapping with a generative adversarial network (Goodfellow et al., 2014), which consists of two networks that are trained adversarially. The generated dataset is taken to train a network that maps RGB images and actions to grasp success probabilities (Levine et al., 2016). The results show that the incorporation of simulation data leads to a consistent improvement of the overall grasp success rate. The work by Rusu et al. (2017) transfers policies from simulated to real robots by using progressive neural networks (Rusu et al., 2016). The first column of the progressive neural network is trained in simulation. All further layers are trained on the real robot while the first column is kept fixed. The training is done with the asynchronous advantage actor-critic method (Mnih et al., 2016). The input to the network is an RGB image and the outputs are a discrete velocity signal for each joint plus a value function. Instead of following an end-to-end approach, we propose a more structured way of transferring skills by extracting a representation that is suitable as input for standard planning methods.

Mitash et al. (2017) presented an object detection algorithm based on a physics simulation and a real robot self learning mechanism. The physics simulator takes CAD models of the objects to generate realistically looking scenes. Each scene is rendered from multiple perspectives and the produced RGB-D images are taken to train a convolutional neural network. The trained network is applied to generate

more labeled training data in a real world environment. In this step, a robot arranges the scene and observes multiple images from different perspectives. The objects that are detected with high confidence are selected to create corresponding labels for all perspectives. Their results show that the simulation part of the algorithm provides the policy with a good starting point for the real robot self learning. Similar to this approach, we also apply the network to generate more training data by using the concatenation property of affine transformations.

### 3.3.3 Integrating 3D Geometry in Neural Networks

There are different ways how neural networks can be designed to handle 3D sensor data and how transformations or rendering operations can be represented within a network. A problem that often occurs is that the observations are taken from a specific viewpoint, which leads to only partial observations of objects. The work by Eitel et al. (2015) does object detection based on RGB-D data. The model consists of a two-stream convolutional neural network with an RGB image input and a depth image input. The output is a probability of how likely an object is detected in the image. They compare different ways to represent depth images and to transfer pre-trained networks trained on RGB data. Byravan and Fox (2017) proposed to learn rigid body motions with deep neural networks based on depth data. The model inputs are a point cloud shaped as an XYZ image and a force vector. The network combines both inputs in a late fusion architecture and outputs a transformed point cloud image. The transformation is done by predicting a fixed amount of object masks and corresponding rigid body transformations.

Spatial Transformer Networks (STNs) are a network module to transform an input feature map to an output feature map (Jaderberg et al., 2015). STNs are differentiable and can be put as a layer in a network. The parameters of the affine transformation are predicted based on the input feature map. Using STNs leads to invariance regarding translation, scale, and rotation. Rezende et al. (2016) applied STNs to extract 3D structure from images. A conditional latent variable model with a low-dimensional codec is used to map observed data to an abstract code that a decoder maps to volume representations. Similar to KMN, they also convert 3D representations to images with an OpenGL renderer. In KMN, the low-dimensional representation are interpretable kinematic parameters that are suitable as inputs for planning methods. Discretizing the 3D space may be possible for certain tasks

such as object detection. However, the achieved accuracy strongly depends on the resolution of the grid.



# Chapter 4

## Inverse KKT

In this chapter, we present the inverse optimal control method Inverse KKT (IKKT) that extracts a cost function from a low amount of demonstrations. The goal of IKKT is to capture the essential features of the demonstrated task in a cost function that is generalizable to various environment configurations. For this purpose, IKKT assumes that the demonstrations are optimal with respect to an underlying constrained optimization problem. Based on this assumption, we derive an inverse optimal control objective that results in a constrained quadratic program for linear cost function parametrizations. Afterwards, we describe various regularization types, weight parametrizations, and a nonparametric IKKT variant.

### 4.1 Inverse KKT Objective

We now present the Inverse KKT objective (Englert et al., 2017), which is a way to solve the inverse problem for the constrained trajectory optimization formulation KOMO (see Section 2.1). To this end, the weight vector  $\mathbf{w}$  in Equation (2.5) is learned from demonstrations. We assume that  $D$  demonstrations of a skill are provided with the robot body (e.g., through teleoperation or kinesthetic teaching) and are given in the form  $\mathcal{D} = \{(\bar{\mathbf{x}}^{(d)}, \mathbf{y}^{(d)})\}_{d=1}^D$ , where  $\bar{\mathbf{x}}^{(d)}$  is the demonstrated trajectory and  $\mathbf{y}^{(d)}$  is the environment configuration (e.g., object position). Another assumption we make is that the constraints  $\mathbf{g}(\bar{\mathbf{x}}, \mathbf{y})$  and  $\mathbf{h}(\bar{\mathbf{x}}, \mathbf{y})$  as well as a set of potential features  $\Phi(\bar{\mathbf{x}}, \mathbf{y})$  are provided as input. Inverse KKT learns the weight vector  $\mathbf{w}$  of these features from the demonstrations. Figure 4.1 shows the concept of skill learning with Inverse KKT and KOMO.

The IOC objective is derived from the Lagrange function of the problem in Equation (2.5)

$$L(\bar{\mathbf{x}}, \mathbf{y}, \boldsymbol{\lambda}, \mathbf{w}) = \mathcal{F}(\bar{\mathbf{x}}, \mathbf{y}, \mathbf{w}) + \boldsymbol{\lambda}^\top \underbrace{\begin{bmatrix} \mathbf{g}(\bar{\mathbf{x}}, \mathbf{y}) \\ \mathbf{h}(\bar{\mathbf{x}}, \mathbf{y}) \end{bmatrix}}_{c(\bar{\mathbf{x}}, \mathbf{y})} \quad (4.1)$$

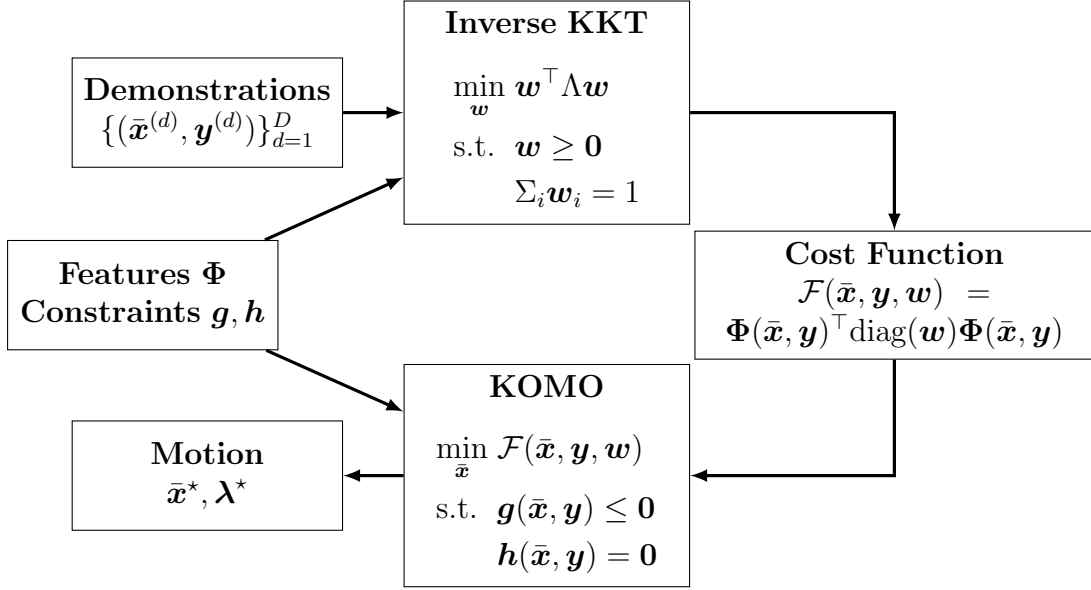


Figure 4.1: Concept of skill learning with inverse optimal control, in which the cost function plays the central role of encoding the demonstrated behavior. In this work, we present the Inverse KKT formulation of learning a cost function for a k-order Markov optimization problem.

and the Karush-Kuhn-Tucker (KKT) conditions (Kuhn and Tucker, 1951; Karush, 1939). The first KKT condition states that for an optimal solution  $\bar{\mathbf{x}}^*$  the equation

$$\nabla_{\bar{\mathbf{x}}} L(\bar{\mathbf{x}}^*, \mathbf{y}, \lambda, \mathbf{w}) = \mathbf{0} \quad (4.2)$$

has to be fulfilled. With Equation (2.6) this leads to

$$2\mathbf{J}(\bar{\mathbf{x}}, \mathbf{y})^\top \text{diag}(\mathbf{w}) \Phi(\bar{\mathbf{x}}, \mathbf{y}) + \mathbf{J}_c(\bar{\mathbf{x}}, \mathbf{y})^\top \lambda = \mathbf{0} \quad (4.3)$$

where the matrix  $\mathbf{J}_c$  is the Jacobian of all constraints  $\mathbf{c}$ . We assume that the demonstrations are optimal and should fulfill this condition. Therefore, the IOC problem can be viewed as searching for a parametrization  $\mathbf{w}$  such that this condition is fulfilled for all the demonstrations  $\bar{\mathbf{x}}$ .

The idea is expressed in the loss function

$$\ell(\mathbf{w}, \lambda) = \sum_{d=1}^D \ell^{(d)}(\mathbf{w}, \lambda^{(d)}) \quad (4.4)$$

with

$$\ell^{(d)}(\mathbf{w}, \lambda^{(d)}) = \left\| \nabla_{\bar{\mathbf{x}}} L(\bar{\mathbf{x}}^{(d)}, \mathbf{y}^{(d)}, \lambda^{(d)}, \mathbf{w}) \right\|^2, \quad (4.5)$$



where we sum over  $D$  demonstrations of the scalar product of the first KKT condition. In Equation (4.4),  $d$  enumerates the demonstrations and  $\boldsymbol{\lambda}^{(d)}$  is the dual to the demonstration  $\bar{\boldsymbol{x}}^{(d)}$  under the problem defined by  $\boldsymbol{w}$ . Note that the dual demonstrations are initially unknown and, of course, depend on the underlying cost function  $\mathcal{F}$ . More precisely,  $\boldsymbol{\lambda}^{(d)} = \boldsymbol{\lambda}^{(d)}(\bar{\boldsymbol{x}}^{(d)}, \boldsymbol{y}^{(d)}, \boldsymbol{w})$  is a function of the primal demonstration  $\bar{\boldsymbol{x}}^{(d)}$ , the environment configuration of that demonstration  $\boldsymbol{y}^{(d)}$ , and the underlying parameters  $\boldsymbol{w}$ . As a result  $\ell^{(d)}(\boldsymbol{w}, \boldsymbol{\lambda}^{(d)}(\boldsymbol{w})) = \ell^{(d)}(\boldsymbol{w})$  becomes a function of the parameters only (we think of  $\bar{\boldsymbol{x}}^{(d)}$  and  $\boldsymbol{y}^{(d)}$  as given, fixed quantities, as in Equations (4.4)–(4.5)).

Given that we want to minimize  $\ell^{(d)}(\boldsymbol{w})$ , we can substitute  $\boldsymbol{\lambda}^{(d)}(\boldsymbol{w})$  for each demonstration by choosing the dual solution that analytically minimizes  $\ell^{(d)}(\boldsymbol{w})$  subject to the KKT's complementarity condition

$$\nabla_{\boldsymbol{\lambda}^{(d)}} \ell^{(d)}(\boldsymbol{w}, \boldsymbol{\lambda}^{(d)}) = \mathbf{0} \quad (4.6)$$

$$\Rightarrow \boldsymbol{\lambda}^{(d)}(\boldsymbol{w}) = -(\tilde{\boldsymbol{J}}_c \tilde{\boldsymbol{J}}_c^\top)^{-1} \tilde{\boldsymbol{J}}_c \boldsymbol{J}^\top \text{diag}(\boldsymbol{\Phi}) \boldsymbol{w} . \quad (4.7)$$

Note that here the matrix  $\tilde{\boldsymbol{J}}_c$  is a subset of the full Jacobian of the constraints  $\boldsymbol{J}_c$  that contains only the active constraints during the demonstration, which we can evaluate as  $\boldsymbol{g}$  and  $\boldsymbol{h}$  are independent of  $\boldsymbol{w}$ . This ensures that Equation (4.7) is the minimizer subject to the complementarity condition. The number of active constraints at each time point has a limit. This limit would be exceeded if more degrees of freedom of the system are constrained than available.

By inserting Equation (4.7) into Equation (4.5), we get

$$\begin{aligned} \ell^{(d)}(\boldsymbol{w}) &= 4\boldsymbol{w}^\top \text{diag}(\boldsymbol{\Phi}) \boldsymbol{J} \boldsymbol{J}^\top \text{diag}(\boldsymbol{\Phi}) \boldsymbol{w} \\ &\quad + 4\boldsymbol{w}^\top \text{diag}(\boldsymbol{\Phi}) \boldsymbol{J} \tilde{\boldsymbol{J}}_c^\top (\tilde{\boldsymbol{J}}_c \tilde{\boldsymbol{J}}_c^\top)^{-1} (\tilde{\boldsymbol{J}}_c \tilde{\boldsymbol{J}}_c^\top) (\tilde{\boldsymbol{J}}_c \tilde{\boldsymbol{J}}_c^\top)^{-1} \tilde{\boldsymbol{J}}_c \boldsymbol{J}^\top \text{diag}(\boldsymbol{\Phi}) \boldsymbol{w} \\ &\quad - 8\boldsymbol{w}^\top \text{diag}(\boldsymbol{\Phi}) \boldsymbol{J} \tilde{\boldsymbol{J}}_c^\top (\tilde{\boldsymbol{J}}_c \tilde{\boldsymbol{J}}_c^\top)^{-1} \tilde{\boldsymbol{J}}_c \boldsymbol{J}^\top \text{diag}(\boldsymbol{\Phi}) \boldsymbol{w} \\ &= 4\boldsymbol{w}^\top \text{diag}(\boldsymbol{\Phi}) \boldsymbol{J} \boldsymbol{J}^\top \text{diag}(\boldsymbol{\Phi}) \boldsymbol{w} \\ &\quad - 4\boldsymbol{w}^\top \text{diag}(\boldsymbol{\Phi}) \boldsymbol{J} \tilde{\boldsymbol{J}}_c^\top (\tilde{\boldsymbol{J}}_c \tilde{\boldsymbol{J}}_c^\top)^{-1} \tilde{\boldsymbol{J}}_c \boldsymbol{J}^\top \text{diag}(\boldsymbol{\Phi}) \boldsymbol{w} \\ &= 4\boldsymbol{w}^\top \text{diag}(\boldsymbol{\Phi}) \boldsymbol{J} \underbrace{\left( \boldsymbol{I} - \tilde{\boldsymbol{J}}_c^\top (\tilde{\boldsymbol{J}}_c \tilde{\boldsymbol{J}}_c^\top)^{-1} \tilde{\boldsymbol{J}}_c \right)}_{\boldsymbol{\Lambda}^{(d)}} \boldsymbol{J}^\top \text{diag}(\boldsymbol{\Phi}) \boldsymbol{w} , \end{aligned} \quad (4.8)$$

which is the IOC cost per demonstration. Adding up the loss per demonstration

---

**Algorithm 1** Inverse KKT

---

**inputs:**

Demonstrations  $\mathcal{D} = \{(\bar{\mathbf{x}}^{(d)}, \mathbf{y}^{(d)})\}_{d=1}^D$   
 Set of features  $\Phi$   
 Constraints  $\mathbf{g}, \mathbf{h}$

**optimize:**

$$\begin{aligned} \min_{\mathbf{w}} \quad & \mathbf{w}^\top \Lambda \mathbf{w} \\ \text{s.t.} \quad & \mathbf{w} \geq \mathbf{0} \\ & \sum_i \mathbf{w}_i = 1 \end{aligned} \tag{4.12}$$

**output:**

Optimal feature weights  $\mathbf{w}^*$

---

and plugging them into Equation (4.4) gives the total Inverse KKT loss

$$\ell(\mathbf{w}) = \mathbf{w}^\top \Lambda \mathbf{w} \tag{4.9}$$

$$\text{with } \Lambda = 4 \sum_{d=1}^D \Lambda^{(d)}. \tag{4.10}$$

The resulting optimization problem is

$$\begin{aligned} \min_{\mathbf{w}} \quad & \mathbf{w}^\top \Lambda \mathbf{w} \\ \text{s.t.} \quad & \mathbf{w} \geq \mathbf{0} \end{aligned} \tag{4.11}$$

Note that we constrain the parameters  $\mathbf{w}$  to be positive. This reflects that we want squared cost features to only positively contribute to the overall cost in Equation (2.2). The proposed approach also works in the unconstrained case. In this case, the constraint term vanishes in Equation (4.3) and the remaining part is the optimality condition of unconstrained optimization, which says that the gradient of the cost function should be equal to zero.

## 4.2 Regularization & Sparsity

The above formulation may lead to the singular solution  $\mathbf{w} = \mathbf{0}$  where zero costs are assigned to all demonstrations, trivially fulfilling the KKT conditions. This calls

for a regularization of the problem. In principle, there are two ways to regularize the problem to enforce a non-singular solution: First, we can impose positive-definiteness of Equation (2.2) at the demonstrations (cf. (Levine and Koltun, 2012)). Second, as the absolute scaling of Equation (2.2) is arbitrary, we may additionally add a constraint

$$\sum_i \mathbf{w}_i = 1 \quad (4.13)$$

to the problem formulation in Equation (4.11). We select the latter option for Inverse KKT and summarize it in Algorithm 1. The overall optimization problem in Equation (4.12) is a (convex) quadratic program (QP), for which efficient solvers exist. The gradient  $\mathbf{w}^\top \Lambda$  and Hessian  $\Lambda$  are structured and sparse, which we exploit in the implementation. There exist different ways to modify the problem so that the solutions become sparse. One possibility is to subtract the regularization term  $\mathbf{w}^\top \mathbf{w}$  from the IOC loss function in Equation (4.12). Another possibility to achieve sparse solutions is to change the equality constraint into  $\sum_i \mathbf{w}_i^p = 1$  with a  $p > 2$ . However, in this case the problem is not convex anymore.

IKKT is evaluated in Section 8.1 on a synthetic benchmark problem and compared to state-of-the-art imitation learning methods. In Section 6.2, we describe a generic set of features and constraints that can be used to represent manipulation skills. Their performance with Inverse KKT is presented on various manipulation tasks in simulation and on a real robot.

### 4.3 Weight Parametrizations

In practice, we usually define parametrizations on  $\mathbf{w}$ . This is helpful since in the extreme case, when for each time step a different parameter is used, this leads to a very high-dimensional parameter space (e.g., 10 tasks and 300 time steps lead to 3000 parameters). This space can be reduced by using the same weight parameter over all time steps or to activate a task only at certain time points. The simplest variant is to define a linear parametrization  $\mathbf{w}(\boldsymbol{\rho}) = A\boldsymbol{\rho}$ , where  $\boldsymbol{\rho}$  are the parameters that the IOC method learns. This parametrization allows a flexible assignment of one parameter to multiple task costs. Further linear parametrizations are radial basis functions or B-spline basis functions over time to more compactly describe

smoothly varying cost parameters. For such linear parametrization the problem in Equation (4.12) remains a QP that can be solved very efficiently.

Another option we will consider in the evaluations is to define a non-linear mapping  $\mathbf{w}(\boldsymbol{\rho}) = \mathcal{A}(\boldsymbol{\rho})$  to more compactly represent all parameters (see Section 8.1.2). For instance, the parameters  $\mathbf{w}$  can be of a Gaussian shape (as a function of time  $t$ ), where the mean and variance of the Gaussian is described by  $\boldsymbol{\rho}$ . Such a parametrization would allow the algorithm to learn directly the time point when costs should be active. In this case, the problem is not convex anymore. We address such problems by using a general non-linear programming method (again, augmented Lagrangian) and doing multiple restarts with different initializations of the parameter.

## 4.4 Nonparametric Inverse KKT

We present a nonparametric variant of the Inverse KKT method. The difference to the previously presented parametric variant is that a kernel function measures the similarity to the demonstrations and no feature set has to be constructed by hand. In Equation (2.2), the objective function  $\mathcal{F}$  is represented as weighted sum of squared features  $\Phi(\bar{\mathbf{x}}, \mathbf{y})$  that are defined on the robot trajectory  $\bar{\mathbf{x}}$  and the environment  $\mathbf{y}$ . In nonparametric Inverse KKT, we assume that the environment  $\mathbf{y}$  is contained in the state  $\mathbf{x}_t$  and represent the objective function as

$$\mathcal{F}(\bar{\mathbf{x}}) = \sum_{t=1}^T \mathcal{C}_t(\mathbf{x}_t) \quad (4.14)$$

where each  $\mathcal{C}_t$  is a functional in a reproducing kernel Hilbert space (RKHS)  $\mathcal{H}$  with a reproducing kernel  $k$ . Similar to the previous parametric formulation, we define  $\mathcal{C}_t$  as

$$\mathcal{C}_t(\mathbf{x}_t) = \mathbf{w}_t^\top \boldsymbol{\psi}(\mathbf{x}_t) \quad (4.15)$$

where  $\boldsymbol{\psi}(\mathbf{x}_t) = \phi^2(\mathbf{x}_t)$ . According to the representer theorem (Schölkopf and Smola, 2002), the parameter vector  $\mathbf{w}_t$  can be represented with the demonstrations as

$$\mathbf{w}_t = \sum_{d=1}^D \alpha_t^{(d)} \boldsymbol{\psi}(\mathbf{x}_t^{(d)}) . \quad (4.16)$$

Hence, the function  $\mathcal{C}_t$  in RKHS is defined as

$$\begin{aligned}\mathcal{C}_t(\mathbf{x}_t) &= \sum_{d=1}^D \alpha_t^{(d)} \langle \boldsymbol{\psi}(\mathbf{x}_t^{(d)}), \boldsymbol{\psi}(\mathbf{x}_t) \rangle \\ &= \sum_{d=1}^D \alpha_t^{(d)} k(\mathbf{x}_t^{(d)}, \mathbf{x}_t)\end{aligned}\quad (4.17)$$

with a kernel  $k$  and coefficients  $\alpha_t^{(d)} \in \mathbb{R}$ . This means, we can define a kernel to represent the cost function and that the search for  $\mathcal{C}_t$  is equal to directly searching for optimal coefficients  $\alpha_t^{(d)}$ .

Similar to the derivation of the Inverse KKT loss function in Equations (4.4)–(4.8), we apply the KKT conditions to formulate the loss function in the nonparametric case. In case of the squared exponential kernel (cf. Equation (2.11)), the gradient of the objective function is

$$\nabla_{\bar{\mathbf{x}}} \mathcal{F}(\bar{\mathbf{x}}) = \sum_{t=1}^T \nabla_{\bar{\mathbf{x}}} \mathcal{C}_t(\mathbf{x}_t) \quad (4.18)$$

$$= \left[ \frac{\partial \mathcal{C}_1(\mathbf{x}_1)}{\partial \mathbf{x}_1}, \dots, \frac{\partial \mathcal{C}_T(\mathbf{x}_T)}{\partial \mathbf{x}_T} \right]^\top \quad (4.19)$$

with

$$\frac{\partial \mathcal{C}_t(\mathbf{x}_t)}{\partial \mathbf{x}_t} = - \sum_{d=1}^D 2\alpha_t^{(d)} k(\mathbf{x}_t, \mathbf{x}_t^{(d)}) (\mathbf{x}_t - \mathbf{x}_t^{(d)})^\top \boldsymbol{\Sigma}^{-1}. \quad (4.20)$$

The resulting loss function for a demonstration is

$$\ell^{(d)}(\boldsymbol{\alpha}) = \nabla_{\bar{\mathbf{x}}} \mathcal{F}^\top \left( \mathbf{I} - \tilde{\mathbf{J}}_c^\top (\tilde{\mathbf{J}}_c \tilde{\mathbf{J}}_c^\top)^{-1} \tilde{\mathbf{J}}_c \right) \nabla_{\bar{\mathbf{x}}} \mathcal{F} \quad (4.21)$$

$$= \boldsymbol{\alpha}^\top \boldsymbol{\Omega}^{(d)} \boldsymbol{\alpha} \quad (4.22)$$

where  $\boldsymbol{\Omega}^{(d)}$  contains all the terms that are independent of  $\boldsymbol{\alpha}$ . Similar to the previous section, we sum over all demonstrations and add a regularization term that results in the nonparametric IKKT optimization problem

$$\begin{aligned}\min_{\boldsymbol{\alpha}} \quad & \boldsymbol{\alpha}^\top \boldsymbol{\Omega} \boldsymbol{\alpha} \\ \text{s.t.} \quad & \sum_i \alpha_i = 1\end{aligned}\quad (4.23)$$

with  $\boldsymbol{\Omega} = \sum_{d=1}^D \boldsymbol{\Omega}^{(d)}$ . This problem can be optimized very efficiently and leads to a unique solution. The performance in the nonparametric case strongly depends

on the choice of a suitable kernel  $k$  and its hyperparameters. In practice, cross-validation on a test and training set, hyperparameter learning, or multiple kernel learning methods would be necessary to solve this problem (Gönen and Alpaydın, 2011). In Section 8.1.1, we present a comparison between the parametric and the nonparametric variants of Inverse KKT on a 2D planning problem.



# Chapter 5

## Combined Optimization and Reinforcement Learning

In the previous chapter, we proposed the algorithm IKKT that learns a skill by imitating human demonstrations. IKKT is sample-efficient and leads to generic skills, but it does not necessarily lead to skills with a higher performance than the demonstrations. In this chapter, we address this issue and present the algorithm combined optimization and reinforcement learning (CORL), which autonomously improves a skill with respect to different objective criteria (Englert and Toussaint, 2016). CORL is a structured learning algorithm that starts with a single demonstration and improves a skill by combining finite horizon optimal control and episodic reinforcement learning.

### 5.1 Optimal Control & Reinforcement Learning

Optimal control (Bertsekas, 2001) and reinforcement learning (Sutton and Barto, 1998) have a similar goal of finding an optimal policy of a system that maximizes a certain objective. In optimal control, we assume that the transition model of the system is known, as well as the objective function. In the case of linear dynamics, quadratic costs and Gaussian noise (LQG), the problem can be solved analytically (Athans, 1971). In the non-linear case one approach is a Laplace approximation, which first computes the most likely path leveraging classical motion optimization (e.g., KOMO (Toussaint, 2017)) and then approximates and solves the LQG problem around the path (e.g., iterative LQG (Todorov and Li, 2005)). Optimal control works well as long the system model and objective function are known analytically. However, this is rarely the case, since dynamics are very complex and the objectives cannot be defined analytically. Reinforcement learning is another approach that does not assume to know the system model and objective function. In contrast to optimal control, the system behavior as well as rewards can only be observed by doing rollouts on the real system. See Section 3.1 for related work on reinforcement



learning.

In the following formulation, we combine the two approaches into a single algorithm CORL. The main idea behind this algorithm is to exploit the benefits of a transition model and analytic cost function when they are available and the flexibility of black-box objectives. We specifically aim to deal with cases where the policy parameters are high-dimensional ( $> 1000$ ). However, at the same time we aim for efficient skill learning from only few ( $< 100$ ) rollouts on the real system. Clearly, for this to be a well-posed problem we need to assume a certain structure in the problem.

## 5.2 Problem Formulation

The problem formulation of CORL consists of an analytically known cost function

$$J : \mathbb{R}^n \rightarrow \mathbb{R} , \quad (5.1)$$

a  $q$ -dimensional equality constraint

$$\mathbf{h}(\bar{\mathbf{x}}, \mathbf{y}, \boldsymbol{\theta}) = \mathbf{0} \quad (5.2)$$

that ties every policy parameters  $\bar{\mathbf{x}}$  and environment configuration  $\mathbf{y}$  to a low-dimensional projection  $\boldsymbol{\theta} \in \mathbb{R}^m$ , a black-box return function

$$R : \mathbb{R}^m \rightarrow \mathbb{R} , \quad (5.3)$$

and a black-box success constraint

$$S : \mathbb{R}^m \rightarrow \{0, 1\} . \quad (5.4)$$

With these four ingredients, we define the structured learning problem

$$\begin{aligned} \min_{\bar{\mathbf{x}}, \boldsymbol{\theta}} \quad & J(\bar{\mathbf{x}}) - R(\boldsymbol{\theta}) \\ \text{s.t.} \quad & \mathbf{h}(\bar{\mathbf{x}}, \mathbf{y}, \boldsymbol{\theta}) = \mathbf{0} \\ & S(\boldsymbol{\theta}) = 1 . \end{aligned} \quad (5.5)$$

That is, we want the best policy parameters  $(\bar{\mathbf{x}}^*, \boldsymbol{\theta}^*)$  (measured with  $J(\bar{\mathbf{x}})$  and  $R(\boldsymbol{\theta})$ ) that fulfill a skill (measured with  $S(\boldsymbol{\theta}) = 1$ ). In contrast to the standard optimal control and reinforcement learning problems, which only optimize a single

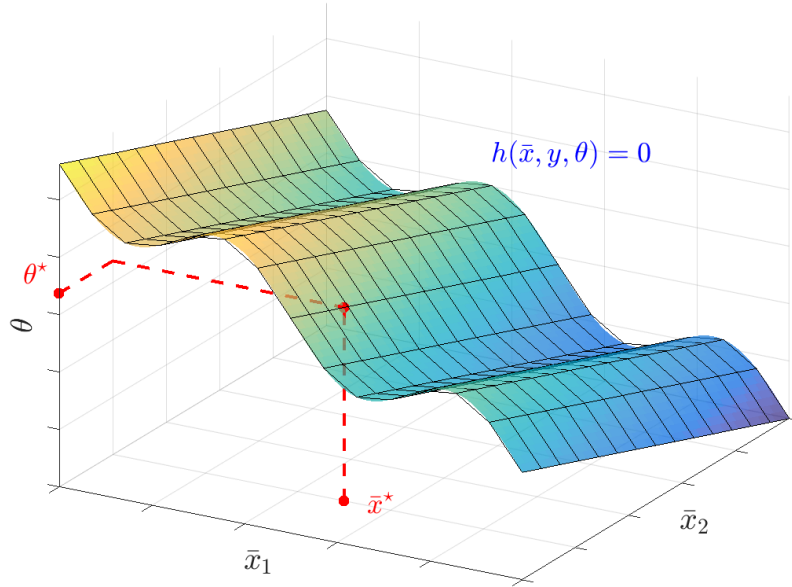


Figure 5.1: Projection of a two-dimensional  $\bar{\mathbf{x}}$  to a one dimensional  $\boldsymbol{\theta}$  with a projection constraint  $\mathbf{h}(\bar{\mathbf{x}}, \mathbf{y}, \boldsymbol{\theta}) = \mathbf{0}$ .

objective function, this formulation splits the objective in an analytic part  $J(\bar{\mathbf{x}})$ , a black-box part  $R(\boldsymbol{\theta})$ , and a black-box success constraint  $S(\boldsymbol{\theta})$ . The analytic cost function  $J(\bar{\mathbf{x}})$  contains all the costs we know a priori in analytic form. The black-box return function  $R(\boldsymbol{\theta})$  and success constraint  $S(\boldsymbol{\theta})$  are a priori unknown and are only observable in form of noisy samples by doing rollouts for a given input.

The equality constraint  $\mathbf{h}(\bar{\mathbf{x}}, \mathbf{y}, \boldsymbol{\theta}) = \mathbf{0}$  is incorporated to define the relation between the high-dimensional  $\bar{\mathbf{x}}$  and the environment  $\mathbf{y}$  to the lower dimensional  $\boldsymbol{\theta}$ . We assume that  $\mathbf{h}$  is smooth and that, for given  $\bar{\mathbf{x}}$  and  $\mathbf{y}$ ,  $\mathbf{h}(\bar{\mathbf{x}}, \mathbf{y}, \boldsymbol{\theta}) = \mathbf{0}$  identifies a unique  $\boldsymbol{\theta}(\bar{\mathbf{x}}, \mathbf{y}) = \boldsymbol{\theta}$ . In that sense,  $\boldsymbol{\theta}$  is a projection of  $\bar{\mathbf{x}}$  and  $\mathbf{y}$ . Figure 5.1 shows a simple example of such a projection. This projection is formulated in terms of an equality constraint so that, for given  $\boldsymbol{\theta}$  and environment  $\mathbf{y}$ , the remaining problem on  $\bar{\mathbf{x}}$  is a standard constrained optimization problem. In the application domain of robot manipulation skills, we employ the low-dimensional projection  $\boldsymbol{\theta}$  to parametrize the interactions with the environment. The details on how  $\mathbf{h}$  is defined in practice are described in Section 6.1.

---

**Algorithm 2** CORL

---

**inputs:**

Demonstration  $(\bar{\mathbf{x}}^{(0)}, \boldsymbol{\theta}^{(0)}, \mathbf{y})$

Analytic cost:  $J(\bar{\mathbf{x}})$

Black-box return:  $R(\boldsymbol{\theta})$

Black-box success:  $S(\boldsymbol{\theta})$

Projection constraint:  $\mathbf{h}(\bar{\mathbf{x}}, \mathbf{y}, \boldsymbol{\theta})$

Init dataset  $\mathcal{D} = \{(\bar{\mathbf{x}}^{(0)}, \boldsymbol{\theta}^{(0)}, R(\bar{\mathbf{x}}^{(0)}), J(\boldsymbol{\theta}^{(0)}), S(\boldsymbol{\theta}^{(0)}))\}$

**repeat:**

Reinforcement learning of  $R(\boldsymbol{\theta})$  and  $S(\boldsymbol{\theta})$ : (see Section 5.3.1)

$$\boldsymbol{\theta}^* = \arg \max_{\boldsymbol{\theta}} a_{\text{PIBU}}(\boldsymbol{\theta}; \mathcal{D})$$

Motion optimization for constrained  $\boldsymbol{\theta}$ : (see Section 5.3.2)

$$\begin{aligned} \bar{\mathbf{x}}^* &= \arg \min_{\bar{\mathbf{x}}} J(\bar{\mathbf{x}}) \\ \text{s.t.} \quad &\mathbf{h}(\bar{\mathbf{x}}, \mathbf{y}, \boldsymbol{\theta}^*) = \mathbf{0} \end{aligned}$$

Perform rollout with policy parameters  $\bar{\mathbf{x}}^*$

Add  $(\bar{\mathbf{x}}^*, \boldsymbol{\theta}^*, J(\bar{\mathbf{x}}^*), R(\boldsymbol{\theta}^*), S(\boldsymbol{\theta}^*))$  to  $\mathcal{D}$

**until** no change in policy parameter

**output:**

Dataset  $\mathcal{D}$

---

## 5.3 Combining Motion Optimization and Reinforcement Learning

We solve the problem in Equation (5.5) by using optimal control methods to improve the policy w.r.t. the high-dimensional  $\bar{\mathbf{x}}$  and black-box Bayesian optimization to improve the policy w.r.t. the low-dimensional  $\boldsymbol{\theta}$ . A summary of the policy update steps of CORL can be found in Algorithm 2. We assume to have an initial policy parametrization  $(\bar{\mathbf{x}}^{(0)}, \boldsymbol{\theta}^{(0)})$  that fulfills the skill ( $S(\boldsymbol{\theta}^{(0)}) = 1$ ) as input. Further inputs are the objectives and constraints of the problem in Equation (5.5). The dataset is initialized with these parameters and their objective values  $R(\bar{\mathbf{x}}^{(0)})$ ,  $J(\boldsymbol{\theta}^{(0)})$ , and  $S(\boldsymbol{\theta}^{(0)})$ . The learning loop of CORL consists of two steps.

The first step is black-box Bayesian optimization over  $\boldsymbol{\theta}$  that aims at improving

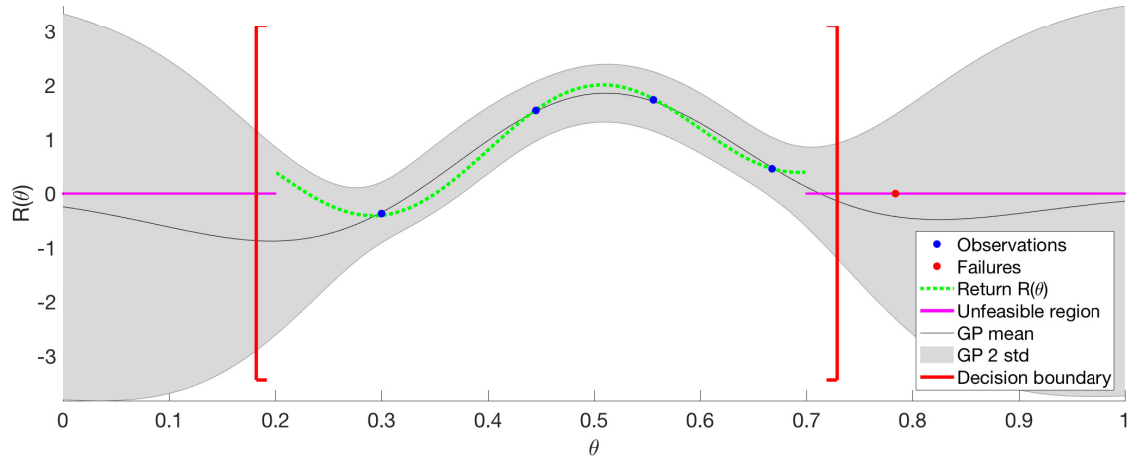
$R(\boldsymbol{\theta})$  and fulfilling the constraint  $S(\boldsymbol{\theta})$ . In Section 5.3.1, we define the acquisition function  $a_{\text{PIBU}}(\boldsymbol{\theta})$  in such a way that it explores the parameter space in a secure and data-efficient manner by finding a good tradeoff between making large steps that potentially lead to risky policies and small steps that would require many rollouts. In order to achieve this goal, we learn a binary classification model of  $S(\boldsymbol{\theta})$  that estimates the boundary between policies that lead to success or failure. This classifier is used to keep the exploration around the feasible region and reduce the number of (negative) interactions with the system.

The second step in CORL is constrained optimization and acts on the high-dimensional  $\bar{\mathbf{x}}$  to improve the analytic cost function  $J(\bar{\mathbf{x}})$ . For this we employ the constrained trajectory optimization framework KOMO (see Section 2.1). There, the low-dimensional parameters  $\boldsymbol{\theta}$  are kept fixed with the equality constraint  $\mathbf{h}(\bar{\mathbf{x}}, \mathbf{y}, \boldsymbol{\theta}) = \mathbf{0}$ . Fixing the low-dimensional parameters  $\boldsymbol{\theta}$  means that the resulting policy fulfills a certain property that is defined by the constraint. We assume that the success of a skill only depends on  $\boldsymbol{\theta}$ , which implies that all policy parameters  $\bar{\mathbf{x}}$  and environments  $\mathbf{y}$  that fulfill the constraint for a fixed  $\boldsymbol{\theta}$  lead to the same outcome (i.e., success or failure).

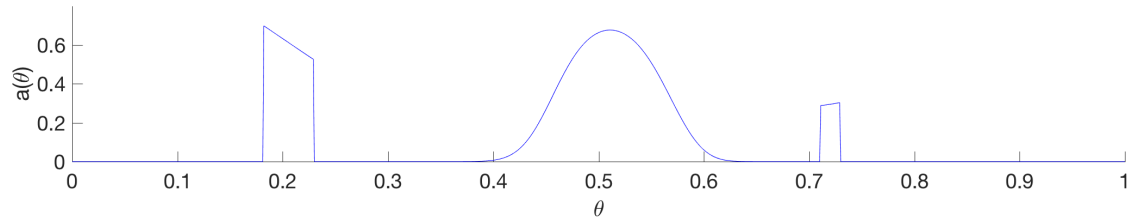
In the following two sections, first the Bayesian optimization and afterwards the motion optimization part are described in detail.

### 5.3.1 Reinforcement Learning over $\boldsymbol{\theta}$ with Unknown Success Constraints

We introduce an episodic reinforcement learning method to improve the policy with respect to the low-dimensional projection  $\boldsymbol{\theta}$ . The goal of this improvement strategy is to optimize the black-box return function  $R(\boldsymbol{\theta})$  under the success constraint  $S(\boldsymbol{\theta})$  so as to have a safe interaction with the system. We utilize Bayesian optimization (see Section 2.3) to learn a binary classifier for the success constraint  $S(\boldsymbol{\theta})$  and a regression model for the return function  $R(\boldsymbol{\theta})$ . We propose a novel acquisition function  $a_{\text{PIBU}}(\boldsymbol{\theta}; \mathcal{D})$  that combines both models in such a way that the next policy is selected in a secure and data-efficient manner. The collected dataset for this strategy is of the form  $\mathcal{D} = \{(\boldsymbol{\theta}^{(i)}, R(\boldsymbol{\theta}^{(i)}), S(\boldsymbol{\theta}^{(i)}))\}_{i=1}^D$  where  $\boldsymbol{\theta}^{(i)}$  are the parameters at iteration  $i$ ,  $R(\boldsymbol{\theta}^{(i)})$  is the observed return, and  $S(\boldsymbol{\theta}^{(i)})$  is the observed binary skill outcome. Based on this data, the next sample  $\boldsymbol{\theta}^*$  is selected by maximizing the acquisition function  $a_{\text{PIBU}}(\boldsymbol{\theta}; \mathcal{D})$ . Both function approximations are done with



(a) Return function and decision boundary



(b) Acquisition function

Figure 5.2: Example of PIBU acquisition function.

Gaussian processes (see Section 2.2). A GP  $g_R$  models the return function  $R(\boldsymbol{\theta})$  and a GP  $g_S$  is the discriminative function of a binary classifier that models the success function  $S(\boldsymbol{\theta})$ . The regression GP only contains the data points that are feasible and lead to successful motions. The classification GP contains all data points and describes the feasible regions of  $\boldsymbol{\theta}$ . This region is incrementally explored with the goal to find the best projection  $\boldsymbol{\theta}$  that maximizes  $R(\boldsymbol{\theta})$  and leads to success ( $S(\boldsymbol{\theta}) = 1$ ). In the regression model  $g_R$  we define a constant prior mean function of 0. For the classification model  $g_S$  we define a constant prior mean function  $m(\boldsymbol{\theta}) = c$  to incorporate knowledge that regions where no data points are available yet the unfeasible class is predicted. Therefore, we select a constant  $c < 0$  that allows keeping the exploration close to the region where data points are available.

We combine properties from the GPs  $g_R$  and  $g_S$  to define the acquisition function

$$a_{\text{PIBU}}(\boldsymbol{\theta}; \mathcal{D}) = [g_S(\boldsymbol{\theta}) > 0] a_{\text{PI}}(\boldsymbol{\theta}; \mathcal{D}) + [g_S(\boldsymbol{\theta}) = 0] \nabla_{g_S}(\boldsymbol{\theta}). \quad (5.6)$$

This function combines the probability of improvement with a boundary uncertainty

criteria (PIBU). In Equation (5.6)  $[\cdot]$  denotes the Iverson bracket. The first term describes the probability of improvement (cf. Equation (2.12)) of  $g_R$  in the inner region of the classifier  $g_S$ . The second term is the predictive variance of the GP classifier  $g_S$  on the decision boundary. The first term focuses on exploiting improvement inside the feasible region and the second term focuses on exploring safely on the decision boundary. Figure 5.2 shows an example of the PIBU acquisition function on a one dimensional problem. Each iteration of CORL optimizes Equation (5.6) to select the next low-dimensional parameter vector  $\theta^*$ . This optimization is implemented as gradient ascent of Equation (5.6) from multiple starting points.

### 5.3.2 Motion Optimization over $\bar{\mathbf{x}}$ with Fixed $\theta$

After the next candidate of the low-dimensional policy parameters  $\theta^*$  is selected, a backprojection to the full policy representation  $\bar{\mathbf{x}}^*$  is necessary to perform a rollout on the actual system. This backprojection is done by selecting the  $\bar{\mathbf{x}}^*$  that optimizes  $J(\bar{\mathbf{x}})$  and fulfills the constraint  $\mathbf{h}(\bar{\mathbf{x}}, \mathbf{y}, \theta^*) = \mathbf{0}$ . This leads to the optimization problem

$$\begin{aligned} \bar{\mathbf{x}}^* &= \arg \min_{\bar{\mathbf{x}}} J(\bar{\mathbf{x}}) \\ \text{s.t.} \quad & \mathbf{h}(\bar{\mathbf{x}}, \mathbf{y}, \theta^*) = \mathbf{0} . \end{aligned} \tag{5.7}$$

We employ the KOMO framework (see Section 2.1) to optimize this problem and define the analytic cost as  $J(\bar{\mathbf{x}}) = \sum_{t=0}^T \mathbf{w}_t^\top \phi_t(\tilde{\mathbf{x}}_t)^2$ . The resulting policy parameters  $(\theta^*, \bar{\mathbf{x}}^*)$  are executed on the real robot and the observed objectives  $(R(\bar{\mathbf{x}}^*), J(\theta^*), S(\theta^*))$  are added to the dataset. These steps are repeated until there is no change in the policy parameters. In the following sections, we define two cost features for the problem in Equation (5.7), which are used in the experimental evaluation of CORL.

#### Optimizing Smoothness of Unconstrained Motion

The first objective criterion measures smoothness in the robot's configuration space. We define the feature as the finite difference acceleration

$$\phi_t(\tilde{\mathbf{x}}_t) = (\mathbf{x}_t - 2\mathbf{x}_{t-1} + \mathbf{x}_{t-2})/\Delta t^2 . \tag{5.8}$$

Alternative smoothness criteria could be the torque or jerk. This feature is used in the objective function in Equation (5.7) to minimize the squared accelerations at

each time step while keeping the low-dimensional projection  $\theta$  fixed. This leads to a smoother motion of the unconstrained part of the motion, which is in the case of manipulation skills often the motion before or after contacts are established.

### Optimizing the Interaction Phase Profile

The second objective criterion that we want to achieve is a smoother motion also regarding the time course of the constraints (e.g., during the interaction phase). In order to achieve such motions, the phase of the trajectory is optimized while the path is kept fixed. In this step, we assume that the trajectory  $\bar{\mathbf{x}}$  of duration  $d$  can be evaluated at time  $t$  by interpolating it with splines. The phase profile  $p(t) : [0, d] \rightarrow [0, 1]$  of the trajectory is optimized with respect to smoothness. The phase profile  $p(t)$  is discretized in  $T + 1$  points  $\bar{p} = [p_0, p_1, \dots, p_T]$  with the boundary conditions  $p_0 = 0$  and  $p_T = 1$ . Again, the squared configuration space accelerations are used as smoothness term which results in an overall cost

$$J(\bar{p}) = \sum_{i=0}^T \left\| \frac{\bar{\mathbf{x}}(p_i d) - 2\bar{\mathbf{x}}(p_{i-1} d) + \bar{\mathbf{x}}(p_{i-2} d)}{\Delta t^2} \right\|^2 + (p_i - 2p_{i-1} + p_{i-2})^2 . \quad (5.9)$$

The first is the finite difference estimate of the acceleration at time step  $t$  and the second term is a cost term directly on the acceleration of the phase variable. The resulting phase profile  $\bar{p}^*$  defines a new trajectory  $\bar{\mathbf{x}}^*$  that is executed on the real system.

The constrained Bayesian optimization method from Section 5.3.1 was employed in (Driess et al., 2017) to learn interaction controllers for complex manipulation tasks (e.g., establishing a contact, sliding on a surface). They defined relevant parameters of a force/task space controller and used the proposed acquisition function PIBU to efficiently and safely improve a skill with respect to compliance and contact force. The evaluation of Algorithm 2 can be found in Section 8.2 where it is compared to alternative methods on a synthetic optimization problem. The performance on a real robot is also demonstrated by learning various manipulation skills (e.g., opening a door).





# Chapter 6

## Prior Knowledge for Manipulation Skill Learning

In this thesis, the algorithms IKKT (Chapter 4) and CORL (Chapter 5) are used to learn robot manipulation skills. Both algorithms allow the incorporation of prior knowledge, which we design in such a way that the inherent structure of the manipulation problem is exploited. A key element of this structure is that external degrees of freedom are manipulated through contacts. In the following sections, we propose projection constraints for CORL and cost features for IKKT that implement such prior domain knowledge. Further, we present a way to concatenate CORL and IKKT with the goal to learn generalizable manipulation skills from a single demonstration.

### 6.1 Projection Constraints for Sample-Efficient Manipulation Skill Learning

In the application domain of robot manipulation skills, we design the low-dimensional projection  $\theta$  of CORL (see Chapter 5) as interaction parameters between the robot and the environment. In this case, the optimization over  $\theta$  in Algorithm 2 learns which interactions are efficient and lead to success. This follows the assumption that the interactions are the most relevant parts to achieve success in manipulation tasks. The interactions are also often the most difficult part to integrate into the motion planning method and the analytic cost functions. Another reason behind this definition is that the interaction parameter space is much lower dimensional than the full robot motion  $\bar{x}$ , which allows the optimization method to focus on the relevant parts of the skill and converge with fewer samples. In the case of CORL, the optimization is performed with the constrained Bayesian optimization method presented in Section 5.3.1. This method optimizes the projection parameters in a safe manner, which is important for not damaging the robot or environment. Defining

the projection as interaction parameters allows to specify the black-box objectives in various forms. We propose to define the black-box return function  $R(\boldsymbol{\theta})$  as a performance measure of a manipulation (e.g., used force) and the black-box success function  $S(\boldsymbol{\theta})$  to check if a certain interaction parameter led to a successful manipulation (e.g., door is open).

A specific constraint  $\mathbf{h}(\bar{\mathbf{x}}, \mathbf{y}, \boldsymbol{\theta}) = \mathbf{0}$  that we use in the door opening experiment in Section 8.2.2, specifies the contact points of the robot gripper on the door handle. In this case, the projection constraints can be computed with robot kinematics that defines the relationship between the trajectory  $\bar{\mathbf{x}}$  and environment  $\mathbf{y}$  to the low-dimensional projection  $\boldsymbol{\theta}$ . If  $\phi_{\text{CP}}(\tilde{\mathbf{x}}_{t_c}, \mathbf{y})$  is the forward kinematics of the robot's contact point at the time of contact  $t_c$  and  $\boldsymbol{\theta}$  is the point where the robot should grasp the door handle, then the projection constraint can be defined as

$$\mathbf{h}(\bar{\mathbf{x}}, \mathbf{y}, \boldsymbol{\theta}) = \phi_{\text{CP}}(\tilde{\mathbf{x}}_{t_c}, \mathbf{y}) - \boldsymbol{\theta} . \quad (6.1)$$

This concept is transferable to different manipulation skills where the contact locations are crucial for performance and success. An underlying assumption is that there is no sliding on the objects during manipulation. Further types of projections  $\boldsymbol{\theta}$  are the desired state of an external degree of freedom or the contact point velocity during manipulation. For example, in the experiment in Section 8.2.3 the robot learns the unlocking mechanism of a lockbox where one joint first has to reach a specific value before the next joint gets unlocked and can be manipulated. In this experiment, we choose  $\boldsymbol{\theta}$  to represent the contact point location and its velocity.

## 6.2 Cost Features for Representing Generalizable Manipulation Skills

In this thesis, we represent skills as constrained optimization problems in the form of Equation (2.5). This representation consists of a set of cost features  $\Phi$ , equality constraints  $\mathbf{h}$ , and inequality constraints  $\mathbf{g}$  that describe the manipulation task. Inverse KKT (see Chapter 4) requires a set of potential cost features as inputs and learns the weights of them from expert demonstrations. Extracting such a set of potential features and constraints from demonstrations is non-trivial. In the real robot experiments of this thesis, the following feature types are used:

- Transition features: Represent the smoothness of the motion (e.g., sum of squared acceleration or torques)
- Position features: Represent a body position relative to another body (e.g., between robot gripper and door handle).
- Orientation features: Represent orientation of a body relative to another body (e.g., by aligning two vectors).

These features are defined at relevant timesteps (e.g., before or after contact change) and for different bodies, whereby a body is either a part of the robot or belongs to an object in the environment. They are defined relative to the manipulated objects so that a transfer between different configurations is possible. Further, we define the following inequality constraints:

- Avoiding collisions between robot and the environment.
- Not violating the robot hardware limits (e.g., joint limits).
- Restricting the movement direction of external objects (e.g., pushing in a certain direction).

The equality constraints are defined for tasks that should be exactly fulfilled. We define the following equality constraints:

- Achieving the contact points between robot and environment.
- Fixing external degrees of freedom (e.g., a door joint) when they are not being manipulated.
- Reaching the desired states of external degrees of freedom.

Figure 6.1 shows a sketch of various features and constraints. The features that receive a weight  $w$  larger than 0 are extracted into the final policy. In the evaluation in Section 8.2, we demonstrate the generalization capabilities of this policy representation on various real robot tasks. These capabilities are for example the generalization to different initial configurations or target configurations of the environment. A further generalization capability is that a robot can perform a skill from various initial states.

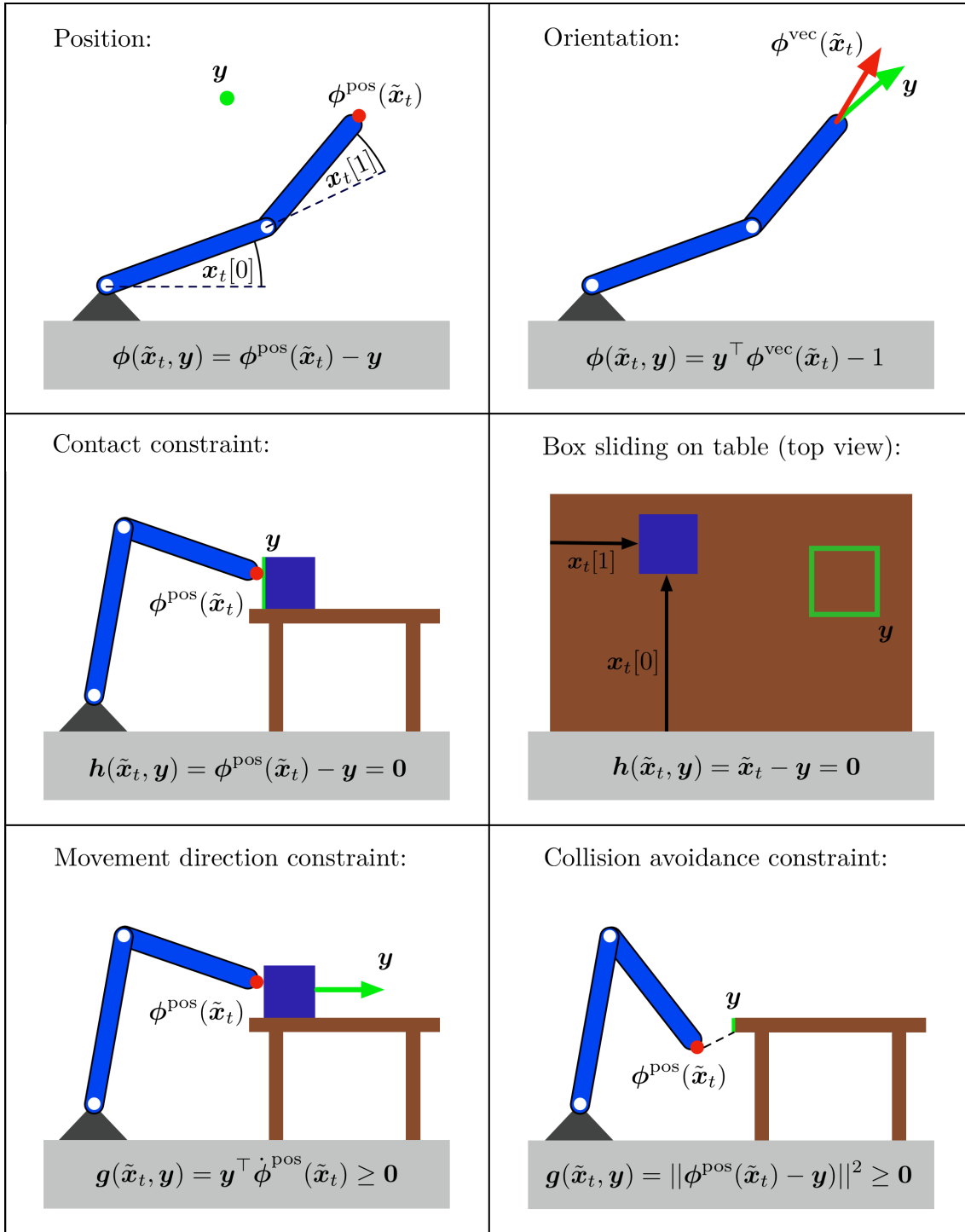


Figure 6.1: The sketches show various features of cost and constraints that we use to define manipulation skills.

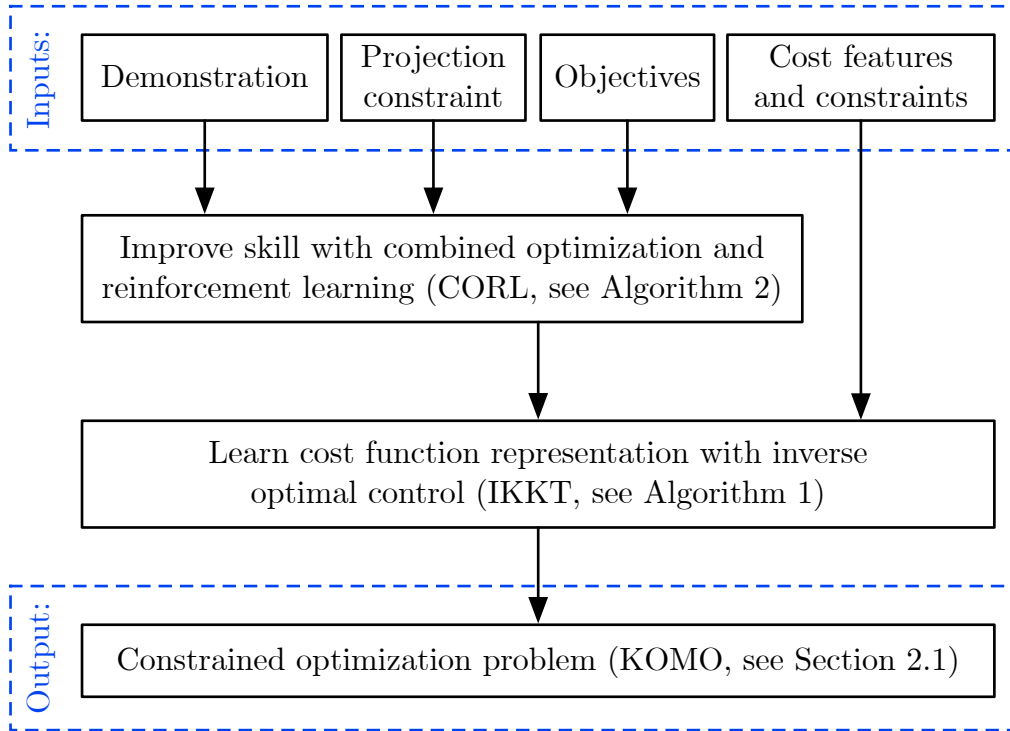


Figure 6.2: Concept of learning skills from a single demonstration.

### 6.3 Learning Skills from a Single Demonstration

We consider the scenario in which a robot is demonstrated a manipulation skill once and should then only use few own trials to reproduce, improve, and generalize that same skill (Englert and Toussaint, 2018b). The underlying idea is that the robot explores the skill by trying different manipulations and thereby collects a dataset. This dataset contains information about the skill that is afterwards used to extract a high level skill representation. The diagram in Figure 6.2 shows an overview of this approach that combines Inverse KKT (Chapter 4) and CORL (Chapter 5). The inputs to CORL are a single demonstration, a projection constraint, and the objective criteria (see Section 6.1). Further inputs for Inverse KKT are a set of cost features and constraints. The demonstration data, which is also required as input for Inverse KKT, is in this scenario created via CORL. The desired output representation is a constrained optimization problem that can be optimized to generate motions for different environment configurations  $\mathbf{y}$ .

In the first step, the structured reinforcement learning method CORL is applied for the given inputs. The dataset  $\mathcal{D}$  is initialized with the input demonstration

$(\bar{\mathbf{x}}^{(0)}, \boldsymbol{\theta}^{(0)})$  and its corresponding performance. Afterwards, we iterate the CORL policy update (see Algorithm 2), which first selects a new low-dimensional projection  $\boldsymbol{\theta}^*$  by maximizing Equation (5.6) that is then mapped on the full policy representation  $\bar{\mathbf{x}}^*$  by optimizing Equation (5.7). This policy is executed on the real system and the observed objectives are added to the dataset  $\mathcal{D}$ . This procedure is repeated until there is no more change in the policy parameters. The output of CORL is a dataset  $\mathcal{D}$  that contains all the rollouts and their corresponding performance.

In the second step, the dataset  $\mathcal{D}$  is taken as input to the inverse optimal control method IKKT (Algorithm 1). The best successful data points of  $\mathcal{D}$  are selected to define a smaller dataset that is used to learn feature weights of a higher level cost function. Inverse KKT extracts the cost function of a constrained optimization problem such that the dataset fulfills the Karush-Kuhn-Tucker conditions. We define a set of generic features and constraints that allows generalizing the skill regarding different environment configurations (see Section 6.2). We also always adopt the projection constraint  $\mathbf{h}(\bar{\mathbf{x}}, \mathbf{y}, \boldsymbol{\theta}) = \mathbf{0}$  of CORL (e.g., contact points) as an equality constraint into IKKT. Afterwards, the corresponding weights  $\mathbf{w}$  are learned with the optimization problem in Equation (4.12) such that the resulting cost function fulfills the KKT conditions for the motions in the dataset. The generalization capabilities we want to achieve are different initial states of the robot and the environment as well as different final states of the external degrees of freedom of the environment.

A concrete skill that we consider in the experiments is to open a cabinet with a PR2 robot (see Section 8.3). The learning is initialized with a single demonstration recorded via kinesthetic teaching. The low-dimensional projection is defined as the force during the opening and the rotation angle of the door knob. The goal is to achieve a successful opening motion with a low amount of force. Afterwards, we take the acquired trajectory data to learn a cost function that allows the robot to generalize the skill to different initial robot states (see Figure 8.20a) and desired door states (see Figure 8.20b).



# Chapter 7

## Kinematic Morphing Networks

The proposed methods IKKT (Chapter 4) and CORL (Chapter 5) are able to generalize and improve a skill. Both methods assume to have access to the structure and state of the robot’s environment  $\mathbf{y}$  (e.g., object shape and location). However, most robots are equipped with sensors, such as a camera or laser scanner, that only allow them to perceive their environment in an abstract and high-dimensional form. Such raw sensor signals are not directly usable as an input for most policy representations. In this chapter, we present an approach to convert the raw sensor signal into a suitable policy input representation: kinematic morphing networks (KMNs; Englert and Toussaint (2018a)), which are deep neural networks that predict the geometric relation between different environment configurations. The main objective behind KMNs is to enable generalization between different geometric environments by extracting parameters of a kinematic model from an observed environment and use them as an input to skill policies. The proposed approach relies on a prototype that serves as a reference environment for which a policy is available. The skill transfer is achieved by morphing an observed environment to this prototype. Concretely, a deep neural network is trained to map depth image observations of the environment to morphing parameters, which include the transformation and configuration of the prototype model. We combine the concatenation property of affine transformations with the ability to convert point clouds to depth images in order to apply the network in an iterative manner. The network is trained on data generated in a simulator as well as augmented data that is created by its own predictions.

### 7.1 Skill Transfer with Kinematic Morphing Networks

We consider the scenario in which a robot observes a manipulation environment with a depth sensor. The observation is taken from a specific viewpoint, which often results in a measurement that only covers certain parts of the environment. We train



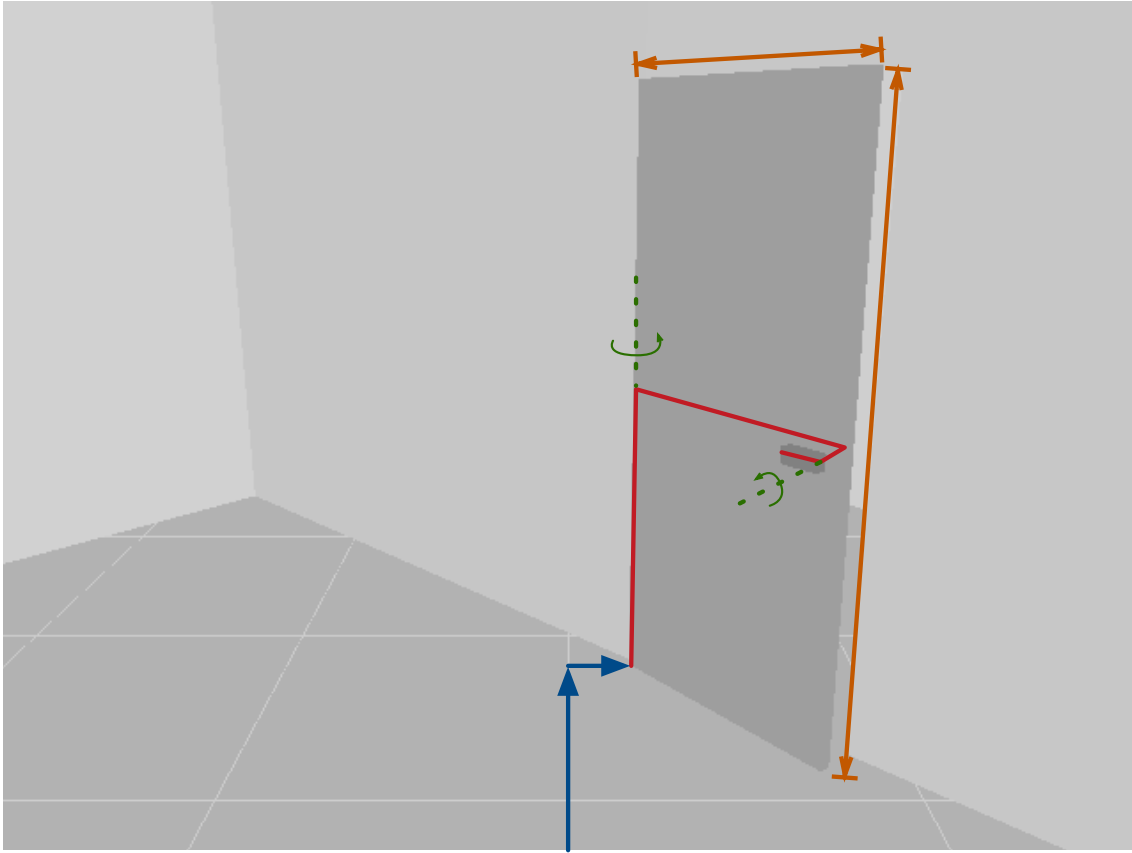


Figure 7.1: Kinematic model of a door that we want to extract from depth images and use to transfer manipulation skills.

a deep neural network that predicts a compact representation from these observations. There, it is assumed that the kinematic model structure of the manipulated environment is known and that a simulator is available to create large amounts of supervised training data. Each of these components is described in detail in one of the following sections.

### 7.1.1 Environment Parametrization

We represent the manipulated environment with a kinematic model that consists of rigid bodies and joints. Figure 7.1 illustrates such a kinematic model for a door environment with its parametrization (e.g., width, position, handle location). A key element of this approach is a prototype that serves as a reference for other models. The models are parametrized by morphing parameters that define the mapping from each point on a model to a corresponding point on the prototype. These parameters

describe the 3D transformation as well as the configuration of the prototype, such as the height of a door handle. We assume that if both a policy for the prototype model and the morphing parameters of an observed environment are known, then the policy can be transferred from the prototype to the observed environment. Specifically, we will utilize the trajectory optimization method KOMO (see Section 2.1) to define costs and constraints depending on the morphing parameter. These costs and constraints describe how the robot should interact with the environment to manipulate it into a desired state.

### 7.1.2 Kinematic Morphing Model

The goal of this work is to extract the morphing parameters from sensor observations. We propose kinematic morphing networks (KMNs) and define them as convolutional neural networks that map depth images to morphing parameters. Further, we introduce a data augmentation method that employs the network predictions to generate more training data. This augmentation applies the predicted morphing parameters on the input point cloud and generates a new depth image from it. The same mechanism is applied to make predictions with the network in an iterative manner. This can be viewed as a controller that changes the inputs in multiple steps until a steady point is reached. In this case, the steady point is the prototype model and the goal is to transform all observed models to this prototype. The advantage is that the model does not have to predict the morphing parameters in a single step. We show in the experiment in Section 8.4.1 that this results in an increased prediction accuracy.

### 7.1.3 Data Generation in Simulation

Training the parameters of a deep neural network with supervised learning requires a large amount of labelled data, which is difficult to collect in the real world since the labels would have to be provided by hand. In this work, we follow a recent trend to generate synthetic data with simulators (see related work in Section 3.3.2). A kinematic engine and an OpenGL renderer are used to create 3D representations of environments for a given set of morphing parameters. This functionality allows to generate a large supervised dataset consisting of point clouds, depth images, and morphing parameters. This work is focussed on the morphing of the kinematic mod-

els and therefore it is assumed that the background has already been removed from the data. The kinematic morphing model is jointly trained on the data generated in simulation and on the augmented data created with the KMN predictions.

Combining all these ingredients creates a tool that extracts a compact representation from high-dimensional sensor data. This tool can be employed to transfer robot skills between different environments. We will demonstrate these transfer abilities in the experiment in Section 8.4.3, where a skill defined as a constrained optimization problem is applied to manipulate doors of different shapes and locations.

## 7.2 Kinematic Model of the Manipulated Environment

We introduce a parametrized kinematic model of the environment  $\mathbf{m}(\boldsymbol{\xi}, \boldsymbol{\gamma}) \subset \mathbb{R}^3$ , which is defined as a set of points of the manipulated environment (e.g., a door) with parameters  $\boldsymbol{\xi} \in \mathbb{R}^n$  and  $\boldsymbol{\gamma} \in \mathbb{R}^m$ . A prototype model  $\mathbf{m}(\boldsymbol{\xi}_0, \boldsymbol{\gamma}_0)$  is defined as a reference for other models, where  $\boldsymbol{\gamma}_0$  and  $\boldsymbol{\xi}_0$  are usually set to  $\mathbf{0}$ . In this thesis, the term morphing describes the mapping of a model  $\mathbf{m}(\boldsymbol{\xi}, \boldsymbol{\gamma})$  to the prototype  $\mathbf{m}(\boldsymbol{\xi}_0, \boldsymbol{\gamma}_0)$ . The parameters of the kinematic model are:

1. The *transformation parameter*  $\boldsymbol{\xi}$  of an affine transformation  $\mathbf{T}_{\boldsymbol{\xi}} \in \text{Aff}(3)$  that describes the linear mapping between two models. The parameters  $\boldsymbol{\xi}$  are specific rotation, translation, and scale parameters around or along a certain axis.
2. The *configuration parameter*  $\boldsymbol{\gamma}$  describe the non-linear mapping between two models that cannot be represented with an affine transformation of the complete model. An example is the relative position between two bodies of the environment.

The affine transformation of the morphing operator is

$$\mathbf{m}(\boldsymbol{\xi}_0, \boldsymbol{\gamma}) = \mathbf{T}_{\boldsymbol{\xi}}^{-1} \mathbf{m}(\boldsymbol{\xi}, \boldsymbol{\gamma}) \quad (7.1)$$

for a given configuration  $\boldsymbol{\gamma}$ . This equation describes how all points of a model parametrized by  $\boldsymbol{\xi}$  transform onto a prototype  $\boldsymbol{\xi}_0$ . The goal of this work is to

predict model parameters  $(\boldsymbol{\xi}, \boldsymbol{\gamma})$  from sensor observations of the model  $\boldsymbol{m}$ . These parameters relate the currently observed model to the prototype model, which will be used to adapt the policy from the prototype to the observed model. We use the kinematic model  $\boldsymbol{m}$  as an input to the cost and constraint functions of the policy representation in Section 2.1, which then generalizes the skill to different models.

## 7.3 Kinematic Morphing Networks

In this section, we propose a technique to extract the morphing parameters  $(\boldsymbol{\xi}, \boldsymbol{\gamma})$  described in the previous section from environment observations. The robot observes an instance of the kinematic model  $\boldsymbol{m}(\boldsymbol{\xi}, \boldsymbol{\gamma})$  in form of a depth image  $\boldsymbol{D} \in \mathbb{R}^{W \times H}$  and a corresponding point cloud  $\boldsymbol{P} \in \mathbb{R}^{3 \times (WH)}$ . We define the function

$$f : \mathbb{R}^{W \times H} \rightarrow \mathbb{R}^n \times \mathbb{R}^m \quad (7.2)$$

that maps depth images  $\boldsymbol{D}$  to morphing parameters  $(\boldsymbol{\xi}, \boldsymbol{\gamma})$ . In this thesis,  $f(\boldsymbol{D}; \boldsymbol{\omega})$  is represented as a convolutional neural network with parameters  $\boldsymbol{\omega} \in \mathbb{R}^B$  (see Section 2.4 for background on neural networks). In the proposed approach, a dataset of the form  $\mathcal{D} = \{(\boldsymbol{D}^{(i)}, \boldsymbol{P}^{(i)}, \boldsymbol{\xi}^{(i)}, \boldsymbol{\gamma}^{(i)})\}_{i=1}^D$  is collected to optimize the network parameters  $\boldsymbol{\omega}$ . In the following sections, we describe the network prediction, training, and architecture.

### 7.3.1 Iterative Network Predictions

We introduce a mechanism that iteratively applies the network from Equation (7.2) on a given input  $(\boldsymbol{D}, \boldsymbol{P})$  to predict transformation parameter  $\boldsymbol{\xi}$ :

1. Predicting parameters  $\boldsymbol{\xi}$  for  $\boldsymbol{D}$  with Equation (7.2).
2. Applying the transformation  $\boldsymbol{T}_{\boldsymbol{\xi}}$  to the point cloud  $\boldsymbol{P}$ .
3. Rendering a new depth image  $\boldsymbol{D}$  from  $\boldsymbol{P}$ .

This concept is visualized in Figure 7.2. These three steps are repeated until a fixed point is reached, i.e. no significant change in  $\boldsymbol{\xi}$ . The resulting point cloud after  $t$  iterations is

$$\boldsymbol{P}_t = \boldsymbol{T}_{\boldsymbol{\xi}_t}^{-1} \dots \boldsymbol{T}_{\boldsymbol{\xi}_2}^{-1} \boldsymbol{T}_{\boldsymbol{\xi}_1}^{-1} \boldsymbol{P} . \quad (7.3)$$

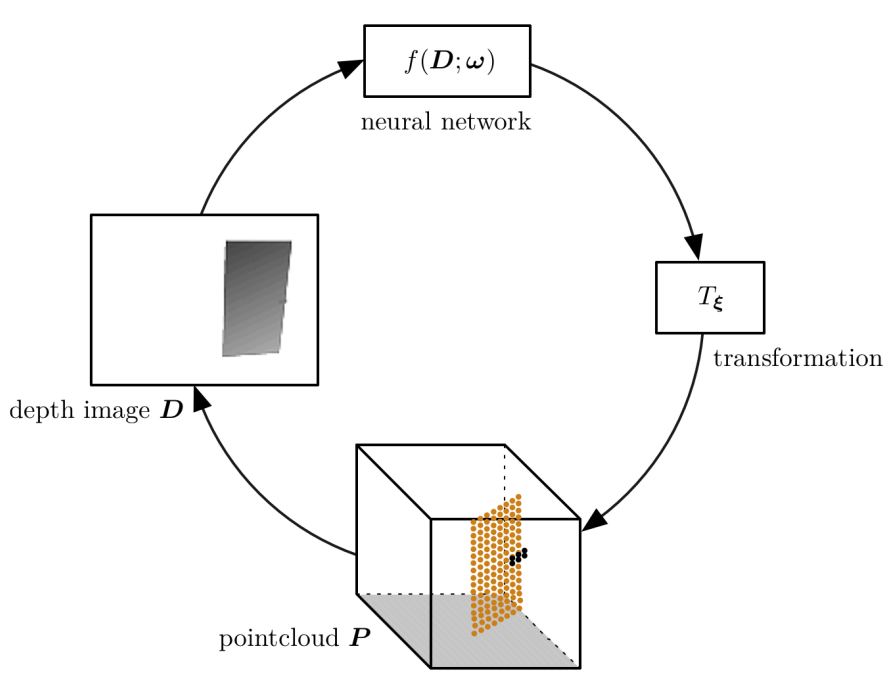


Figure 7.2: Multi-step network predictions.

The idea is that in each step the point cloud is transformed a bit closer towards the prototype. After convergence, this point cloud should overlay with the prototype model  $\xi_0$ . The necessary steps are summarized in Algorithm 3. The inputs are a depth image  $\mathbf{D}$ , a point cloud  $\mathbf{P}$ , a previous transformation  $\xi$  (by default  $\xi_0$ ), network parameters  $\omega$ , and number of predictions  $N$ . In the first step of the for-loop, the network predicts transformation parameters for the given depth image. Afterwards, the corresponding point cloud is transformed with the predicted transformation, which is then mapped to a new depth image. Finally, the predicted transformations are concatenated which we express with the symbol  $\circ$ . This procedure is repeated  $N$  times. The output are a new depth image and point cloud with the corresponding morphing parameter. An alternative to the fixed number of iterations  $N$  would be to repeat the steps until the network predicts a transformation that is close to the identity transformation, which indicates convergence. The algorithm requires a converting function *pointCloudToDepth* that renders a depth image  $\mathbf{D}$  from the transformed point cloud  $\mathbf{P}$ . Because the configuration parameters  $\gamma$  cannot be predicted in an iterative manner, only the last prediction of the network is returned.

**Algorithm 3:** Multi-step Network Predictions

---

```

function predict( $\mathbf{D}, \mathbf{P}, \boldsymbol{\xi}, \boldsymbol{\omega}, N$ ) :
  for  $n = 1 : N$ 
     $(\bar{\boldsymbol{\xi}}, \gamma) = f(\mathbf{D}; \boldsymbol{\omega})$ 
     $\mathbf{P} = \mathbf{T}_{\bar{\boldsymbol{\xi}}}^{-1} \mathbf{P}$ 
     $\mathbf{D} = \text{pointCloudToDepth}(\mathbf{P})$ 
     $\boldsymbol{\xi} = \bar{\boldsymbol{\xi}}^{-1} \circ \boldsymbol{\xi}$ 
  end
  return  $(\mathbf{D}, \mathbf{P}, \boldsymbol{\xi}, \gamma)$ 

```

---

**7.3.2 Data Generation and Network Training**

Throughout training, the current state of the network is used to augment the training data by applying Algorithm 3. The full iterative data generation and network training mechanism is summarized in Algorithm 4. The inputs are the parametrized model of the environment  $\mathbf{m}(\boldsymbol{\xi}, \gamma)$ , the neural network  $f(\mathbf{D}; \boldsymbol{\omega})$ , the initial dataset size  $N_{\text{data}}$ , the augmented dataset size  $N_{\text{aug}}$ , and the limits of the model parameters ( $L^{\text{up}}, L^{\text{low}}$ ). In the first part of the algorithm, a dataset is generated with an OpenGL renderer that creates a depth image  $\mathbf{D}$  and a point cloud  $\mathbf{P}$  for a given model  $\mathbf{m}(\boldsymbol{\xi}, \gamma)$ . The parameters  $(\boldsymbol{\xi}, \gamma)$  are sampled uniformly in the feasible range of parameters defined by  $L^{\text{up}}$  and  $L^{\text{low}}$ . Afterwards, the network is trained on the generated dataset. In the second phase, the trained model is taken to augment the dataset by applying the model on a subset  $N_{\text{aug}}$  of the initial data. In this step, the multi-step network prediction algorithm is applied to generate a new datapoint by applying  $N_{\text{pred}}$  predictions of the network. The augmented data points are appended to  $\mathcal{D}$  and the network is retrained. This second part can be seen as a fine-tuning of the network parameters for data points that are close the prototype. The data generation and retraining procedure is repeated with larger values of  $N_{\text{pred}}$  until there is no further change in network parameters  $\boldsymbol{\omega}$ .

**7.3.3 Network Architecture**

The function  $f$  is parametrized as a multi-layer convolutional neural network (see Section 2.4). The dataset consists of depth images with width  $W = 640$ , height  $H = 480$ , and corresponding point clouds with  $W \cdot H$  points. The depth images are downsampled to a resolution of  $128 \times 96$  before using them as an input to the net-

---

**Algorithm 4:** Data Generation and Network Training

---

**inputs:**

Kinematic model  $\mathbf{m}(\boldsymbol{\xi}, \gamma)$   
 Neural network  $f(\mathbf{D}; \boldsymbol{\omega})$   
 Initial dataset size  $N_{\text{data}}$   
 Augmented dataset size  $N_{\text{aug}}$   
 Upper and lower parameter limits  $(L^{\text{up}}, L^{\text{low}})$

Initialize dataset  $\mathcal{D} = \emptyset$

**// Generate an initial dataset**

**for**  $d = 1 : N_{\text{data}}$   
    $(\boldsymbol{\xi}, \gamma) \sim \mathcal{U}(L^{\text{up}}, L^{\text{low}})$   
    $(\mathbf{D}, \mathbf{P}) = \text{render}(\mathbf{m}(\boldsymbol{\xi}, \gamma))$   
    $\mathcal{D} = \mathcal{D} \cup \{(\mathbf{D}, \mathbf{P}, \boldsymbol{\xi}, \gamma)\}$

**end**

**// Train network**

$\boldsymbol{\omega} = \arg \min_{\boldsymbol{\omega}} \sum_{i=1}^{|\mathcal{D}|} \left\| (\boldsymbol{\xi}^{(i)}, \gamma^{(i)}) - f(\mathbf{D}^{(i)}; \boldsymbol{\omega}) \right\|^2$

**// Generate data with model predictions**

$N_{\text{pred}} = 1$

**repeat**

**for**  $d = 1 : N_{\text{aug}}$   
      $(\mathbf{D}, \mathbf{P}, \boldsymbol{\xi}, \gamma) = \text{predict}(\mathbf{D}^{(d)}, \mathbf{P}^{(d)}, \boldsymbol{\xi}^{(d)}, \boldsymbol{\omega}, N_{\text{pred}})$   
      $\mathcal{D} = \mathcal{D} \cup \{(\mathbf{D}, \mathbf{P}, \boldsymbol{\xi}, \gamma^{(d)})\}$

**end**

**// Retrain network**

$\boldsymbol{\omega} = \arg \min_{\boldsymbol{\omega}} \sum_{i=1}^{|\mathcal{D}|} \left\| (\boldsymbol{\xi}^{(i)}, \gamma^{(i)}) - f(\mathbf{D}^{(i)}; \boldsymbol{\omega}) \right\|^2$

$N_{\text{pred}} = N_{\text{pred}} + 1$

**until** no change in  $\boldsymbol{\omega}$

**output:**

Optimal network weights  $\boldsymbol{\omega}^*$

---

work while the point cloud dimensionality is kept the same. This downsampling has two benefits: 1) A reduction of the number of network parameters  $\boldsymbol{\omega}$ ; 2) The conversion from dense point clouds to low-resolution depth images leads to less holes that may occur through scaling or rotation operations. The depth values are normalized between a value of 0 and 1, where a depth value of 0 belongs to the background and

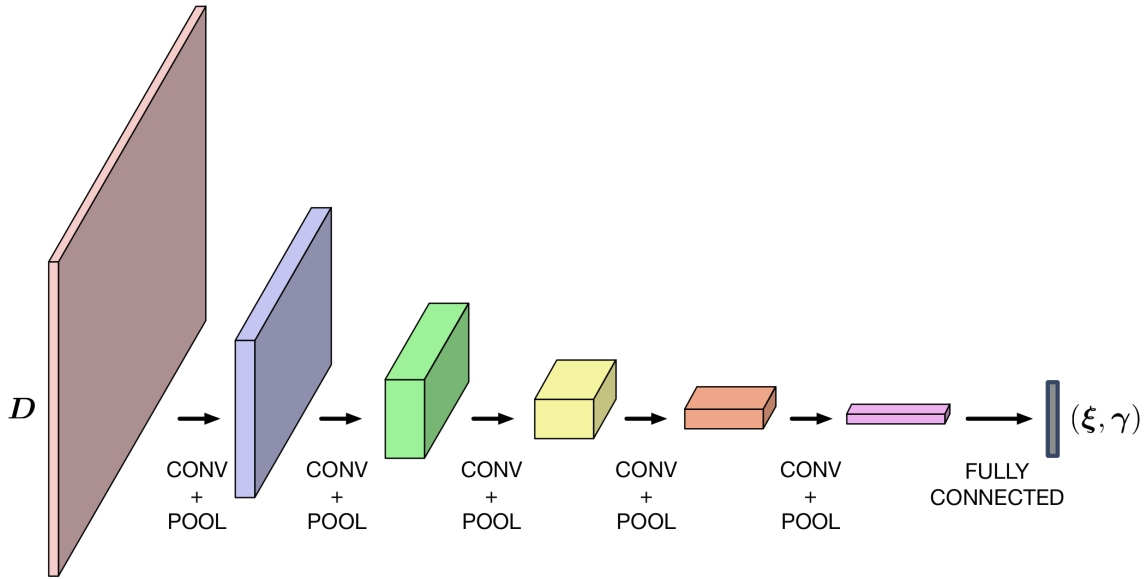


Figure 7.3: Network architecture.

all other values to the environment. The basic structure of the network is visualized in Figure 7.3 and consists of 5 convolutional layers where each layer is followed by a max-pooling layer. We use a kernel size of  $3 \times 3$  and a  $2 \times 2$  max pooling (see Section 2.4 for background on CNN). The number of filters in the convolutional layers are chosen depending on the model complexity. The convolution layers use a ReLU activation function and the last layer of the network is a fully connected linear layer that outputs the morphing parameters  $(\xi, \gamma)$ . The loss function is the mean squared error and is minimized with the Adam optimizer (Kingma and Ba, 2014).

The kinematic morphing networks are evaluated in Section 8.4. The experiments show that the proposed iterative prediction scheme leads to a higher accuracy than one-step predictions. They also demonstrate the performance of KMN on real sensor data from a Kinect camera.





# Chapter 8

## Experiments

In this chapter, we report on the experimental results of the proposed algorithms IKKT, CORL, and KMN. The experiments were performed on synthetic and real robot problems. The real robot manipulation experiments are done with a Willow Garage PR2 robot. The experiments show the properties of the methods and their performance in comparison to alternative methods. An overview of the experiments is given in Table 8.1.

	<b>Domain</b>	<b>IKKT</b>	<b>CORL</b>	<b>KMN</b>
Simulated	2D motion planning (Section 8.1.1)	✓		
	2D constrained optimization (Section 8.2.1)		✓	
	Via-point motion (Section 8.1.2)	✓		
	Box (Section 8.1.4 & 8.4.1)	✓		✓
	Door (Section 8.4.1 & 8.4.3)			✓
Real	Door (Section 8.1.5 & 8.2.2)	✓	✓	
	Cabinet (Section 8.3)	✓	✓	
	Lockbox (Section 8.2.3)		✓	
	Box (Section 8.4.2)			✓

Table 8.1: Overview of the experiments in this thesis.

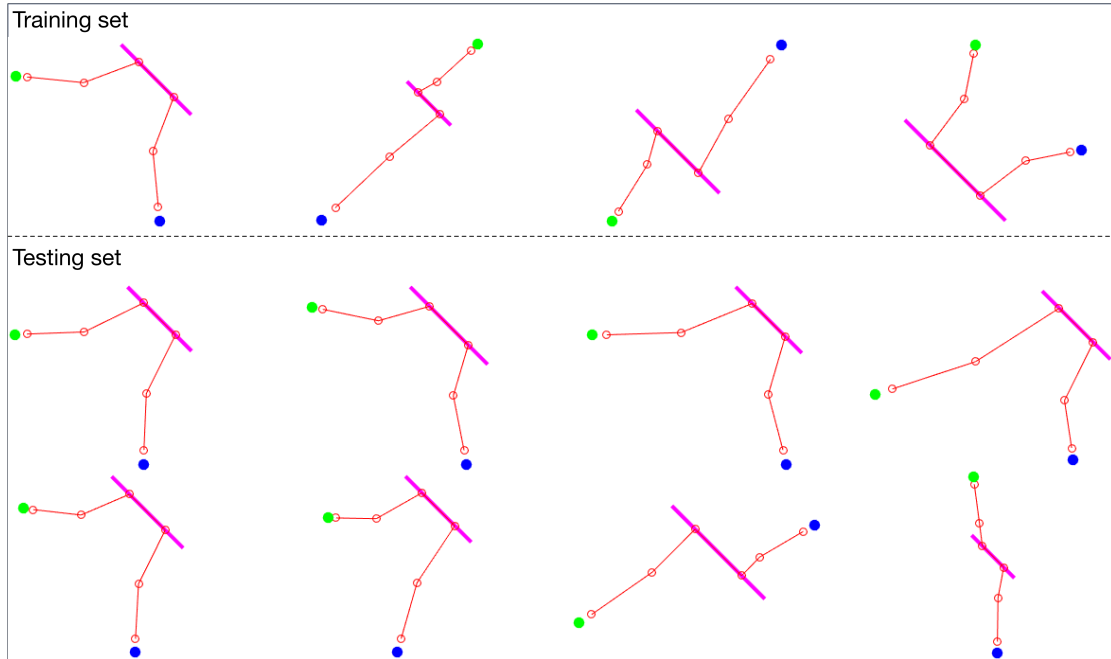


Figure 8.1: **2D motion planning task** (see Section 8.1.1) — These images show the 2D motion planning problem. The task is to go from a start state (green dot) to a goal state (blue dot). During the motion a contact with the magenta line has to be established. The four data points on the top row are used as training data and the other data points are used for testing.

## 8.1 Evaluation of Inverse KKT

Inverse KKT (Chapter 4) is evaluated based on different experiments. These experiments show different properties (e.g., parametrization, noise robustness) of the algorithm and demonstrate its performance on robot manipulation skills.

### 8.1.1 IOC on a 2D Problem with Constraints

In this experiment, we compare different IOC algorithms on a 2D problem. We will compare:

- Inverse KKT with a set of hand-defined features (Section 4.1).
- Nonparametric Inverse KKT with a squared exponential kernel (Section 4.4).
- Continuous inverse optimal control (CIOC) which was proposed by Levine and Koltun (2012). A description of this algorithm can be found in the related

work section 3.2.1.

The task is a 2D trajectory optimization problem of a point mass. The state of the point mass is two-dimensional and the motion consists of six time steps, which lead to a trajectory  $\bar{\mathbf{x}} \in \mathbb{R}^{12}$ . The goal of this task is that the robot moves from an initial state to a goal state. At time step three and four of the trajectory the robot should be in contact with a line. During this contact phase the robot should move 1 unit in the vertical direction downwards. The state of the environment  $\mathbf{y}$  contains the initial position, goal position, and line parametrization. The domain is visualized in Figure 8.1.

CIOC and the proposed feature variant of Inverse KKT use transition features and a set of linear features around four points in the 2D world. In the two IKKT variants, we represent the contact in form of equality constraints  $\mathbf{h}(\bar{\mathbf{x}}, \mathbf{y})$ . Since the CIOC formulation does not incorporate constraints, we add the contacts as cost features  $\Phi(\bar{\mathbf{x}}, \mathbf{y})$ . Initially, 12 motions are created for different scenarios  $\mathbf{y}$  which are visualized in Figure 8.1. In the nonparametric IKKT variant, the trajectory  $\bar{\mathbf{x}}$  is augmented with the environment  $\mathbf{y}$ . The dataset is split in four motions for training the IOC methods and eight motions for evaluating it. To evaluate the methods, we first take the training data as input to the IOC methods and learn a weight vector  $\mathbf{w}$ . Afterwards, we insert the learned weight vector in the optimal control problem to generate motions for the test scenarios. The resulting motions are compared to the ground truth motions of the test scenarios. We compare the error of the trajectories and the violation of the constraints on the training and test set. The results are reported in Table 8.2. We also visualize the resulting motions of all three variants in Figure 8.2 for a training scenario (left) and a test scenario (right). The results confirm that CIOC and IKKT (feature) reach for the same feature set a similar performance (see discussion in Section 3.2.1). The nonparametric variant of IKKT achieves a much lower performance. It manages to reach a reasonable training error. However, the generalization abilities are very limited, which is due to the simple RBF kernel at each time step. In order to improve the performance it would be necessary to consider multiple or more complex kernels. In the following experiments, we will therefore focus on the parametric variant of IKKT. In this evaluation, CIOC reached a lower training error and IKKT reached a lower test error. The visual comparison shows that both methods reproduce the trajectories of the training set and also come very close to the trajectories of the test set. The

Algorithm	Error (train)	Error (test)	Constraint violation (train)	Constraint violation (test)
IKKT (feature)	0.027475	<b>0.46944</b>	1.1102e-15	1.6653e-15
IKKT (kernel)	0.94625	66.065	<b>4.4409e-16</b>	<b>8.2469e-16</b>
CIOC	<b>0.014732</b>	0.64592	0.00058039	0.001128

Table 8.2: **2D motion planning task** (see Section 8.1.1) — Results from the 2D motion planning task. The error is the sum of absolute differences between the reference motion and the motion that was optimized with the learned weights  $w$ . The constraint violation is the distance to the magenta line.

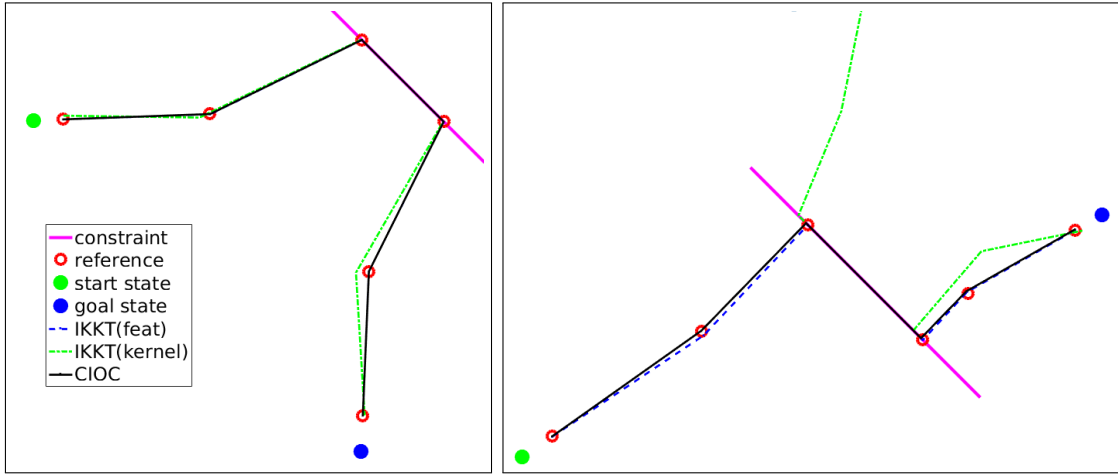


Figure 8.2: **2D motion planning task** (see Section 8.1.1) — Evaluation of the learned parameters of the 2D motion planning task. The left image shows the performance on a training scenario and the right image shows the performance on a test scenario.

constraint violation of CIOC is higher compared to the two IKKT methods since it has to weight the contact features with the other features whereas IKKT can incorporate them separately as constraints.

### 8.1.2 Comparison of Different IKKT Weight Parametrizations

The goal of Inverse KKT is to learn cost functions for finite horizon constrained optimal control problems, including when and how long the costs should be active. In this experiment, Inverse KKT is tested on a simple benchmark scenario with

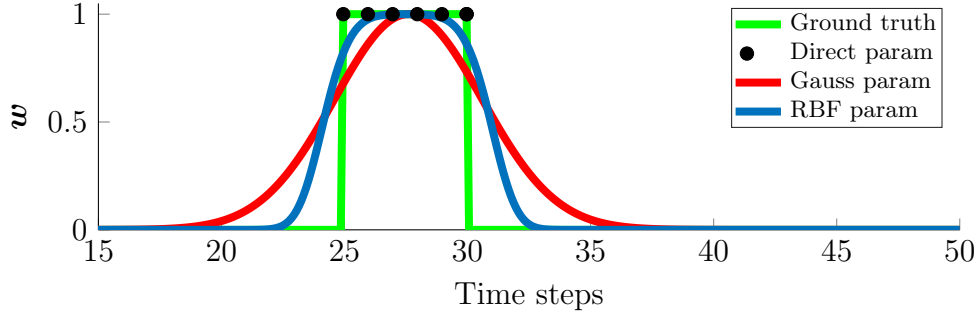


Figure 8.3: **Weight parametrization experiment** (see Section 8.1.2) — Learned time profiles of different weight parametrizations  $\mathbf{w}(\boldsymbol{\rho})$ .

different parametrizations (see Section 4.3) of the weights  $\mathbf{w}(\boldsymbol{\rho})$ . Therefore, we create synthetic demonstrations by optimizing the forward problem with a known ground truth parameter set and test if it is possible to reestimate these parameters from the demonstrations. For this experiment a simple robot arm with seven degrees of freedom is employed. Three demonstrations with 50 time steps are created with KOMO by optimizing the ground truth cost function. In this experiment, the cost function is chosen such that the robot endeffector is close to a target point during the time steps 25 to 30. We compare the three parametrizations:

- **Direct parametrization:** A different parameter is defined at each time step (i.e.,  $\mathbf{w} = \boldsymbol{\rho}$ ) that results in  $\boldsymbol{\rho} \in \mathbb{R}^{50}$ .
- **Radial basis function (RBF):** The basis functions are equally distributed over the time horizon. We defined 30 Gaussian basis functions with a standard deviation of 0.8. This results in  $\boldsymbol{\rho} \in \mathbb{R}^{30}$ .
- **Non-linear Gaussian:** A single unnormalized Gaussian weight profile where we have  $\boldsymbol{\rho} \in \mathbb{R}^3$  with the weight as linear parameter and the non-linear parameters are the mean and standard deviation. In this case, the mean directly corresponds to the time where the activation is highest.

The first two parametrizations are linear parametrizations of the form  $\mathbf{w}(\boldsymbol{\rho}) = A\boldsymbol{\rho}$  and the last one is a non-linear parametrization  $\mathbf{w}(\boldsymbol{\rho}) = \mathcal{A}(\boldsymbol{\rho})$ . The demonstrations are taken as input to the Inverse KKT algorithm (see Algorithm 1) and the weights are initialized randomly.

A comparison of the learned parameters and the ground truth parameters is shown in Figure 8.3. The green line represents the ground truth weights used to

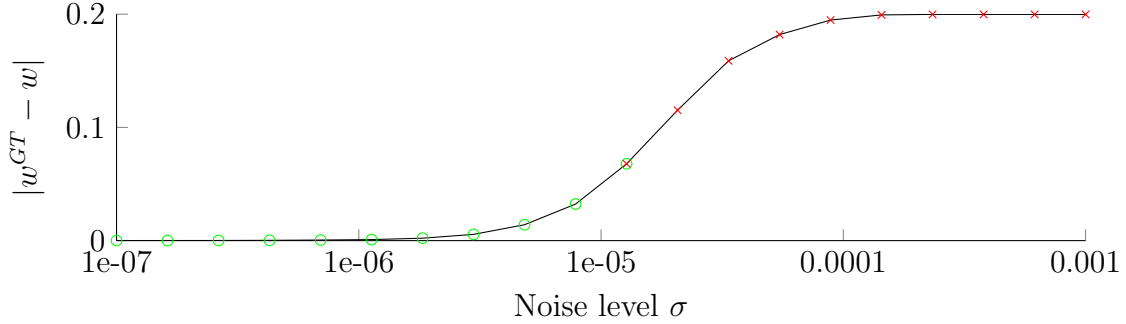


Figure 8.4: **Noisy demonstration experiment** (see Section 8.1.3) — Mean absolute error between the estimated and ground truth parameters for various noise levels on the demonstrations.

create the demonstrations. The black dots show the learned parameters of the direct parametrization. The red line shows the learned Gaussian activation and the blue line shows the RBF network. The figure shows that all parametrizations approximate the ground truth profile and detect the correct activation region between the time steps 25 and 30. The Gaussian and RBF parametrizations also give some weight to the region outside the actual cost region, which is reasonable since at those time steps the robot is still close to the target position. The results show that the linear RBF network are well suited to learn time profiles of cost functions. The main reason for this is the linearity of the parametrization that makes the Inverse KKT problem convex and the versatility of the RBF network to take on more complex forms. Directly learning the time with the non-linear Gaussian-shaped parametrization was more difficult and required multiple restarts with different initialization. This experiment demonstrates that the framework of constrained trajectory optimization and its counterpart Inverse KKT works well for reestimating cost weights of optimal demonstrations.

### 8.1.3 IKKT with Noisy Demonstrations

A core assumption of IKKT is that the demonstrations are optimal. This experiment investigates what happens if this is not the case. Therefore, we create a scenario with non-optimal demonstrations and evaluate if IKKT is still able to estimate the underlying ground truth cost parameters. The scenario is equivalent to the one in the previous experiment in which the robot has to reach a target position with its endeffector. In this experiment, different levels of Gaussian noise are added to the

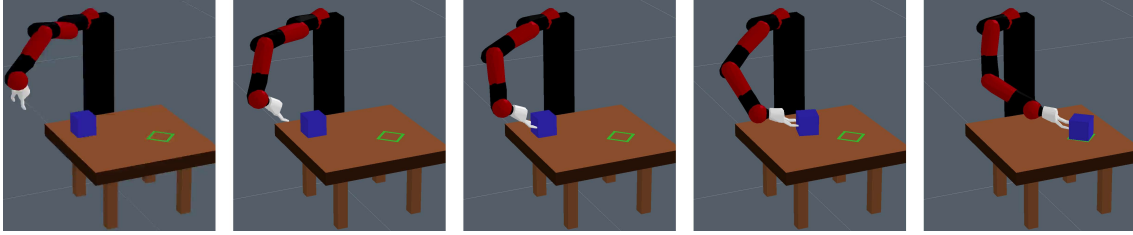


Figure 8.5: **Box sliding experiment** (see Section 8.1.4) — These images show the box sliding motion of Section 8.1.4 where the goal of the task is to slide the blue box on the table to the green target region.

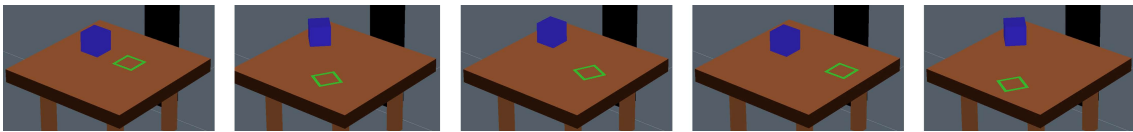


Figure 8.6: **Box sliding experiment** (see Section 8.1.4) — Each image shows a different instance of the box sliding task. We were able to generalize to different initial box states (blue box) and to different final box targets (green area).

optimal demonstrations. We want to test if IKKT is still able to extract the ground truth parameters that were initially used to create the noise-free demonstrations. In Figure 8.4 the mean absolute error is visualized for different standard deviations  $\sigma$  of the Gaussian noise. The values are averaged over 100 different random seeds. The cases in which the task could still be performed are visualized with green circles and failures are visualized with red crosses. A successful run is defined when the target was reached inside a 1 cm tolerance. The results show that the error increases continuously with the noise level and if  $\sigma$  is above  $2e-05$ , then the task begins to fail. This demonstrates the strong requirement to apply Inverse KKT only with near optimal demonstrations.

### 8.1.4 Learning a Box Sliding Skill from Demonstration

In this experiment, Inverse KKT is applied to learn a cost function for sliding a box on a table. This task is depicted in Figure 8.5. The goal is to move the blue box on the table to the green marked target position and orientation. The robot consists of a fixed base and a hand with 2 fingers. In total, the robot has 10 degrees of freedom. In addition to these internal degrees of freedom, we also model the box as part of the configuration state, which adds three more degrees of freedom (two translational + one rotational). The final box position and orientation is provided



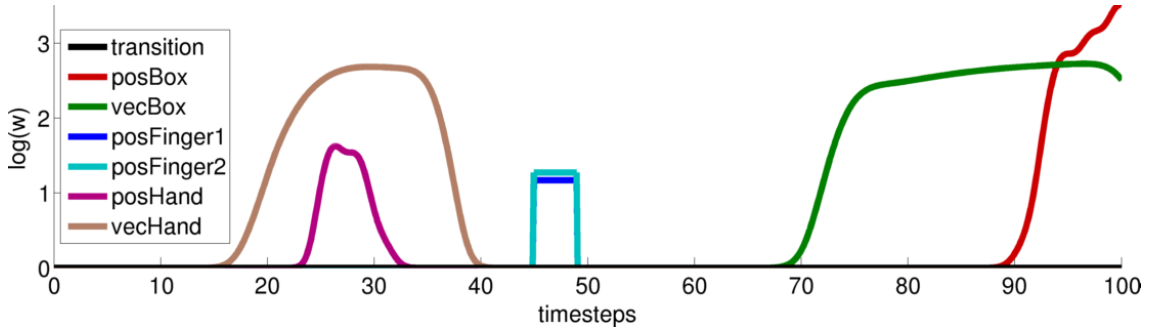


Figure 8.7: **Box sliding experiment** (see Section 8.1.4) — The learned parameters  $\mathbf{w}$  over time.

as input to the algorithm as part of the environment configuration  $\mathbf{y}$ . We recorded three synthetic demonstrations of the task and created a set of features with the approach described in Section 6.2 that led to  $\mathbf{w} \in \mathbb{R}^{537}$  parameters. The relevant features extracted with IKKT are:

- **transition:** Squared acceleration at each time step in joint space.
- **posBox:** Relative position between the box and the target.
- **vecBox:** Relative orientation between the box and the target.
- **posFinger1/2:** Relative position between the robots fingertips and the box.
- **posHand:** Relative position between robot hand and box.
- **vecHand:** Relative orientation between robot hand and box.

Figure 8.7 shows the learned weights  $\mathbf{w}$  of these features over time. The contacts between the fingers and the box during the sliding are modeled with equality constraints. They ensure that during the sliding the contact is maintained. To achieve realistic motions, we define an inequality constraint that restricts the movement direction during contact into the direction in which the contact is applied. This ensures that no unrealistic motions such as sliding backwards or sideways are created. For clarity we would like to note that we are not doing a physical simulation of the sliding behavior in this experiment. The goal was more to learn a policy that executes a geometric realistic trajectory from an initial to a final box position. Figure 8.5 shows one of the sliding motions resulting from the learned cost function.

**Algorithm 5** Black-box IOC

---

```

repeat
  Resample parameters  $\{\mathbf{w}^{(n)}\}_{n=1}^N$  with CMA-ES
  for all parameters  $\mathbf{w}^{(n)}$  do
    Init fitness value  $f^{(n)} = 0$ 
    for all demonstration  $\bar{\mathbf{x}}^{(d)}$  do
      Compute optimal motion  $\bar{\mathbf{x}}^*$  on environment  $\mathbf{y}^{(d)}$  with cost param.  $\mathbf{w}^{(n)}$ 
      Update fitness value  $f^{(n)} = f^{(n)} + \|\bar{\mathbf{x}}^* - \bar{\mathbf{x}}^{(d)}\|^2$ 
    Update CMA-ES distribution with fitness values
  until no change in  $\mathbf{w}$ 

```

---

Algorithm	Trajectory error	Computation time
Inverse KKT	<b>0.00021</b>	<b>49.29 sec</b>
Black-box IOC	0.00542	7116.74 sec

Table 8.3: **Box sliding experiment** (see Section 8.1.4) — Comparison of Inverse KKT and black-box IOC. The results of the evaluation show that Inverse KKT is superior in terms of squared error between the trajectories and computation time.

The learned cost function was able to generalize to a wide range of different start and goal locations of the box (Figure 8.6).

We compare the proposed method Inverse KKT to a black-box optimization approach (similar to (Mombaur et al., 2010; Rückert et al., 2013)). This approach is implemented with the black-box method CMA-ES in the outer loop and the constrained trajectory optimization method KOMO (see Section 2.1) in the inner loop. The resulting method is summarized in Algorithm 5. The fitness function for CMA-ES is the squared error between the demonstration  $\bar{\mathbf{x}}^{(d)}$  and the optimized motion  $\bar{\mathbf{x}}^*$  (with cost weights  $\mathbf{w}^{(n)}$ ). We compare this method with Inverse KKT by computing the trajectory error and the computational time, which are reported in Table 8.3. The black-box method took around 4900 iterations of the outer loop until it converged to a solution. This comparison shows that using structure and optimality conditions of the solution can enormously improve the learning speed. Further difficulties with black-box methods is that they cannot naturally deal with constraints (e.g.,  $\mathbf{w} > \mathbf{0}$ ) and that their initialization is non-trivial.

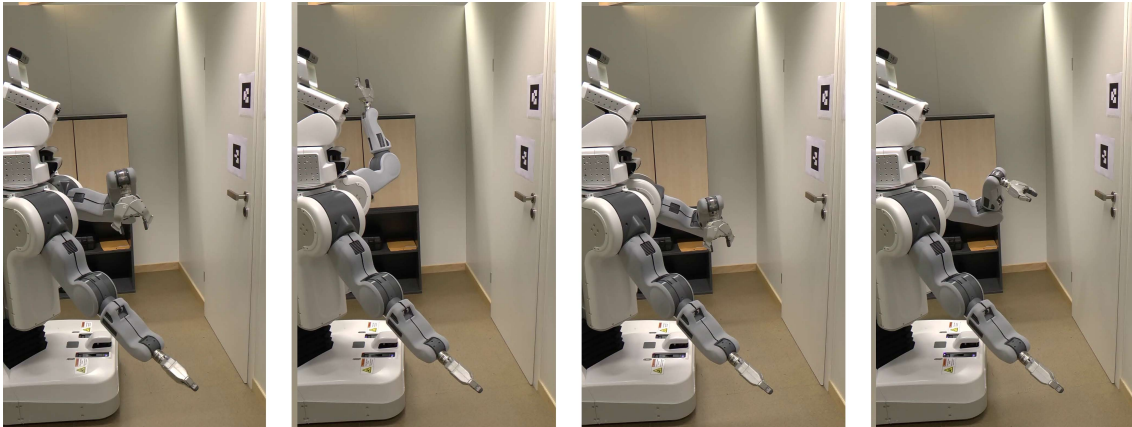


Figure 8.8: **Door experiment** (see Section 8.1.5) — Demonstration of a skill via kinesthetic teaching.

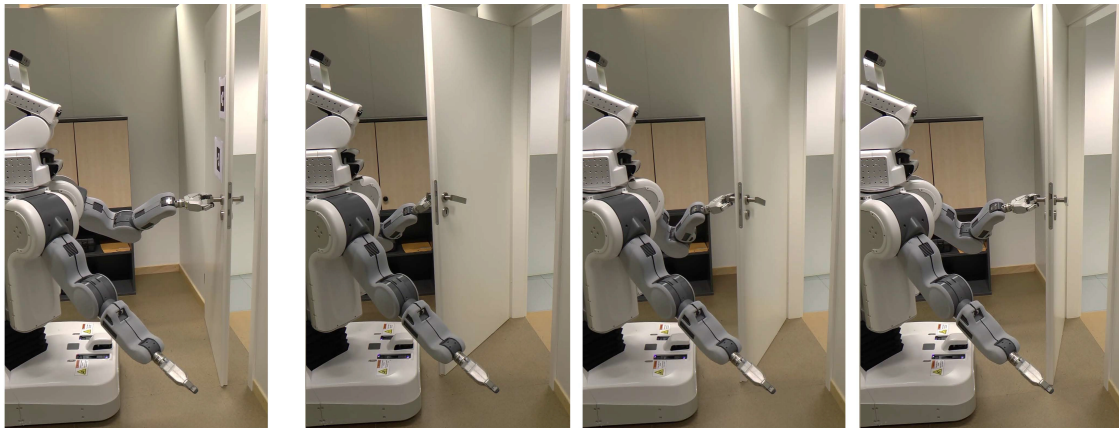
### 8.1.5 Learning a Door Skill with Inverse KKT

We apply the introduced Inverse KKT method on a skill with the goal to open a door with a real PR2 robot. The model of the door is known in advance and the state can be observed with AR marker. The configuration state consists of the seven rotational joints of the robot's left arm. Additionally, the angle of the door and handle are also added to the configuration state, which allows the definition of cost functions directly on the state of the door. The gripper joint is kept fixed during the whole motion. Two demonstrations of unlocking and opening the door from different initial positions are recorded with kinesthetic teaching (see Figure 8.8). These demonstrations and a feature set similar to the box sliding motion from the previous experiment are the inputs to Inverse KKT. The Inverse KKT algorithm extracted the features:

- Relative position & orientation between gripper and handle before and after unlocking the handle.
- Endeffector orientation during the whole opening motion.



(a) Different initial states



(b) Different final door angles

Figure 8.9: **Door experiment** (see Section 8.1.5) — These images show the generalization abilities of learned IKKT cost functions. The pictures in (a) show different initial states of the robot and the pictures in (b) show different final door angle positions. After learning the weight parameters  $\mathbf{w}^*$  with Inverse KKT it was possible to generalize to all these instances of the door opening task.

- Position of the final door state.

The equality constraints are, similar to the box sliding experiment, defined to keep the contact between endeffector and door. Furthermore, an inequality constraints to avoid collisions with the rest of the robot body is incorporated. The resulting cost function was able to robustly produce motions that could generalize to different initial positions and different target door angles (see Figure 8.9).

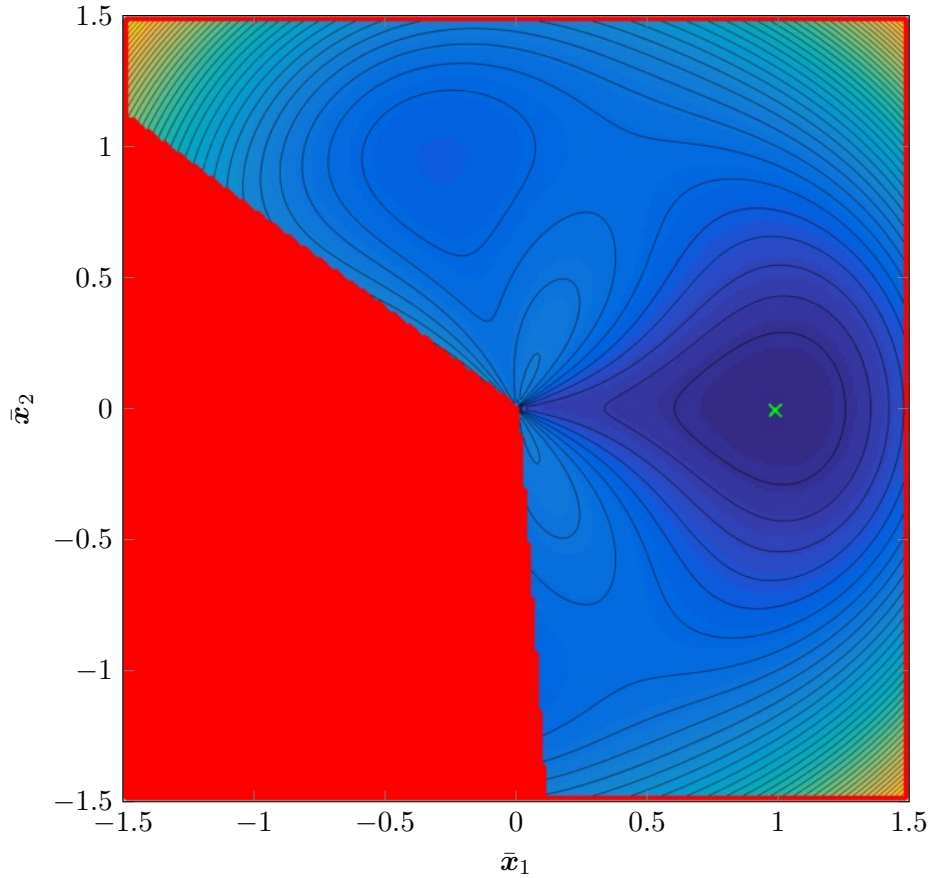


Figure 8.10: **Synthetic benchmark experiment** (see Section 8.2.1) — Contours of  $J(\bar{\mathbf{x}}) - R(\boldsymbol{\theta})$ . The red area is the infeasible region where  $S(\boldsymbol{\theta}) = 0$  and the green cross shows the optima  $\bar{\mathbf{x}}^* = (1, 0)$ .

## 8.2 Evaluation of CORL

CORL (Chapter 5) is evaluated first on a synthetic benchmark and afterwards on two real robot tasks.

### 8.2.1 Evaluation of CORL on a Synthetic Benchmark Problem

In this evaluation, we compare CORL (Algorithm 2) to different algorithms on a synthetic benchmark problem. To allow for reproducible quantitative comparison, we define a structured reinforcement learning problem in the form of Equation (5.5) with parameters  $\bar{\mathbf{x}} \in \mathbb{R}^2$  and a projection  $\boldsymbol{\theta} \in \mathbb{R}$ . The problem is defined with an

analytic cost

$$J(\bar{\mathbf{x}}) = (\bar{\mathbf{x}}_1^2 + \bar{\mathbf{x}}_2^2 - 1)^2, \quad (8.1)$$

a black-box return

$$R(\boldsymbol{\theta}) = -0.5\boldsymbol{\theta}^2 + \cos(3\boldsymbol{\theta}), \quad (8.2)$$

and a black-box success

$$S(\boldsymbol{\theta}) = [-1.5 < \boldsymbol{\theta} < 2.5]. \quad (8.3)$$

The projection is defined with the constraint

$$h(\bar{\mathbf{x}}, \boldsymbol{\theta}) = \boldsymbol{\theta} - \text{atan} \left( \frac{\bar{\mathbf{x}}_1}{\bar{\mathbf{x}}_2} \right). \quad (8.4)$$

The total objective we want to minimize is  $J(\bar{\mathbf{x}}) - R(\boldsymbol{\theta})$  under the constraint that  $S(\boldsymbol{\theta}) = 1$  (see Equation (5.5)). We limit the search space to the region  $\bar{\mathbf{x}} \in [-1.5, 1.5] \times [-1.5, 1.5]$ . This problem has multiple local optima and a global optima at  $\bar{\mathbf{x}}^* = (1, 0)$  with a value of  $-1$ . The contours of  $J(\bar{\mathbf{x}}) - R(\boldsymbol{\theta})$  are visualized in Figure 8.10, where the red area denotes the infeasible region  $S(\bar{\mathbf{x}}) = 0$  and the green cross the optima  $\bar{\mathbf{x}}^*$ .

We compare two different type of algorithms with each other. The first type employs the CORL framework we proposed in Chapter 5 with different reinforcement-learning algorithms (noted as CORL + *<method>*). The second type are standard reinforcement learning methods that optimize  $\bar{\mathbf{x}}$  and ignore the specific problem structure of using an analytic cost term and a black-box success constraint. Here is a summary of all evaluated algorithm configurations:

- **PIBU:** Bayesian optimization with the acquisition function PIBU (see Equation (5.6)).
- **PoWER:** Policy learning by weighting exploration with the returns (Kober and Peters, 2008).
- **UCB:** Bayesian optimization with the acquisition function upper confidence bound (Brochu et al., 2010).
- **CMA-ES:** Covariance matrix adaptation evolution strategy (Hansen and Ostermeier, 2001).

- **CORL + PIBU:** CORL algorithm with PIBU.
- **CORL + UCB:** CORL algorithm with UCB.
- **CORL + CMA-ES:** CORL algorithm with CMA-ES.
- **CORL + SAL:** CORL algorithm with safe active learning (Schreiter et al., 2015).

The CMA-ES algorithm is defined with a population size of 6 and the number of offsprings is 3. Only the PIBU and SAL variants aim for a safe exploration during the optimization process. Note that SAL assumes to observe the distance to the feasibility boundary in critical (but feasible) regions, which all other methods do not observe. To enable testing those methods that cannot cope with different objectives and success constraints (i.e., CMA-ES, UCB, and PoWER), we define the combined objective function

$$o(\bar{\mathbf{x}}) = [S(\boldsymbol{\theta}) = 1](J(\bar{\mathbf{x}}) - R(\boldsymbol{\theta})) + [S(\boldsymbol{\theta}) = 0]15 . \quad (8.5)$$

The return and cost function can only be observed for parameters that lead to success, such that it is consistent with the other methods. Failures receive a constant cost of 15. The optimization step in CORL is done with a Newton method. The acquisition function uses for both GPs a squared exponential kernel (Equation (2.11)). The regression GP  $g_R$  is configured with the hyperparameters  $l = 0.4$ ,  $\sigma_f = 10$ , and  $\sigma_n = 0.11$ . The classification GP  $g_S$  is configured with  $l = 0.4$ ,  $\sigma_f = 10$ , and a constant prior mean  $m(\boldsymbol{\theta}) = -7$ . All algorithms are applied on the problem from 100 different initial parameters, which are sampled uniformly in the feasible search region. Table 8.4 shows the results of this experiment. We compare the metrics:

- **Global optima found:** This metric describes how many times the algorithm found the global optima  $\bar{\mathbf{x}}^*$ .
- **Max distance to safe region:** The maximum distance of all failure samples to the safety region. All values are given by mean and standard deviation.
- **Number of failures:** The number of failure samples  $S(\boldsymbol{\theta}) = 0$  that were selected by the algorithm until convergence. All values are given by the mean and standard deviation over the 100 trials.



Method	Global optima found	Max distance to safe region	Number of failures
PIBU	99/100	$0.64 \pm 0.40$	$5.27 \pm 0.68$
PoWER	88/100	$1.12 \pm 0.44$	$6.95 \pm 4.45$
UCB	95/100	$1.48 \pm 0.11$	$14.53 \pm 1.08$
CMA-ES	85/100	$1.20 \pm 0.38$	$7.19 \pm 4.50$
CORL + PIBU	<b>100/100</b>	<b><math>0.10 \pm 0.04</math></b>	$2.05 \pm 0.26$
CORL + UCB	<b>100/100</b>	$1.26 \pm 0.69$	<b><math>1.38 \pm 0.98</math></b>
CORL + CMA-ES	95/100	$0.97 \pm 0.53$	$3.73 \pm 2.53$
CORL + SAL	96/100	<b><math>0.06 \pm 0.12</math></b>	<b><math>1.57 \pm 3.38</math></b>

Table 8.4: **Synthetic benchmark experiment** (see Section 8.2.1) — Results of the CORL synthetic benchmark experiment.

The best two algorithms of each metric are marked bold in Table 8.4. The CORL + SAL method is as expected the safest method with a mean of 1.57 failure samples— but recall that it assumes it can observe the distance to the boundary in critical regions. The proposed method CORL + PIBU always finds the global optima and exhibits a very low number of near-boundary failures even without observing critical distance. The methods that do not take safety into account reach higher number of failure samples (between 5 and 15) that are also located far away from the safety region.

## 8.2.2 Evaluation of CORL on a Door Opening Skill

In this experiment, we apply Algorithm 2 on the task to open a door. The motion also includes the unlocking of the door by turning the handle first. We define the low-dimensional projection  $\theta$  as two parameters in the contact space of the door handle (see Section 6.1). The first parameter is the finger position on the handle relative to the demonstration. The second parameter is the finger opening widths. Different grasps of such a projection are shown in Figure 8.11. The analytic cost function  $J(\bar{x})$  is defined as the sum of squared accelerations in configuration space of the motion with the goal to reach a smooth motion. The black-box return  $R(\theta)$  is defined as the amount of force to open the door, which is measured with a force torque sensor at the robot wrist. The success criteria  $S(\theta)$  is a Boolean function that tells if the door was opened successfully. In order to achieve an autonomous



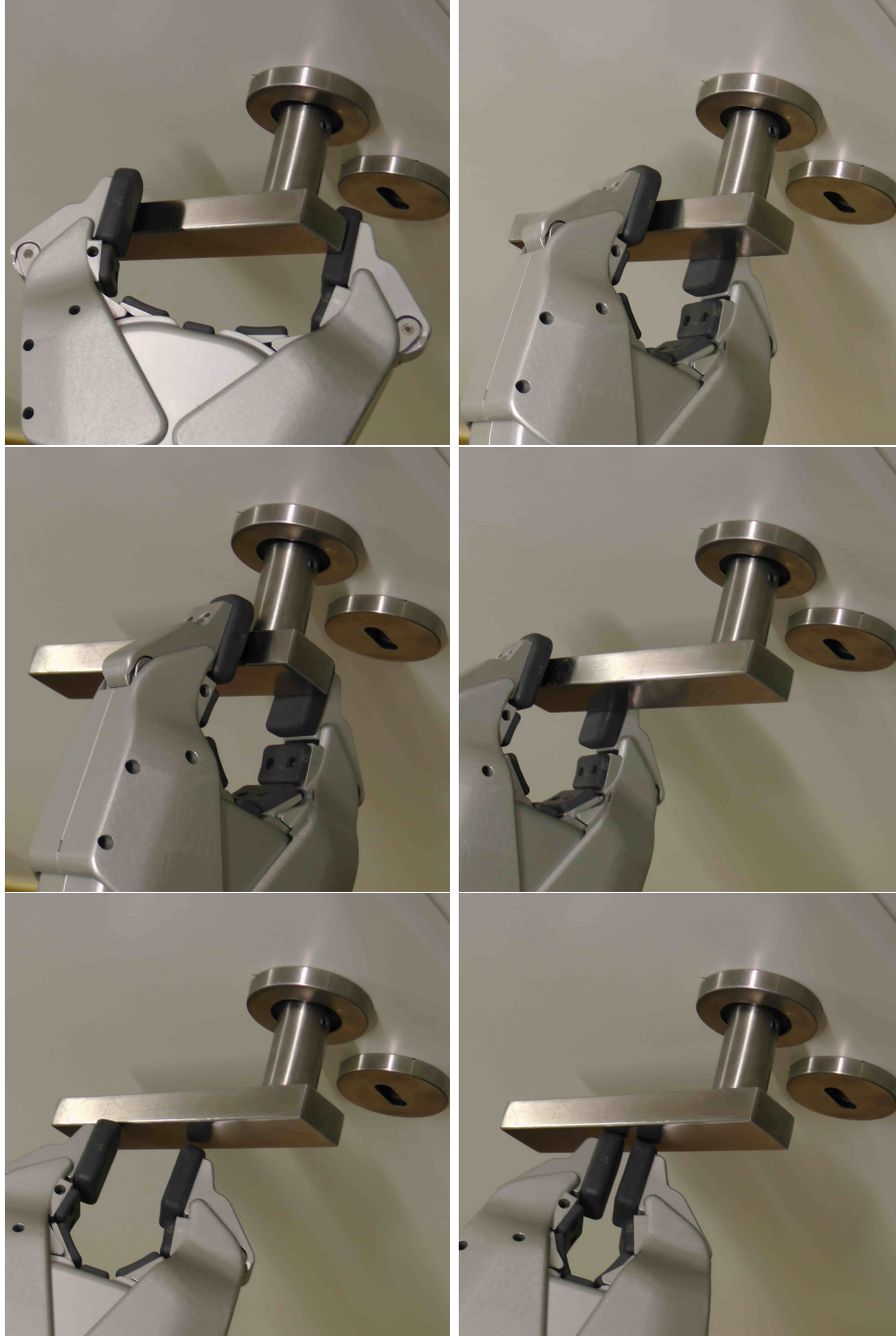


Figure 8.11: **Door experiment** (see Section 8.2.2) — The images show different grasps that were executed during learning.

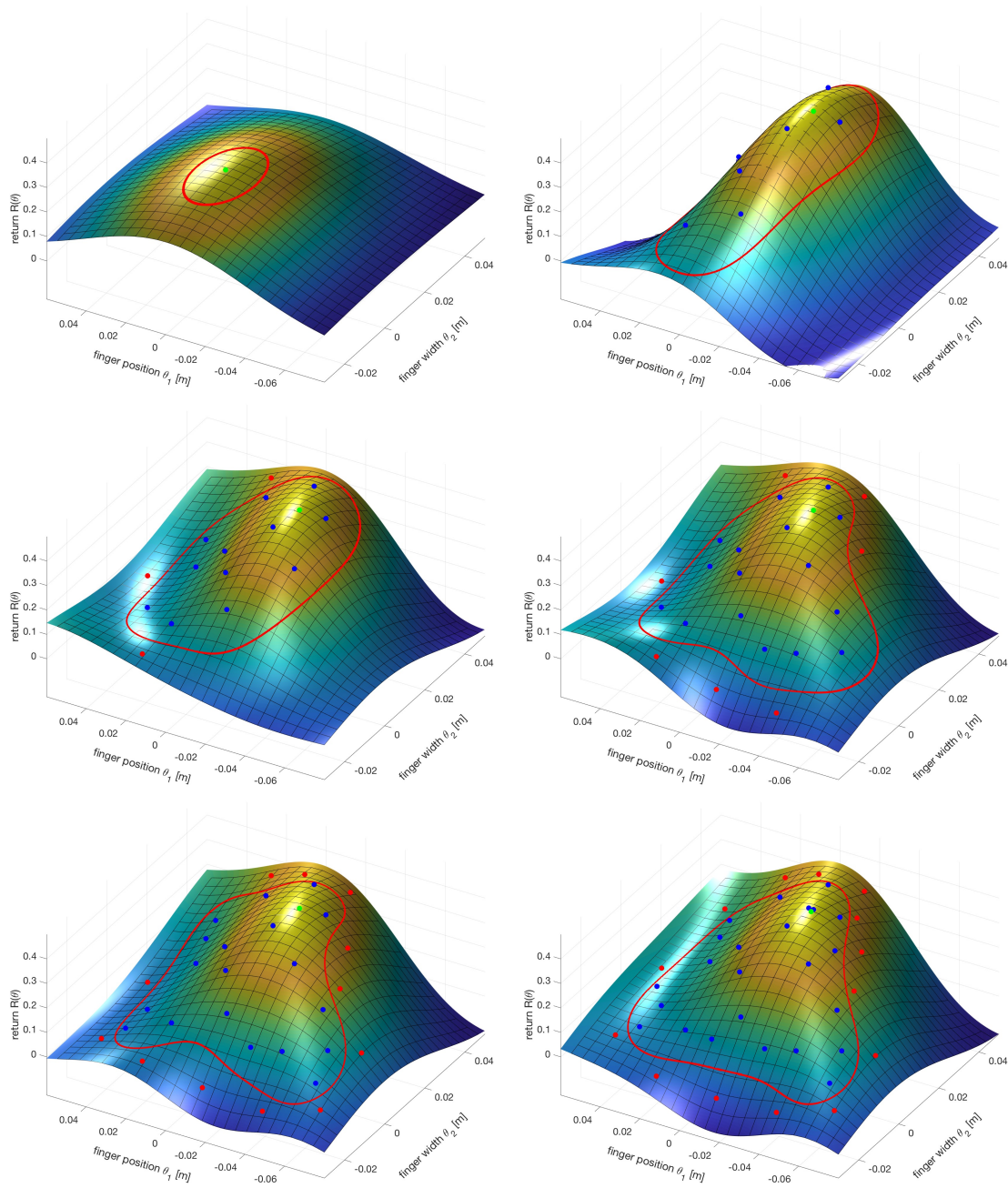


Figure 8.12: **Door experiment** (see Section 8.2.2) — The figures show the mean function of the GP  $g_R$ . Blue points denote successful rollouts, red points denote failures and the green point denotes the best parameters so far. The red line denotes the decision boundary of the GP classifier  $g_S$ .

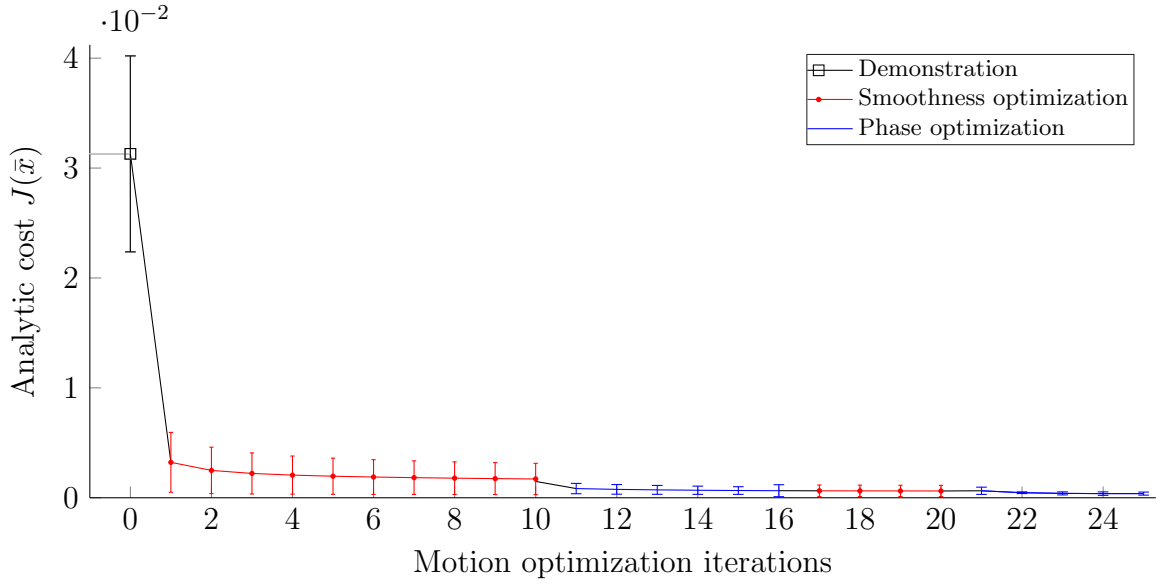


Figure 8.13: **Door experiment** (see Section 8.2.2) — The graph shows the analytic cost function over iterations. The skills starts from a demonstration and is improved with respect to smoothness and phase.

learning behavior we put markers on the door to measure if the door is open and add a simple motion that closes the door after each trial. This allows the robot to perform the learning on its own without human intervention. The parameters of the regression GP  $g_R$  are  $l = 0.042$ ,  $\sigma_f = 0.168$ , and  $\sigma_n = 0.012$ . The classification GP  $g_S$  uses the hyperparameters  $l = 0.02$ ,  $\sigma_f = 10$ , and a constant prior mean of  $-7$ . We compare this approach to a CORL + CMA-ES variant, where both CMA-ES and PIBU operate on the low-dimensional projection  $\theta$  exploiting the combination with the analytic motion optimization. The return and success function during different stages of the learning are shown in Figure 8.12. The proposed method converged in this run after 40 rollouts. From these 40 rollouts 26 were successes and 14 were failures. The blue dots are successful rollouts, the red dots are failures, and the green dot shows the best parameters so far. The red line denotes the current estimate of the classification boundary.

In this experiment, the constrained motion optimization framework is applied in a step wise manner. The problem in Equation (5.7) is solved iteratively with an additional constraint  $\|\bar{\mathbf{x}}^{(i+1)} - \bar{\mathbf{x}}^{(i)}\| < \epsilon_{\max}$  that limits the change between two trajectories. After each iteration, the trajectory is tested on the actual system to see if it still fulfills the task. Figure 8.13 shows the analytic cost of the demonstra-

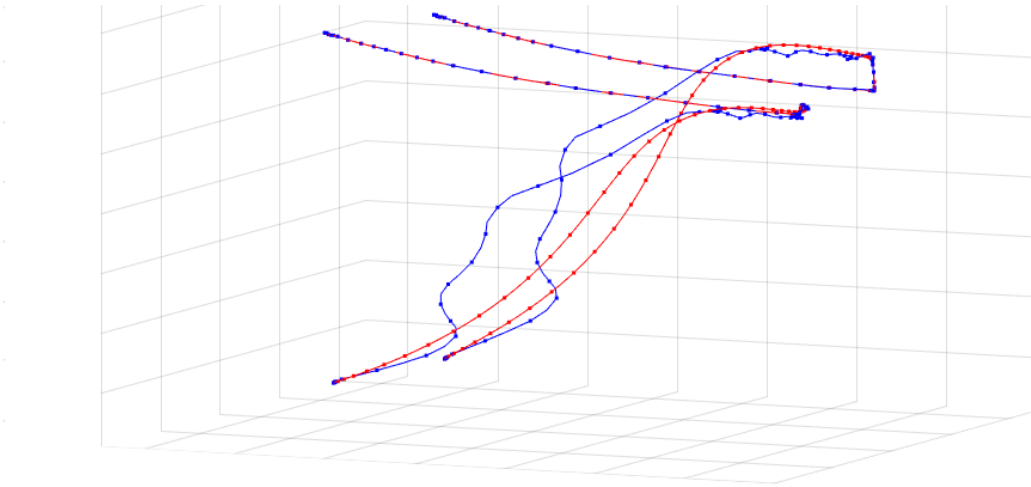


Figure 8.14: **Door experiment** (see Section 8.2.2) — The blue lines are the contact points trajectories of the demonstration. The red lines are the smoothed trajectories after the constrained motion optimization step is applied.

Algorithm	Highest return	Failure rate	Max distance to safety region
CORL + PIBU	$0.45 \pm 0.032$	$0.350 \pm 0.025$	$0.0146 \pm 0.006$
CORL + CMA-ES	$0.41 \pm 0.017$	$0.397 \pm 0.131$	$0.0358 \pm 0.030$

Table 8.5: **Door experiment** (see Section 8.2.2) — Comparison of CORL + PIBU with CORL + CMA-ES on the door opening skill.

tion trajectory as well as the costs of the trajectories that were optimized regarding smoothness and phase (see Section 5.3.2). The values denote the mean and double standard deviation of three experimental runs. Figure 8.14 depicts how the constrained motion optimization step improves the trajectories of the contact points.

The results of this experiment are summarized in Table 8.5. The reported performance measures are the highest return, the failure rate with the system, and the maximum distance to the safety region. All values are reported as mean and standard deviation over four runs on the real system. The results show that CORL + PIBU reaches a lower failure rate with a small standard deviation. The failures of PIBU are also closer to the safety region compared to CMA-ES. This results from the fact that the boundary is explored with the PIBU acquisition function (see Equation (5.6)). CORL + CMA-ES also finds a slightly worse policy than CORL. We tried alternative approaches that do not rely on this low-dimensional

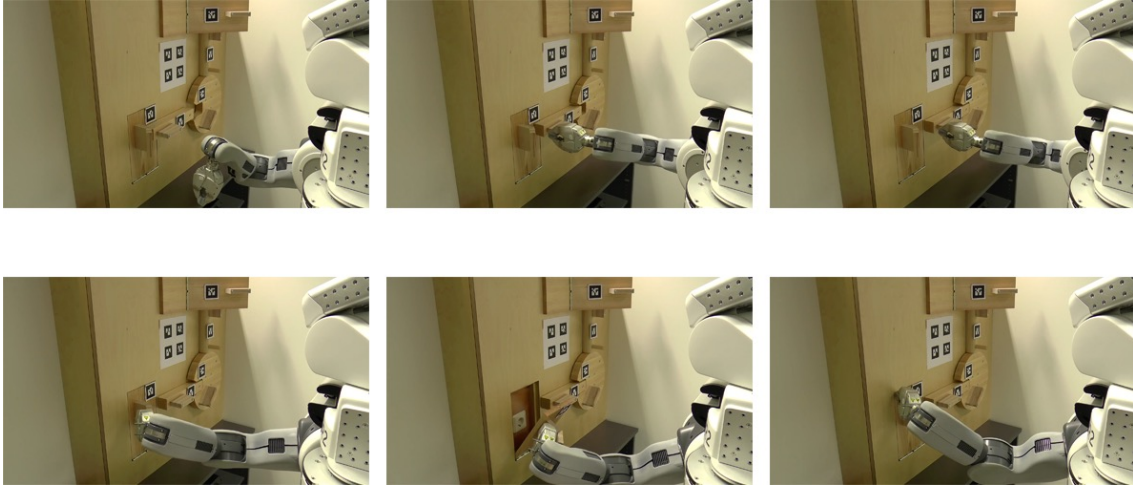


Figure 8.15: **Lockbox experiment** (see Section 8.2.3) — The top three images show a sequence of the sliding skill that we want to learn. The bottom three images show a sequence of the success constraint motion that checks if the sliding was successful, which is the case when the rotational joint can be manipulated.

projection and the combination with an analytic motion optimizer: We performed experiments with dynamic movement primitives and PoWER similar to (Kober and Peters, 2008). For this we parametrized the shape and goal of the DMP, leading to a 96 dimensional parameter space. However, we could not achieve a noticeable learning performance after 150 rollouts. We assume that the black-box return function that combines the applied forces with the path smoothness is not informative enough for this large parameter space. This reinforces the motivation for the general approach of dissecting objectives into high-dimensional analytical and low-dimensional black-box parts.

### 8.2.3 Learning Unlocking Mechanisms of a Lockbox

We evaluate CORL on an experiment where the objective is to learn the manipulation of a translational joint (see Figure 8.15). The lockbox was designed to do research on different physical exploration strategies (Baum et al., 2017) and consists of multiple rotational and translational degrees of freedom that lock each other. In this experiment, we focus on a translational joint that should be opened with a sliding motion. The low-dimensional projection  $\theta$  is defined as the vertical location of the contact point and the sliding velocity during manipulation (see Section 6.1). The goal is to manipulate the translational joint, such that the next joint is unlocked



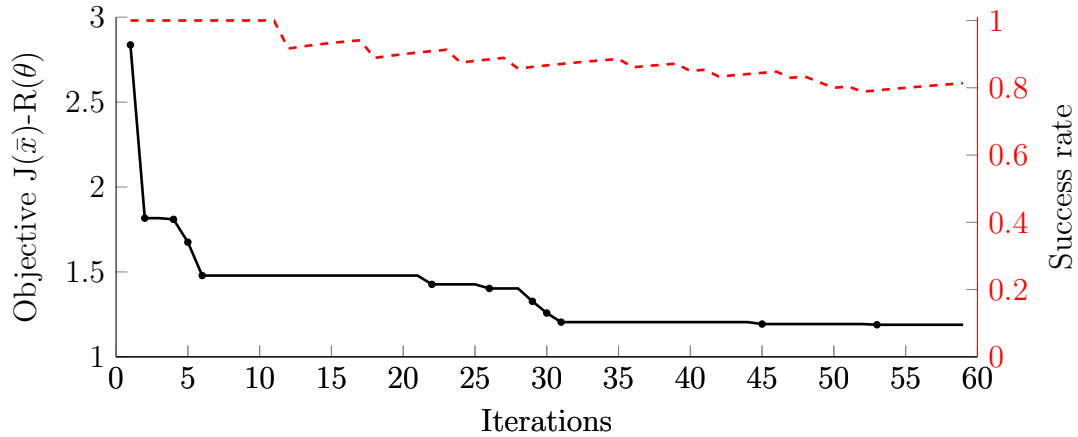


Figure 8.16: **Lockbox experiment** (see Section 8.2.3) — The figure shows the best objective (black solid line) and the success rate (red dashed line) over iterations.

(as shown in Figure 8.15 (top images)). The success constraint is evaluated by executing another motion that checks if the next joint is manipulatable (as shown in Figure 8.15 (bottom images)). Further, we also added a motion that closes both joints again, which again allows the robot to achieve a complete autonomous learning without human supervision. The GP hyperparameters and objectives are the same as in the previous section. CORL + PIBU converged after 59 iterations with 48 successes and 11 failures. The total interaction time of the robot was 61 minutes. In Figure 8.16, the learning curve (black line) and success rate (red dashed line) are visualized.

### 8.3 Learning a Manipulation Skill from a Single Demonstration

We evaluate the concept of learning a skill from a single demonstration (see Section 6.3) on the task of opening a cabinet with a PR2. The experimental setup is visualized in Figure 8.17. The skill consists of grasping the knob, rotating the knob to unlock the door joint, and opening the door. The full policy parametrization is a trajectory  $\bar{\mathbf{x}}$  that consists of 200 timesteps and 11 degrees of freedom (9 belong to the robot and 2 to the cabinet). The trajectory is executed with a duration of 15 seconds. We recorded a single demonstration with kinesthetic teaching as initialization.

The low-dimensional projection  $\boldsymbol{\theta} \in \mathbb{R}^3$  is defined as interaction parameters with the cabinet (see Section 6.1). The first parameter is the opening angle of the cabinet at the end of the skill. The second parameter is the reference gripper opening, which corresponds to the amount of force that is applied while grasping the knob. The third parameter is the final angle of the knob, which is important for unlocking the cabinet door joint. All parameters are defined relative to the initial demonstration. The analytic cost function  $J(\bar{\mathbf{x}})$  measures the sum of squared accelerations over the complete trajectory. The black-box return  $R(\boldsymbol{\theta})$  is the negative amount of force applied during the opening, which is measured with a force/torque sensor in the wrist of the robot. The black-box success  $S(\boldsymbol{\theta})$  is the binary signal if the cabinet door was successfully opened to a desired degree.

In the first part, we use CORL to improve the skill with respect to the defined objectives. We compare four different algorithm configurations of CORL on this problem:

1. **CORL + PIBU** ( $l = 0.3$ ): Bayesian optimization with the acquisition function PIBU and a wide kernel length scale  $l$ . The hyperparameters are  $\sigma_f = 0.5$  and  $\sigma_n = 0.01$  for the regression GP  $g_R$  and  $\sigma_f = 6$  for the classification GP  $g_S$ .
2. **CORL + PIBU** ( $l = 0.15$ ): PIBU with a narrow kernel length scale  $l$ . The other hyperparameter are identical to configuration 1.
3. **CORL + CMA-ES** ( $\sigma = 0.3$ ): CMA-ES with a high initial variance  $\sigma^2$ , a population size of 7 and 3 parents.

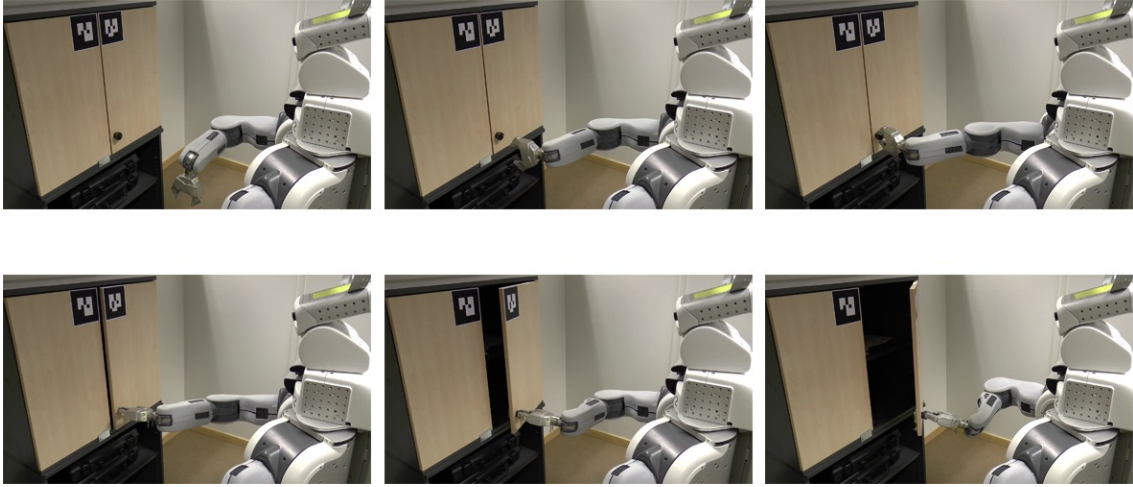


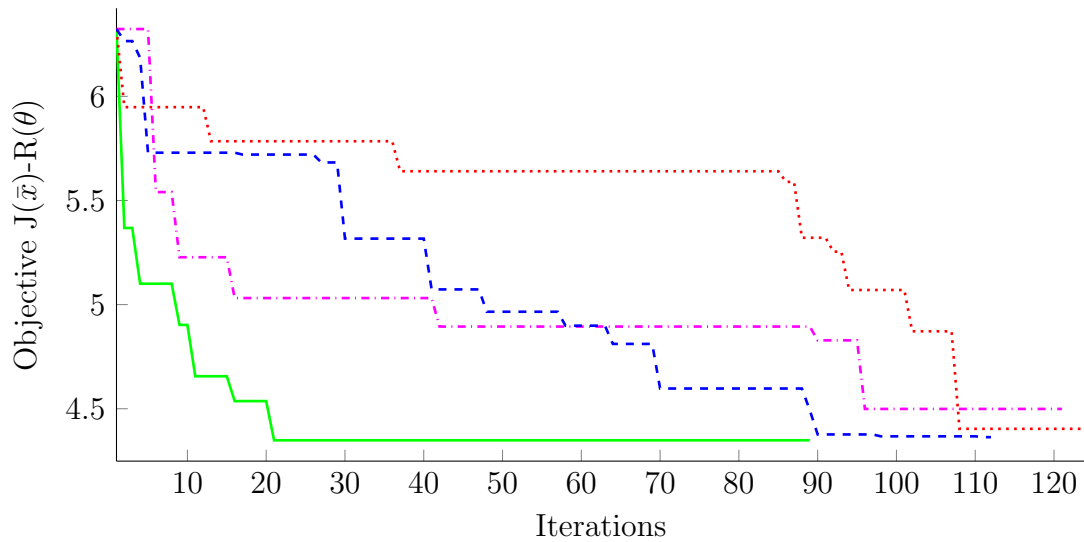
Figure 8.17: **Cabinet experiment** (see Section 8.3) — The images show a sequence of the learned manipulation skill on the cabinet environment.

4. **CORL + CMA-ES** ( $\sigma = 0.1$ ): CMA-ES with a small initial variance  $\sigma^2$ . The other parameters are identical to configuration 3.

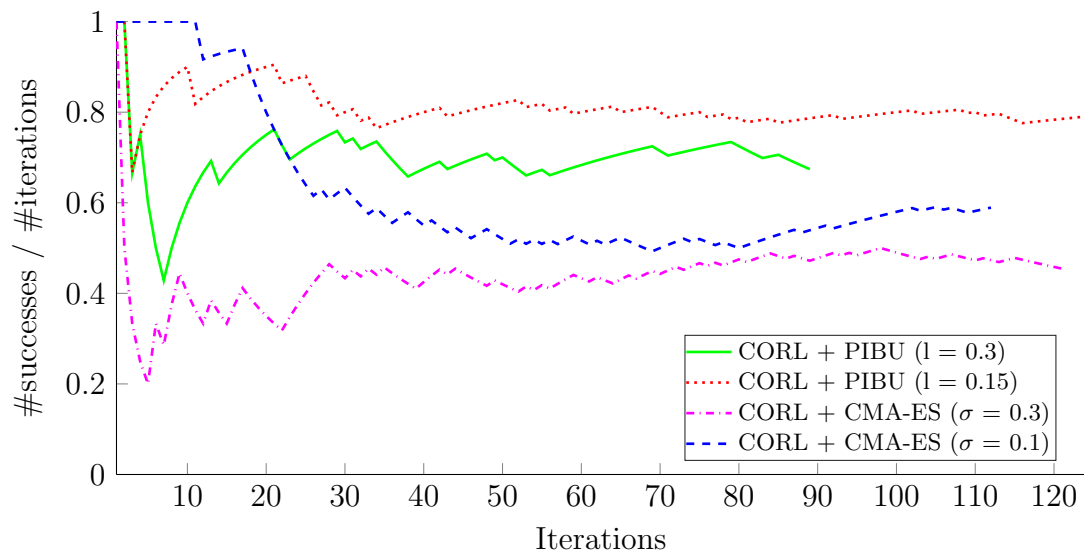
CORL + PIBU is evaluated with two different hyperparameter configurations of the squared exponential kernel (Equation (2.11)), which corresponds to how far a datapoint extrapolates its value. In the CMA-ES case, we evaluate a configuration with a small initial variance and a configuration with a wide initial variance of the samples.

The results of the experiment are visualized in Figures 8.18–8.20. Figure 8.18a shows the best candidate and Figure 8.18b shows the success rate over iterations. All methods lead to a similar best policy and the learning behavior depends on the hyperparameters of each method. The best policy receives an objective of 4.33 where the analytic cost  $J(\bar{\mathbf{x}})$  is 0.96 and the black-box return  $R(\boldsymbol{\theta})$  is  $-3.37$ . The configuration CORL + PIBU ( $l = 0.30$ ) already finds its best solution after 20 iterations, but requires more iterations until convergence to explore the whole success region. The configuration CORL + PIBU ( $l = 0.15$ ) leads to the highest success rate of 0.8, but also requires more iterations until convergence than other variants. This result shows the tradeoff that has to be made between convergence speed and safe exploration. Figure 8.19 shows the three dimensional projection space  $\boldsymbol{\theta}$  for the configuration CORL + PIBU ( $l = 0.3$ ). The success region  $g_S(\boldsymbol{\theta}) = 0$  is visualized as gray surface and all the points inside lead to success and the points outside to failure. The successful samples are visualized as dots and the color denotes the return





(a) Best candidate



(b) Success rate

Figure 8.18: **Cabinet experiment** (see Section 8.3) — The graph in (a) shows the best candidate over iterations. The graph in (b) shows the success rate over iterations that measures how many failures were executed on the system compared to the number of iterations so far.

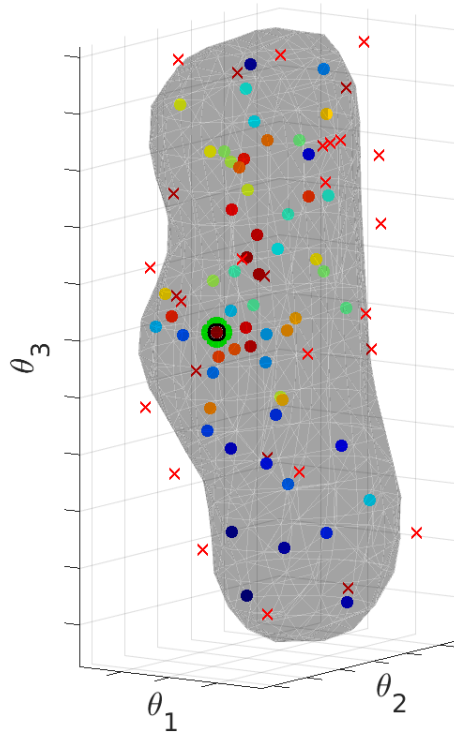
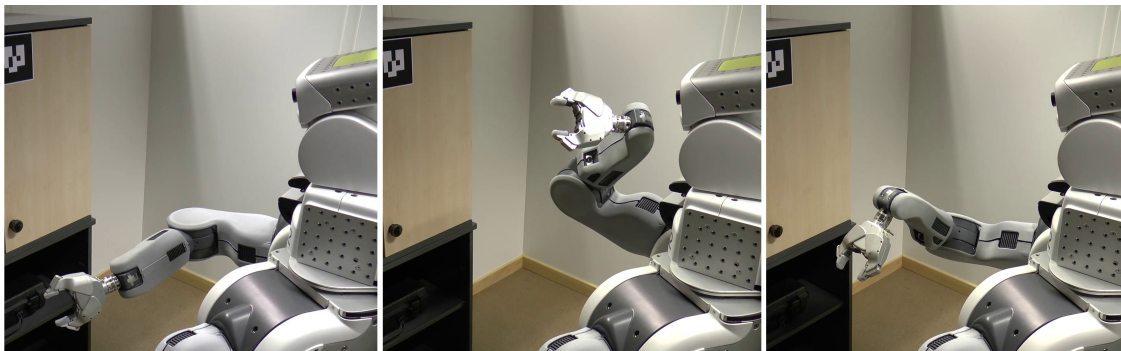


Figure 8.19: **Cabinet experiment** (see Section 8.3) — This image shows the classification boundary in the projection space  $\theta$  for the cabinet skill, which classifies the successful parameters (dots) from the failures (red crosses).

value. Blue dots have the lowest and red dots have the highest return value. The failures are visualized as red crosses. The best candidate is visualized with a green circle around it. The sequence of the resulting motion is visualized in Figure 8.17. In the second part, the collected dataset is taken to learn a higher-level cost function representation of the skill. We select three successful trajectories from the dataset  $\mathcal{D}$  and applied the IKKT algorithm with the features described in Section 6.2. The resulting cost function was able to generalize to different initial positions and door angles of the cabinet (see Figure 8.20a and 8.20b).



(a) Generalization to different initial states



(b) Generalization to different final door angles

Figure 8.20: **Cabinet experiment** (see Section 8.3) — The images (a) and (b) show the generalization of the constrained optimization problem to different initial positions and final door angles.

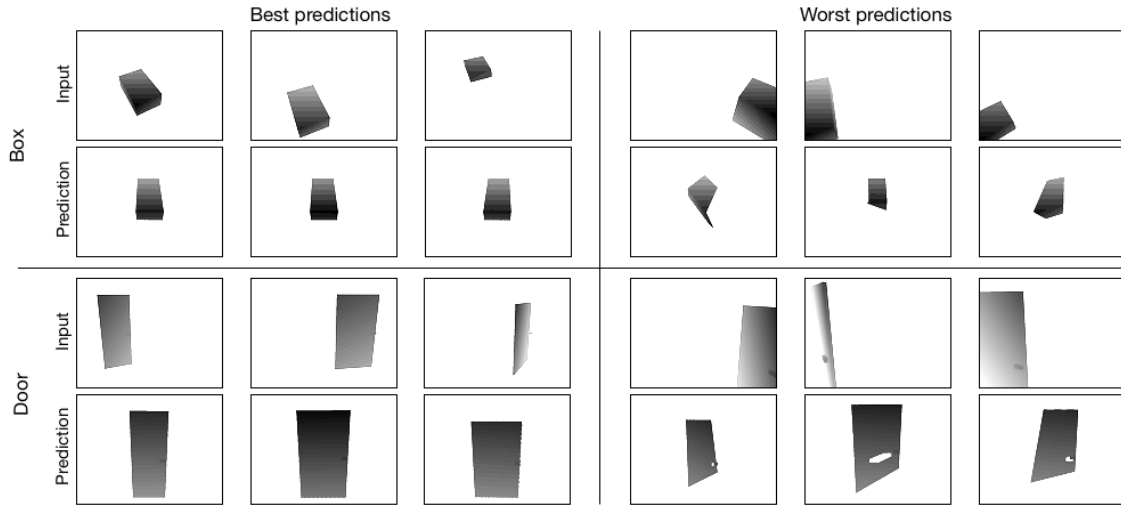


Figure 8.21: **KMN prediction accuracy** (see Section 8.4.1) — In the top row are the input depth images and on the bottom are the corresponding transformed images from the network predictions. The three samples on the left show the best predictions on the test dataset.

## 8.4 Evaluation of Kinematic Morphing Networks

In this section, kinematic morphing networks (Chapter 7) are evaluated based on three experiments. In the first experiment, the performance is compared to alternative strategies on multiple tasks with varying complexity. The second experiment shows the adaption to real world sensor data and the third experiment demonstrates the transfer of a policy between different simulated door environments.

### 8.4.1 Evaluation of Prediction Accuracy

The prediction accuracy of KMNs is evaluated on two tasks:

1. **box:** Three different box parametrizations are defined with varying complexity: A)  $n = 2$ : the box is only translated along the horizontal  $x$  and  $y$  direction; B)  $n = 3$ : the box is additionally rotated around the  $z$  axis; C)  $n = 5$ : the box is additionally scaled in its width and height.
2. **door:** This environment has  $n = 3$  transformation parameters and  $m = 4$  configuration parameters. The transformation parameters  $\xi$  are, similar to the box, the translation along  $x$  and  $y$  and the rotation around the  $z$  axis.

The configuration parameters  $\gamma$  are the size of the door and the location of the door handle. A sketch of this parametrization is shown in Figure 7.1.

Figure 8.21 shows several depth images of both tasks. The prototype is, in both tasks, defined at  $\xi_0 = \mathbf{0}$  and  $\gamma_0 = \mathbf{0}$ , which corresponds to the configuration where the object is directly in front of the robot. We compare different algorithms on both environments that predict the morphing parameters  $(\xi, \gamma)$  from depth images and point clouds. The algorithms are:

- **Kinematic morphing network (KMN):** This is the method proposed in this thesis (see Section 7.3). It is applied with  $N_{\text{pred}} = 5$  number of predictions.
- **Baseline:** Uses the same neural network as KMN. However, the network is trained only on the initially generated data without retraining and applied with a single prediction step ( $N_{\text{pred}} = 1$ ).
- **Iterative closest point (ICP):** The iterative closest point algorithm by Chen and Medioni (1992) with 100 iterations.

The results of this experiment are reported in Table 8.6. The table lists the environment parameters, number of filters in the five convolutional layers, and prediction error for all tasks and algorithms. The network architecture is described in Section 2.4. The dataset is split into 80% train and 20% test data. The initially generated amount of data  $N_{\text{data}}$  and the network architecture is chosen heuristically according to the complexity of the task. The number of augmented data points  $N_{\text{aug}}$  is always set to 20% of  $N_{\text{data}}$ . The reported error metric is the mean absolute error over 1000 data points and reported on the train and test set for all algorithms.

The results indicate that the proposed approach KMN achieves the lowest prediction errors on all tasks. The difference between KMN and Baseline comes through the iterative prediction and retraining mechanisms, since both variants have the same network architecture. Figure 8.22a compares the training error between Baseline and KMN on the box environment C. A different color denotes a new iteration of the KMN retraining loop in Algorithm 4. The data augmentation of KMN leads to a faster decrease of the training error. Figure 8.22b shows the mean prediction error with standard deviation of Baseline and KMN over number of network predictions on the test set. The first prediction of both networks achieves a similar error. However, the KMN method improves the prediction by applying the network

Scenario + Parameter	$L^{\text{low}}$	$L^{\text{up}}$	# Filter in Conv Layer	Baseline (Train)	Baseline (Test)	KMN (Train)	KMN (Test)	ICP (Train)	ICP (Test)
<b>box A</b> $N_{\text{data}} = 40000$ $\theta$ :			[2 4 6 8 10]	4.521e-03	4.512e-03	<b>1.429e-03</b>	<b>1.417e-03</b>	1.668e-02	1.681e-02
x translation	-0.40	0.40		2.124e-03	2.075e-03	6.870e-04	7.007e-04	7.419e-03	7.183e-03
y translation	-0.40	0.40		2.397e-03	2.437e-03	7.417e-04	7.165e-04	9.258e-03	9.623e-03
<b>box B</b> $N_{\text{data}} = 60000$ $\theta$ :			[2 4 6 8 10]	7.125e-02	6.968e-02	<b>7.335e-03</b>	<b>7.345e-03</b>	6.370e-01	6.281e-01
x translation	-0.40	0.40		9.530e-03	9.559e-03	1.590e-03	1.629e-03	1.023e-02	1.057e-02
y translation	-0.40	0.40		1.161e-02	1.129e-02	1.790e-03	1.763e-03	1.281e-02	1.332e-02
z axis rotation	-1.05	1.05		5.010e-02	4.883e-02	3.955e-03	3.953e-03	6.140e-01	6.042e-01
<b>box C</b> $N_{\text{data}} = 100000$ $\theta$ :			[4 8 10 12 14]	1.886e-01	1.911e-01	<b>4.242e-02</b>	<b>4.295e-02</b>	-	-
x translation	-0.40	0.40		1.348e-02	1.345e-02	2.904e-03	2.807e-03	-	-
y translation	-0.40	0.40		1.307e-02	1.345e-02	2.835e-03	2.727e-03	-	-
z axis rotation	-1.05	1.05		8.080e-02	8.430e-02	9.675e-03	1.094e-02	-	-
length scaling	-0.40	0.40		4.213e-02	3.956e-02	1.383e-02	1.298e-02	-	-
height scaling	-0.50	1.50		3.913e-02	4.029e-02	1.318e-02	1.350e-02	-	-
<b>door</b> $N_{\text{data}} = 100000$ $\theta$ :			[2 4 8 16 32]	1.385e-01	1.394e-01	<b>5.253e-02</b>	<b>5.307e-02</b>	-	-
x translation	-0.80	0.80		1.217e-02	1.238e-02	2.954e-03	2.940e-03	-	-
y translation	-0.80	0.80		8.174e-03	8.471e-03	3.236e-03	3.192e-03	-	-
z axis rotation	-1.05	1.05		1.976e-02	1.948e-02	5.275e-03	5.512e-03	-	-
$\gamma$ :									
door height	-0.40	0.20		1.504e-02	1.473e-02	1.199e-02	1.177e-02	-	-
door width	-0.20	0.20		1.662e-02	1.686e-02	4.539e-03	4.474e-03	-	-
handle y	-0.04	0.04		1.858e-02	1.903e-02	1.058e-02	1.068e-02	-	-
handle z	-0.10	0.10		4.818e-02	4.849e-02	1.395e-02	1.449e-02	-	-

Table 8.6: **Prediction accuracy** (see Section 8.4.1) — Results of the kinematic morphing network experiment.

multiple times. After 3 iterations there is no significant change in the accuracy anymore. The prediction error of Baseline increases when applied iteratively since it only was trained on the initial dataset. This shows that the retraining mechanism is necessary to successfully apply the iterative predictions. The ICP algorithm is only applied on box A and box B since ICP cannot handle configuration parameters. ICP achieves a reasonable performance on the translation parameters of both environments. However, ICP has difficulties on predicting the rotation parameter in both tasks. It did not always detect the correct rotation direction or led to rotations that flipped the box.

Figure 8.21 shows different samples of the dataset (top row) with the corresponding network prediction (bottom row) separated in best and worst predictions. The

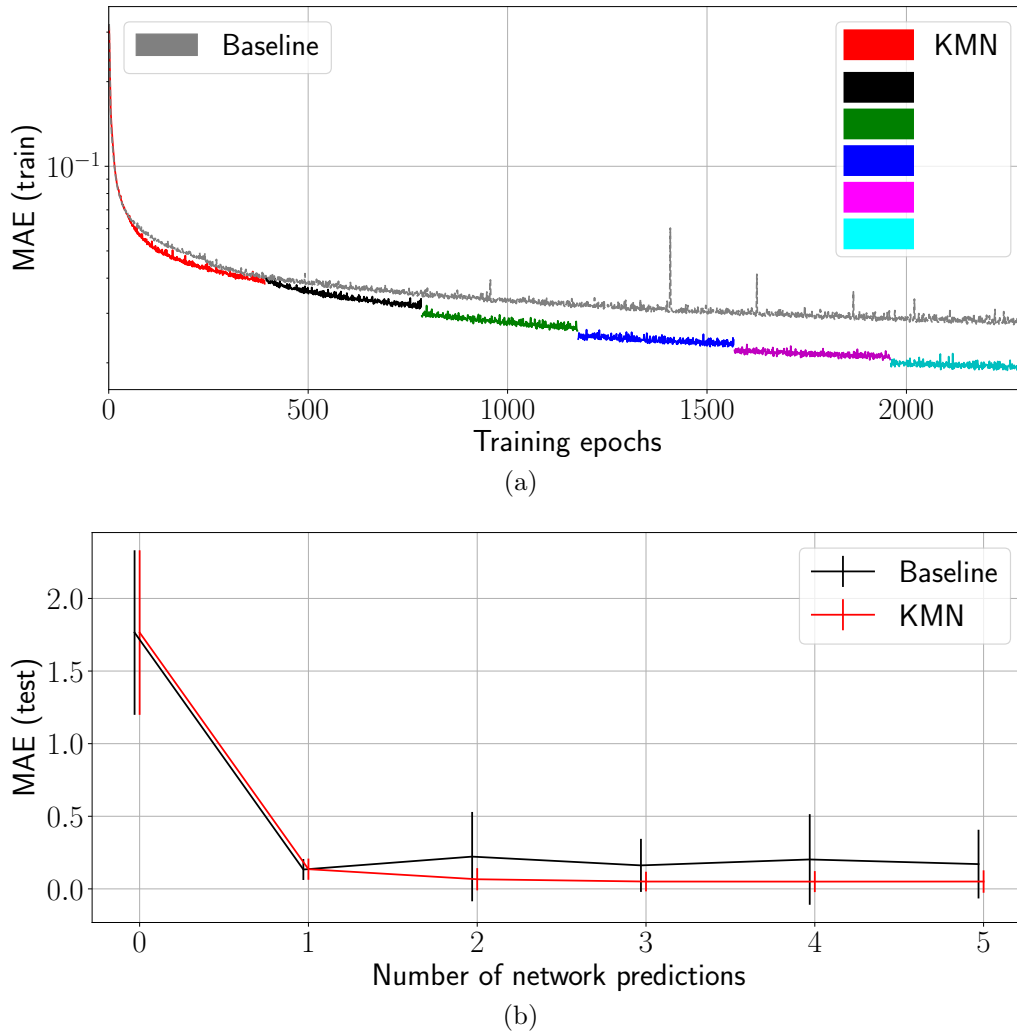


Figure 8.22: **KMN prediction accuracy** (see Section 8.4.1) — The graphs in (a) show the training error of the Baseline and KMN variant. The alternating colors denote a new retraining iteration in Algorithm 4. The plot in (b) shows the prediction error with standard deviation over multiple network predictions.

worst predictions occurred when the box/door was only partially observed. This makes sense since it is difficult to estimate the height of a door when it is not fully visible. The samples also show that the point cloud transformation and the subsequent rendering can lead to holes in the depth image. However, since the KMN network also has such data points in the training phase, it could handle them better than Baseline.

### 8.4.2 Evaluation on Real Sensor Data

In this experiment, we evaluate how the KMN model performs on real sensor data. The trained model of the box C task from the previous section is applied on data recorded with a Kinect v1 camera. The point clouds are recorded with the IR depth-finding camera of the Kinect. We tried to reproduce the simulated environment in the real world (e.g., same camera pose). The point clouds are preprocessed by transforming the points from camera into world frame and removing the points that do not belong to the object. We put the box at 15 different locations inside the field of view of the camera. The network was able to predict morphing parameters for all 15 samples that transform the observed box close to the prototype. Figure 8.23 shows different simulated and real point clouds overlaid with their predicted box location (green box). The reached accuracy was lower and the number of network predictions  $N_{\text{pred}}$  until convergence was slightly higher when applying the network on real sensor data.

### 8.4.3 Skill Transfer on the Door Task

We use the trained KMN model to transfer a skill policy between different doors. The policy is defined on the kinematic model  $\mathbf{m}(\boldsymbol{\xi}, \boldsymbol{\gamma})$  as a constrained motion optimization problem (see Section 2.1). The robot is a PR2 and the trajectory  $\bar{\mathbf{x}}$  consists of  $T = 200$  configurations of the robot base (3 DOF), left arm (7 DOF), gripper (1 DOF), and door (2 DOF). We defined the features  $\Phi$  of the cost function as

- Base pose in front of the door.
- Pre-grasp pose of gripper in front of door handle.
- Target state of handle joint.
- Target state of door joint.

The equality constraint  $\mathbf{h}$  consists of a feature that describes the contact between door handle and gripper (similar to the experiment in Section 8.1.5). Specifically, two points are defined on the door handle and on the robot gripper. The constraint measures the difference between a point pair that should be zero during the manipulation. Further constraints are defined to avoid collisions and to fix joints when



they are not being manipulated. This policy generalized to various geometric door instances. Figure 8.24 shows the environment morphing and opening motion on one instance of the door environment.

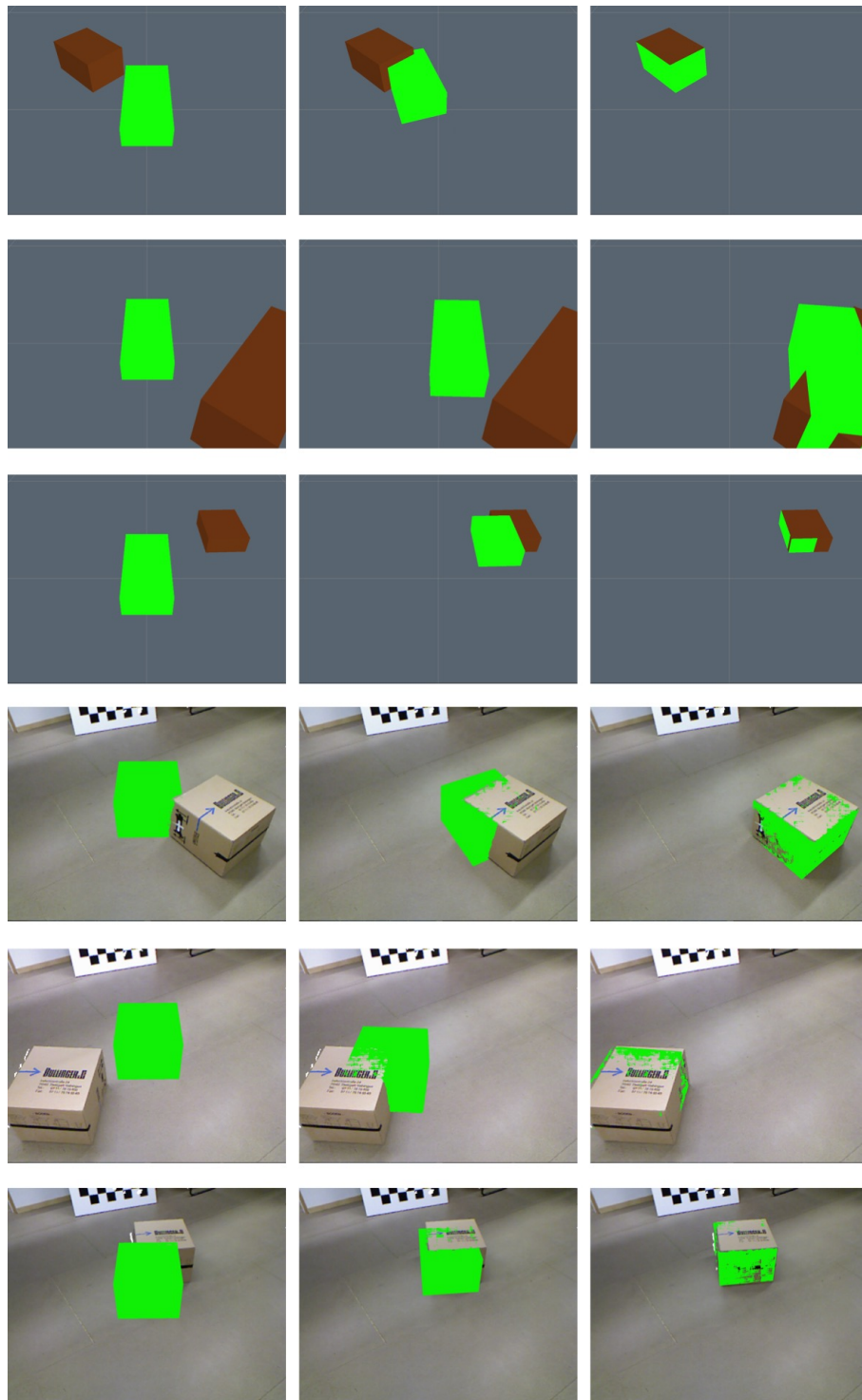


Figure 8.23: **KMN predictions** (see Section 8.4.2) — Kinematic morphing network predictions (green box) overlaid with point clouds from a simulated or real Kinect camera. The images show the morphing from the prototype to the observed box.

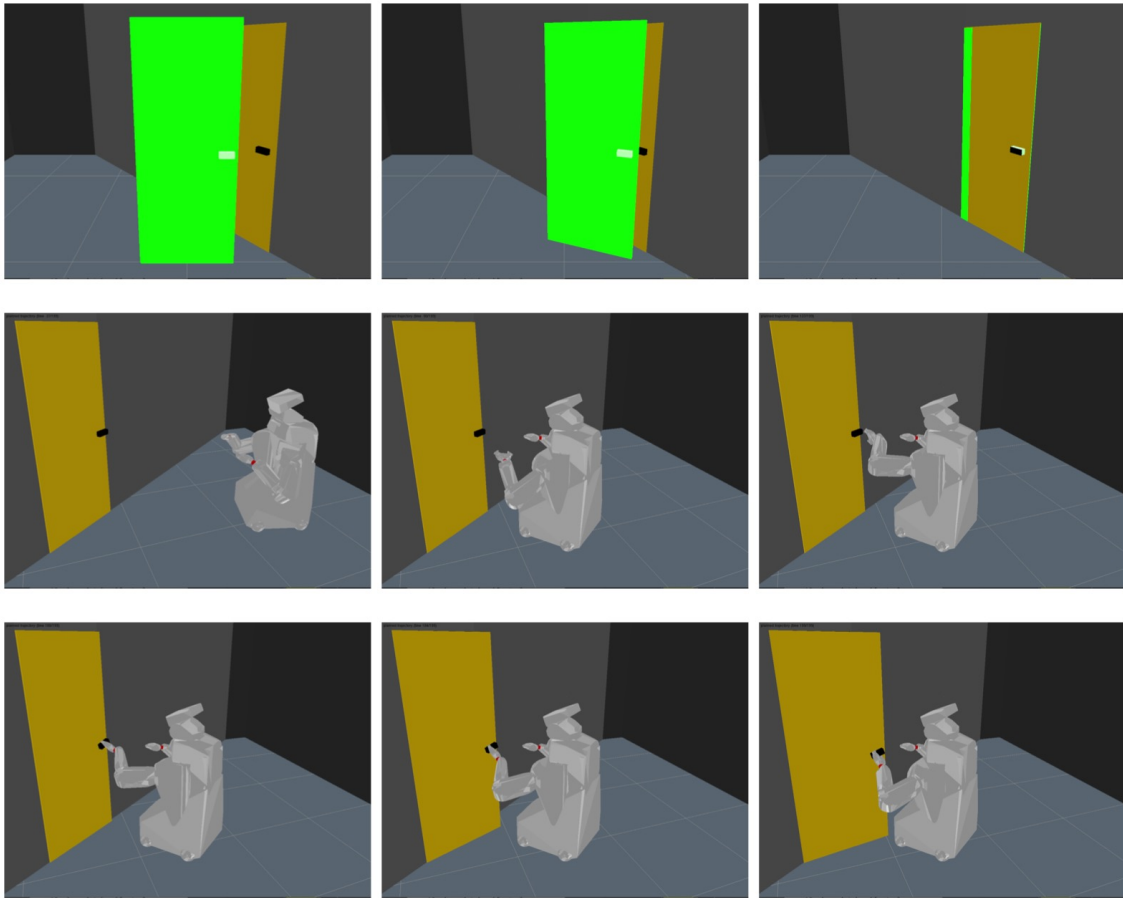


Figure 8.24: **KMN skill transfer** (see Section 8.4.3) — Sequence of door morphing transformation and opening motion.



# Chapter 9

## Conclusion

In this chapter, the contributions and results of this thesis are summarized. Afterwards, we give an outlook on potential future research directions.

### 9.1 Summary

In this thesis, we proposed various algorithms for structured manipulation skill learning in robotics. The algorithms rely on data-driven techniques and have the goal to advance manipulation skill learning towards sample-efficiency and generalization.

#### 9.1.1 Inverse KKT

In Chapter 4, we proposed the algorithm IKKT to learn skills by imitating human demonstrations. The main purpose of this strategy is to learn a high-level skill representation in form of a cost function that is generalizable to various environment configurations. The method is formulated as a counterpart to the finite horizon constrained optimal control method KOMO and finds optimal cost functions such that the demonstrations fulfill the KKT optimality conditions. We presented two representations of the cost function: 1) A weighted sum of squared features; 2) A functional in a reproducing kernel Hilbert space. In Section 6.2, we described how IKKT can be applied for robot manipulation tasks by defining a set of features and constraints that describe the interaction between the robot and the environment. The experimental results showed that Inverse KKT is able to extract cost function parameters from demonstrations under the assumptions that the demonstrations are optimal and that the feature set is rich enough. The algorithm was applied to various robot manipulation tasks, such as sliding a box or opening a door, where it learned cost functions that generalized the skills to various task variations. The comparison to black-box inverse optimal control indicated that Inverse KKT requires a lower computation time and finds cost functions that lead to motions with a high similarity to the demonstrations.

### 9.1.2 Combined Optimization and Reinforcement Learning

Afterwards, we introduced the algorithm CORL (Chapter 5), which combines different objectives and optimization methods to improve a skill. The idea behind CORL is to give robots the ability to autonomously improve a manipulation skill in a safe and data-efficient manner. A main component of the formulation is the low-dimensional projection of the policy that parametrizes the interaction with the environment. In Section 6.1, we describe how CORL can be efficiently configured for manipulation skills by exploiting the underlying problem structure. CORL improves this interaction with episodic reinforcement learning and the full motion with analytic motion optimization. One advantage of this separation is that it is not necessary to specify models of the physical interaction between the robot and the environment, which are difficult to obtain. Another advantage is that analytic optimization can very efficiently improve a skill with respect to the full motion while the learning part can focus on improving the black-box objectives regarding the low-dimensional projection. We also presented an approach in Chapter 6 to concatenate CORL and Inverse KKT in order to learn a generalizable skill from a single demonstration. The experimental evaluation revealed that CORL is able to improve real robot manipulation skills with a reasonable amount of rollouts. The results on the benchmark problem showed that CORL finds a tradeoff between a safe and data-efficient learning, which depends on the algorithm configuration and hyperparameters.

### 9.1.3 Kinematic Morphing Networks

Finally, in Chapter 7, we presented kinematic morphing networks to extract a low-dimensional representation of the environment that can be used as input for manipulation skills. The low-dimensional representation is defined as parameters of the kinematic structure of the manipulated environment. Kinematic morphing networks extract these parameters from depth images and are trained on data generated with a simulator. The network prediction and training steps are implemented in an iterative manner by applying transformations on point clouds and converting them to depth images. The approach was evaluated on different kinematic environments such as boxes and doors. The results showed that the iterative predictions lead to a higher accuracy in comparison to single predictions. A comparison to ICP suggested

that the kinematic morphing network achieves a better performance in terms of prediction accuracy. Further experiments demonstrated the network performance on real sensor data and the transfer of a skill by taking the extracted parametrization as input to a motion planning method.

## 9.2 Outlook

The proposed algorithms address specific issues in manipulation skill learning. However, there are still open problems that need to be addressed to further improve the skill acquisition abilities of robots. In the following sections, we will discuss potential future research directions.

### 9.2.1 Towards Autonomous Learning in Robotics

A major goal in skill learning research is to achieve a fully autonomous learning behavior that does not require human supervision. One of the difficulties in reaching such a behavior lies in the problem that the overall search space of a skill is quite high-dimensional and in order to achieve a sample-efficient learning it is necessary to incorporate existing knowledge that reduces the search space. In this thesis, we incorporated this knowledge by providing demonstrations of a skill, defining low-dimensional projections, and employing robot simulators. These points were significant for the proposed algorithms to achieve a successful learning behavior. However, there is still room for improvement in all of these points in order to achieve an even more autonomous learning behavior.

The experimental results of this thesis show that the usage of demonstrations was very important for achieving a sample-efficient learning on the real system. In CORL, a single demonstration was recorded as input to have a good initial guess of the policy parameters for the subsequent local optimization methods. Inverse KKT used multiple expert demonstrations to extract generalizable features from them. In both methods, we tried to reduce the amount of human demonstrations as far as possible. An assumption that we made is that the demonstrations are performed directly on the robot body via kinesthetic teaching. This assumption is quite strong and requires a human to actively provide demonstration of a task. Future research should try to relax it towards a passive observation of human behavior (e.g., through video). There, a main challenge lies in extracting a suitable representation (e.g.,

kinematic model of the human/environment) from such observations in order to use it in imitation learning frameworks. Another challenge lies in addressing the correspondence problem (Nehaniv and Dautenhahn, 2002), which occurs when the body of the teacher and robot have different geometric or physical properties. A potential solution might be to search for a feasible matching of features between the different bodies and to use Inverse KKT purely on those features. Another issue that may occur in certain tasks is that they cannot be demonstrated by humans due to various reasons. In such a case, a solution may be to initialize learning by taking motions from similar tasks instead of human demonstrations.

Another way to further automate the skill learning process is to integrate expert domain knowledge in an abstract form into the learning algorithms, such that it is usable for many tasks in that domain. We defined the inputs of the learning methods in a task independent manner. For example, the same projection constraint in CORL (Section 6.1) or set of cost features in Inverse KKT (Section 6.2) could be applied in multiple manipulation tasks. Such a definition of task independent features is important to avoid the need of human adaptation for specific tasks. In the proposed algorithms, the integration of expert knowledge was only possible by designing the learning algorithm such that they mirror the problem structure (e.g., contacts as constraints). Further research might define this expert knowledge for other application domains or try to relax certain assumptions we made (e.g., rigid body) to enhance the range of potential applications.

### 9.2.2 Increasing the Interaction between Algorithms

The aim of skill learning in robotics is to achieve multiple objectives such as performance, safety, and generalization at the same time. An open question in this area is how to find a good tradeoff between the various objective criteria. In this thesis, the interplay of different types of learning algorithm and policy representations was crucial to learn complex skills. In Section 6.3, we combined imitation learning, black-box reinforcement learning, and analytic motion optimization with each other. CORL was employed to improve and explore a skill while Inverse KKT was used to learn a generalizable representation. Future research should focus on increasing the interaction of different algorithms in a sequential or hierarchical way to create more intelligent behavior. Therefore, the interface of these methods has to be defined in a proper way and it may be necessary to design algorithms that do



not directly produce relevant outputs but instead provide usable inputs for other algorithms.

Many algorithms and objective criteria require specific policy representations, which make it necessary to provide a functionality that can convert various policy representations into each other. The work in this thesis also showed that certain policy representations are more suited for learning specific types of objectives. For example, in CORL we used a trajectory as a representation of a skill that was well suited for analytic motion optimization methods. In Inverse KKT on the other hand, we employed abstract cost functions as skill representations since their generalization capabilities are higher. A goal of future research might be to investigate other types of policy representations like deep neural networks or higher level planning policies on a symbolic level.

### 9.2.3 Learning Complex Manipulation Models

Providing robots with intelligent behavior makes it necessary that they get an understanding about how the world works and how their actions can change the world in a desired way. In this thesis, we designed and configured the algorithms for environments that consist of rigid bodies and joints. The structure and state of such environments can be represented in a low-dimensional manner, which we exploited during the feature design to achieve an efficient learning. We also used this low-dimensional representation as output for the proposed kinematic morphing networks where we assumed to have a generative model in form of a simulator that can generate synthetic observations for given parameters. However, the rigid body assumption also limits the range of real world tasks on which the algorithms can be applied. The extension of the proposed techniques to tasks with more complex objects (e.g., deformable, elastic, liquid) is an interesting question for future research. The interaction models of such environments are even more complex than the models considered in this thesis. It would be interesting to investigate if the proposed CORL approach still performs well on such environments. An alternative to CORL, which maps interaction parameters to return values, may be to learn interaction models from data and use them as a simulator to optimize or test policies before applying them on the real system. Recent advances in the area of deep learning may be a promising way to achieve this goal by providing a powerful representation and training it with large amounts of data. However, the collection of such large

datasets is difficult to do on the real system. A solution may be to use active learning strategies to specifically select actions that reduce the model uncertainty and close the gap between the simulated and the real world.



# List of Symbols

	<b>General</b>
$m(\mathbf{x})$	GP mean function
$k(\mathbf{x}, \mathbf{x}')$	GP covariance function
$\mathbf{K}$	GP Gram matrix
$\sigma_f$	GP signal standard deviation
$\sigma_n$	GP noise standard deviation
$l$	GP kernel length scale
$\mu(\mathbf{x}_*)$	GP predictive mean
$\mathbb{V}(\mathbf{x}_*)$	GP predictive variance
$a(\mathbf{x}; \mathcal{D})$	acquisition function
$f(\mathbf{x}; \boldsymbol{\omega})$	neural network
$\boldsymbol{\omega}$	neural network parameters
$\mathbf{W}^{(i)}$	neural network weight parameters
$\mathbf{b}^{(i)}$	neural network bias parameters
$\mathbf{z}^{(i)}$	neural network hidden states
$\mathbf{a}^{(i)}$	neural network activations
$h^{(i)}(\mathbf{z}^{(i)})$	neural network activation function
$\sigma(x)$	sigmoid function
$[<cond>]$	Iverson bracket

	<b>KOMO/IKKT</b>
$\mathbf{x}_t$	robot configuration at time t
$\tilde{\mathbf{x}}_t$	$k$ -order robot configuration
$\bar{\mathbf{x}}$	robot configuration trajectory
$d$	trajectory duration
$T$	number of trajectory time steps
$\Delta t$	trajectory time step size
$p(t)$	phase profile of trajectory
$\mathbf{y}$	environment
$\phi_t(\tilde{\mathbf{x}}_t, \mathbf{y})$	robot state kinematic map
$\mathbf{J}(\tilde{\mathbf{x}}_t, \mathbf{y})$	Jacobian of state kinematic map
$\Phi(\bar{\mathbf{x}}, \mathbf{y})$	robot trajectory kinematic map
$\mathbf{J}(\bar{\mathbf{x}}, \mathbf{y})$	Jacobian of trajectory kinematic map
$\mathcal{F}(\bar{\mathbf{x}}, \mathbf{y}, \mathbf{w})$	trajectory cost function
$\mathbf{w}$	cost function weights
$\rho$	cost function weight parametrization
$\mathbf{g}(\bar{\mathbf{x}}, \mathbf{y})$	inequality constraints
$\mathbf{h}(\bar{\mathbf{x}}, \mathbf{y})$	equality constraints
$\mathbf{c}(\bar{\mathbf{x}}, \mathbf{y}) = \begin{bmatrix} \mathbf{g}(\bar{\mathbf{x}}, \mathbf{y}) \\ \mathbf{h}(\bar{\mathbf{x}}, \mathbf{y}) \end{bmatrix}$	constraints
$\mathbf{J}_c(\bar{\mathbf{x}}, \mathbf{y})$	Jacobian of constraints
$\tilde{\mathbf{J}}_c(\bar{\mathbf{x}}, \mathbf{y})$	Jacobian of active constraints
$\boldsymbol{\lambda}$	Lagrange multipliers
$L(\bar{\mathbf{x}}, \mathbf{y}, \boldsymbol{\lambda}, \mathbf{w})$	Lagrange function
$\ell(\mathbf{w}, \boldsymbol{\lambda})$	IKKT loss function
$\mathcal{C}_t(\mathbf{x}_t)$	cost functional

	<b>CORL</b>
$\boldsymbol{\theta}$	projection parameters
$\mathbf{h}(\bar{\mathbf{x}}, \mathbf{y}, \boldsymbol{\theta})$	projection constraint
$J(\bar{\mathbf{x}})$	analytic cost function
$R(\boldsymbol{\theta})$	black-box return function
$S(\boldsymbol{\theta})$	black-box success constraint
$g_S$	GP model of success constraint $S$
$g_R$	GP model of return function $R$
	<b>KMN</b>
$f(\mathbf{D}; \boldsymbol{\omega})$	kinematic morphing network
$\boldsymbol{\xi}$	transformation parameters
$\boldsymbol{\gamma}$	configuration parameters
$\mathbf{m}(\boldsymbol{\xi}, \boldsymbol{\gamma})$	kinematic model
$(L^{\text{up}}, L^{\text{low}})$	parameter limits
$T_{\boldsymbol{\xi}}$	affine transformation
$\mathbf{D}$	depth image
$\mathbf{P}$	point cloud

# Bibliography

- Achiam, J., Held, D., Tamar, A., and Abbeel, P. (2017). Constrained policy optimization. In *Proceedings of the International Conference on Machine Learning*.
- Albrecht, S., Ramirez-Amaro, K., Ruiz-Ugalde, F., Weikersdorfer, D., Leibold, M., Ulbrich, M., and Beetz, M. (2011). Imitating human reaching motions using physically inspired optimization principles. In *Proceedings of the International Conference on Humanoid Robots*.
- Aldous, D. J. (1985). Exchangeability and related topics. In *École d'Été de Probabilités de Saint-Flour XIII—1983*, pages 1–198. Springer.
- Argall, B. D., Chernova, S., Veloso, M., and Browning, B. (2009). A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57(5):469–483.
- Athans, M. (1971). The role and use of the stochastic linear-quadratic-gaussian problem in control system design. *IEEE Transactions on Automatic Control*, 16(6):529–552.
- Baum, M., Bernstein, M., Martín-Martín, R., Höfer, S., Kulick, J., Toussaint, M., Kacelnik, A., and Brock, O. (2017). Opening a lockbox through physical exploration. In *Proceedings of the International Conference on Humanoid Robots*.
- Berkenkamp, F. and Schoellig, A. P. (2015). Safe and Robust Learning Control with Gaussian Processes. In *Proceedings of the European Control Conference*.
- Bertsekas, D. P. (2001). *Dynamic Programming and Optimal Control*. Athena Scientific.
- Bousmalis, K., Irpan, A., Wohlhart, P., Bai, Y., Kelcey, M., Kalakrishnan, M., Downs, L., Ibarz, J., Pastor, P., Konolige, K., Levine, S., and Vanhoucke, V. (2017a). Using simulation and domain adaptation to improve efficiency of deep robotic grasping. *arXiv:1709.07857*.
- Bousmalis, K., Silberman, N., Dohan, D., Erhan, D., and Krishnan, D. (2017b). Unsupervised pixel-level domain adaptation with generative adversarial networks. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*.

- Bradley, D. (2009). *Learning In Modular Systems*. PhD thesis, Carnegie Mellon University.
- Bristow, D. A., Tharayil, M., and Alleyne, A. G. (2006). A survey of iterative learning control. *IEEE Control Systems*, 26(3):96–114.
- Brochu, E., Cora, V. M., and De Freitas, N. (2010). A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning. *arXiv:1012.2599*.
- Byravan, A. and Fox, D. (2017). SE3-nets: Learning rigid body motion using deep neural networks. In *Proceedings of the International Conference on Robotics and Automation*.
- Calandra, R., Seyfarth, A., Peters, J., and Deisenroth, M. (2015). Bayesian optimization for learning gaits under uncertainty. *Annals of Mathematics and Artificial Intelligence*, 76(1):5–23.
- Calinon, S., Alizadeh, T., and Caldwell, D. G. (2013). On improving the extrapolation capability of task-parameterized movement models. In *Proceedings of the International Conference on Intelligent Robots and Systems*.
- Chen, Y. and Medioni, G. (1992). Object modelling by registration of multiple range images. *Image and Vision Computing*, 10(3):145–155.
- Choi, J. and Kim, K.-E. (2013). Bayesian nonparametric feature construction for inverse reinforcement learning. In *Proceedings of the International Joint Conference on Artificial Intelligence*.
- Deisenroth, M. P., Fox, D., and Rasmussen, C. E. (2015). Gaussian processes for data-efficient learning in robotics and control. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(2):408–423.
- Deisenroth, M. P., Neumann, G., and Peters, J. (2013). A Survey on Policy Search for Robotics. *Foundations and Trends in Robotics*, 2:1–142.
- Doerr, A., Ratliff, N., Bohg, J., Toussaint, M., and Schaal, S. (2015). Direct Loss Minimization Inverse Optimal Control. In *Proceedings of Robotics: Science and Systems*.



- Driess, D., Englert, P., and Toussaint, M. (2017). Constrained bayesian optimization of combined interaction force/task space controllers for manipulations. In *Proceedings of the International Conference on Robotics and Automation*.
- Eitel, A., Springenberg, J. T., Spinello, L., Riedmiller, M., and Burgard, W. (2015). Multimodal deep learning for robust rgb-d object recognition. In *Proceedings of the International Conference on Intelligent Robots and Systems*.
- Englert, P. and Toussaint, M. (2016). Combined Optimization and Reinforcement Learning for Manipulations Skills. In *Proceedings of Robotics: Science and Systems*.
- Englert, P. and Toussaint, M. (2018a). Kinematic morphing networks for manipulation skill transfer. *Accepted at the International Conference on Intelligent Robots and Systems*.
- Englert, P. and Toussaint, M. (2018b). Learning manipulation skills from a single demonstration. *International Journal of Robotics Research*, 37(1):137–154.
- Englert, P., Vien, N. A., and Toussaint, M. (2017). Inverse kkt — learning cost functions of manipulation tasks from demonstrations. *International Journal of Robotics Research*, 36(13-14):1474–1488.
- Finn, C., Levine, S., and Abbeel, P. (2016). Guided cost learning: Deep inverse optimal control via policy optimization. In *Proceedings of the International Conference on Machine Learning*.
- Ganin, Y., Ustinova, E., Ajakan, H., Germain, P., Larochelle, H., Laviolette, F., Marchand, M., and Lempitsky, V. (2016). Domain-adversarial training of neural networks. *The Journal of Machine Learning Research*, 17(1):2096–2030.
- Garcia Polo, F. J. and Fernandez-Rebollo, F. (2011). Safe reinforcement learning in high-risk tasks through policy improvement. In *Symposium on Adaptive Dynamic Programming And Reinforcement Learning*.
- Gardner, J., Kusner, M., Xu, Z., Weinberger, K., and Cunningham, J. (2014). Bayesian Optimization with Inequality Constraints. In *Proceedings of the International Conference on Machine Learning*.

- Gelbart, M. A., Snoek, J., and Adams, R. P. (2014). Bayesian Optimization with Unknown Constraints. In *Uncertainty in Artificial Intelligence*.
- Gold, S., Rangarajan, A., Lu, C.-P., Pappu, S., and Mjolsness, E. (1998). New algorithms for 2d and 3d point matching: pose estimation and correspondence. *Pattern Recognition*, 31(8):1019 – 1031.
- Gönen, M. and Alpaydm, E. (2011). Multiple kernel learning algorithms. *The Journal of Machine Learning Research*, 12:2211–2268.
- Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press.
- Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In *Advances in Neural Information Processing Systems*.
- Gramacy, R. B. and Lee, H. K. H. (2011). Optimization under unknown constraints. In *Bayesian Statistics 9*. Oxford University Press.
- Griffiths, T. L. and Ghahramani, Z. (2011). The indian buffet process: An introduction and review. *Journal of Machine Learning Research*, 12(Apr):1185–1224.
- Grubb, A. and Bagnell, J. A. (2010). Boosted backpropagation learning for training deep modular networks. In *International Conference on Machine Learning*.
- Hansen, N. and Ostermeier, A. (2001). Completely Derandomized Self-Adaptation in Evolution. Strategies. *Evolutionary Computation*, 9(2):159–195.
- International Federation of Robotics (2017a). Executive summary world robotics 2017 industrial robots.
- International Federation of Robotics (2017b). Executive summary world robotics 2017 service robotics.
- Jaderberg, M., Simonyan, K., Zisserman, A., et al. (2015). Spatial transformer networks. In *Advances in Neural Information Processing Systems*.
- Jetchev, N. and Toussaint, M. (2014). TRIC: Task space retrieval using inverse optimal control. *Autonomous Robots*, 37(2):169–189.

- Kalakrishnan, M., Pastor, P., Righetti, L., and Schaal, S. (2013). Learning Objective Functions for Manipulation. In *Proceedings of the International Conference on Robotics and Automation*.
- Kalakrishnan, M., Righetti, L., Pastor, P., and Schaal, S. (2011). Learning force control policies for compliant manipulation. In *Proceedings of the International Conference on Intelligent Robots and Systems*.
- Kalman, R. E. (1964). When is a linear control system optimal? *Journal of Basic Engineering*, 86(1):51–60.
- Karush, W. (1939). Minima of functions of several variables with inequalities as side constraints. *M. Sc. Dissertation. Dept. of Mathematics, Univ. of Chicago*.
- Kingma, D. and Ba, J. (2014). Adam: A method for stochastic optimization. In *Proceedings of the International Conference on Learning Representations*.
- Kober, J. and Peters, J. (2008). Policy Search for Motor Primitives in Robotics. In *Advances in Neural Information Processing Systems*.
- Kuhn, H. and Tucker, A. (1951). Nonlinear programming. In *Second Berkeley Symposium on Mathematical Statistics and Probability*.
- Kupcsik, A. G., Deisenroth, M. P., Peters, J., and Neumann, G. (2013). Data-Efficient Generalization of Robot Skills with Contextual Policy Search. In *Proceedings of the National Conference on Artificial Intelligence*.
- Kushner, H. J. (1964). A New Method of Locating the Maximum Point of an Arbitrary Multipeak Curve in the Presence of Noise. *Journal of Fluids Engineering*, 86(1):97–106.
- Le Cun, Y., Jackel, L., Boser, B., Denker, J., Graf, H., Guyon, I., Henderson, D., Howard, R., and Hubbard, W. (1989). Handwritten digit recognition: Applications of neural network chips and automatic learning. *IEEE Communications Magazine*, 27(11):41–46.
- Levine, S. and Koltun, V. (2012). Continuous inverse Optimal Control with Locally Optimal Examples. In *Proceedings of the International Conference on Machine Learning*.

- Levine, S., Pastor, P., Krizhevsky, A., and Quillen, D. (2016). Learning Hand-Eye Coordination for Robotic Grasping with Deep Learning and Large-Scale Data Collection. In *International Symposium on Experimental Robotics*.
- Levine, S., Popovic, Z., and Koltun, V. (2011). Nonlinear inverse reinforcement learning with gaussian processes. In *Advances in Neural Information Processing Systems*.
- Lizotte, D. J., Wang, T., Bowling, M. H., and Schuurmans, D. (2007). Automatic Gait Optimization with Gaussian Process Regression. In *International Joint Conference on Artificial Intelligence*.
- Marco, A., Hennig, P., Bohg, J., Schaal, S., and Trimpe, S. (2016). Automatic LQR Tuning Based on Gaussian Process Global Optimization. In *Proceedings of the International Conference on Robotics and Automation*.
- Marinho, Z., Dragan, A. D., Byravan, A., Boots, B., Srinivasa, S. S., and Gordon, G. J. (2016). Functional gradient motion planning in reproducing kernel hilbert spaces. *arXiv:1601.03648*.
- Martin-Martin, R. and Brock, O. (2014). Online interactive perception of articulated objects with multi-level recursive estimation based on task-specific priors. In *Proceedings of the International Conference on Intelligent Robots and Systems*.
- McCulloch, W. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 7:115–133.
- Michini, B. and How, J. P. (2012). Bayesian nonparametric inverse reinforcement learning. In *Proceedings of the European Conference on Machine Learning*.
- Mitash, C., Bekris, K., and Boularias, A. (2017). A self-supervised learning system for object detection using physics simulation and multi-view pose estimation. In *Proceedings of the International Conference on Intelligent Robots and Systems*.
- Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In *Proceedings of the International Conference on Machine Learning*.
- Mockus, J., Tiesis, V., and Zilinskas, A. (1978). The application of Bayesian methods for seeking the extremum. *Towards Global Optimization*, 2(117–129).

- Mombaur, K., Truong, A., and Laumond, J.-P. (2010). From human to humanoid locomotion—an inverse optimal control approach. *Autonomous Robots*, 28(3):369–383.
- Muhlig, M., Gienger, M., Steil, J. J., and Goerick, C. (2009). Automatic selection of task spaces for imitation learning. In *Proceedings of the International Conference on Intelligent Robots and Systems*.
- Nair, V. and Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In *Proceedings of the International Conference on Machine Learning*.
- Nehaniv, C. L. and Dautenhahn, K. (2002). The Correspondence Problem. In *Imitation in Animals and Artifacts*.
- Ng, A. Y. and Russell, S. (2000). Algorithms for Inverse Reinforcement Learning. In *Proceedings of the International Conference on Machine Learning*.
- Nickisch, H. and Rasmussen, C. E. (2008). Approximations for Binary Gaussian Process Classification. *Journal of Machine Learning Research*, 9(10):2035–2078.
- Paraschos, A., Daniel, C., Peters, J., and Neumann, G. (2013). Probabilistic Movement Primitives. In *Advances in Neural Information Processing Systems*.
- Pastor, P., Kalakrishnan, M., Chitta, S., Theodorou, E., and Schaal, S. (2011). Skill learning and task outcome prediction for manipulation. In *Proceedings of the International Conference on Robotics and Automation*.
- Peters, J., Mülling, K., and Altun, Y. (2010). Relative Entropy Policy Search. In *Proceedings of the Conference on Artificial Intelligence*.
- Puydupin-Jamin, A.-S., Johnson, M., and Bretl, T. (2012). A Convex Approach to Inverse Optimal Control and its Application to Modeling Human Locomotion. In *Proceedings of the International Conference on Robotics and Automation*.
- Ramachandran, D. and Amir, E. (2007). Bayesian inverse reinforcement learning. In *Proceedings of the International Joint Conference on Artificial Intelligence*.
- Rasmussen, C. E. and Williams, C. K. I. (2006). *Gaussian Processes for Machine Learning*. MIT Press.

- Ratliff, N. D., Bagnell, J. A., and Zinkevich, M. (2006). Maximum margin planning. In *Proceedings of the International Conference on Machine Learning*.
- Ratliff, N. D., Silver, D., and Bagnell, J. A. (2009). Learning to search: Functional gradient techniques for imitation learning. *Autonomous Robots*, 27(1):25–53.
- Rezende, D. J., Eslami, S. A., Mohamed, S., Battaglia, P., Jaderberg, M., and Heess, N. (2016). Unsupervised learning of 3d structure from images. In *Advances in Neural Information Processing Systems*.
- Rosenblatt, F. (1962). *Principles of Neurodynamics*. Spartan Book.
- Rückert, E. A., Neumann, G., Toussaint, M., and Maass, W. (2013). Learned graphical models for probabilistic planning provide a new class of movement primitives. *Frontiers in Computational Neuroscience*, 6(138).
- Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning Internal Representations by Error Propagation. In *Parallel Distributed Processing*, volume 1, pages 318–362. MIT Press.
- Rusu, A. A., Rabinowitz, N. C., Desjardins, G., Soyer, H., Kirkpatrick, J., Kavukcuoglu, K., Pascanu, R., and Hadsell, R. (2016). Progressive neural networks. *arXiv:1606.04671*.
- Rusu, A. A., Vecerik, M., Rothörl, T., Heess, N., Pascanu, R., and Hadsell, R. (2017). Sim-to-real robot learning from pixels with progressive nets. In *Proceedings of the Conference on Robot Learning*.
- Schaal, S., Ijspeert, A., and Billard, A. (2003). Computational Approaches to Motor Learning by Imitation. *Philosophical Transactions of the Royal Society of London*, 358:537–547.
- Schölkopf, B. and Smola, A. J. (2002). *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press.
- Schonlau, M., Welch, W. J., and Jones, D. R. (1998). Global Versus Local Search in Constrained Optimization of Computer Models. *Lecture Notes-Monograph Series*, 34:11–25.

- Schreiter, J., Nguyen-Tuong, D., Eberts, M., Bischoff, B., Markert, H., and Toussaint, M. (2015). Safe Exploration for Active Learning with Gaussian Processes. In *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*.
- Siciliano, B. and Khatib, O., editors (2016). *Springer Handbook of Robotics*. Springer, 2nd edition.
- Srinivas, N., Krause, A., Kakade, S., and Seeger, M. (2012). Information-Theoretic Regret Bounds for Gaussian Process Optimization in the Bandit Setting. *IEEE Transactions on Information Theory*, 58(5):3250–3265.
- Stulp, F., Raiola, G., Hoarau, A., Ivaldi, S., and Sigaud, O. (2013a). Learning compact parameterized skills with a single regression. In *Proceedings of the International Conference on Humanoid Robots*.
- Stulp, F. and Sigaud, O. (2013). Robot Skill Learning: From Reinforcement Learning to Evolution Strategies. *Paladyn, Journal of Behavioral Robotics*, 4(1):49–61.
- Stulp, F., Sigaud, O., and Others (2013b). Policy Improvement Methods: Between Black-box optimization and Episodic Reinforcement Learning. *Journées Franco-phones Planification, Décision, et Apprentissage pour la conduite de systèmes*.
- Sturm, J., Stachniss, C., and Burgard, W. (2011). A Probabilistic Framework for Learning Kinematic Models of Articulated Objects. *Journal of Artificial Intelligence Research*, 41:477–526.
- Sui, Y., Gotovos, A., Burdick, J. W., and Krause, A. (2015). Safe Exploration for Optimization with Gaussian Processes. In *Proceedings of International Conference on Machine Learning*.
- Sutton, R. S. and Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. MIT Press.
- Theodorou, E., Buchli, J., and Schaal, S. (2010). A Generalized Path Integral Control Approach to Reinforcement Learning. *Journal of Machine Learning Research*, 11:3137–3181.

- Todorov, E. and Li, W. (2005). A generalized iterative lqg method for locally-optimal feedback control of constrained nonlinear stochastic systems. In *Proceedings of the American Control Conference*.
- Toussaint, M. (2017). A tutorial on Newton methods for constrained trajectory optimization and relations to SLAM, Gaussian Process smoothing, optimal control, and probabilistic inference. In *Geometric and Numerical Foundations of Movements*. Springer.
- Toussaint, M., Ratliff, N., Bohg, J., Righetti, L., Englert, P., and Schaal, S. (2014). Dual Execution of Optimized Contact Interaction Trajectories. In *Proceedings of the International Conference on Robotics and Automation*.
- Vuga, R., Nemeč, B., and Ude, A. (2015). Enhanced Policy Adaptation Through Directed Explorative Learning. *International Journal of Humanoid Robotics*, 12(3).
- Widrow, B. and Hoff, M. E. (1960). Adaptive switching circuits. Technical report, Stanford.
- Wright, S. J. and Nocedal, J. (1999). *Numerical Optimization*, volume 2. Springer New York.
- Zhifei, S. and Joo, E. M. (2012). A survey of inverse reinforcement learning techniques. *International Journal of Intelligent Computing and Cybernetics*, 5(3):293–311.
- Zhou, X., Wan, Q., Zhang, W., Xue, X., and Wei, Y. (2016). Model-based deep hand pose estimation. In *Proceedings of the International Joint Conference on Artificial Intelligence*.
- Zhou, Y. and Chellappa, R. (1988). Computation of optical flow using a neural network. In *Proceedings of International Conference on Neural Networks*.
- Ziebart, B. D., Maas, A., Bagnell, J. A., and Dey, A. K. (2008). Maximum Entropy Inverse Reinforcement Learning. *Proceedings of the Conference on Artificial Intelligence*.





# Eigenständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Dissertation selbstständig und ausschließlich auf der Grundlage der in der Arbeit angegebenen Hilfsmittel und Hilfen verfasst habe.

Peter Englert

Stuttgart, den 02.07.2018

