Machine Learning for Robot Grasping and Manipulation

Maschinelles Lernen fuer Robotisches Greifen und Manipulation
Zur Erlangung des Grades eines Doktors der Naturwissenschaften (Dr. rer. nat.)
genehmigte Dissertation von M.Eng. Oliver Kroemer aus Celle
2015 — Darmstadt — D 17

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Department of Computer Science
Intelligent Autonomous Systems

Machine Learning for Robot Grasping and Manipulation
Maschinelles Lernen fuer Robotisches Greifen und Manipulation

Genehmigte Dissertation von M.Eng. Oliver Kroemer aus Celle

1. Gutachten: Prof. Dr. Jan Peters
2. Gutachten: Prof. Dr. Oliver Brock

Tag der Einreichung: 17 Oct 2014
Tag der Prüfung: 16 Dec 2014

Darmstadt — D 17

Abstract

Robotics as a technology has an incredible potential for improving our everyday lives. Robots could perform household chores, such as cleaning, cooking, and gardening, in order to give us more time for other pursuits. Robots could also be used to perform tasks in hazardous environments, such as turning off a valve in an emergency or safely sorting our more dangerous trash. However, all of these applications would require the robot to perform manipulation tasks with various objects. Today's robots are used primarily for performing specialized tasks in controlled scenarios, such as manufacturing. The robots that are used in today's applications are typically designed for a single purpose and they have been preprogrammed with all of the necessary task information. In contrast, a robot working in a more general environment will often be confronted with new objects and scenarios. Therefore, in order to reach their full potential as autonomous physical agents, robots must be capable of learning versatile manipulation skills for different objects and situations. Hence, we have worked on a variety of manipulation skills to improve those capabilities of robots, and the results have lead to several new approaches, which are presented in this thesis

Learning manipulation skills is, however, an open problem with many challenges that still need to be overcome. The first challenge is to acquire and improve manipulation skills with little to no human supervision. Rather than being preprogrammed, the robot should be able to learn from human demonstrations and through physical interactions with objects. Learning to improve skills through trial and error learning is a particularly important ability for an autonomous robot, as it allows the robot to handle new situations. This ability also removes the burden from the human demonstrator to teach a skill perfectly, as a robot is allowed to make mistakes if it can learn from them. In order to address this challenge, we present a continuum-armed bandits approach for learning to grasp objects. The robot learns to predict the performances of different grasps, as well as how certain it is of this prediction, and selects grasps accordingly. As the robot tries more grasps, its predictions become more accurate, and its grasps improve accordingly.

A robot can master a manipulation skill by learning from different objects in various scenarios. Another fundamental challenge is therefore to efficiently generalize manipulations between different scenarios. Rather than relearning from scratch, the robot should find similarities between the current situation and previous scenarios in order to reuse manipulation skills and task information. For example, the robot can learn to adapt manipulation skills to new objects by finding similarities between them and known objects. However, only some similarities between objects will be relevant for a given manipulation. The robot must therefore also learn which similarities are important for adapting the manipulation skill. We present two object representations for generalizing between different situations. Contacts between objects are important for many manipulations, but it is difficult to define general features for representing sets of contacts. Instead, we define a kernel function for comparing contact distributions, which allows the robot to use kernel methods for learning manipulations. The second approach is to use warped parameters to define more abstract features, such as areas and volumes. These features are defined as functions of known object models. The robot can compute these parameters for novel objects by warping the shape of the known object to match the unknown object.

Learning about objects also requires the robot to reconcile information from multiple sensor modalities, including touch, hearing, and vision. While some object properties will only be observed by specific sensor modalities, other object properties can be determined from multiple sensor modalities. For example, while color can only be determined by vision, the shape of an object can be observed using vision or touch. The robot should use information from all of its senses in order to quickly learn about objects. We explain how the robot can learn low-dimensional representations of tactile data by incorporating cues

from vision data. As touching an object usually occludes the surface, the proposed method was designed to work with weak pairings between the data in the two sensor modalities.

The robot can also learn more efficiently if it reuses skills between different tasks. Rather than relearn a skill for each new task, the robot should learn manipulation skills that can be reused for multiple tasks. For an autonomous robot, this would require the robot to divide tasks into smaller steps. Dividing tasks into smaller parts makes it easier to learn the corresponding skills. If a step is a part of many tasks, then the robot will have more opportunities to practice the associated skill, and more tasks will benefit from the resulting performance improvement. In order to learn a set of useful subtasks, we propose a probabilistic model for dividing manipulations into phases. This model captures the conditions for transitioning between different phases, which represent subgoals and constraints of the overall tasks. The robot can use the model together with model-based reinforcement learning in order to learn skills for moving between phases.

When confronted with a new task, the robot will have to select a suitable sequence of skills to execute. The robot must therefore also learn to select which manipulation to execute in the current scenario. Selecting sequences of motor primitives is difficult, as the robot must take into consideration the current task, state, and future actions when selecting the next motor skill to execute. We therefore present a value function method for selecting skills in an optimal manner. The robot learns the value function for the continuous state space using a flexible non-parametric model-based approach.

Learning manipulation skills also poses certain challenges for learning methods. The robot will not have thousands of samples when learning a new manipulation skill, and must instead actively collect new samples or use data from similar scenarios. The learning methods presented in this thesis are, therefore, designed to work with relatively small amounts of data, and can generally be used during the learning process. Manipulation tasks also present a spectrum of different problem types. Hence, we present supervised, unsupervised, and reinforcement learning approaches in order to address the diverse challenges of learning manipulations skills.

Zusammenfassung

Die Robotik hat als Technologie ein unglaubliches Potenzial für die Verbesserung unseres Alltags. Roboter könnten Hausarbeiten, wie Putzen, Kochen und Gartenarbeiten durchführen, um uns mehr Zeit für andere Aktivitäten zu geben. Roboter könnten auch Aufgaben in gefährlichen Umgebungen übernehmen, wie z.B. das Ausschalten eines Ventils in einem Notfall oder das Sortieren von gefährlichen Abfällen. Alle diese Anwendungsgebiete fordern von dem Roboter jedoch das Manipulieren von verschiedenen Objekten. Heutzutage werden Roboter primär für spezielle Aufgaben in kontrollierten Szenarien, beispielsweise der Produktion verwendet. Die Roboter, die in heutigen Anwendungen verwendet werden, werden in der Regel für einen bestimmten Zweck entworfen und sind mit allen notwendigen Informationen zu der Aufgabe vorprogrammiert. Im Gegensatz dazu wird ein Roboter in einer allgemeineren Umgebung oft mit neuen Objekten und Szenarien konfrontiert. Um ihr volles Potenzial zu erreichen, müssen Roboter in der Lage sein eine Vielzahl von unterschiedlichen Objekten in unterschiedlichen Situationen manipulieren zu können. Basierend auf einer Vielzahl von Manipulationsaufgaben, haben wir daran gearbeitet diese Fähigkeiten von Robotern zu verbessern. Die Ergebnisse davon haben zu einigen neuen Ansätzen geführt, die wir in dieser Arbeit vorstellen.

Manipulationsfähigkeiten zu lernen ist jedoch ein offenes Problem mit vielen Herausforderungen, welche noch zu überwinden sind. Die erste Herausforderung ist das Lernen oder Verbessern der Manipulationsfähigkeiten mit wenig bis keiner menschlichen Unterstützung. Anstatt das der Roboter vorprogrammiert wird soll dieser in der Lage sein von menschlichen Demonstrationen oder durch physische Interaktion mit den Objekten zu lernen. Zu lernen wie Fähigkeiten durch Ausprobieren verbessert werden können ist eine besonders wichtige Voraussetzung für einen autonomen Roboter, da es ihm erlaubt neue Situationen zu bewältigen. Durch diese Fähigkeit muss die Demonstration nicht perfekt sein, da der Roboter Fehler machen darf, wenn er von diesen lernen kann. Um dieser Herausforderung zu begegnen, stellen wir ein continuum-armed bandits Ansatz vor um das Greifen von Objekten zu lernen. Der Roboter lernt vorherzusagen wie gut verschiedene Griffe sind, als auch wie sicher er über die Vorhersage ist. Basierend darauf wählt er einen Griff aus. Durch Ausprobieren mehrerer Griffe werden die Vorhersagen genauer und somit die Griffe besser.

Ein Roboter kann eine Manipulationsfähigkeit meistern indem er von unterschiedlichen Objekten in unterschiedlichen Szenarien lernt. Eine weitere zentrale Herausforderung ist daher das effiziente Generalisieren von Manipulationen zwischen verschiedenen Szenarien. Anstatt immer wieder von Grund auf zu lernen sollte der Roboter Ähnlichkeiten zwischen der momentanen Situation und vorherigen Szenarien erkennen, um Manipulationsfähigkeiten und Aufgabeninformationen wiederzuverwenden. Beispielsweise kann der Roboter lernen eine Manipulationsfähigkeit an neue Objekte anzupassen, indem er Ähnlichkeiten zwischen diesen und bekannten Objekten findet. Allerdings können nur einige Ähnlichkeiten zwischen Objekten für eine bestimmte Manipulation relevant sein. Der Roboter muss daher auch lernen welche Ähnlichkeiten wichtig sind, um die Manipulationsfähigkeit anzupassen. Wir präsentieren zwei Objektdarstellungen zur Generalisierung zwischen verschiedenen Situationen. Kontakte zwischen Objekten sind wichtig für viele Manipulationen, aber es ist schwierig generelle Eigenschaften für die Repräsentation von Kontakten zu definieren. Stattdessen definieren wie eine Kernelfunktion zum Vergleichen von Kontaktverteilungen, welche es dem Roboter erlaubt Kernelmethoden zum Lernen der Manipulationen zu verwenden. Der zweite Ansatz ist warped Parameter zu verwenden, um abstraktere Eigenschaften, wie Flächen und Volumen, zu definieren. Diese Eigenschaften sind definiert als Funktionen von bekannten Objektmodellen. Der Roboter kann diese Parameter für neue Objekte berechnen indem die Form des bekannten Objekts so verzerrt wird, dass es mit dem unbekannte Objekt übereinstimmt.

Um Objekteigenschaften zulernen muss der Roboter Informationen von mehreren Sensoren abgleichen, einschließlich Tasten, Hören und Sehen. Während einige Objekteigenschaften nur von bestimmten Sensoren beobachtet werden, können andere Objekteigenschaften von unterschiedlichen Sensoren bestimmt werden. Während Farbe zum Beispiel nur optisch bestimmt werden kann, kann die Form sowohl optisch als auch durch Berührung bestimmt werden. Der Roboter sollte die Informationen von allen Sinnen verwenden, um schnell über Objekte zu lernen. Wir erklären, wie der Roboter niedrig dimensionale Darstellungen von taktilen Daten lernen kann mit der Hilfe von visuellen Daten. Das Berühren eines Objekts verdeckt normalerweise die Oberfläche. Deswegen benötigt die vorgeschlagene Methode nur schwache Paarungen zwischen den visuellen und taktilen Daten.

Der Roboter kann auch effizienter lernen, wenn er Fähigkeiten zwischen verschiedenen Aufgaben wiederverwendet. Anstatt eine Fähigkeit für jede Aufgabe wieder zu lernen, sollte der Roboter Fähigkeiten lernen die er für mehrere Aufgaben verwenden kann. Ein autonomer Roboter sollte die Aufgabe in mehrere Unteraufgaben teilen. Die Aufteilung in Unteraufgaben macht es einfacher die entsprechende Fähigkeiten zu erlernen. Wenn eine Unteraufgabe Teil von vielen Aufgaben ist, hat der Roboter mehr Möglichkeiten die zugehörige Fähigkeit zu üben. Dadurch profitieren mehr Aufgaben von der Leistungsverbesserung. Um eine Reihe von nützlichen Unteraufgaben zu lernen, schlagen wir ein probabilistisches Modell vor zur Unterteilung der Manipulation in Phasen. Dieses Modell erfasst die Bedingungen für den Übergang verschiedener Phasen, die die Teilziele der Aufgaben darstellen. Der Roboter kann das Modell zusammen mit modellbasierten Reinforcement Learning verwenden, um Fähigkeiten für den Übergang zwischen den Phasen zu lernen.

Wird dem Roboter eine neue Aufgabe gestellt, muss er eine geeignete Folge von Fähigkeiten auswählen um die Aufgabe durchzuführen. Der Roboter muss daher auch lernen welche Manipulation im momentanen Szenario am Besten ausgeführt werden sollte. Der Roboter muss dabei den momentanen Zustand und die Aufgabe berücksichtigen, sowie die zukünftigen Aktionen. Um die Beste Fähigkeit auszuwählen nutzen wir eine Wertfunktion Methode. Der Roboter lernt die Wertfunktion für den kontinuierlichen Zustandsraum mit einem flexiblen nicht-parametrischen modellbasierten Ansatz.

Das Lernen von Manipulationsfähigkeiten birgt auch gewisse Herausforderungen für die Lernmethoden. Der Roboter hat nicht Tausende von Beispielen beim Erlernen einer neuen Manipulationsfähigkeit, sondern muss aktiv neue sammeln oder Beispiele von ähnlichen Szenarien benutzen. Die Lernmethoden in dieser Arbeit können daher mit relativ geringen Mengen an Daten arbeiten, und können während des Lernprozesses verwendet werden. Manipulationsaufgaben präsentieren deswegen auch ein Spektrum verschiedener Problemtypen. Wir präsentieren supervised, unsupervised und reinforcement learning Ansätze, um die vielfältigen Herausforderungen anzugehen.

Acknowledgements

I have received an incredible amount of support during my time as a Ph.D. student, and had the pleasure to work with many talented researchers. I would therefore like to thank:

- Jan Peters for all of the support and supervision that he has given me over the years. Thanks to his guidance and his belief in me, I have found my voice as a researcher.

- my thesis referees, Prof. Dr. Jan Peters and Prof. Dr. Oliver Brock for evaluating this thesis.

- Prof. Dr. Stefan Roth for heading the thesis committee, and Prof. Dr. Oskar von Stryk, Prof. Dr. Johannes Fürnkranz, and Prof. Dr Gerhard Neumann for participating in the defence.

- a special thanks goes to Justus Piater and Christoph Lampert for all of their help and contributions to this thesis.

- Erhan Oztop and Emre Ugur for hosting me at ATR

- my former colleagues at the Max Planck Institute for Intelligent Systems for the lively discussions. In particular, I would like to thank Katherina Muelling, Jens Kober, Duy Nguyen-Tuong, and Abdeslam Boularias for their support and making my time in Tuebingen memorable.

- my colleageues at the TU Darmstadt for providing a wonderful environment for both teaching and research. It has been an absolute pleasure working with them. In particular, I would like to thank my officemates Herke van Hoof and Christian Daniel for their support and good humor, as well as Alexandros Paraschos for helping me with the real-time kernel.

- I would like to thank all of my co-authors for the many wonderful collaborations over the years, and in particular, Abdeslam Boularias, Ayse Naz Erkan, Christian Daniel, Christoph Lampert, Duy Nguyen-Tuong, Emre Ugur, Erhan Oztop, Gerhard Neumann, Guilherme Maeda, Heni Ben Amor, Herke van Hoof, Jens Kober, Katherina Muelling, Marco Ewerton, Renaud Detry, Rudolf Lioutikov, Sascha Brandl, Ullrich Hillenbrand, Yasemin Altun, and Yevgen Chebotar.

- my parents, my brother, and Sandra for their love and support, and my parents for inspiring me to pursue a Ph.D. in robot learning.

- Finally, I would like to thank my robots Darias and RDaneel for their hard work, without which this thesis would not have been possible. It has been a pleasure and an inspiration watching them learn.

Contents

1 Introduction

There are countless applications for robots that can manipulate a wide range of different objects in everyday environments. Such robots could be used to perform household chores and care for the elderly. Their versatility and adaptability would allow them to provide assistance in emergency situations, e.g., by clearing debris. These advanced robots could also be used in service industries such as serving food and cleaning up litter. However, there are still many open problems that need to be solved before such robots can be realized. These robots will be confronted with new objects and tasks, and they will need to be capable of adapting to new scenarios. Hence, the necessary skill knowledge cannot be preprogrammed into these robots. Instead, they will need to learn the skills for manipulating objects autonomously.

In this thesis, we explore different learning approaches for robot grasping and manipulation. There are many challenges that a robot will need to overcome in order to learn a versatile set of manipulation skills. In order to illustrate some of these challenges, we can consider a bartender robot as an example application for a future service robot.

To begin its new job, the robot will first be shown a few of the tasks that it will need to perform with a couple of objects. For example, a human may show how to prepare and serve a beverage. This task however consists of multiple subtasks, such as placing a glass on the counter, grasping a bottle, and sliding the glass across the counter. The first challenge is therefore to decompose the demonstrations into smaller subtasks. It is not only easier to learn skills for smaller tasks, but it is also more straightforward to reuse skills between tasks. Taking inspiration from human manipulation skills [1], we propose a probabilistic model for segmenting tasks into phases. The model captures how the robot's actions change the state of the object in each phase, as well as the conditions needed to transition to different phases. Using this model, the robot can determine a set of subtasks and learn manipulation skills for each of them.

The next challenge is to learn the individual manipulation skills. Grasping is one of the most fundamental skills for manipulating objects, and one of the first skills the robot will need to master. Instead of relying on additional human assistance, the robot will have to learn grasping by interacting with the various glasses, bottles, and other assorted objects in its environment. The robot should not execute the same grasp every time, but rather explore new grasps in order to improve its grasping abilities. However, the robot should also not try completely random actions either. We therefore frame grasping as a continuum-armed bandits problem in order to find a balance between exploring new grasps and exploiting known grasps. We present a policy for actively selecting new grasps based on the outcomes of previous grasp attempts.

The robot can learn skills more efficiently by generalizing between different objects. Many tasks, such as operating a bar tap or stacking objects on a tray, are based on physical contacts between different objects. A contact on the backside of a bar tap allows the robot to pull it, and a region of contact under the base of a glass provides stability. Similar contacts on other objects will allow for similar interactions. In order to learn which contacts are needed for a particular interaction, the robot will need to be able to represent contacts between objects. Instead of defining various features, we propose a kernel that allows the robot to directly compute the similarity between different contacts. This kernel allows the robot to apply kernel methods for classifying and clustering interactions between objects.

Pouring is another important ability for a bartender robot, but it is not based on direct physical contact. The important features for pouring are related to more abstract parameters of the objects, such as the volume of the container and the size of its opening. The robot will therefore need to estimate these parameters for new containers in order to generalize its pouring skills. We therefore propose warped parameters for computing geometric features of objects. Warped parameters are defined as functions on the point cloud of a known object. Applying transformations to the point cloud, e.g., scaling, changes the value of the parameter. The parameter can thus be computed for new objects using a warping process.

Once the robot has learned various manipulation skills, the next challenge is to combine them in order to perform different tasks. For example, the robot may have been shown how to prepare a single drink as part of its initial training. However, the robot should not simply execute the same sequence multiple times when asked for multiple drinks. Instead, the drinks should be prepared in parallel. In order to select skills in an optimal manner, we present a non-parametric method for learning value functions in continuous state spaces.

Overtime, objects in the bar will experience natural wear and tear from being used. The bartender robot should therefore be able to detect if a glass is chipped or if the counter is dirty or damaged. Such defects can sometimes be observed by visually inspecting the relevant surfaces. However, these changes in the surface texture can often be detected more reliably using the sense of touch. In order to accurately classify different surfaces, we propose a method for reducing the dimensionality of tactile data by learning from visual information when it is available. Once the robot has learned a suitable representation for the tactile data, the vision information is no longer needed.

As the example of the bartender robot demonstrates, there are many different challenges that a service robot would need to overcome in order to learn versatile manipulation skills. We therefore propose a variety of learning methods for addressing these diverse problems. The approaches presented in this thesis include supervised, unsupervised, and reinforcement learning methods. The bartender application also illustrates how learning manipulation skills is a process. The robot will need to time to build up its experiences to master different manipulation skills. In order to help the robot during the learning process, we focus on methods that can be applied to relatively small amounts of data.

1.1 Contributions

This thesis presents several methods that have been specifically designed for robot manipulation tasks. The methods have been applied to learning manipulation tasks, including pouring, stacking, and grasping. In this section, we outline the key contributions of this thesis to the state-of-the-art in robot learning for manipulation.

Learning to Grasp using Reinforcement Learning

We proposed the continuum Gaussian bandits (CGB) policy for learning to grasp. While most grasp learning methods focus on supervised learning approaches [2], we frame grasping as a continuum-armed bandits problem and use a reinforcement learning approach. In this manner, the robot actively selects grasps to evaluate and uses the outcomes of these grasps to select better grasps in the future. The robot can thus autonomously improve its grasping abilities over time, and optimize grasps for specific objects. The approach was inspired by the upper confidence bound (UCB) policies used for multi-armed bandits problems [3]. The resulting algorithm is closely related to the UCB policies used in Bayesian optimization [4]. However, rather than performing a global search, the proposed method performs a sample-efficient local search for optimal grasps. This approach is well-suited for learning grasps without relying on explicit information regarding the object's shape and size. By applying this approach, the robot was able to learn dexterous grasps that would have been difficult to program by hand.

Contact-Distribution Kernels and Warped Parameters

Contacts between objects are fundamental to many manipulations. A common problem is therefore to determine whether a set of contacts allow for the desired manipulation or interaction between objects. However, defining general features to represent sets of contacts is not trivial. We therefore proposed modeling the contact distributions using Gaussians and computing a kernel between these distributions. By defining a kernel function, the robot can use a wide range of kernel methods. Using these kernels, the robot learned to stack assorted blocks into towers and segment grasping actions into phases.

In order to define more abstract features for objects, we also present warped parameters. These geometric parameters are defined as functions on the point clouds of known objects, and can be computed for novel objects using a warping process. In the context of robot manipulation, warping has previously been used to determine specific points in a scene, e.g. a contact point [5] or a via point [6]. In contrast, warped parameters can represent more abstract parameters, such as the length of a blade or the volume of a container. The robot could learn pouring skills, that generalize between different objects, by using the warped parameters.

A Probabilistic Model for Decomposing Manipulations into Phases

Humans seem to separate manipulation skills into distinct phases, with phase transitions often corresponding to the making and breaking of contacts [7, 1]. Such a change in contact state usually results in the agent's actions having different effects on its environment. In order to segment manipulation tasks, we present the state-based transitions auto-regressive hidden Markov model (STARHMM) for representing phases. Rather than segmenting manipulations based on the actions [8, 9, 10], the proposed model splits the task into parts based on the effects of the actions. Each phase is represented as a linear system model. The proposed model also learns the conditions in which a phase transition is more likely to occur. These conditions are then used to define individualized reward functions for learning motor primitives for transitioning between different phases.

A Non-Parametric Approach to Learning Value Functions

Value functions are useful for selecting motor primitives, as they reduce the problem of selecting sequences of actions to greedily selecting the next action. However, the exact computation of the value function remains an open problem for most systems with continuous state spaces [11]. We therefore present non-parametric dynamic programming (NPDP) for computing value functions in a flexible manner. The NPDP algorithm models the system dynamics using a kernel density estimate, and we show that the value function for this model has a Nadaraya-Watson kernel regression form. We also show how different modeling assumptions give rise to other common methods for learning value functions, such as *least-squared temporal difference learning* [12] and *kernelized temporal difference learning* [13]. The NPDP approach was used to learn high-level policies for sequencing motor primitives for performing manipulation tasks.

Using Vision Data to Learn Low-Dimensional Representations of Tactile Data

Dynamic tactile sensing plays an important part in manipulation tasks, as it is used both for determining object properties as well as detecting key events during manipulation tasks [14, 15]. Data from dynamic tactile sensing is however noisy, high-dimensional, and often includes vibrations from other sources. We therefore present weakly-paired maximum covariance analysis (WMCA) and mean maximum covariance analysis (μMCA) for reducing the dimensionality of the tactile data. These methods allow the robot to incorporate vision information when learning a lower-dimensional representation of the tactile data. As images of the surfaces capture the texture information but not the other vibrations, the vision data helps the robot to extract the relevant components of the tactile data. Unlike sensor fusion approaches [16, 17, 18], the vision information is only needed when learning the dimensionality reduction. Afterwards, the dimensionality reduction can be applied even when only tactile data is available.

	Chapter 2	Chapter 3	Chapter 4	Chapter 5	Chapter 6
Single Manipulations	●	●	○	○	○
Manipulation Sequences	○	○	●	●	○
Learning Object Properties	○	●	○	○	●
Reinforcement Learning	●	○	○	●	○
Manipulation Phases	○	●	●	○	○
Biological Inspiration	●	○	●	○	●

Figure 1.1: An overview of some of the overarching themes presented in this thesis, and the chapters that they relate to.

1.2 Outline

The chapters of this thesis were written such that they can be read independently. We recommend readers that are unfamiliar with dynamic motor primitives (DMPs) to read Chapter 2 first, as DMPs are used throughout the majority of this thesis to represent manipulation skills. Chapters 2 and 3 present methods for learning individual manipulation skills. Chapters 4 and 5 describe methods for learning sequences of motor primitives. In Chapter 6, we discuss methods for performing dimensionality reduction on tactile data by exploiting information from vision data. In Chapter 7, we summarize the main contributions of the thesis and discuss some of the open challenges for learning grasping and manipulation skills. Figure 1.1 presents an overview of some of the topics covered in the thesis and their corresponding chapters.

Chapter 2 presents the Continuum Gaussian Bandits policy for learning how to grasp objects. The chapter also gives an overview of dynamic motor primitives, and how they can be used for grasping. This chapter is based on [19].

Chapter 3 describes a kernel between contact distributions, which generalizes between different objects. The kernel is evaluated on a block stacking task with assorted toy blocks. This chapter also discusses warped parameters, and how they can be used to compute geometric parameters of containers for pouring tasks. This chapter is based on [20, 21].

Chapter 4 explains a modified autoregressive hidden Markov model for segmenting manipulation into smaller subtasks based on phases. The model forms the bases for learning libraries of motor primitives for transitioning between different phases. The method was tested on a bimanual grasping task. This chapter is based on [22, 23].

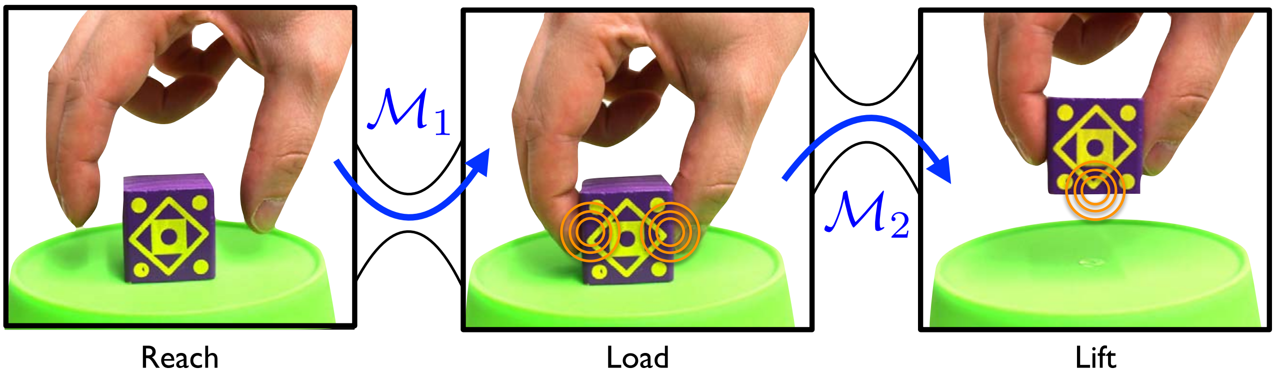


Figure 1.2: The figure illustrates the three phases of a basic grasping task: reach, load, and lift. The orange circles indicate the locations where contacts are made or broken during the phase transition. These mechanical events represent subgoals of the overall task. The changes in contact result in small vibrations. The blue arrows indicate motor primitives $\mathscr{M}$ used to transition from one phase to another.

Chapter 5 proposes a non-parametric model-based approach to learning value functions, and explains how the value function can be used to learn a high-level controller for selecting motor primitives. This approach was used to sequence the motor primitive library learned in Chapter 4. This chapter is based on [24].

Chapter 6 presents methods for learning low-dimensional representations of dynamic tactile data using vision information. The proposed method is evaluated on a material classification task. This chapter is based on [25].

Chapter 7 presents the main conclusions of the thesis. It also presents key ideas for developing and selecting methods for learning manipulation skills. The chapter ends with a discussion on open problems for grasping and manipulation.

Chapters 2 to 6 each end with a section on "Potentially Helpful Insights". These sections are meant to give additional insights into the work for future Ph.D. students. We explain the motivations for pursuing the individual projects and try to provide deeper insights into the methods. We also discuss connections to other approaches, and give ideas for how insights from these projects could be extended into new research directions.

Although the chapters investigate different aspects of manipulation tasks, there are two overarching topics that span this thesis. The first topic is phases, and deals with the general structure of manipulations. The second topic is representations, which deals with defining manipulations in a useful way for learning.

Manipulation Phases

The first overarching topic in this thesis is phases. Phases are a way of dividing manipulation tasks into smaller parts, such that the transitions between phases correspond to mechanical events, e.g., making or breaking contacts between objects [7, 1]. These events represent subgoals of the overall task. As an example, we can consider learning a basic grasp from a continuous demonstration [7]. The three phases of the grasping task, i.e. reach, load, and lift, are illustrated in Fig. 1.2.

The first challenge is to divide the demonstrated grasping action into phases, and learn the conditions needed for transitioning between them. We present a modified autoregressive hidden Markov model for exactly this purpose in Chapter 4. Dividing the basic grasping task into phases results in two subtasks: transitioning from the reach phase to the load phase, and transitioning from the load phase to the lift phase.

The next step is to learn a motor primitive for each of these subtasks. For some tasks, the robot can use the model to learn the motor primitive, as described at the end of Chapter 4. However, the robot can also learn the motor primitive by attempting to grasp the actual object. In Chapter 2, we present a reinforcement learning method for learning to grasp. This method implicitly learns how to approach an object in order to transition to the load phase. By physically interacting with an object, the robot can gather more information about which conditions result in the desired phase transitions.

The transitions between phases are often characterized by mechanical events, which the robot can sense. For example, making and breaking contacts results in vibrations that can be detected through dynamic tactile sensing. In Chapter 6, we discuss methods for removing irrelevant components of dynamic tactile sensing data by learning with visual cues. This approach was evaluated on a texture classification task.

In order to grasp different objects, the robot will need to generalize the phase transition conditions between objects. For the grasping task, these conditions are linked to the contacts between the hand, the object, and the supporting surface. Rather than designing a set of features for representing these contacts, the robot can use a kernel function for directly computing the similarity between sets of contacts, as explained in Chapter 3. The robot can then learn to predict phases using methods such as kernel logistic regression. Not all phase transitions correspond to making and breaking contacts though. For example, a phase transition occurs when a container starts pouring. This phase transition depends on the volume of the container and how much liquid it is holding. These kinds of features can be computed using warped parameters, which are also explained in Chapter 3.

The robot has now decomposed the task into phases and learned a motor primitive for each phase transition. The next step is to recombine these motor primitives to achieve the task. For the grasping example, there is only one sequence of valid motor primitives. Let us instead consider the situation where the robot has another motor primitive for sliding the object to the right in the load phase. If the task gives the robot a reward for the height of the object, then the robot should quickly transition to the lift phase. If the robot gets an additional reward for the horizontal position of the object, then it may transition to the load phase, execute the sliding motor primitive, and then transition to the lift phase. In Chapter 5, we present a dynamic programming method for learning motor primitive sequences.

As this example demonstrates, even basic manipulations can often be decomposed into phases. We provide methods that allow the robot to learn motor primitives for transitioning between phases, and for generalizing phase transition conditions between objects. We also explain how a demonstration can be divided into phases and the resulting motor primitives then resequenced to perform other tasks.

Representations for Learning Manipulations

The second overarching topic in this thesis is representations. A robot can only learn a skill if it can represent it in a meaningful way. One of the key challenges of learning manipulation skills is the need to represent the various different components of manipulation tasks. The robot will needs to be capable of representing actions, objects, and tasks.

In order to represent actions, we use dynamic motor primitives (DMPs). In Chapter 2, we describe how reaching and grasping movements can be represented using DMPs. In Chapter 3, we also explain how motor primitives can be scaled to account for differences in object size. When selecting an action to execute, the robot must also take into consideration the future actions in the sequence. The influence of these future actions can be concisely represented using a value function, such as the one described in Chapter 5.

For generalizing manipulations, the robot needs to represent object properties such as the shape and material. Geometric parameters of objects, e.g. lengths and volumes, can be grounded in point clouds

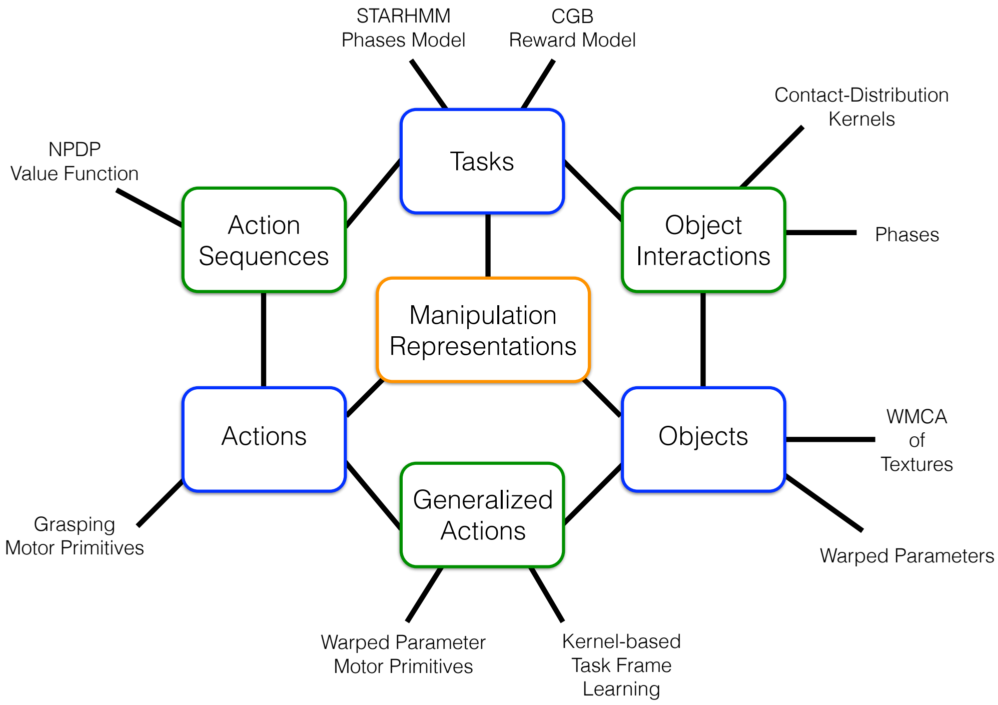


Figure 1.3: The figure illustrates the diversity of representations needed for learning manipulations, and how they are linked to the different topics in this thesis.

using the warped parameters described in Chapter 3. This chapter also explains a kernel for implicitly representing the shape of contacts between objects. In Chapter 6, we describe how the robot can learn sensory representations of textured surfaces.

The robot also needs to represent different tasks. In general, a task can be defined using a reward function. In Chapter 5, we explain how a robot can learn to select sequences of actions in order to maximize the task reward. The reward function defines the goal of the task, but not necessarily its structure. In order to represent the structure of tasks, we present a probabilistic model of manipulation phases in Chapter 4.

An overview of the different representations presented in this thesis is shown in Fig. 1.3. As the figure indicates, some of the representations fall between the object, action, and task labels. These representations allow the robot to generalize actions between different tasks and objects. The robot can learn versatile manipulation skills using the proposed representations.

2 Learning to Grasp Through Trial and Error

Learning to perform a task through trial-and-error learning is a fundamental skill for robots working in everyday environments. By learning from its own experiences, aß robot can autonomously acquire new information about objects and improve its manipulation skills. However, performing completely random exploratory movements will only rarely yield information that is relevant to the current task. Performing the same action will also not result in a lot of new information. Hence, in order to learn the manipulation skill quickly, the robot must find a balance between performing the current best action and exploring new actions [26]. We therefore present a continuum-armed bandits approach for learning to grasp, which explicitly models the exploration-exploitation trade-off. The performance of different grasps is modeled using Gaussian process regression. The robot selects the next grasp to execute using an upper confidence bound (UCB) policy, and uses the outcome of the grasp to improve its model. The improved model allows the robot to select better grasps in the future. In this project, we also wanted to explore whether the robot could optimize grasps without making assumptions about important object properties, e.g., shape, appearance, or material, when selecting a grasp. The grasp is therefore simply defined as the 3D position and orientation of the hand relative to the object frame.

2.1 Combining Active Learning and Reactive Control for Robot Grasping

One of the key challenges for robotics is the large variability inherent in the tasks and environments that a robot may encounter. Preparing a robot completely beforehand for all possible situations is probably impossible as it is prohibitively difficult to foresee all scenarios. Such a preparation is also inefficient, as only a few of the situations will be required by the robot. Due to these limitations, it is important to design robots that can adapt and learn from their own experiences.

Grasping an unknown object is an example of a task that is made particularly difficult by the large variety of objects (see Figure 2.2). Many approaches have been proposed for robot grasping. Early work [27, 28] found analytical solutions to the problem, but these approaches require precise information about the environment (e.g., external forces, surface properties) that may not be accessible. Supervised learning can be used to train robots how to recognize good grasping points [29], but requires a considerable initial input from a human supervisor. Active and reinforcement learning methods have focused on exploring the object to acquire complete affordance models [30, 31], but not on optimizing grasps. However, finding good grasp locations is only a part of the problem.

The robot grasping task can be decomposed into two problems: deciding where to grasp the object, and determining how to perform the grasping movement. These two sub-problems are closely related and must be addressed together in order to perform a successful grasp. The choice of where to grasp an object sets the context for determining how to grasp it. However, the execution of the grasp ultimately determines whether the grasp location was well-chosen.

In this chapter, we present a hierarchical controller that reflects the structure of these two task components, as shown in Figure 2.1. The upper level decides where to grasp the object, and the lower level determines how to perform the grasping movements given the context of these grasp parameters and the scene. The upper level subsequently receives a reward based on the grasp execution, and takes this into consideration when selecting future grasps.

The system employs a hybrid architecture that uses reinforcement learning, imitation learning, and reactive control. The core of the upper level is a reinforcement learning approach that uses the successfulness of evaluated grasps to determine future grasps. It is crucial that its state-action space is

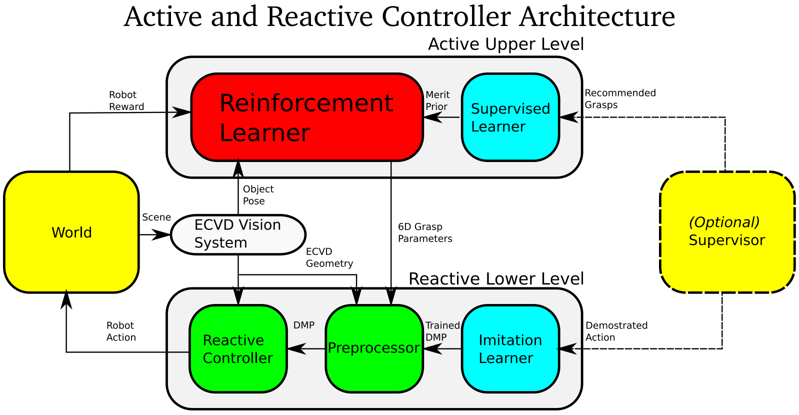


Figure 2.1: The controller architecture consists of a upper level based on reinforcement learning and a bottom level based on reactive control. Both levels are supported by supervised/imitation learning. The World and Supervisor are external elements of the system.

low dimensional for faster convergence [32, 33], and that information from other sources (e.g., demonstrated grasps) can easily be incorporated. To reduce the action space, the reinforcement learner specifies a grasp as a six dimensional hand pose in the object's reference frame, and all remaining variables inherent to the grasping movements are handled by a lower level controller.

The lower level controller is responsible for action execution. A straightforward method of acquiring an arbitrary motion policy is by imitation learning. One approach to imitation learning is to transform a demonstrated trajectory into a standard dynamical systems motor primitive (DMP) [34, 35]. This policy is adapted, in a task specific manner, to the grasp parameters specified by the reinforcement learner. The resulting DMP is augmented by a reactive controller that takes the geometry of the object and scene into consideration. The resulting action is executed by the robot, which returns a corresponding reward to the upper level of the controller.

The complete hybrid controller is illustrated in Figure 2.1. It uses its own experiences to quickly converge on good grasping locations. The grasping motions are taught by demonstration and adapted to different grasp locations and the surrounding geometry. A key feature of this hybrid approach is that the reactive controller is incorporated in the reinforcement learner's action-reward feedback loop. Thus, the hybrid system will learn an appropriate grasping action together with a corresponding grasp location, and solve both of the sub-problems.

In the following sections, we discuss the proposed controller in a top-down manner. The active learner and the reactive bottom level of the controller are detailed in Sections 2.2 and 2.3 respectively. In Section 2.4, the system is evaluated both in simulation and on the robot platform shown in Figure 2.2.

2.2 High-Level Active Learner

The high-level controller chooses where on the object to apply the next grasp, and improves the grasp locations using the acquired data. The reinforcement learning approach is inspired by the grasp learning exhibited by infants [36, 37, 38], requiring relatively little prior knowledge and making few assumptions. Young infants have a grasp reflex that allows them to crudely grasp objects [36]. They learn to improve their grasps through trial and error, allowing them to later be able to perform precision grasps. The reactive controller of the hybrid system represents a vision-based grasp reflex. The initial grasps may be crude, but the learning system will adapt to the object and can learn to perform precision grasps.

To keep the number of assumptions low, we define the state as the object being grasped, and learn a model for each object. The robot's grasps are learned in the object's reference frame, allowing the object to be repositioned in the workspace. Similar to a young infant [36], learning to grasp an object is treated

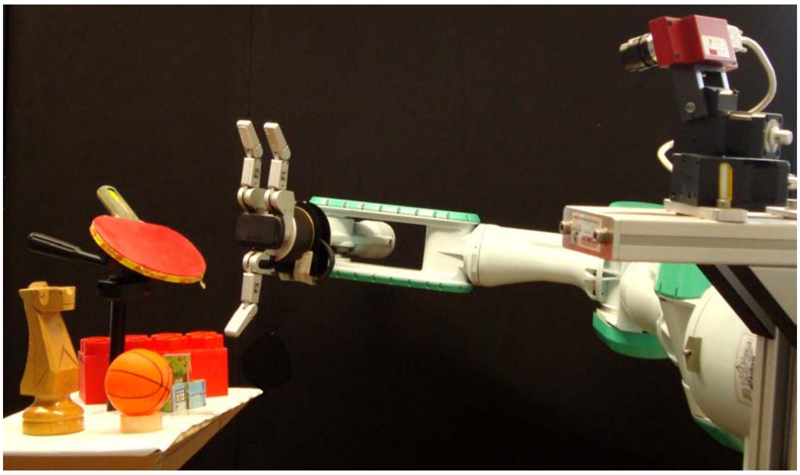


Figure 2.2: The robot used in our experiments and an example of a grasping task in a cluttered environment.

as context independent and only based on the task constraints it has encountered. Thus, if an object has always been presented as hanging on a string, both the robot and infant would initially not know that grasping it from below does not work when the object is on a table [36]. The robot will assign an expected reward to the grasp that reflects both situations and how often it has encountered each.

Another infant-like feature is that the robot has no vision-grasp mapping. Infants under nine months do not orientate their hands to the orientation of object parts [38]. The robot also does not assume that the geometry of a cup's handle will imply a certain orientation of the hand as appropriate. Instead, it will try different orientations and find one that is well-suited for it. Hence, several object properties do not need to be modeled explicitly, e.g., friction. Ultimately, the reinforcement learning approach is highly adaptive and is applicable to a wide range of situations.

In contrast, supervised learning of grasps has focused on methods using internal models of the world [39, 40], or mappings between visual features of objects and grasps [29]. These approaches are more characteristic of adult human grasping, and thus require large amounts of prior information.

To converge quickly to high rewarding grasp locations, the system must balance the exploitation of good grasping points and the exploration of new, possibly better, ones [26]. From a machine learning perspective, this selecting of grasps can be interpreted as a continuum-armed bandits problem [41].

The continuum-armed bandit problem is a generalization of the traditional n-armed bandit problem [32] where the agent must choose from a continuous range of locally dependent actions, instead of a finite number. Under this interpretation, the action is given by the grasp applied and the reward is a measure of the success of this grasp.

To date, most methods [42, 43] that solve the continuum-armed bandit problem are based on discretizing the space. For high-dimensional domains, such as robot grasping, any discrete segmenting will scale badly due to the *curse of dimensionality* [33]. The hard segmentation will result in unnatural borders and make the use of prior knowledge complicated. We propose a sample-based reinforcement learner that models the distribution of expected rewards over the continuous space of actions using Gaussian process regression (GPR) [44]. The proposed learner then searches for the most promising grasp to evaluate next, using a method inspired by mean-shift clustering [45]. The resulting policy is called Continuum Gaussian Bandits (CGB), and is outlined in Algorithm 1.

The following four sections detail the active learner and present the employed policy (Section 2.2.1), the modeling of the expected rewards (Section 2.2.2), how the learner selects the next grasp (Sections 2.2.3), and then the method for implementing this selection on the reward model (Sections 2.2.4 and 2.2.5). Finally, Section 2.2.6 explains how supervised data can be incorporated into the active learner as prior knowledge.

2.2.1 Upper Confidence Bound Policy

Choosing where to grasp a novel object suffers from an exploration-exploitation problem. The traditional machine learning framework for studying this dilemma is the n-armed bandits problem, wherein an agent must repeatedly choose from a finite set of n possible actions to maximize the accumulated reward.

Among the more successful strategies [32] are upper confidence bound (UCB) policies. While there are different versions of UCB policies [32, 3], the principle idea is to assign each action two variables, i.e., the expected reward μ for taking that action, and a confidence bound $\pm\sigma$ indicating the range in which the actual mean reward is. Both μ and σ indicate how desirable executing the action is. A high expected reward μ is valuable in the sense of exploitation and receiving rewards, while a large confidence bound σ indicates an informative action that is good for exploration. Using the exploration variable σ leads to a more structured exploration than regular randomized policies (e.g., ϵ-greedy [32]). UCB policies also provide performance guarantees, and have an upperbound on the expected regret that scales only logarithmically with the number of trials [3].

The sum of the expected reward μ and the standard deviation σ indicates how desirable executing the action is overall. Adopting Bayesian optimization terminology, we refer to the value $\mu + \sigma$ as the acquisition function value. A UCB policy always selects the action for which the acquisition function is the greatest [3]. Intuitively, a UCB policy optimistically chooses the action which *could* be the best, and will thus only converge to an action when it knows that no other action could be better.

Adapting a UCB policy to the continuum-armed bandits requires a new approach that scales to the high-dimensional spaces of grasping tasks. The first step towards realizing this approach is to create a sample-based model of the exploration σ and exploitation μ variables.

2.2.2 Expected Reward and Confidence Modeling with Gaussian Process Regression

Modeling the upper confidence bound for continuous actions requires the expected reward function and its standard deviation to be approximated. A well-suited approach that satisfies these requirements is Gaussian process regression (GPR) [44].

Rather than mapping inputs to specific output values, GPR returns a Gaussian distribution of the expected rewards. This Gaussian distribution is characterized by its mean $\mu(\mathbf{x})$ and standard deviation $\sigma(\mathbf{x})$, where the standard deviation is a confidence bound on the expected reward. This technique is non-parametric, which implies that $\mu(\mathbf{x})$ and $\sigma(\mathbf{x})$ are functions that directly incorporate all previous samples. Non-parametric methods are very adaptable, and apply few constraints on the model. The GPR approach incorporates a prior that keeps the mean and variance bounded in regions without data. Unexplored regions will thus have a large confidence bound $\sigma(\mathbf{x})$ and small expected rewards $\mu(\mathbf{x})$. Sampling from these regions will shift $\mu(\mathbf{x})$ towards the actual expected reward at $\mathbf{x}$, and decrease the standard deviation $\sigma(\mathbf{x})$.

We employ the common Gaussian kernels $k(\mathbf{x}, \mathbf{y}) = \sigma_a^2 \exp(-0.5(\mathbf{x}-\mathbf{y})^T \mathbf{W}(\mathbf{x}-\mathbf{y}))$ where $\mathbf{W}$ is a diagonal matrix of kernel widths. The parameter σ_a affects the convergence rate of the policy, as explained in Section 2.2.6.

For grasping, the vectors $\mathbf{x} \in \mathbb{R}^6$ and $\mathbf{y} \in \mathbb{R}^6$ each contain three position and three orientation parameters of grasps, which describe the final position of the hand in the object's reference frame. Working in the object's reference frame allows the object to be repositioned and reorientated in the workspace without altering the grasp parameters. Additional grasp parameters are excluded to keep the number of parameters minimal, and thus allow for rapid learning. All of the other motion parameters are handled by the reactive low level controller, which modifies these parameters depending on the object and the scene, as well as the parameters in $\mathbf{x}$.

The proposed UCB policy will base its decisions on the acquisition function $M(\mathbf{x}) = \mu(\mathbf{x}) + \sigma(\mathbf{x})$, where $\mu(\mathbf{x})$ and $\sigma(\mathbf{x})$ are the expected reward and standard deviation at grasp $\mathbf{x}$ respectively. The standard GPR model [44] for the mean μ, variance σ^2, and standard deviation σ, are

$$\begin{aligned}\mu(\mathbf{x}) &= \mathbf{k}(\mathbf{x},\mathbf{Y})^T \left(\mathbf{K} + \sigma_s^2 \mathbf{I}\right)^{-1} \mathbf{t},\\ \sigma(\mathbf{x}) &= \sqrt{\sigma^2(\mathbf{x})} = \sqrt{k(\mathbf{x},\mathbf{x}) - \mathbf{k}(\mathbf{x},\mathbf{Y})^T \left(\mathbf{K} + \sigma_s^2 \mathbf{I}\right)^{-1} \mathbf{k}(\mathbf{x},\mathbf{Y})},\end{aligned}$$

where $[\mathbf{K}]_{i,j} = k(\mathbf{y}_i, \mathbf{y}_j)$ is the Gram matrix, the kernel vector $\mathbf{k}$ decomposes as $[\mathbf{k}(\mathbf{x},\mathbf{Y})]_j = k(\mathbf{x},\mathbf{y}_j)$, the hyperparameter σ_s^2 indicate the noise variance, and the N previous data points are stored in $\mathbf{Y} = [\mathbf{y}_1, \ldots, \mathbf{y}_n]$ with corresponding rewards $\mathbf{t} = [t_1, \ldots, t_n]$.

Both the mean and variance equations can be rewritten as the weighted sum of Gaussians, giving

$$\begin{aligned}\mu(\mathbf{x}) &= \textstyle\sum_{j=1}^{N} k\left(\mathbf{x},\mathbf{y}_j\right)\alpha_j,\\ \sigma^2(\mathbf{x}) &= k(\mathbf{x},\mathbf{x}) - \textstyle\sum_{i=1}^{N}\sum_{j=1}^{N} k'\left(\mathbf{x}, 0.5\left(\mathbf{y}_i + \mathbf{y}_j\right)\right)\gamma_{ij},\end{aligned}$$

where $k'(\mathbf{x},\mathbf{y}) = \sigma_a^2 \exp(-(\mathbf{x}-\mathbf{y})^T \mathbf{W}(\mathbf{x}-\mathbf{y}))$, and the constants are defined as $\alpha_j = [(\mathbf{K} + \sigma_s^2 \mathbf{I})^{-1}\mathbf{t}]_j$ and $\gamma_{ij} = [(\mathbf{K} + \sigma_s^2 \mathbf{I})^{-1}]_{i,j} \exp(-0.25(\mathbf{y}_i - \mathbf{y}_j)^T \mathbf{W}(\mathbf{y}_i - \mathbf{y}_j))$. Different upper confidence intervals σ have been used in UCB policies [46], and can be used by modeling them with a second GPR [47].

The previous rewards $\mathbf{t}$ occur in the exploitation term $\mu(\mathbf{x})$, but not in the standard deviation $\sigma(\mathbf{x})$ as it represents the exploration, which is independent of the rewards. A similar acquisition function has previously been employed for multi-armed bandits in metric spaces, wherein GPR was used to share knowledge between discrete bandits [48].

Having chosen a UCB policy framework and a GPR reward model, the implementation of the policy has to be adapted to the acquisition function.

2.2.3 UCB Policy for GPR Model

Given a model of the UCB acquisition function, the system requires a suitable method for determining the action with the highest acquisition function value. Executing this grasping action will acquire the greatest combination of reward and information.

The acquisition function will most likely not be concave and will contain an unknown number of maxima with varying magnitudes [44]. Determining the global maximum of the acquisition function analytically is therefore usually intractable [44]. However, numerically, we can determine a set of locally optimal grasps. Such sets of grasps will contain many maxima of the acquisition function, especially near the previous data points. Given a set of local maxima, the acquisition function is evaluated for each candidate grasp, and the robot executes the grasp with the highest value.

The method for finding the local maxima was inspired by mean-shift [45], which is commonly used for both mode detection of kernel densities and clustering. Mean-shift converges onto the local maxima of a given point by iteratively applying

$$\mathbf{x}_{n+1} = \frac{\sum_{j=1}^{N} \mathbf{y}_j k\left(\mathbf{x}_n, \mathbf{y}_j\right)}{\sum_{j=1}^{N} k\left(\mathbf{x}_n, \mathbf{y}_j\right)}, \tag{2.1}$$

where $k(\mathbf{x}_n, \mathbf{y}_j)$ is the kernel function, and $\mathbf{y}_j$ are the N previously tested maxima candidates as before. The monotonic convergence via a smooth trajectory can be proven for mean-shift [45]. To find all of the local maxima, mean-shift initializes the update sequence with all previous data point. The global maximum is then determined from the set of local maxima, which is guaranteed to include the global maximum [49].

The intuition behind this approach for grasping is that all of the previous grasp attempts are locally re-optimized based on the current empirical knowledge, as modeled by the acquisition function. Subsequently, we choose the best of these optimized grasps to execute and evaluate.

Mean-shift is however limited to kernel densities and does not work directly in cases of regression, because the α_j and $\gamma_{i,j}$ weights are not always positive [45]. In particular, the standard update rule (2.1) can not be used, nor can we guarantee that the global maximum will be one of the detected maxima. However, the global maximum is only excluded from the set of found maxima if it is isolated from all previous samples by regions of low value.

As Equation (2.1) is not applicable in our regression framework, a new update step had to be developed, which monotonically converges upon the local maximum of our acquisition function.

2.2.4 Local Maxima Detection for GPR

Given the model in Section 2.2.2, the acquisition function takes the form

$$M(\mathbf{x}) = \textstyle\sum_{j=1}^{N} k(\mathbf{x},\mathbf{y}_j)\alpha_j + \sqrt{k(\mathbf{x},\mathbf{x}) - \sum_{i=1}^{N}\sum_{j=1}^{N} k'\left(\mathbf{x}, 0.5\left(\mathbf{y}_i + \mathbf{y}_j\right)\right)\gamma_{ij}}.$$

To use the policy described in Section 2.2.3 with this acquisition function, a monotonically converging update rule is required that can determine local maxima. We propose an update rule consisting of the current gradient of the acquisition function, divided by a local upper bound of the acquisition function's second derivative; Specifically, we propose

$$\mathbf{x}_{n+1} = \frac{\partial_x \mu + \partial_x \sigma}{q(\mu) + \frac{q(\sigma^2)}{\sqrt{p(\sigma^2)}}} + \mathbf{x}_n = \mathbf{s} + \mathbf{x}_n, \tag{2.2}$$

where $\partial_x \mu = \sum_{j=1}^{N} \mathbf{W}\left(\mathbf{y}_j - \mathbf{x}_n\right) k\left(\mathbf{x}_n, \mathbf{y}_j\right)\alpha_j$ and

$$\partial_x \sigma = \sum_{i=1}^{N}\sum_{j=1}^{N} \frac{2}{\sigma}\gamma_{ij}\mathbf{W}\left(\frac{\mathbf{y}_i + \mathbf{y}_j}{2} - \mathbf{x}_n\right) k'\left(\mathbf{x}, \frac{\mathbf{y}_i + \mathbf{y}_j}{2}\right).$$

The function $q(\cdot)$ returns a local upper bound on the absolute second derivative of the input within the $\mathbf{x}_n$ to $\mathbf{x}_{n+1}$ range. Similarly, $p(\cdot)$ returns a local lower bound on the absolute value of the input.

This form of update rule displays the desired convergence qualities, as explained in Section 2.2.5. The rule is only applicable because the Gaussian kernels have bounded derivatives resulting in finite $q(\mu)$ and $q(v)$, and any real system will have a positive variance giving a real non-zero $\sqrt{p(v)}$.

To calculate the local upper and lower bounds, we first define a region of possible $\mathbf{x}_{n+1}$ values to consider. Therefore, we introduce a maximum step size $m > 0$, where steps with larger magnitudes must be truncated; i.e., $||x_{n+1} - x_n|| \leq m$. Having defined a local neighborhood, $q(\mu)$, $q(v)$, and $p(v)$ need to be evaluated.

In Section 2.2.2, μ and v were represented as the linear weighted sums of Gaussians. Given a linear sum, the rules of superposition can be applied to evaluate $q(\mu)$, $q(v)$, and $p(v)$. Thus, the upper bound of a function in the region is given by the sum of the local upper bounds of each Gaussian, i.e.,

$$q_m\left(\textstyle\sum_{j=1}^{N} k\left(\mathbf{x},\mathbf{y}_j\right)\alpha_j\right) \leq \textstyle\sum_{j=1}^{N} q_m\left(k\left(\mathbf{x},\mathbf{y}_j\right)\alpha_j\right).$$

As Gaussians monotonically tend to zero with increasing distance from their mean, determining an upper bound value for them individually is trivial. In the cases of $q(\mu)$ and $q(v)$, the magnitudes of the second derivatives can be bounded by a Gaussian; i.e.,

$$||\partial_x^2 k\left(\mathbf{x},\mathbf{y}_j\right)|| < \sigma_a^2 \exp\left(-(\mathbf{x}-\mathbf{y}_j)^T \mathbf{W}(\mathbf{x}-\mathbf{y}_j)/6\right),$$

Algorithm 1 Continuum Gaussian Bandits (CGB)

```
Initialize:
          Store $N$ initial points in $\mathbf{Y}$ and $\mathbf{t}$
Loop:
          Calculate $\alpha$ and $\gamma$
          $M_{\text{best}} = 0$
          for $j = 1$ to $N$
            $\mathbf{x}_o = \mathbf{y}_j$
            while not converged
              Calculate update step $\mathbf{s}$
              $\mathbf{x}_{n+1} = \mathbf{s} + \mathbf{x}_n$
            end
            if $M(\mathbf{x}) > M_{\text{best}}$
              $\mathbf{x}_{\text{best}} = \mathbf{x}_n$
              $M_{\text{best}} = M(\mathbf{x}_{\text{best}})$
            end
          end
          Attempt and evaluate $\mathbf{x}_{\text{best}}$
          Store results in $\mathbf{y}_{N+1}$ and $\mathbf{t}_{N+1}$
          $N = N + 1$
```

which can then be used to determine the local upper bound.

We have thus defined an update step and its implementation, which can be used to detect modes of a Gaussian process in a regression framework. The final algorithm has a time complexity of $O(N^3)$, similar to all other exact GPR methods [47]. However, this complexity scales linearly with the number of dimensions, while discretization methods scale exponentially, making the proposed GPR method advantageous when the problem dimensionality is greater than three. The mode detection algorithm can be easily parallelized for efficient implementations on multiple computers or GPUs as an anytime algorithm.

This section concludes the details of the proposed reinforcement learner, which is outlined in Algorithm 1. As shown, the final algorithm is quite compact and straightforward. It consists of modeling the expected rewards using GPR, and applying a parallel search to determine a maximum to evaluate next. The mode detection behavior is analyzed in the next section. Incorporating supervised data from other data sources is described in Section 2.2.6 which completes the upper level of the controller design.

2.2.5 Mode Detection Convergence Analysis

Having specified the method for determining maxima of a GPR in Section 2.2.4, Lyapunov's direct method can be used to show that the method converges monotonically to stationary points. The underlying principle is that an increased lower bound on the acquisition function reduces the set of possible system states and, therefore, a continually increasing acquisition leads to convergence. The following one dimensional analysis will show that only an upper bound on the magnitude of the second derivative is required for a converging update rule.

The increase in the acquisition function value is given by $M(x_{n+1}) - M(x_n)$. Given an upper bound u of the second derivative between x_n and x_{n+1}, and the gradient $g = \partial_x M(x_n)$, the gradient in the region can be linearly bounded as

$$g - \|x - x_n\| u \leq \partial_x M(x) \leq g + \|x - x_n\| u.$$

Considering the case $g \geq 0$ and therefore $x_{n+1} \geq x_n$, the change in the acquisition function is lower bounded by

$$M(x_{n+1}) - M(x_n) \;=\; \int_{x_n}^{x_{n+1}} \partial_x M(x)\, dx \;\geq\; \int_{x_n}^{x_{n+1}} g - (x - x_n)u\, dx.$$

This term is maximal when the linear integrand reaches zero; i.e, $g-(x_{n+1}-x_n)u = 0$. This limit results in a shift of the form $s = x_{n+1}-x_n = u^{-1}g$, as was proposed in Equation (2.2). The same update rule can be found by using a negative gradient and updating x in the negative direction. The acquisition function thus always increases, unless the local gradient is zero or u is infinite. A zero gradient indicates that the local stationary point has been found, and variable u is finite for any practical Gaussian process. In some cases, the initial point may be within the region of attraction of a point at infinity, which can be tested for by determining the distance from the previous data points.

The intuition underlying the results of the analysis is that at each step, the system assumes the gradient will shift towards zero at the maximum possible rate within the region. The estimate of the maximum is then moved to the first point where a zero gradient is possible. This concept can easily be generalized to higher dimensional problems. The update rule guarantees that the gradient cannot shift sign within the update step, and thus ensures that the system will not overshoot nor oscillate about the stationary point. The update rule $x_{n+1} = u^{-1}g + x_n$ therefore guarantees that the algorithm monotonically converges on the local stationary point.

2.2.6 Incorporating Supervised data

Having fully designed the central reinforcement learner, the upper level controller still requires a method for allowing prior task information to be incorporated into the acquisition function to help reduce the search space.

Similar to how a child learns a new task by observing a parent before trying it themselves [36], a robot can use human demonstrations of good grasps to define its starting search region. However, whether these grasps are suitable for the robot is initially unknown.

GPR makes incorporating prior information fairly straightforward. If the supervised data has a reward associated to it, the data can be directly added to the data set. If the region suggested by the demonstration returns only low rewards, the system will begin searching neighboring areas where the acquisition function is still high due to uncertainty. Thus, it defines an initial search region with soft boundaries that can move during the learning process.

The parameter σ_a of the acquisition function specifies how conservative the policy is in expanding these boundaries; i.e., a higher value will encourage more exploration, while a lower value will converge faster. Hence, it can be seen as a learning rate. With the rewards in the grasping task set to be within the range 0 to 1, the parameter is set to 0.75 to encourage exploration but also allow for a reasonable rate of convergence.

The robot experiment was initialized with search regions defined by 7, 10, and 25 demonstrated grasps for the box, watering can, and paddle respectively. The width parameters $\mathbf{W}$ of the Gaussian kernel were also optimized on these initial parameters.

This section concludes the discussion of the upper level controller. It takes the rewards of grasps, the pose of the object, and, optionally, demonstrated data as inputs, and returns the next grasp location to attempt. This grasp location is passed to the robot via a lower level controller, which generates the complete grasping motions based on these parameters.

2.3 Low-Level Reactive Imitation Controller

While the upper level of the controller selected grasp locations, the lower level is responsible for the execution of the grasp, including the reaching and fingers' motions. It is important that the system is adaptive at this level, as the success of a grasp depends on the execution. The finger motions should particularly adapt to the geometry of the object, a process known as preshaping. The robot's motions are learned from human demonstrations, and subsequently modified to incorporate the grasp information from the active learner and the scene geometry from the vision system.

A common approach to the grasp execution problem is to rely on specially designed sensors (e.g., laser scanner, ERFID) to get accurate and complete representations of the object and environment [39, 50], followed by lengthy planning phases in simulation [51]. We restrict the robot to only using stereo cameras, and a fast reactive sensor-based controller [52].

Although densely sampling sensors such as time-of-flight cameras and laser range finders are favored for reactive obstacle avoidance [53], the sparser information of stereo vision systems has also been used for these purposes [54, 55]. Robot grasping research has focused on coarse object representations of novel objects [56, 57, 58, 59], and using additional sensor arrays when in close proximity to the object [60, 61]. Learning to grasp objects is also often done in simulation [56, 40] which allows for many virtual grasp attempts on a model of the object. In contrast, the proposed hybrid system relies on relatively few real-world grasps and does not rely on having accurate dynamics and contact models.

For the lower level system, we propose a sensor-based robot controller that can perform human inspired motions, including preshaping of the hand, smooth and adaptive motion trajectories, and obstacle avoidance, using only stereo vision to detect the environment. Unlike previous approaches, we work with a sparse visual representation of objects, which maintains a high level of geometric details. The controller uses potential field methods [52], which treat the robot's state as a particle in a force field; i.e. the robot is attracted to a goal state, and repelled from obstacles.

The attractor field needs to be capable of encoding complex trajectories and adapting to different grasp locations. We therefore use the dynamical system motor primitive (DMP) [62, 35] framework. The DMPs are implemented as passive dynamical systems superimposed with external forces; i.e.,

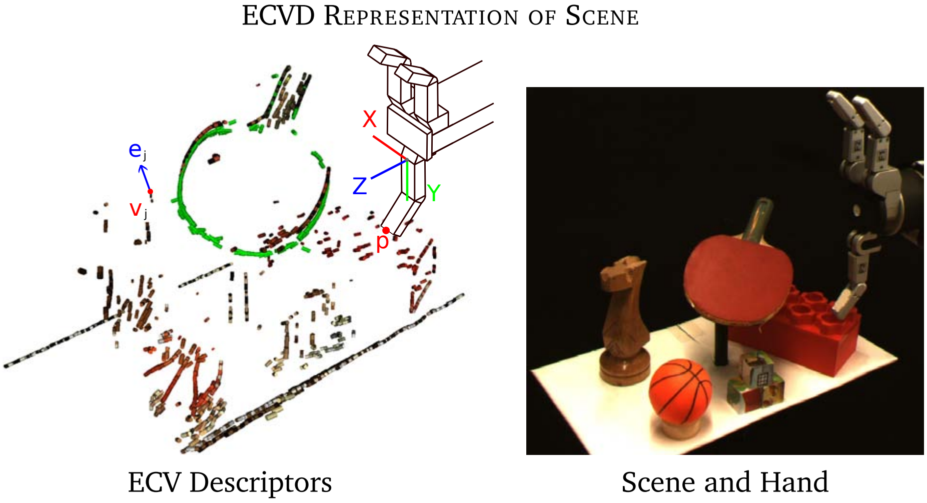
$$\ddot{y} = \alpha_z(\beta_z \tau^{-2}(g - y) - \tau^{-1}\dot{y}) + a\tau^{-2} f(x), \tag{2.3}$$

where α_z and β_z are constants, τ controls the duration of the primitive, a is an amplitude, $f(x)$ is a nonlinear function, and g is the goal for the state variable y. The variable $x \in [0, 1]$ is the state of a canonical system $\dot{x} = -\tau x$, which acts as a shared clock amongst different DMPs; i.e. it ensures that the finger and arm motions are synchronized. The function $f(x)$ encodes the trajectory for reaching the goal state, and takes the form

$$f(x) = \frac{\sum_{j=1}^{M} \psi_j(x) w_j x}{\sum_{i=1}^{M} \psi_i(x)},$$

where $\psi(x)$ are M Gaussian basis functions, and w are weights. The weights w are acquired by imitation learning, using locally weighted regression [62, 34]. The DMPs treat the goal state g as an adjustable variable and ensure that this state is always reached. However, their capability to generalize can be further improved by using a task-specific reference frame based on the active learner's grasp parameters, as detailed in Section 2.3.2. This adaptation of the action to different goals allows the object to be repositioned and reorientated in the robot's workspace.

More important is the choice of the scene's visual representation, which is used to augment the attractor field and forms the basis of the detractor field. The scene description needs to be in 3D, work at a fine scale to maintain geometric details, and represent the scenes sparsely to reduce the number of calculations required per time step. The Early Cognitive Vision system of Pugeault et al. [63, 64] (see Figure 2.3) fulfills these requirements by extracting edge features from the observed scene. The system subsequently localizes and orientates these edges in 3D space [65], with the resulting features known as early cognitive vision descriptors (ECVD) [63]. By using a large amount of small ECVDs, any arbitrary object or scene can be represented. Given an ECVD model of an object, the object's position and orientation can be determined [66] and the ECVDs of the object model can be superimposed into the scene representation.


ECVD Representation of Scene

ECV Descriptors

Scene and Hand

Figure 2.3: The left image shows the ECVD representation of the scene on the right. The paddle is the object to be grasped, while the surrounding objects clutter. The coordinate frame of the third finger of the lower finger in the image and the variables used in Section 2.3 are shown. The x-y-z coordinate system is located at the base of the finger, with z orthogonal to the palm, and y in the direction of the finger. The marked ECVD on the left signifies the j^{th} descriptor, with its position at $\mathbf{v}_j = (v_{jx}, v_{jy}, v_{jz})^T$, and edge direction $\mathbf{e}_j = (e_{jx}, e_{jy}, e_{jz})^T$ of unit length. The position of the finger tip is given by $\mathbf{p} = (p_x, p_y, p_z)^T$.

As a hybrid system, the lower-level controller supplies a complex adaptive action policy that the upper level can indirectly modify. The top level controller only needs to modify the action for a given object, which can be done more efficiently than having to learn the entire action. To allow for quick learning, the actions given by the reactive controller should be repeatable, while still adaptive. By making the rewards for grasps depend on the reactive controller, the reinforcement learner finds both good grasp locations as well as matching grasp executions.

In Sections 2.3.1 and 2.3.2, we describe the DMPs for grasping, followed by their augmentation using the ECVD based detractor field in Section 2.3.3.

2.3.1 Attractor Fields based on Dynamical Systems Motor Primitives

Generating the grasp execution begins with defining an attractor field as a DMP, which encodes the desired movements given no obstacles. The principle features that need to be defined for these DMPs are the goal positions, and the generic shape of the trajectories.

The high-level grasp controller gives the goal location and orientation of the hand, but not the fingers. Using the ECVDs, the goal position of each finger is approximated by first estimating a locally linearized contact plane for the object in the finger coordinate system (see Figure 2.3). The purpose of this step is to get the fingers close to the object's surface during preshaping to allow for more control of the object during grasping. It is not intended to infer exact surface properties or whether the grasp is suitable. If the selected surface is unsuitable for grasping, a low reward will be received and the upper level controller will adapt its policy accordingly.

A contact plane is approximated for each finger to allow for a range of object shapes. The influence of the i^{th} ECVD is weighted by $w_i = \exp(-\sigma_x^{-2} v_{ix}^2 - \sigma_y^{-2} v_{iy}^2 - \sigma_z^{-2} v_{iz}^2)$, where σ_x, σ_y, and σ_z are length constants that reflect the finger's length and width, and $\mathbf{v}_i$ is the position of the ECVD in the finger reference frame. The hand orientation is such that the Z direction of the finger should be approximately parallel to the contact plane, which reduces the problem to describing the plane as a line in the 2D X-Y space. The X-Y gradient of the plane is approximated by $\phi = (\sum_{i=1}^{N} w_i)^{-1} \sum_{i=1}^{N} w_i \arctan(e_{iy}/e_{ix})$, where

N is the number of vision descriptors, and $\mathbf{e}_i$ is the direction of the i^{th} edge. The desired Y position of the fingertip is then given by

$$\tilde{p}_y = \frac{\sum_{i=1}^{N}(w_i v_{iy} - \tan(\phi) w_i v_{ix})}{\sum_{i=1}^{N} w_i},$$

which can be converted to joint angles using the inverse kinematics of the hand. The proposed method selects the goal postures of the fingers in a deterministic manner, which depends on the object's geometry as well as the grasp parameters specified by the active learner. Thus, the hybrid system's active learner indirectly selects the posture of the fingers through a reactive mechanism based on the visual model of the object.

The next step defines the reaching and grasping trajectories. Many beneficial traits of human movements, including smooth motions and small overshoots for obstacle avoidance [67, 68, 37], can be transferred to the robot through imitation learning. To demonstrate grasping motions, we used a VICON motion tracking system to record human movements during a grasping task. The grasped object can be different to the robot's. VICON markers were only required at the hand and finger tips. The tracking system samples the human's motions, generating position $\mathbf{q}$, velocity $\dot{\mathbf{q}}$, and acceleration $\ddot{\mathbf{q}}$ data, as well as the samples' time stamps. The weights w_i of the DMP are then given by

$$w_i = \left(\sum_{k=1}^{T} \psi_i(x_k) x_k^2\right)^{-1} \sum_{j=1}^{T} \psi_i\left(x_j\right) x_j \left(\tau^2 \ddot{q}_j - \alpha_z(\beta_z(g - q_j) - \tau^1 \dot{q}_j)\right) a^{-1},$$

where x_j is the state of the canonical system corresponding to the j^{th} time stamp. The solution is eaily computed in closed form. Further information on imitation learning of DMPs can be found in Ijspeert's paper [34]. As the reaching trajectories are encoded in task space the correspondence problem of the arm was not a problem.

The DMPs are provably stable [35] and the goal state, as specified by the upper level controller, will always be achieved. Alterations added by the reactive controllers must stay within the bounds of the framework to ensure that this stability is maintained.

2.3.2 Transformed Dynamical Motor Primitives for Grasping

While DMPs generalize to arbitrary goal positions, the grasps' approach direction can not be arbitrarily defined, and the amplitude of the trajectory is unnecessarily sensitive to changes in the start position y_0 and the goal position g if $y_0 \approx g$ during training. These limitations can be overcome by including a preprocessor that modifies the DMPs' hyperparameters.

The system can maintain the correct approach direction by using a task-specific coordinate system. Due to the translation invariance of DMPs, only a rotation $\mathbf{R} \in \mathbb{SO}(3)$ between the two coordinate systems needs to be determined. The majority of the reaching motions will lie in a plane defined by the start and goal locations, and the final approach direction. These components of the plane are supplied by the high level controller, with the approach direction defined by the final hand orientation.

The first new in-plane axis $\mathbf{x}_p$ is set to be along the approach direction of the grasp; i.e., $\mathbf{x}_p = -\mathbf{a}$ as shown in Figure 2.4 . The approach direction is thus easily defined and only requires that the Y_p and Z_p DMPs reach their goal before the X_p primitive. The second axis, $\mathbf{y}_p$, must be orthogonal to $\mathbf{x}_p$ and also in the plane, as shown in Figure 2.4. It is set to $\mathbf{y}_p = b^{-1}((\mathbf{g}-\mathbf{s}) - \mathbf{x}_p(\mathbf{g}-\mathbf{s})^T\mathbf{x}_p)$, where b^{-1} is a normalization term, and $\mathbf{s}$ and $\mathbf{g}$ are the motion's 3D start and goal positions respectively. The third axis vector is given by $\mathbf{z}_p = \mathbf{x}_p \times \mathbf{y}_p$. The DMPs can thus be specified by the preprocessor in the X_p-Y_p-Z_p coordinate system, and mapped to the X_w-Y_w-Z_w world reference frame by multiplying by $\mathbf{R}^T = [\mathbf{x}_p, \mathbf{y}_p, \mathbf{z}_p]^T$.

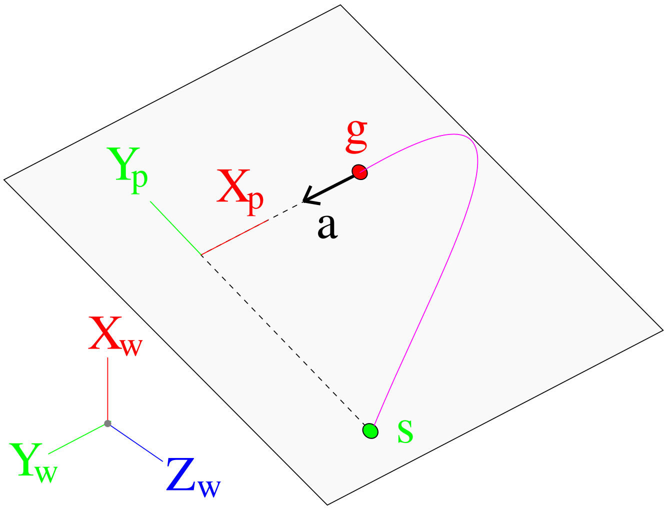


Figure 2.4: The diagram shows the the change in coordinate systems for the reaching DMPs. The axes X_w-Y_w-Z_w are the world coordinate system, and X_p-Y_p-Z_p is coordinate system in which the DMP is specified. The trajectory of the DMP is shown by the curved line, starting at **point s**, and ending at **point g**. X_p is parallel to the approach direction of the hand, the **arrow a**. The axis Y_p is perpendicular to X_p, and pointing from s towards **g**.

The change of coordinate system is a fundamental step for the hybrid system. It places the reactive controller, together with all of its modifications, within the reinforcement learner's action-reward feedback loop. Therefore, the system learns pairings of grasp locations and grasp executions that lead to high rewards.

The second problem relates to the scaling of motions with ranges greater than $||y_0 - g||$, which are required to move around the outside of objects. In the standard form $a = g - y_0$ [62], which leads to motions that are overly sensitive to changes in g and y_0 if $g \approx y_0$ during training. The preprocessor can reduce the sensitivity by using a more robust scaling term, for which we propose the amplitude

$$a = ||\eta(g - y_0) + (1 - \eta)(g_T - y_{0T})||,$$

where g_T and y_{0T} are the goal and start positions of the training data respectively, and $\eta \in [0, 1]$ is a weighting hyperparameter. This amplitude is always between the training amplitude and the standard generalization value $a = g - y_0$, and η controls how conservative the generalization is to new goals (see Figure 2.5) . By taking the absolute value of the amplitude, the approach direction is never reversed (see Figure 2.5). The amplitude previously proposed by Park et al. [58] corresponds to the special case of $\eta = 0$. Example generalizations of a reaching trajectory are shown in Figure 2.6.

The described transformations allow a single DMP to perform a larger range of grasps, which implies that fewer DMPs are required in total. Using different DMPs for different sections of the object or workspace should be avoided as it creates unnecessary discontinuities in the rewards, which can slow down the hybrid system's learning process. Only one grasp had to be learned for the entire robot experiment, which was then adapted to the various situations.

2.3.3 Detractor Fields based on ECVDs

Detractor fields refine the motions generated by the DMPs to avoid obstacles during the reaching motion and ensure that the finger tips do not collide with the object during the hand's approach.

The detractor field is based on ECVDs, which represent small line segments of an object's edges localized in 3D, as shown in Figure 2.3. The detractive forces of multiple ECVDs describing a single line

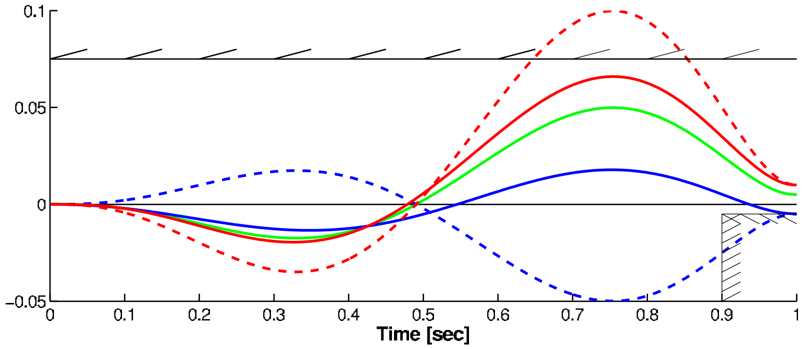


Figure 2.5: This is a demonstration of the effects of transforming the amplitude variable a of DMPs. The hashed black lines represent boundaries. The **solid green** line shows the trained trajectory of the DMP going to 0.05. If goal is then placed at 0.1 and the workspace is limited to ±0.075 (top boundary), the **dashed red** line is the standard generalization to a larger goal, while the **solid** plot uses the new amplitude. If the goal is −0.05, and needs to be reached from above (lower right boundary), then the **dashed blue** line is the standard generalization to a negative goal, and the **solid grey** trajectory uses the new amplitude. Both of the new trajectories were generated with $\eta = 0.25$.

should not superimpose, nor should the field stop DMPs from reaching their ultimate goals. The system therefore uses a Nadaraya-Watson model [69, 70] of the form

$$u_a = -\upsilon(x)\frac{\sum_{i=1}^{N} r_i c_{ai}}{\sum_{j=1}^{N} r_j},$$

to generate a suitable detractor field, where r_i is a weight assigned to the i^{th} ECVD, s is the strength of the overall field, x is the state of the DMPs' canonical system, c_{ai} is the detracting force for a single descriptor, and subscript a specifies if the detractor field is for the finger motions or the reaching movements.

The weight of an ECVD for collision avoidance is given by $r_i = \exp(-(\mathbf{v}_i - \mathbf{p})^{\mathrm{T}}\mathbf{h}(\mathbf{v}_i - \mathbf{p}))$, where $\mathbf{v}_i$ is the position of the i^{th} ECVD in the local coordinate system, $\mathbf{h}$ is a vector of positive length scale hyperparameters, and $\mathbf{p}$ is the finger tip position, as shown in Figure 2.3. The detractor puts more importance on ECVDs in the vicinity of the finger.

The reaching and finger movements react differently to edges and employ different types of basis functions c_i for their respective potential fields. For the fingers, the individual potential fields are logistic sigmoid functions about the edge of each ECVD of the form $\rho(1+\exp(d_i\sigma_c^{-2}))^{-1}$, where $d_i = \left\|(\mathbf{p}-\mathbf{v}_i)-\mathbf{e}_i(\mathbf{p}-\mathbf{v}_i)^{\mathrm{T}}\mathbf{e}_i\right\|$ is the distance from the finger to the edge, $\rho \geq 0$ is a scaling parameter, and $\sigma_c \geq 0$ is a length parameter. Differentiating the potential field results in a force of

$$c_{fi} = \rho\left(1+\exp\left(d_i\sigma_c^{-2}\right)\right)^{-2}\exp\left(d_i\sigma_c^{-2}\right).$$

As the sigmoid is monotonically increasing, the detractor always forces the fingers open further to move their tips around the ECVDs and ensure that they approach the object from the outside. A similar potential function can be employed to force the hand closed when near ECVDs pertaining to the scene rather than the object.

The reaching motion uses the Gaussian basis functions of the form $\varrho\exp(-0.5\mathbf{d}_i^{\mathrm{T}}\mathbf{d}_i\sigma_d^{-2})$, where $\mathbf{d}_i = (\mathbf{q}-\mathbf{v}_i)-\mathbf{e}_i(\mathbf{q}-\mathbf{v}_i)^{\mathrm{T}}\mathbf{e}_i$ is the distance from the end effector position, $\mathbf{q}$, to the edge, and $\varrho \geq 0$ and $\sigma_d \geq 0$ are scale and length parameters respectively. Differentiating the potential with respect to $\mathbf{d}_i$ gives a force term in the Y direction of

$$c_{hi} = \varrho(\mathbf{d}_i.\mathbf{Y})\sigma_d^{-2}\exp(-0.5\mathbf{d}_i^{\mathrm{T}}\mathbf{d}_i\sigma_d^{-2}),$$

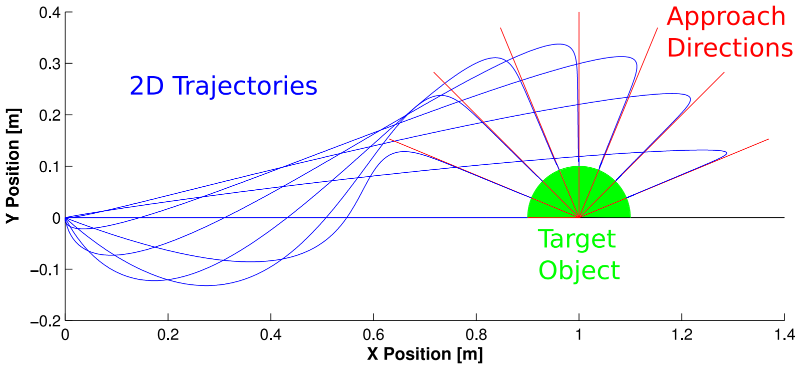


Figure 2.6: Workspace trajectories where the x and y values are governed by two synchronized DMPs. The semicircle indicates the goal positions, with desired approach directions indicated by the red straight lines. The approach direction DMP was trained on an amplitude of one, and $\eta = 0.25$.

which thus apply a radial force from the edge with an exponentially decaying magnitude.

The strength factor $s(\mathbf{x})$ controls the precision of the movements, ensuring that the detractor forces tend to zero at the end of a movement and do not obstruct the DMPs from achieving its goal state. Therefore, the strength of the detractors is coupled to the canonical system of the DMP. Hence, $v(x) = (\sum_{j=1}^{M} \psi_j)^{-1} \sum_{i=1}^{M} \psi_i w_i x$, where x is the value of the canonical system, ψ are its basis functions, and w specify the varying strength of the field during the trajectory.

Modelling the human tendency towards more precise movements during the last 30% of a motion [67], the strength function, $v(x)$, was set to give the highest strengths during the first 70% of the motion for the reaching trajectories, and the last 30% for the finger movements. Setting the strength in this manner is also beneficial to the reinforcement learner. The reward of the learner depends mainly on the final position of the hand, and the closing of the fingers. If these parts of the motion are more repeatable, then it is easier for the upper-level controller to learn.

The detractor fields of both the grasping and reaching components have been defined, and are superimposed into the DMP framework as

$$\ddot{y} = \left(\alpha_z(\beta_z \tau^{-2}(g - y) - \tau^{-1}\dot{y}) + a\tau^{-2} f(x)\right) - \tau^{-2} u_a,$$

which represents the entire ECVD and DMP based potential field.

Combining the ECVD based DMPs with the new coordinate system for reaching and motion amplitude, we have fully defined the low-level controller. Its main contribution is to learn a grasping movement by imitation and then to reactively adapt these motions to new situations in a manner suited to the task and specified by the upper level controller.

2.4 Evaluations

The following sections evaluate the system both in simulation and on a real robot platform. The first part of the evaluation (Section 2.4.1) tests the upper-level controller against other continuum UCB policies on a simulated benchmark problem. The real world evaluation, presented in Section 2.4.2, demonstrates the complete controller working on a real robot grasping novel objects in cluttered environments.

2.4.1 Comparative UCB Analysis

This section focuses on the reinforcement learner and shows that the CGB algorithm (see Algorithm 1) performs well in practice, and can be scaled to the more complex domain of grasp learning. The

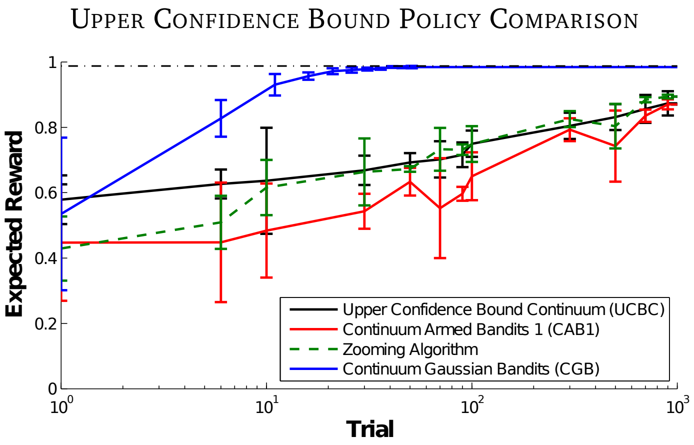


Figure 2.7: The expected rewards over 100 experiments are shown for the four compared methods. The results were filtered for clarity. Due to the differences in experiment lengths, the x-axis uses a logarithmic scale. The dashed horizontal line represents the maximum expected reward given the noise.

comparison is between four UCB policies, including our proposed method, on a 1D benchmark example of the continuum-armed bandits problem. The policies were tested on the same set of 100 randomly generated 7$^{\text{th}}$ order spline reward functions. The rewards were superimposed with uniform noise of width 0.1, but restricted to a range of $[0, 1]$. The space of bandits was also restricted to a range between 0 and 1. None of the policies were informed of the length of the experiment in advance, and each policy was tuned to achieve high rewards.

2.4.1.1 Compared Methods

The tested competing policies are UCBC [42], CAB1 [43], and Zooming [71]. These algorithms represent standard UCB policy implementations for continuum-armed bandits in the literature. A key issue for any policy that uses discretizations is selecting the number of discrete bandits to use. Employing a coarser structure will lead to faster convergence, but the expected rewards upon convergence are also further from the optimal. Balancing this trade-off is therefore important for a policy's success.

The UCBC policy of Auer [42] divides the bandits space into regular intervals and treats each interval as a bandit in a discrete UCB policy. After choosing an interval, a uniform distribution over the region selects the bandit to attempt. The number of intervals sets the coarseness of the system, and was tuned to 10.

Instead of using entire intervals, the CAB1 policy of Kleinberg [43] selects specific grasps at uniform grid points. A discrete UCB policy is then applied to these points, for which we chose UCB1 [3], as suggested in [43]. The discretization trade-off is dealt with by resetting the system at fixed intervals with larger numbers of bandits, thus ensuring that the points becomes denser as the experiment continues.

The zooming algorithm, of Kleinberg et al. [71], also uses a grid structure to discretize the bandits. In contrast to CAB1, the grid is not uniform and additional bandits can be introduced at any time in high rewarding regions. A discrete policy is then applied to this set of active bandits. Similar to CAB1, the zooming algorithm works in time intervals and resets its grid after fixed numbers of trials.

Our proposed Continuum Gaussian Bandits (CGB) method was initialized with 4 equispaced points. Demonstrated data was not used in order to test its performance without the benefits of such data. All four methods were initially run for 55 trials, as shown in Figure 2.7. The CAB1, UCBC, and Zooming methods extended to 1000 trials to demonstrate their convergence behavior.

	UCBC	CAB1	Zoom	CGB
Mean Reward	0.6419	0.4987	0.6065	**0.9122**
1D computation time	46 μs	47 μs	**27 μs**	2.9 sec
6D computation time	4.6 sec	6.7 ms	**5.6ms**	17.6 sec
1D initialization run time	10 min	12 min	24 min	**4 min**
6D initialization run time	1.9 yrs	1.2 days	4.2 days	**24 min**

Table 2.1: These results pertain to the first 50 grasp attempts in the benchmark problem. The table shows the mean computation times for the different algorithms, and how they would scale to six dimensions, given the computational complexity of the algorithms [43, 42, 71]. Similarly, the table shows the amount of time needed to initialize the systems by trying each of the initial grasps once.

2.4.1.2 Results

The expected rewards for the four UCB policies during the experiment can be seen in Figure 2.7. The computation and run times were also acquired for the experiments for comparison, and estimated for the 6 dimensional problem, as shown in Table 2.1.

Apart from our proposed policy, Zooming was the most successful over the 1000 trials at achieving high rewards, as it adapts its grid to the reward function. However, only CGB consistently determined the high rewarding regions and converged on them. In several trails, the reward function had two distinct peaks with near-optimal rewards, and the CGB policy converged onto both.

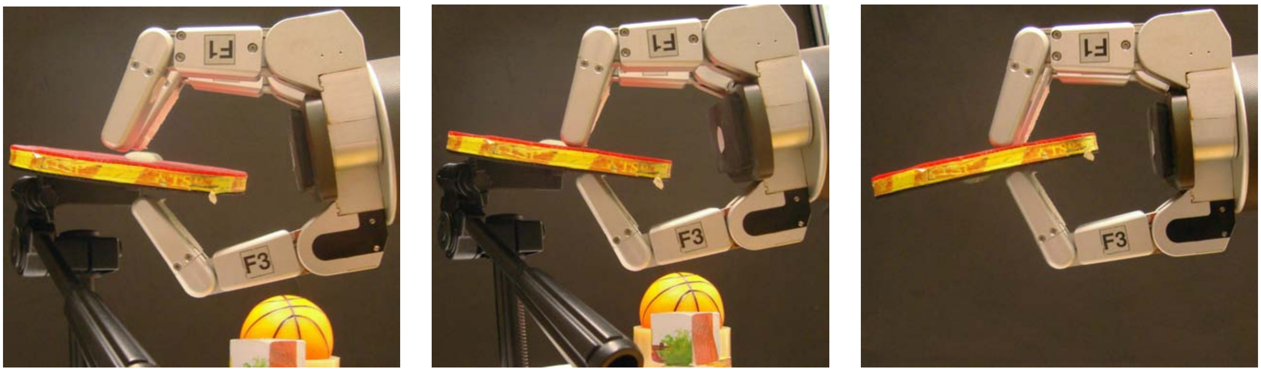
The convergence of UCB policies is frequently described by the acquisition function's percentage of exploitation $\mu(\mathbf{x}^*)/(\mu(\mathbf{x}^*)+\sigma(\mathbf{x}^*))$, where $\mathbf{x}^*$ is the current action selected by the policy. This value is initially zero and increases as the policy returns to previously explored actions with high rewards. The 97.5% exploitation mark was reached by the CGB policy on average at the 33^{rd} trial. Another measure of convergence is found by directly comparing the different maxima found by CGB. The policy converges when the expected value $\mu(\mathbf{x}^*)$ of the selected action is greater than the highest acquisition function value $\mu(\mathbf{x})+\sigma(\mathbf{x})$ of the other candidate actions. This criterion is based on the fact that the acquisition function $\mu(\mathbf{x})+\sigma(\mathbf{x})$ tends to $\mu(\mathbf{x})$ as the exploration of an action is exhausted. Using this criteria, the policy converged on average at the 37^{th} trial.

As parametric policies, the standard methods assume that the optimal solution can be represented by their fixed features and corresponding parameters. These policies can therefore only converge to an optimal solution if it is representable by these features. Both CAB1 and the Zooming algorithm will converge onto the true optimum, but only as the number of samples tends to infinity, as indicated in Figure 2.7.

In terms of computation times, the previous methods were faster than the proposed method, although CGB and UCBC exhibit similar orders of magnitude. One reason for CGB being slower is that this implementation performs the parallel search for maxima sequentially. Parallelizing this search would reduce the expected 6D computation time of CGB to 0.65 seconds.

Most of the system's time is however used to perform the actions (i.e., the run times). For this comparison we focused on the time required to initialize the systems by trying each initial grasp once. Not only is the proposed method the fastest in terms of run times (see Table2.1), it also shows that implementing the other methods for grasping is not practical due to the curse of dimensionality.

The UCBC algorithm has both longer computation and running times than CAB1 and the Zooming algorithm. However, as CAB1 and the Zooming algorithm increase the number of active actions throughout the experiment, these would ultimately exhibit computation and run times greater than UCBC.



A. Preshaping B. Grasping C. Lifting

Figure 2.8: The three main phases of a basic grasp are demonstrated. (A) Preshaping the hand poses the fingers to match the object's geometry. (B) Grasping closes the three fingers at the same rate to secure the object. (C) The object is lifted and the fingers adjust to the additional weight. The objects at the bottom of A and B are clutter.

The memory requirements of the previous methods increases exponentially with the dimensionality, and CGB will only require more memory than UCBC once it has performed a million grasps. The memory requirements of CGB scale with the number of samples, and sufficient memory should be made available depending on the difficulty of the learned task.

In cases where large numbers of samples have been accumulated, suitable implementations of GPR (e.g., Sparse GP [72]) reduce the computational complexity. The loss of accuracy incurred by such implementations is comparable to the accuracy limits inherent to discretization methods, making these methods suitable alternatives to standard GPR.

Ultimately the experiment shows that the proposed method outperforms the other methods in a low dimensional setting, and is the most practical method for higher dimensions due to the curse of dimensionality.

2.4.2 Robot Grasping Task

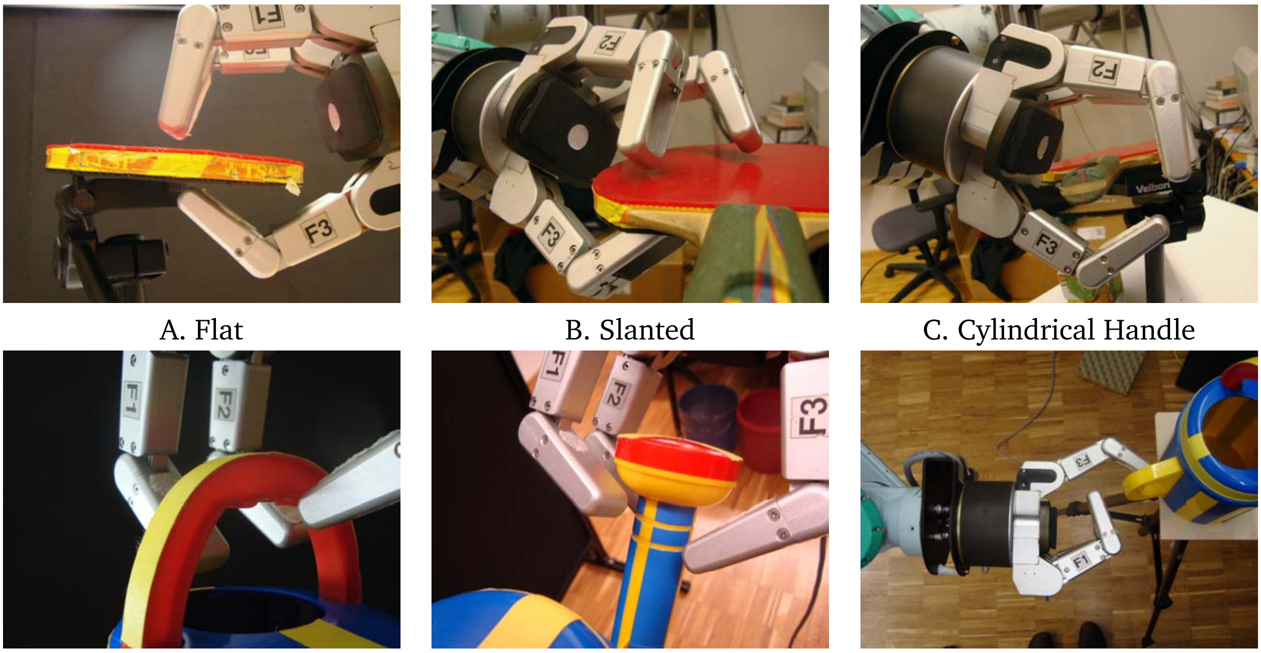
Having shown that the proposed CGB algorithm is an efficient UCB policy, the robotics evaluation focuses on including the lower level controller for improved actions in a robot grasping scenario. This experiment involves the complete system being implemented on a real robot platform. The following sections detail the running of the experiment (Section 2.4.2.1) and the results of the experiment (Section 2.4.2.2).

In this experiment, we implement only the methods proposed in this chapter. The methods described in Section 2.4.1.1 were not tested on the real system.

2.4.2.1 Grasping Experiment

The robot is a basic hand-eye system consisting of a 7 degrees of freedom Mitsubishi PA-10 arm, a Barrett hand, and a Videre stereo camera. The robot only uses sensors essential for the task and forgoes additional hardware such as tactile sensors and laser rangefinders. The robot's task was to learn several good grasps of novel objects through trial and error. All grasps were executed on the real robot and not in simulation.

Each trial begins by estimating the object's position and orientation to convert between world and object reference frames, and to project the ECVD model of the object into the scene representation. The stereo camera allows the object position and orientation to be reliably estimated using the pose estimation method of Detry et al. [66].



A. Flat B. Slanted C. Cylindrical Handle

D. Arched Handle E. Knob F. Extreme Point

Figure 2.9: Various preshapes are shown. **A** and **B** show the system adjusting to different plane angles. **C** and **D** demonstrate the preshaping for different types of handles. **E** shows the preshaping for a circular disc structure, such as a door knob, and gets its fingers closely behind the object. **F** shows where the object was out of the reach of two fingers, but still hooks the object with one finger.

The CGB algorithm then determines the parameters of the next grasp, which the reactive lower level controller uses to modify the grasping action. If the robot grasps the object, the robot attempts to lift the object from the table, as shown in Fig. 2.8. Thus, the robot ensures that the table is not supporting the object. Trials are given rewards depending on how little the fingers moved while lifting the object, thereby encouraging more stable grasps. The rewards are not deterministic due to errors in pose estimation and effects caused by the placement of the object.

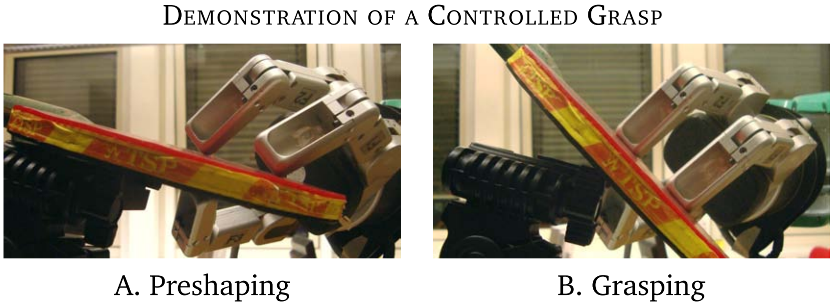
The robot task was made more difficult by adding clutter to the scene. After each grasp attempt, the hand reverses along the same approach direction, but without employing the detractor fields or preshaping of the hand, to determine if collisions would have occurred if the reactive controller had not been used.

The system was run three times on a table tennis paddle to show that it is repeatable. To show that the system can adapt to various scenarios and objects, the experiment was also run twice on both a toy watering can and a wooden box.

The experiments for learning to grasp a paddle consisted of 55 trials, while only 40 trials were required for the watering can and box experiments. Overall 325 different grasp attempts were executed with the combined active and reactive system.

2.4.2.2 Results

The active learner and reactive controller were successfully integrated and the complete system converged onto high-rewarding grasp regions in all of the trials. The imitation learning was straightforward, requiring only one demonstration and allowing for continuous smooth motions to be implemented. Examples of the estimated finger goal locations can be seen in Fig. 2.9. The preshaping adapted to a range of geometries, and consistently placed the fingers close enough to the object for a controlled grasp to be



Demonstration of a Controlled Grasp

A. Preshaping

B. Grasping

Figure 2.10: A controlled grasp, made possible by the hybrid system's preshaping ability. (A) The preshaping matchs the geometry of the object. When grasping, the two fingers on the left pinch the paddle. The finger on the right turns the paddle clockwise about the pinched point. (B) The grasping ends when the paddle has become aligned with all three finger tips.

executed. This preshaping gave more control over the object when grasping, leading to higher rewards and allowing for more advanced grasps to be performed (see Fig. 2.10). Similar to human grasps [73], some of the learned grasps implicitly exploited constraints in order to improve the grasp success rate.

The detractor field and preshaping of the hand allowed the system to work in cluttered environments, which was not a trivial task. The hand came into contact with the clutter for an estimated 8.3% of the grasp attempts, but never more than a glancing contact. These contacts were usually with visually occluded parts of the objects, and thus not fully modelled by the ECVDs. Accumulating the scene representation from multiple views solves this problem. During the reversing phases, when the reactive controller is deactivated, the hand collided with one or more pieces of clutter during 85.4% of the attempts. Thus, the reactive control decreases the number of contacts with the clutter by a factor of ten. The fingers always opened sufficiently to accept the object without colliding with it.

The rewards during the experiment are shown in Figure 2.11. In all of the experiments, the proposed hybrid system found suitable grasps for the object. The watering can and box experiments converged faster than the paddle experiments, due to their initial search region being smaller. While all experiments acquired low rewards for the initial grasps, the soft boundaries allowed the system to explore beyond these regions and find neighbouring regions of better grasps.

Thus, by limiting the robot to evaluating local optima of the acquisition function and using fixed hyperparamters, the policy resulted in a local search for better grasps. This is in contrast to the global optimization performed by UCB Bayesian optimization approaches [48]. The local search was well-suited for the proposed task, as the robot had to select grasps without explicit information regarding the shape and size of the object. A global search could potentially learn more grasps with higher performance, but it would also result in a lot of exploratory grasps that do not even make contact with the object.

Amongst the most important results of this experiment is that the central loop of the hybrid controller works in practice. The system did not just quickly learn a graspable location on an object, but rather the hybrid system quickly learned an entire fluid motion for grasping the object, including preshaping. The system took a single demonstrated action and learned modifications that generalized the action to three different objects. The learning process was significantly hastened by the hybrid approach, as the reactive controller allowed the dimensionality of the reinforcement learner to be kept relatively low, while simultaneously performing complicated grasping motions.

One of the main motivations of this project was to create a reinforcement learner that could generate new grasps for training a supervised grasp learner. Although the system successfully learned grasps which could be used as training data, learning from these grasps is not trivial. Some of the grasps, e.g. Fig. 2.10, involved moving the object around with the fingers before achieving the final grasp. These grasps do not just depend on the local shape of the object, but also the interactions between the object and the environment. Hence, these grasps are not guaranteed to succeed if the object were resting in the middle of a table or wedged between two other objects. The supervised learner would also need to

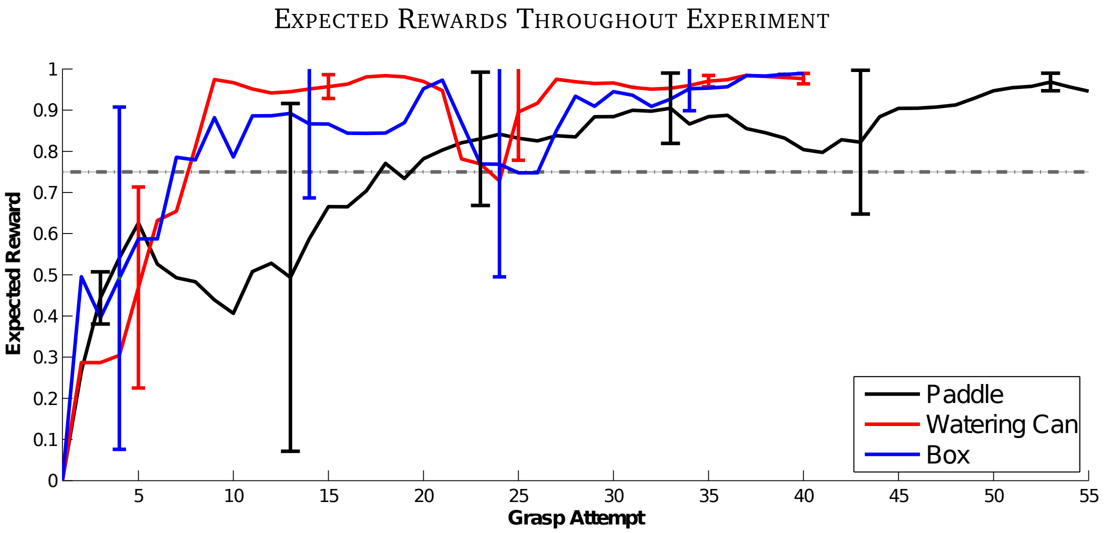


Figure 2.11: The graph shows the expected reward of the attempted grasps over the run of the experiment for the three different objects. All values are averaged over the runs of the experiment, with error bars of +/- two standard deviations. The dashed horizontal line indicates the upper confidence bound of a point at infinity.

take into consideration the interactions between the object and its environment when selecting a grasp. In the future, it would be useful to investigate whether similar types of grasps would be learned if the robot used a kernel that was based on the features used by the supervised learner, e.g., the local shape of the object.

The upper and lower levels divide the grasping problem into two sub-problems: determining where to grasp an object and deciding how to correctly execute the grasp. By incorporating the reactive controller in the learning loop, the hybrid system learned an action that solves both of these sub-problems.

2.5 Conclusion

The first contribution of this chapter was to formulate grasping as a reinforcement learning problem. Other approaches have mainly focused on supervised learning paradigms [74, 75], or active learning approach to guide exploration [30, 31]. By using a reinforcement learning approach, the robot can optimize grasps for specific objects.

The second contribution was to present a hierarchical hybrid controller that can efficiently determine good grasps of objects and execute them. The upper level controller is based on reinforcement learning to allow the robot to learn from its own experiences, but capable of incorporating supervised data from other sources if available. Grasp execution is handled by a lower level controller based on imitation learning and reactive control. This hybrid structure allowed the system to learn both good grasp locations and corresponding grasp executions simultaneously, while keeping the dimensionality of the learning problem low.

We have shown that the presented algorithms and learning architectures work well both in simulation and on a real robot. In simulation, the active learner outperformed several standard UCB policies designed for the continuum-armed bandits problem. The entire system was successfully implemented on a real robot platform, which consistently found high-reward grasps for various objects.

2.6 Potentially Helpful Insights

The main goal of this project was to develop a method that would allow robots to learn better grasps of objects without relying on prior assumptions regarding “good” grasps. We also wanted to determine if learning grasps in this manner is feasible given the limited amount of information provided to the robot and the precision needed for executing some grasps.

By determining better grasps of an object, the robot can autonomously expand its training dataset for a supervised learner in order to predict grasps of novel objects. These predicted grasps can then be used as the initialization for optimizing grasps of other objects. The robot could then iterate between these supervised and reinforcement learning methods in order to master grasping. Although the robot managed to find better grasps, the results also highlight the need to investigate the interactions between different learning methods.

Supervised learning approaches to grasping often try to predict whether a grasp will succeed based on local features of the object at the grasp point . However, the grasps learned by our robot often involved moving the object during the grasping process. The success of these grasps therefore depends on the interactions between the object and its environment, which may not be captured using only local shape information. In our experiments, the effects of the environment were increased by using a smaller stand to support the object. The effects were also more noticeable for the the table tennis paddle, which tended to be grasped vertically, and could therefore not exploit the sliding movements of the object.

The issue of environmental interactions could be addressed in several different manners. One approach would be to explicitly incorporate the interactions in the learning process. The robot could represent the interactions using an additional kernel, such as the one described in Chapter 3. The kernel would effectively weight samples based on the similarity of the environmental interactions. The supervised learner could also explicitly take into consideration the environmental interactions when selecting a grasp. Including this kind of state information would make the proposed approach more robust to changes in the object’s state during learning. If the robot does not explicitly incorporate the interactions, then the model would treat them as additional sources of noise in the reward. In other words, the expected reward would be lower if a grasp only performs well in a few situations. The robot would thus attempt to learn grasps that are robust to such variations, but may also restrict the robot from exploiting certain environmental interactions.

Rather than using supervised learning to generalize grasps and reinforcement learning to improve grasps, one could also attempt to merge these two components by using a kernel that generalizes between objects. For example, one could use a kernel based on the local shape of the object near the grasp frame. This approach would allow the robot to directly learn to improve grasps from multiple objects. The kernel would however not capture some relevant information, e.g. the object’s mass distribution and material properties, which the current kernel incorporates implicitly. Hence, the robot may not be able to learn optimal grasps of individual objects, but could potentially learn grasps that generalize well between objects. An early attempt at this approach [76], using the expected reward as the acquisition function, gave promising results and could quickly adapt to novel objects.

3 Generalizing Between Objects with Different Geometries

When generalizing manipulation skills between different scenarios, the robot must take into consideration the geometry of the object that it is manipulating. The robot should therefore use object representations that allow it to determine similarities between known and novel objects in order to generalize between them. These representation should capture the parts of the object that are relevant to the manipulation task.

In this chapter, we present two representations for generalizing manipulations between objects. The first part of the chapter presents a kernel for contact distributions. Contacts between objects are fundamental to many manipulations, but defining general features to represent contacts is difficult. The kernel allows the robot to compute the similarity between different contact distributions, and use a wide range of kernel methods for learning. The second part of the chapter presents warped parameters for computing geometric features. This approach is based on finding correspondences between objects by warping one object to fit the other's shape. The warped shape is used to compute geometric parameters such as volumes and lengths. These parameters can then be used to learn manipulations that generalize between different objects.

3.1 Learning From Contact Distributions

Manipulation tasks almost always involve direct physical contact between two or more objects. These contacts can be between different objects in the robot's environment, or between an object and the robot. Depending on the locations of the contacts, different types of interactions and manipulations can occur. For example, a contact on the side of an object may allow for pushing and sliding the object, while a contact on the bottom can be used for lifting or supporting the object. In order to successfully perform a manipulation task, a robot must be able to determine the potential interactions between objects and utilize them to accomplish the task's goal.

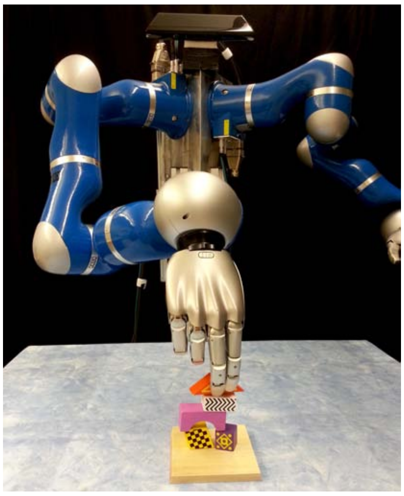


Figure 3.1: The Darias robot performing a block stacking task. The robot learns suitable block placements using a kernel function for comparing contact distributions.

Utilizing contact information in an efficient manner is however not a trivial task. Analytical approaches tend to require accurate models of the objects, and rely on simplified contact models [2]. In an effort to make robots more autonomous, learning approaches have become more widely adopted in the field of robot manipulation [77, 78, 79]. However, representing contacts between objects often relies on hand-crafted features for the given task.

In this chapter, we propose an example-based learning approach to detect interactions between objects from their contact distributions. We pose the problem of detecting interactions as a binary classification problem, wherein the robot has to predict whether or not a certain interaction is occurring based on the geometry and relative poses of the objects. The robot first computes which regions of the objects are in contact with each other. The resulting cloud of contact points is subsequently modeled as a Gaussian distribution. A Bhattacharyya kernel function [80] can then be used to compute the similarities between the contact distributions and, thus, classify them using kernel logistic regression. In this manner, the robot uses the similarity between the current contact distribution and previous distributions in order to classify the potential interaction. The details of the approach are explained in Section 3.2.

Classifying interactions between objects is closely related to learning affordances [81]. If an object allows a robot to perform an action with it, than the object is said to "afford" that action. Affordances have been widely studied in robotics [82, 83, 84], and especially in the field of robot grasp synthesis [2]. Recently, several papers have proposed template-based approaches for detecting where an object can be grasped [85, 86, 87, 88, 76]. These approaches predict where to grasp an object based on the local shape of the object relative to the hand.

Learning symbolic representations of geometric relations between objects, e.g. object A is ON object B, is an important skill for performing complex manipulation tasks. Rosman and Ramamoorthy [89] proposed the use of a contact network to learn the spatial relations between objects. Contact points were detected using a support vector machine to separate the point clouds of the objects. The vectors between the objects' contact points were then computed and used to classify relations such as *on* and *adjacent* using a k-nearest neighbors classifier. Kulick et al. use an active learning approach to efficiently learn a symbolic representation of the relations between objects [90]. Using features such as the heights of objects and the relative positions between objects, they train a Gaussian process classifier to learn in which geometric states the predicate is true.

Contact information can also be represented in the form of tactile sensor readings [91, 92]. Bekiroglu et al. [91] proposed learning to predict stable grasps of objects using kernel logistic regression. Their approach used a product of three separate kernels based on the position of the hand relative to the object, the approach direction of the hand, and moment features of the tactile sensor arrays' readings. In the work of Dang et al. [92], the locations of the sensed contact points are defined relative to the palm, and modeled using a bag-of-words representation. A support vector machine is then trained to classify stable and unstable grasps.

The features used by learning algorithms can also be designed to capture specific aspects of the contacts between objects [79, 77, 93]. In [79], a classifier was trained on simulated data to predict interactions, such as support and location control, between pairs of objects. The classifier was provided with 93 features, such as the total contact patch area, and the vector between the closest contact point and the other object. Automatic relevance determination was then used to effectively select a subset of these features. Jiang et al. [77] addressed the problem of learning to place objects in a scene. The placement of an object was represented by a set of 145 features, including features for modeling supporting contacts and the caging of objects. A support vector machine with a shared sparsity structure was then used to classify good and bad placements of objects.

The proposed kernel approach was implemented on the robot shown in Fig. 3.1. In the first experiment, the robot was given the task of predicting which grasps allow it to steadily pick up an elongated object. The second experiment required the robot to stack assorted blocks. The details of the experiments are given in Section 3.3.

3.2 Computing a Kernel for Contact Distributions

In Sections 3.2.1 to 3.2.3, we explain how contacts between objects are detected and used to create contact distributions. In Sections 3.2.4 and 3.2.8, we provide a kernel function for computing the similarity between contact distributions and explain how it is used to classify the distributions using kernel logistic regression.

3.2.1 Contact Points

In order to determine the contacts between objects, we first need a suitable representation of the object and its geometry. Given an object O_i, where i specifies the index of the object, we define its geometry as a point cloud with n_i points at positions $\mathbf{p}_{ij}$ and corresponding normals $\mathbf{u}_{ij}$ for $j \in \{1, \ldots, n_i\}$. Point clouds are flexible object representations that are widely used in robotics [94]. The normals of the points are straightforward to compute using the covariance of nearby points and the viewing direction.

The point cloud defines the surface of the object and, hence, also where contacts can potentially be made with another object. In order to obtain a set of contact points, each point in the point cloud is classified as either being in contact with the other object or not. In our experiments, we used logistic regression to classify the points, although other methods for detecting contacts are also applicable. The probability of a point $\mathbf{p}_{ic}$ being in contact $\mathscr{C}_{icj}$ with the object O_j is given by

$$p(\mathscr{C}_{icj}|\mathbf{p}_{ic}, \mathbf{u}_{ic}, O_j) = \left(1 + \exp\left(\boldsymbol{\phi}^T \boldsymbol{\rho}\right)\right)^{-1},$$

where $\boldsymbol{\phi}$ is a vector of feature functions and $\boldsymbol{\rho}$ is a vector of corresponding weights. We used three features, including a density estimation

$$\phi_1(\mathbf{p}_{ic}, O_j) = \sum_k \exp\left(-\frac{\left\|\mathbf{p}_{ic} - \mathbf{p}_{jk}\right\|^2}{\sigma^2}\right)$$

and a surface normal density estimation

$$\phi_2(\mathbf{p}_{ic}, \mathbf{u}_{ic}, O_j) = \sum_k (\mathbf{u}_{ic}^T \mathbf{u}_{jk}) \exp\left(-\frac{\left\|\mathbf{p}_{ic} - \mathbf{p}_{jk}\right\|^2}{\sigma^2}\right)$$

where σ is the length scale of the density. We also include a bias term $\phi_3 = 1$.

These three features are well-suited for detecting arbitrary contacts between two objects. Some interactions however require specific types of contacts, e.g., cutting requires contact with a sharp edge. The set of features can be easily extended for more specific types of contacts.

Computing a set of weights $\boldsymbol{\rho}$ that maximizes the likelihood of the training data is a convex optimization problem, and can be solved using iterative reweighted least squares, as explained in [47]. A point is classified as a contact point if the probability of contact is greater than 0.5.

3.2.2 Object Centers

In addition to the shape of the object, we also define a set of *object centers* for each object. Object centers are used to define interaction-relevant coordinate frames for the object. Each center $\mathbf{c}_{ik}$, where k is the index of the center for object O_i, is associated with a position $\mathbf{x}_{ik}$ and at least one axis $\mathbf{a}_{ik}$. For example, the position of an object's center of gravity is given by the mean point of its mass, and an axis pointing down in the direction of gravity. For an articulated object, such as a hand winch or door handle, the position and axis of rotation of the revolute joint defines another center. Although an object may have many centers, usually only one center is used for predicting an interaction. In this thesis, we only consider a single object center $\mathbf{c}_i$, and leave automatically selecting the relevant center to future work.

Once the contact points have been found, they need to be defined with respect to the center's coordinate frame. If the axes of the center already define three orthogonal axes $\mathbf{a}_i^x$, $\mathbf{a}_i^y$, and $\mathbf{a}_i^z$, then this step is trivial. However, the center of gravity or the center of a revolute joint only define a single axis $\mathbf{a}_i^x$ and not a full 3D coordinate frame. In order to define the other two axes, we first project the contact points into a 2D plane, with the normal of the plane given by the first axis of the center $\mathbf{a}_i^x$. We then compute the matrix of second moments about the center position for the contact points, and subsequently compute the eigenvectors of the matrix. The second axis $\mathbf{a}_i^y$ is defined by the eigenvector with the largest eigenvalue, such that the mean of the contact points is in the positive direction. Using this approach, the contact point clouds are aligned according to the radial direction with the largest variance. The third axis is simply given by the cross product of the first two $\mathbf{a}_i^z = \mathbf{a}_i^x \times \mathbf{a}_i^y$.

The positions of the $\tilde{n}_i$ contact points in the object center's coordinate frame are denoted as $\tilde{\mathbf{p}}_{ij}$ with corresponding normals $\tilde{\mathbf{u}}_{ij}$ for $j \in \{1, \ldots, \tilde{n}_i\}$.

3.2.3 Computing Contact Distributions

Having computed a set of contact points, we now want to compare this set of contacts to previously observed ones. Rather than comparing points individually, we first model the set of contact points as a distribution. In particular, we model them as a 6D Gaussian distribution, where the first three dimensions correspond to the positions of points, and the last three model the normals. In the lifting experiment in Section 3.3, we also investigate replacing the normals of each point with an estimate of the force. However, the forces are in most cases not known, especially when the interaction is between two objects and not with the robot.

Given a set of contacts, we now define a distribution over contact points as a Gaussian distribution. The mean vector $\boldsymbol{\mu}_i$ and variance $\boldsymbol{\Sigma}_i$ of the distribution are given as

$$\boldsymbol{\mu}_i = \frac{1}{\tilde{n}_i} \sum_{k=1}^{\tilde{n}_i} \begin{bmatrix} \tilde{\mathbf{p}}_{ik} \\ \tilde{\mathbf{u}}_{ik} \end{bmatrix},$$

$$\boldsymbol{\Sigma}_i = \frac{1}{\tilde{n}_i} \sum_{k=1}^{\tilde{n}_i} \left(\begin{bmatrix} \tilde{\mathbf{p}}_{ik} \\ \tilde{\mathbf{u}}_{ik} \end{bmatrix} - \boldsymbol{\mu}_i \right) \left(\begin{bmatrix} \tilde{\mathbf{p}}_{ik} \\ \tilde{\mathbf{u}}_{ik} \end{bmatrix} - \boldsymbol{\mu}_i \right)^T.$$

Near-singular covariance matrices can lead to numerical issues, which can be alleviated by enforcing a lower limit on the eigenvalues, or by including additional noise as described in Section 3.2.7. The Gaussian model provides a compact representation of the mean contact position and normal orientation, as well as the correlations between the parameters around the mean.

3.2.4 Kernel Between Contact Distributions

Having converted the contact points into a contact distribution, we can now use a kernel to compute the similarity between distributions. We use the Bhattacharyya kernel [80] which is given by

$$k((\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i), (\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)) = \int \sqrt{\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)} \sqrt{\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)} d\mathbf{x}.$$

The computation of the kernel is given in [95], and we include it again here for completeness. The kernel function is computed as

$$k((\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i), (\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)) = C \exp(-M/4),$$

where the values of C and M are given by

$$C = 0.5^{-d/2} |\hat{\boldsymbol{\Sigma}}|^{1/2} |\boldsymbol{\Sigma}_i|^{-1/4} |\boldsymbol{\Sigma}_j|^{-1/4},$$

$$M = \boldsymbol{\mu}_i^T \boldsymbol{\Sigma}_i^{-1} \boldsymbol{\mu}_i + \boldsymbol{\mu}_j^T \boldsymbol{\Sigma}_j^{-1} \boldsymbol{\mu}_j - \hat{\boldsymbol{\mu}}^T \hat{\boldsymbol{\Sigma}} \hat{\boldsymbol{\mu}}.$$

The vector $\hat{\boldsymbol{\mu}}$ is given by $\hat{\boldsymbol{\mu}} = \boldsymbol{\Sigma}_i^{-1} \boldsymbol{\mu}_i + \boldsymbol{\Sigma}_j^{-1} \boldsymbol{\mu}_j$, and the matrix $\hat{\boldsymbol{\Sigma}}$ is computed as $\hat{\boldsymbol{\Sigma}} = (\boldsymbol{\Sigma}_i^{-1} + \boldsymbol{\Sigma}_j^{-1})^{-1}$. The parameter $d = 6$ is the dimensionality of the Gaussians. The kernel function computes a value from zero to one, where a value of one is achieved if the contact distributions are identical. As the overlap between the distributions decreases, the kernel function tends to zero.

3.2.5 Extension to Multiple Gaussians

Although we focus on representing contact distributions using single Gaussians, the proposed framework is straightforward to extend to multiple Gaussians. By representing the contact distribution as a mixture of Gaussians, the model can capture more details of the distribution. The resulting kernel can therefore distinguish between different contact distributions more easily.

However, the Bhattacharyya kernel is not suitable for comparing Gaussian mixture models. Instead, given that the contact distribution of object O_i has the form

$$f_i(\mathbf{x}) = \sum_{h=1}^{H_i} \nu_{ih} \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_{ih}, \boldsymbol{\Sigma}_{ih}),$$

where ν_i are the mixture components of the H_i Gaussians, one can compute the kernel function

$$k(f_i(\mathbf{x}), f_j(\mathbf{x})) = \frac{\int f_i(\mathbf{x}) f_j(\mathbf{x}) \mathrm{d}x}{\sqrt{\int f_i(\mathbf{x}) f_i(\mathbf{x}) \mathrm{d}x} \sqrt{\int f_j(\mathbf{x}) f_j(\mathbf{x}) \mathrm{d}x}},$$

in closed-form. This kernel function also has a value of 1 when the contact distributions are the same, and tends to zero as the overlap decreases. The kernel is based on the expected likelihood kernel [95] and is closely related to the Cauchy-Schwarz divergence [96]. Although multiple Gaussians can model the contact distribution more precisely, this level of detail is often not needed when learning robust manipulation skills [97].

3.2.6 Shape Kernel

The proposed approach can also be used to compare the shapes of object parts [76]. In this case, rather than selecting points that are in contact with another object, the robot selects all of the points within a certain region relative to the object center. Alternatively, the robot can use all of the points in the object's point cloud and weight them according to their location. For example, the robot can use a Gaussian weighting function centered on the object center in order to give the points nearer to the center more importance.

Shape kernels are more useful for comparing parts of objects rather than the shapes of whole objects. In these cases, the object center defines a coordinate frame associated to that part of the object. For example, the object center could define a grasp frame on a handle part. A motor primitive, as described in Chapter 2, can then be used to define a grasping trajectory for moving the hand relative to this grasp frame. Using this approach, the robot can learn to detect parts of objects that afford certain actions based on their shapes [76].

3.2.7 Interaction-Specific Contact Similarity

Although the contact distribution is defined in a 6D space, not all of the dimensions will be equally relevant for predicting a given interaction. For example, when pushing open a door, the horizontal distance from the axis of rotation is more relevant than the vertical position along the axis. As a result, two contacts are more similar if they are offset vertically rather than horizontally from each other.

We can model this additional similarity by adding interaction-specific Gaussian noise $\mathcal{N}(\mathbf{0}, \tilde{\boldsymbol{\Sigma}})$ to the contact points. Thus, each contact point is represented as a Gaussian distribution $\mathcal{N}([\ \tilde{\mathbf{p}}_{ik}^T \ \ \tilde{\mathbf{u}}_{ik}^T\ ]^T, \tilde{\boldsymbol{\Sigma}})$ instead of just a single point. If the offset between two contact points corresponds to a direction with a larger variance, then their distributions will overlap more and they will be considered as more similar. In practice, the interaction-specific covariance matrix $\tilde{\boldsymbol{\Sigma}}$ is added to the standard covariance matrices $\boldsymbol{\Sigma}_i$ and $\boldsymbol{\Sigma}_j$ before computing the kernel value. The experiment in Section 3.3.2 shows that the robot can use this additional similarity information to increase the sample efficiency of the learning algorithm.

3.2.8 Classifying Contact Distributions

Having defined a kernel between contact distributions, we can now use a wide range of kernel methods from machine learning [98]. In order to classify a contact distribution, we use kernel logistic regression. Kernel logistic regression uses the similarity to previously observed distributions, with known labels, to classify new contact distributions. The probability that a contact distribution $\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$ allows for a certain interaction $\mathcal{I}$ is given by

$$p(\mathcal{I}|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) = (1 + \exp(\alpha))^{-1},$$

where

$$\alpha = \theta_0 + \sum_{j=1}^{m} \theta_j k((\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i), (\boldsymbol{\mu}'_j, \boldsymbol{\Sigma}'_j)),$$

and we have m previous examples of contact distributions $\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}'_j, \boldsymbol{\Sigma}'_j)$. The weight parameters θ can be learned using iterative reweighted least squares. Contact distributions that are not similar to any previous distributions will have a probability defined by θ_0. As kernel logistic regression is a probabilistic classifier, it can model a contact distribution that only sometimes allows for the interaction. Previous contact distributions that allowed for the interaction will generally have more negative weights, which will result in a probability closer to one.

3.3 Experiments

The proposed approach was implemented on a real robot, as shown in Fig. 3.1. The robot consists of two Kuka lightweight robot arms, each equipped with a DLR five-fingered hand [99], and a Microsoft Kinect. The robot was evaluated on two tasks: picking up an elongated object, and stacking assorted toy blocks.

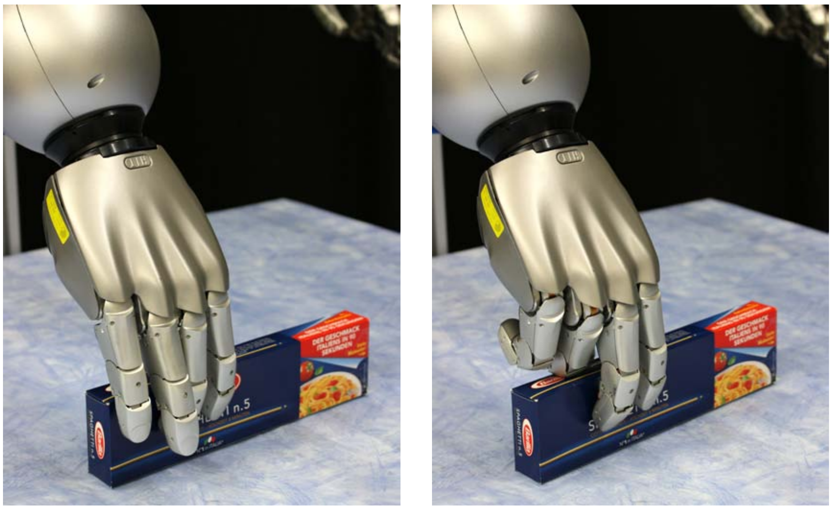
3.3.1 Picking up Elongated Objects

In the first experiment, we applied the framework to the problem of predicting whether a given grasp allows an elongated object to be steadily lifted.

Experimental Setup

The robot performed 60 randomly selected grasps along the length of a spaghetti box. The first half of the grasps were performed with a three-fingered grasp and the other 30 were executed with a four-fingered grasp, as shown in Fig. 3.2. The robot subsequently tried to lift the box 13 cm above the table. The picking up of the box was considered successful if the object was no longer in contact with the table, and a failure otherwise, as shown in Fig. 3.3. Before lifting the box, the robot recorded the state of the scene and computed the contact distribution. Based on this information, the robot had to predict whether or not the lift would be successful. In order to detect contact points, we labeled ten points in one scene to train the contact classifier. The contact distribution is defined relative to the center of gravity. Although the box has a simple shape, which affords a continuous range of grasps along its length, it is not trivial to predict stable lifts for this object. The robot's hand has to match the shape of the object such that it can conteract the torque caused by gravity to achieve a stable lift. The magnitude of the torque depends on where the object is grasped relative to its center of mass.

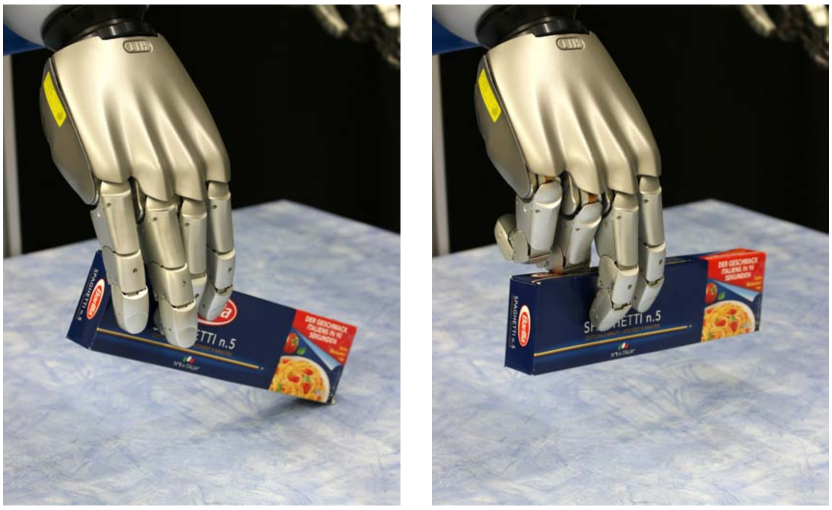
In addition to evaluating the method explained in Section 3.2, referred to here as NORMAL+POS, we also evaluated several benchmark approaches. The first benchmark approach, MEANONLY, performs the



3-Fingered Grasp

4-Fingered Grasp

Figure 3.2: The two types of grasps that were used during the lifting experiment. The three-fingered grasp uses the tips of the thumb, middle, and index fingers in order to pinch the object. The ring and little finger are not touching the box. The four-fingered grasp additionally uses the back of the ring finger on the top of the box in order to provide additional support.



Failed Lift

Successful Lift

Figure 3.3: Examples of failed and successful lifts. A lift was considered a failure if the object was still touching the table at the end of the trial.

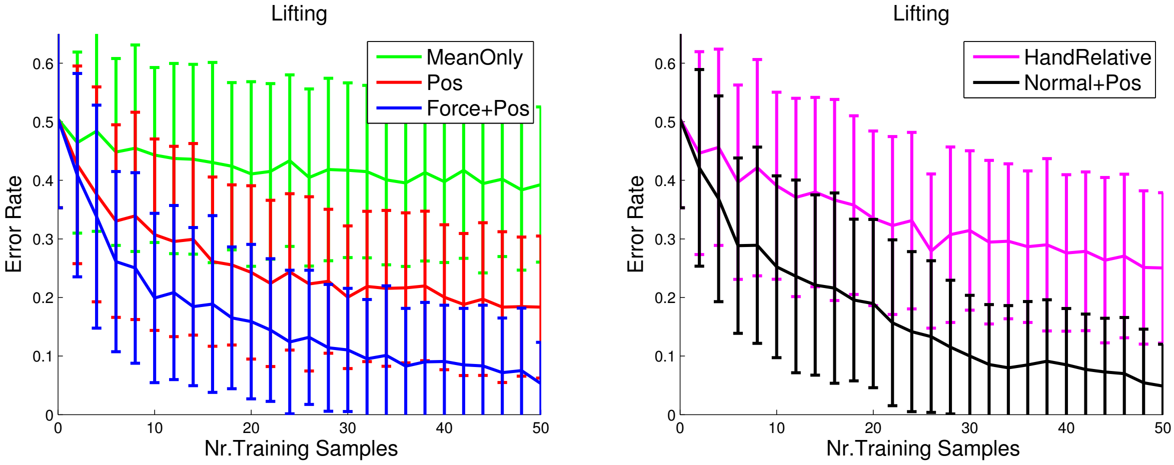


Figure 3.4: The expected error rates for the lifting task. The error bars indicate one standard deviation. An error rate of 1 indicates that none of the test samples were correctly classified, and an error rate of 0 is achieved when the classifier evaluates all of the samples correctly.

classification using only the mean contact $\boldsymbol{\mu}_i$. The POS approach uses only the position distribution of the contact points and not the normals. As a result, the contact distribution is only 3D. Although the fingers do not have tactile sensors, forces can be roughly approximated using the joint torque sensors of the fingers and the relative positions of the contact points. The FORCE+POS approach is the same as NORMAL+POS, except that the normals $\mathbf{u}_i$ have been replaced by force estimates. The final method HANDRELATIVE uses the positions and estimated forces of the contact points, but defines the contact distribution relative to the hand rather than the object center.

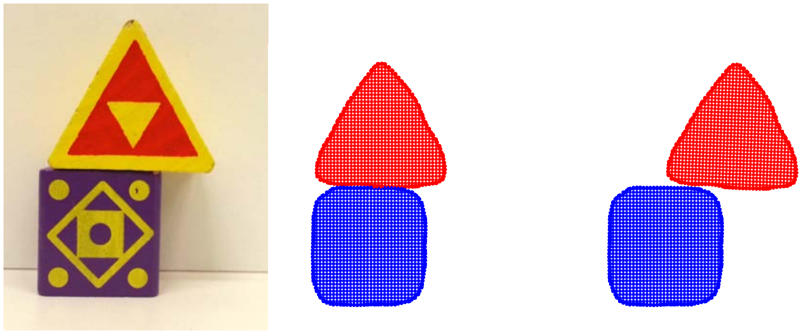
The performance of the various methods were tested for different numbers for training samples. In each evaluation, ten grasps were selected as test samples. From the remaining grasp samples, a subset of samples were selected as training data. The classifier was then trained on the training data and used to classify the test samples. The error rate is given by the percentage of correctly classified grasps in the test set. This process was repeated 250 times for each classifier and each number of training samples. The results of the evaluation are shown in Fig. 3.4 .

Discussion

Using only the mean contact or the distribution relative to the hand resulted in poor performance. The task was especially challenging for the HANDRELATIVE approach, as the object has the same shape along its length. Despite this challenge, the approach still obtained an error rate of 25.04%.

Using only the position of the contact points relative to the object center resulted in an error rate of 18.36%, which is only marginally better than the performance of HANDRELATIVE. In comparison, the NORMAL+POS and the FORCE+POS achieved error rates of 4.88% and 5.28% respectively. The contact normals clearly capture a considerable amount of information, as they allow side contacts to be differentiated from top contacts.

Both NORMAL+POS and FORCE+POS performed well on the task, and learned to accurately predict steady lifts. However, both approaches also have their limitations. The NORMAL+POS approach cannot differentiate between the robot gently placing its fingers on the box and the fingers applying forces at the contacts. This approach can therefore sometimes only predict whether an interaction is possible, given the contacts, but not if the interaction is being performed. The FORCE+POS approach can differentiate between these two scenarios, and using it together with tactile sensing is a promising direction for future research. However, as the forces between objects will often not be directly observed, the NORMAL+POS approach is generally more applicable.



Positive Example Negative Example

Figure 3.5: Point cloud examples of a stable and an unstable stacking of blocks

3.3.2 Stacking Objects

In the second experiment, the robot was given the task of classifying whether one object was supporting another. The robot then used the trained classifier to stack assorted toy blocks.

Classifying Stable Block Placements

The robot was provided with 60 example scenes, each containing two interacting toy blocks, such as the ones shown in Fig. 3.5 . For the 30 negative examples, physically impossible static scenes were created by hand. The models of the blocks were acquired using a turn table setup and a Kinect. The object center is again defined by the center of gravity. To train the contact point classifier, ten points were hand labelled in one scene. The points of the object were classified as contacts based on the features described in Section 3.2.1. Using additional features, such as the position and orientation of the points relative to the object's center, were also tested, but had no significant effects on the outcome of the experiment.

The performance of the contact point classifier was evaluated in the same manner as for the previous experiments. A set of ten test samples were randomly selected and removed from the pool of 60 samples. A subset of the remaining samples were then used to train the classifier. The classifier was subsequently applied to the ten test samples, and the error rate was recorded. The error rate is 1 if all ten samples were incorrectly classified, and 0 if all of them were correctly classified. The test samples were subsequently put back into the pool of samples. This process was repeated 250 times for each number of training samples.

In addition to the standard approach, we also evaluated adding an interaction-specific covariance matrix $\tilde{\Sigma}$, as explained in Section 3.2.7. The elements of the diagonal matrix were recomputed for each trial using a basic hill-climbing approach to minimize the leave-one-out cross-validation error rate on the training set.

The results of this experiment are shown in Fig. 3.6 . Starting with error rates close to 50%, the classifiers' performances gradually improves as more samples are provided. Given 50 samples, the standard classifier achieved an expected error rate of 5.0%, and could accurately predict when the object was being supported. Using the additional interaction-specific covariance matrix, the classifier achieved an expected error rate of 0.4% for 50 samples, and only required 20 samples to achieve an expected error rate of 3.84%. The sample efficiency of the algorithm can therefore be increased by incorporating the interaction-specific covariance. In many of the trials, the covariance matrix $\tilde{\Sigma}$ indicated that the vertical position of the supporting contacts was less relevant than the horizontal position. The experiment demonstrates the classifier's ability to generalize between different object shapes.

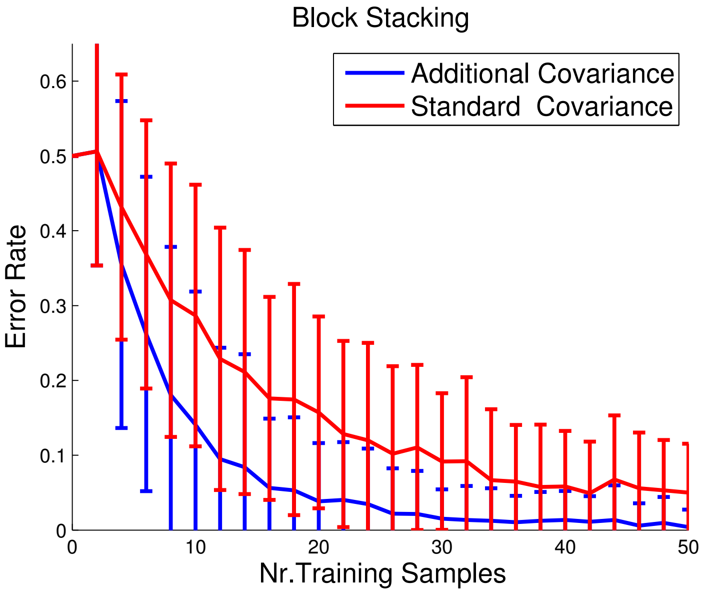


Figure 3.6: The expected error rate for the block stacking task. The red line indicates the performance when using the standard covariance matrix. The blue line shows the performance when adding the interaction-specific covariance matrix. The error bars indicate one standard deviation.

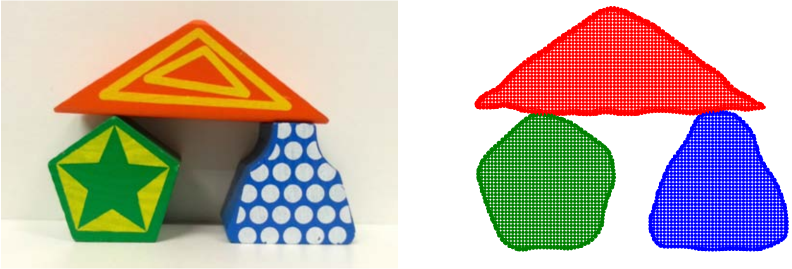


Figure 3.7: An example scene with three objects, wherein the green and blue objects are supporting the triangular red block.

Generalization to Multiple Objects

In order to demonstrate the classifier's ability to generalize to multiple objects, it was applied to the scene of three objects shown in Fig. 3.7. In this scene, the top object is being supported by both of the lower objects. When the classifier is applied to the top block and only one of the bottom blocks, the interaction is classified as not supporting. However, we can also combine the blue and green point clouds of the bottom objects in order to create one compound object. When applying the classifier to the top object and this compound object, the top object is labeled as being supported by the bottom object. Thus, as one would expect, the classifier detects that the top is being supported by both objects jointly, and by neither one separately. The classifier was tested on two more similar scenes of three blocks, with the same results.

Building Block Towers

In the final part of the experiment, the real robot used the classifier from the first part to perform block stacking. The interaction-specific covariance matrix was not used in this experiment. The robot was

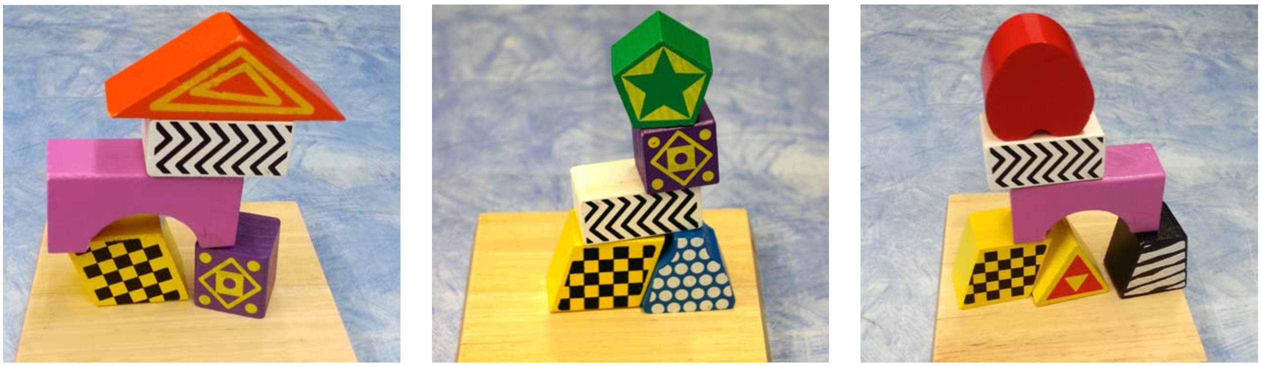


Figure 3.8: examples of block towers constructed by the robot.

provided with a small wooden board, on which to stack the blocks. In order to avoid all of the blocks being placed directly on the board, the placing of the blocks was limited to a single strip along the middle of the board. For every block, the robot observed the current scene using the Kinect and used the resulting point cloud as the supporting object in the interaction. As the focus is not on the planning aspects of the problem, the sequence of blocks was predefined.

In order to determine a suitable placement for the current block, the robot sampled different positions in the scene. For each sample, the contact points were estimated and the probability of the block being supported was computed. The robot then attempted to place the block at the position with the highest probability.

Randomly sampling positions in the scene led to poor performance. One of the main challenges for the robot was the noisy partial point cloud of the current scene. The kinect usually only captured the top and front of the current block stack, but not the back or sides. The lack of reliable points on the sides of objects resulted in unforeseen collisions between blocks. This problem could be alleviated by obtaining more views of the scene, completing the point cloud based on symmetries [75, 100], or applying a penalty for placing the block into occluded regions.

In order to reduce the number of accidental collisions, we also implemented a sampling approach that mimics the movement of the block when it is being put down. The robot sampled 20 horizontal positions at 7.5mm increments across the width of the board. For each horizontal position, the robot sampled vertical placements at 5mm increments in a top-down manner until contact was detected between the block and the stack.

In order to evaluate the proposed approach, the robot was given the task of creating five towers consisting of five blocks each. Using the improved sampling approach, the robot successfully placed 96% of the blocks without knocking any blocks down. Only one block was misplaced by a few millimeters and fell down. The robustness of the system could be further improved by also considering the probability of success of neighboring positions [101].

The robot currently ignores the interactions between blocks further down in the stack. As a result the robot may select a block placement that causes a supporting block to fall down. One potential solution to this problem would be to recheck the interactions between objects further down the stack. For each interaction, the objects higher up in the stack would then be treated as a single compound object, with a corresponding object center. This approach would however require the robot to keep a model of the current scene's geometry.

The results of the experiment show that the robot was able to construct multiple block towers, such as the ones shown in Fig. 3.8 ,using the proposed approach.

3.4 Generalizing Between Objects with Warped Parameters

As objects of the same type may have different shapes and sizes, the robot will have to adapt its actions to the geometry of the specific object that it is manipulating. The shape of objects is particularly important when manipulating liquids, e.g., pouring a glass of water, as liquids conform to the shape of their container. The robot must therefore take into consideration a container's geometry when using it in a pouring task.

Although containers come in a wide variety of shapes and sizes, the important differences can usually be defined by a few geometric parameters [102, 103]. For example, the volume of a container indicates how much fluid it can hold, regardless of whether it has a spherical, or cylindrical shape. A robot can generalize pouring actions between different containers by using these geometric parameters. However, the robot will not be provided with the geometric parameters for most of the novel objects that it encounters. While a human may annotate the geometric information for a couple of objects, the robot will usually need to compute these parameters on its own.

In the remainder of this chapter, we investigate using warped parameters to generalize pouring skills between different objects. A warped parameter is defined as a function on the points of a known object's point cloud. For example, a warped parameter may compute the volume of a set of points' convex hull. When the robot encounters a novel object, it warps the point cloud of the known object to the new object's shape. As a result of the warping, the value of the warped parameter changes to match the geometry of the new object. Once the geometric parameters have been computed, the robot can use them to generalize actions and task constraints between different objects.

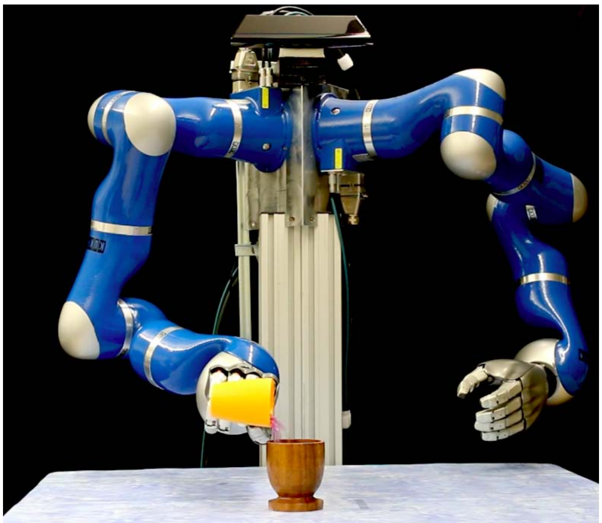


Figure 3.9: The robot performs a pouring task with two previously unknown objects. The pouring action was learned from human demonstrations using a taller cup and a wider container to pour into.

Several previous works have used warping to generalize manipulations between objects. Hillenbrand et al. [5, 104] used warping to map contact points onto novel objects, in order to transfer grasps between objects. A similar approach was used by Rainer et. al [105, 106] for transferring coordinate frames of task constraints between objects. However, the size and shape of the constraint regions were not adapted to the new object's geometry. Rather than warping only the points on the object, Schulman et al. [6] computed a warping functions for the entire scene. The warping was then applied to the demonstrated trajectory of the source scene in order to obtain a trajectory for the current scene. These approaches focus on mapping specific points from the source scene to the target scene, and are therefore especially well-suited for contact-based manipulations. Warped parameters can be used to model more abstract features of the objects, such as areas and volumes.

Several methods have also been proposed for learning to perform pouring tasks. Pastor et al. [107] learned dynamic motor primitives (DMPs) for pouring from human demonstrations, and used these to generalize to different cup placements. Similarly, Muehlig et al. [108] encoded demonstrated bimanual pouring trajectories using Gaussian mixture models. Rozo et al. [109] proposed learning a controller for pouring tasks based on the observed forces. The work on learning pouring from demonstration has mainly focused on learning with the same set of objects. In comparison, we propose learning in a feature space defined by the warped parameters, in order to automatically generalize between objects.

Some work has also been done on generalizing pouring actions between different objects using reinforcement learning. Kroemer et al. [76] learned a pouring DMP from human demonstrations, and then used a trial-and-error approach to learn the location of a novel container's opening. The opening was detected using a shape-similarity kernel. Tamosiunaite et al. [110] used reinforcement learning to learn the shape of the pouring DMP, as well as the goal point. Trial-and-error learning was also used to adapt the learned motion to novel objects, without explicitly considering the differences in geometry.

In Section 3.5, we explain the process of computing the warped parameters. In Section 3.6, we describe how the robot can learn pouring actions and task constraints that generalize between objects using the warped parameters. The proposed method was successfully evaluated both in simulation and on the robot shown in Fig. 3.9. The results of the experiments are detailed in Section 3.7.

3.5 Computing Warped Parameters

In this section, we describe how a robot can compute geometric parameters of an object by warping a known object to match its shape. The object models and the warping process used in this section are described in Sections 3.5.1 to 3.5.3. The computation of the warped parameters for pouring tasks is described in Section 3.5.4.

3.5.1 Geometric Object Models

In order to generalize manipulations to a novel object, the robot first computes correspondences between a known source object O_s and the unknown target object O_t. An object O_i is modeled as a set of c_i points located at positions $\mathbf{p}_{ij} \in \mathbb{R}^3$ with corresponding normals $\mathbf{n}_{ij} \in \mathbb{R}^3$, where $j \in \{1, ..., c_i\}$.

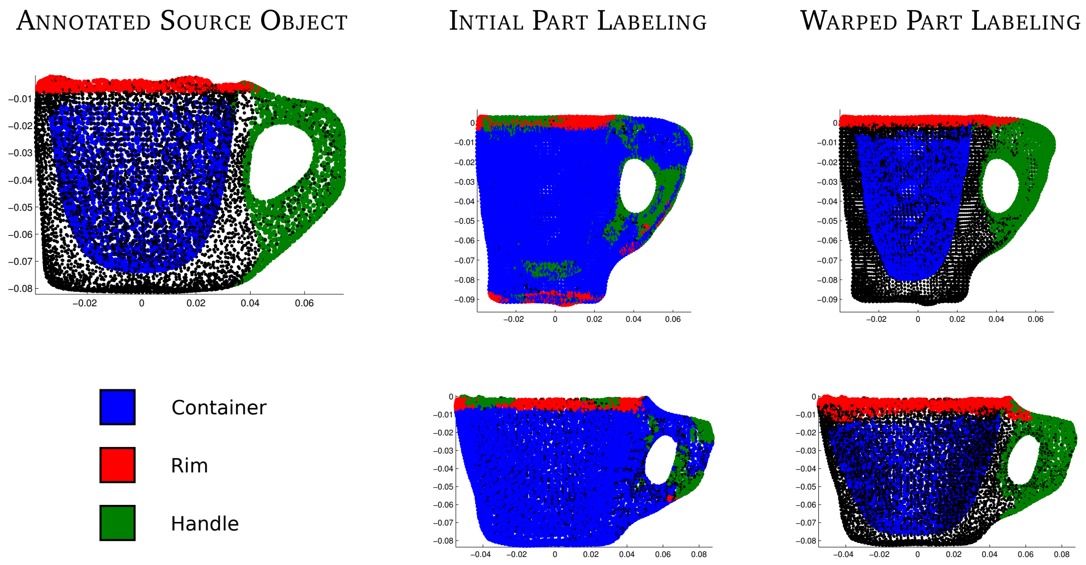
Objects often consist of multiple parts, and a manipulation may only depend on the shape of a part of an object. Hence, geometric parameters often describe the shape of a part rather than the whole object. We therefore also assign each point $\mathbf{p}_{ij}$ a vector $\mathbf{l}_{ij}$ of length ρ with binary labels, which indicate which of the ρ object parts the point corresponds to. The labels of the target object O_t are initially unknown, but can be computed using the warping process.

An example of an annotated cup can be seen in Fig.3.10. The first part is the CONTAINER, which holds the liquids. The second part is the RIM around the opening. We also label the HANDLE as a *dummy* part. As not all containers have handles, it is not used to define any warped parameters for the pouring task, and is only included to help align objects during the warping process.

3.5.2 Warping

Given a source object and a target object, the robot can compute correspondences between the two objects. These correspondences are determined by warping the shape of the source object onto that of the target object. There are various methods for computing 3D warpings between object [111, 112], and the proposed approach does not depend on a specific warping algorithm. We therefore employ a basic warping algorithm for finding correspondences between the containers. The warping process consists of two stages: 1) object alignment, and 2) point mapping

In the first stage, the source object is coarsely aligned with the target object, such that their corresponding object parts are close together. This alignment is accomplished by computing a coordinate system based on the objects' parts. The origin of the coordinate frame is the mean of the container points. The first axis is given by the direction to the mean of the rim points, and the second axis is the orthogonal direction to the mean of the handle points. The third axis is computed by the cross product of the first two axes. As the part labels of the target object $\mathbf{l}_t$ are unknown, an initial estimate of the labels is computed using logistic regression. One classifier is trained for each of the three object parts. Each point $\mathbf{p}_{ti}$ is classified based on the local distribution of points in its neighborhood. The features



Annotated Source Object

Intial Part Labeling

Warped Part Labeling

Figure 3.10: The *left* column shows the point cloud of the source object, annotated by a human user. The *middle* column shows the point clouds of two target objects. The points were labelled using a classifier based on local features. This intial estimate is only used to compute a coarse alignment with the source object. The point clouds were pre-aligned for this figure to show more clearly how the labels change during the warping process. The *right* column shows the final results of the label mapping approach.

used to describe the local distribution of points include the eigenvalues of the covariance matrix, and the distance from the point $\mathbf{p}_{ti}$ to the mean of the neighborhood points. The classifiers were trained on the labelled points of the source object. An example of the initial labeling can be seen in Fig. 3.10 . The coordinate frame of the object is estimated using this initial labeling of points. Once the two objects are aligned, the source object was scaled in each direction such that the variances of its container part matched those of the target object. We denote the aligned source objects and target objects by $\tilde{O}_s$ and $\tilde{O}_t$ respectively.

In the second stage of the warping algorithm, the points from the source object $\tilde{O}_s$ are mapped onto the target object $\tilde{O}_t$. This step is similar to the approach proposed by Hillenbrand [113]. Each point of the aligned source object is mapped to the mean of the k nearest neighbors in the aligned target object. In our experiments, we set $k = 1$. Hence, the warped source point $\mathbf{p}_{wi}$, with corresponding normal $\mathbf{n}_{wi}$ and labels $\mathbf{l}_{wi}$, is given by

$$\mathbf{p}_{wi} = \mathbf{p}_{tj}\ ,\ \mathbf{n}_{wi} = \mathbf{n}_{tj}\ ,\ \text{and } \mathbf{l}_{wi} = \mathbf{l}_{si},$$

$$\text{s.t. } j = \arg\min \left\|\tilde{\mathbf{p}}_{si} - \tilde{\mathbf{p}}_{tj}\right\| \text{ and } \tilde{\mathbf{n}}_{si}^T \tilde{\mathbf{n}}_{tj} > 0.$$

Thus, each source point is mapped to the closest target point with a normal pointing in the same direction. The warped object and its point cloud are denoted by O_w.

3.5.3 Point Mapping vs. Label Mapping

The warping process defines a new position and normal for each of the c_s point of the source object O_s. The location of these new points can be used to define warped parameters, as detailed in the next section. We refer to this approach as *point mapping*, as the points of the source object are mapped onto the target object.

However, if the source object has considerably fewer points than the target object, then some details of the target object may not be captured by the warped object. This issue can be addressed by warping the target object to match the source object. The alignment and scaling of the objects is performed as before. However, the points of the target object are mapped onto the source object. The label of each of the target points is then determined using a k-nearest neighbors classifier. In our experiments, we again used $k = 1$, such that

$$\mathbf{p}_{wi} = \mathbf{p}_{ti}\ ,\ \mathbf{n}_{wi} = \mathbf{n}_{ti}\ ,\ \text{and}\ \mathbf{l}_{wi} = \mathbf{l}_{sj},$$

$$\text{s.t. } j = \arg\min \left\| \tilde{\mathbf{p}}_{si} - \tilde{\mathbf{p}}_{tj} \right\| \text{ and } \tilde{\mathbf{n}}_{si}^{T} \tilde{\mathbf{n}}_{tj} > 0.$$

We refer to this approach as label mapping, as the labels of the source object are mapped onto the target object. When using multiple neighbors $k > 1$, the point is assigned to a part if the majority of its k neighbors belong to that part.

The benefit of using the label mapping approach is that it guarantees that all of the points of the target object are used for computing the warped parameters. However, when using label mapping, points can only be referred to by their label and not as individual points. In comparison, when using point mapping, one can refer to individual points, e.g., $\mathbf{p}_{w72}$, which correspond to specific points on the source object. The right column of Fig. 3.10 shows an example of using label mapping.

3.5.4 Warped Parameters

Having computed the correspondences between the known source object and the novel target object, the robot can compute the warped parameters for the target object. A warped parameter is defined as a function on the warped point cloud $f(O_w)$. Warped parameters can be used to define geometric reference parameters, such as lengths, areas, and volumes, of an object's part. Warped parameters can also be used to define task frames.

For pouring, the task frame is defined by the lip point of the first container, and the center of the second container's opening. The center of the opening is defined as the mean of the rim points. The lip point is defined as the rim point that is the closest to the other container. A pouring motion is defined by the trajectory of the held container's lip point relative to the center of the second container's opening. The trajectory includes the relative 3D position and the tilt of the first container about its lip point. The other two rotation dimensions are usually assumed to be zero. If there is no second container, the lowest rim point is defined as the lip point.

The geometric reference parameters for pouring include the radius of the opening, the volume of the container, the height of the container, and a reference angle for tilting the cup. The radius of the opening is given by the mean distance between the rim points and the center of the opening. The volume of the container is given by the volume of the container points' convex hull. The height of the container is given by the range of all of the points along the first dimension. A tilt reference angle is defined by the amount that the cup must be rotated about the lip point, such that half of the container's volume is above the lip point. As the warping process reshapes the points of the source object, the estimates of the reference parameters will change accordingly. In this manner, the warped parameter function defines how the parameter's value is grounded in the object's geometry.

As the above examples show, warped parameters can be used to define various object properties, and can even build on each other. These parameters can then be automatically computed for new objects using the warping process.

3.6 Learning with Warped Parameters

In this section, we describe how a robot can learn pouring actions and task constraints that generalize to new objects using the warped parameters.

3.6.1 Learning Task Constraints

When performing a pouring task, the liquid should remain in the cup while it is being transported, and it should only be poured out if it will be transferred to another container. These task constraints correspond to phase transitions [22] and can be fulfilled by learning to predict when the held container will start to pour and when the poured liquid will fill the second container. The conditions for pouring and filling are learned by training a classifier for each condition. The classification is performed using logistic regression, which is a form of probabilistic classifier. The probability of pouring $y_p = 1$ from the first container is given by

$$p(y_p = 1|\mathbf{x}_u) = (1 + \exp(-\boldsymbol{\omega}^T \varphi(\mathbf{x}_u)))^{-1}$$

where $\varphi(\mathbf{x})$ is a vector of features describing the state of the container $\mathbf{x}$, and the weight vector $\boldsymbol{\omega}$ is computed from training data using iterative reweighted least squares. The features $\varphi(\mathbf{x})$ are of the form α/α_r, where α is a variable and α_r is a reference value defined by a warped parameter. For predicting pouring, the features include the tilt angle of the cup divided by the tilt reference angle, and the fluid volume divided by the volume of the container. The resulting features are dimensionless quantities that automatically adapt to the geometry of the container.

For predicting when the poured liquid increases the fluid volume in the second container $y_f = 1$, we expand the set of features to include both objects and their relative positions. The vertical distance between the containers is divided by the height of the cup. The horizontal distances between the containers are divided by the radius of the second container. These features allow the robot to learn when the poured liquid will miss the second container, as well as predict when the container will overflow.

3.6.2 Learning Motor Primitives in Warped Spaces

The proposed warping approach can also be used to learn motor primitives that adapt to the shape of the objects being manipulated. Motor primitives are often used to define desired trajectories that can be easily adapted to different situations. In order to model distributions of trajectories, we use the probabilistic motor primitives (ProMPs)[114]. These motor primitives encode correlations between the different dimensions of the trajectory, and can be conditioned on the initial state of the objects.

The learned motor primitive defines a desired trajectory in the task space described in Section 3.5.4. Similar to the features used to generalize task constraints, the trajectories are defined as dimensionless quantities. The vertical distance between the objects is divided by the height of the held cup, and the tilt angle is divided by the reference tilt angle. The horizontal distances are divided by the radius of the second container.

The motor primitives are learned by scaling the demonstrated trajectories according to the warped parameters of the objects used in the demonstrations. In order to execute a pouring action, the robot samples a trajectory from the ProMP, and rescales it according to the current objects' warped parameters.

3.7 Experiments

The proposed method was implemented and evaluated both in simulation and on a real robot. The robot, shown in Fig. 3.9, consists of two Kuka light weight robot arms, each equipped with a five-fingered DLR hand [99]. The robot observes the table-top scene from above using a Microsoft Kinect camera. Ten different cups and bowls were scanned from multiple views, and 3D mesh models were generated using an implicit surface representation and marching cubes [115].

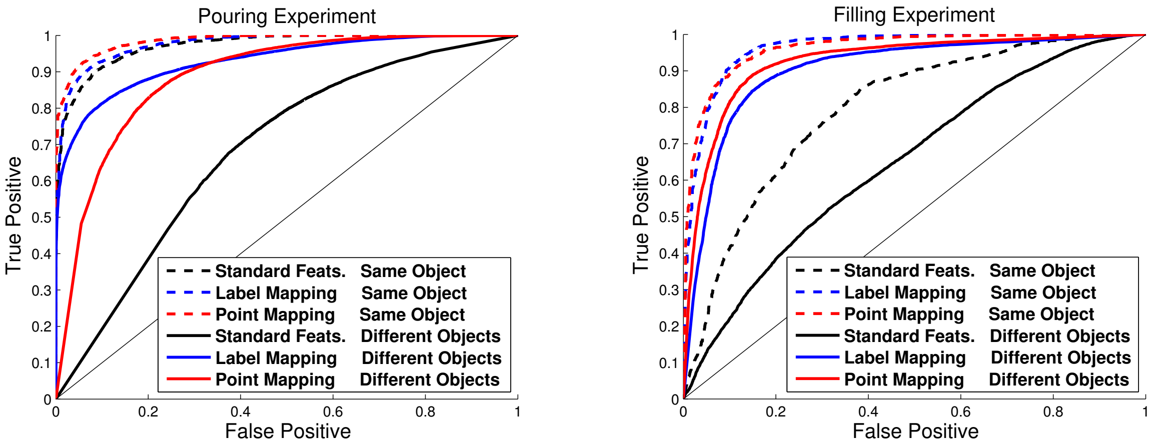


Figure 3.11: The figure shows the ROC curves for the learned classifiers for both the pouring experiment and the filling experiment. The *dashed* lines indicate the performance when the classifier is applied to data from the same object that was used for training the classifier. The *solid* lines indicate the performance when the classifiers are applied to novel objects, for which they had no training data. A classifier is generally considered to perform better if it gets closer to the top left corner. Classifiers were trained using features based on the warped parameters computed using both the label mappings and point mappings approaches. The standard features approach did not use the reference values given by the warped parameters.

3.7.1 Simulated Pouring and Filling Experiments

In the first experiment, we evaluated how well task constraints generalize between objects when using warped parameters. The objects were simulated using the Bullet physics engine [116] together with Fluids 2 for incorporating smoothed particle hydrodynamics [117].

Each object was filled 1000 times with a random amount of liquid, and tilted by a random angle around the lip point. If the volume of the fluid in the cup decreased, the trial was labelled as pouring $y_p = 1$. Otherwise it was labelled as not pouring $y_p = 0$. The classifiers were trained on sets of 50 samples. The classifiers were tested on two test sets: the 950 other samples from the same object, and the 9000 samples from the other objects. The latter dataset is used to test how well the classifiers generalize between different objects.

A similar procedure was used for the filling experiment. However, the cup used for pouring always contained 10 particles at the start of the trial, and the second container was filled by a random amount. The cup was always tilted by 120°. The relative positions of the cups were varied between trials. A trial was considered as successful $y_f = 1$ iff none of the particles ended up outside of the second container.

For each training set, three classifiers were computed. The first two classifiers were trained using the warped parameters from the point mapping and the label mapping approaches respectively. The features used for training the classifiers were described in Section 3.6.1. As a benchmark, we also evaluated the classifiers without using the warped parameters. In this case, all of the reference values α_r were set to one, regardless of the objects being manipulated, and the relative positions of the objects were defined by their centers.

The results of the pouring and filling experiments can be seen in Fig. 3.11 . As one would expect, the classifiers generally achieved similar levels of performance when evaluated on the training object. The standard features performed considerably worse in the filling experiment, as different cups were used for pouring even though the second container remained the same. The ROC curves show that the performance of all three classifiers decreases when generalizing to novel objects. However, the drop in performance is considerably less when using the warped parameters. The features based on the warped

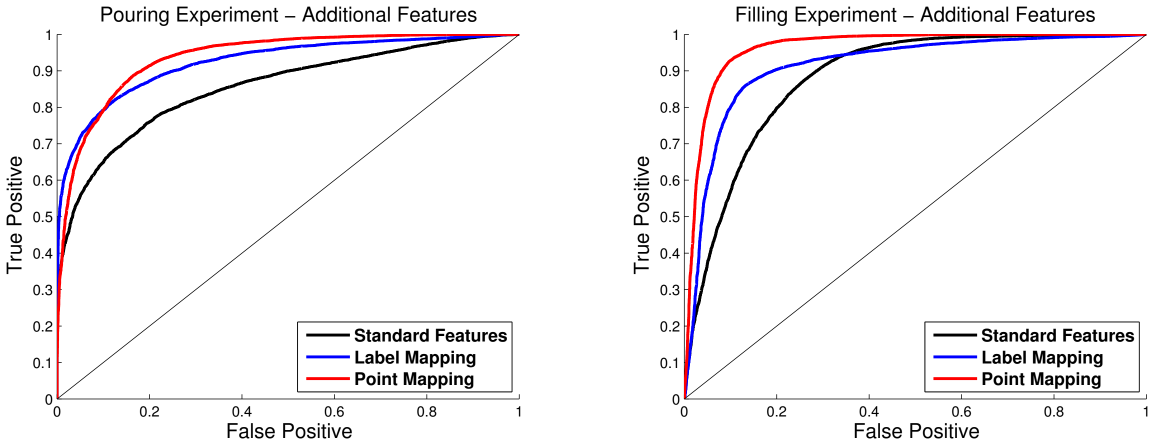


Figure 3.12: The figure shows the ROC curves for the learned classifiers for both the pouring experiment and the filling experiment. Classifiers were trained using the warped parameters of the label mapping and point mapping approaches as additional features. The standard features approach did not use the warped parameters.

parameters are therefore better at separating the positive and negative examples across different objects. While the two warping methods performed similarly well on the filling experiment, the label mapping approach performed better in the pouring experiment, detecting more than 50% of the true positives with almost no false positives. The results show that the warping parameters can be used to reliably generalize the constraints of the pouring task between different containers.

3.7.2 Warped Parameters as Additional Features

Rather than using the warped parameters to construct features, the warped parameters can also be used as additional features for representing the object. However, these features will be the same for the entire training set if the robot only learns from one object. Hence, the robot must learn from multiple objects in order to learn how to generalize.

For the second experiment, we use the same simulated pouring and filling setup as in the previous experiment. However, rather than learning from one object, the robot now learns from nine of the ten objects. The test set is given by 200 samples from one of the objects. The training set consists of 20 samples from each of the other nine objects, giving a total of 180 samples. We cycled through all of the containers, such that each one was used for testing. Rather than using features of the form α/α_r, the α and α_r values were simply concatenated into one extended feature vector. The results of the experiment are shown in Fig. 3.12 .

The standard features perform better than in the previous experiment due to the additional training data. The classifier is also less likely to overfit to a single object when learning from nine different containers. The warped parameter features lead to better performance in both the pouring and the filling tasks. For the pouring experiment, including the warped parameters increased the area under the curve (AUC) from 0.854 to 0.923 and 0.936 for the label mapping and point mapping approaches respectively. Similarly, the warped parameters increased the AUC from 0.880 to 0.913 and 0.963 for the filling experiment. Using the warped parameters there lead to an average increase in the AUC of approximately 6.5% over just using the standard features.

The experiment shows that the robot can use the warped parameters as additional features, when learning from multiple objets. In this manner, the robot can learn to generalize between different objects.

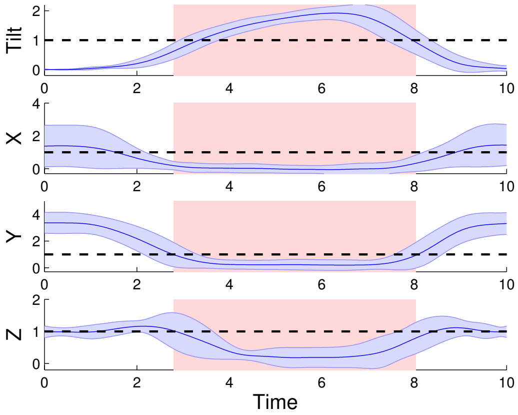


Figure 3.13: The plots show the distribution over trajectories learned by the ProMPs in the generalized space. The blue line indicates the mean trajectory, and the shaded regions correspond to +/- two standard deviations. The black horizontal lines indicate when the value is one. The tilt is one when the cup is tilted such that half of the container's volume is above the lip points. The X and Y values are one when the lip point is one radius away from the second container's center. The Z value is one when the vertical distance between the cup and the container is the same as the height of the cup. The red region indicates when the X-Y position of the cup's lip point is within one radius of the container's center.

3.7.3 Robot Pouring Experiment

In the second experiment, the robot used warped parameters to generalize pouring actions between different objects. The robot was provided with ten demonstrations of a pouring task using kinaesthetic teaching. All of the demonstrations were performed with the same two objects shown in the left picture of Fig. 3.14. For safety reasons, the task was performed with gel balls rather than an actual liquid. The cup was half full at the start of each trial. Using the ten demonstrations, the robot learned a ProMP for pouring, as described in Section 3.6.2. The learned distribution over trajectories is shown in Fig. 3.13. The robot was then given the task of pouring with different objects. The robot successfully learned to pour from a shorter cup into a bigger bowl, a smaller cup, and a square bowl, as shown in Fig. 3.14. Only a couple of gel balls were spilled during the experiments.

As the cups were half-full, pouring usually commenced when the tilt value went above one. Figure 3.13 shows that the distribution over trajectories remains safely below this value until the lip point is above the opening. When moving the cup back, most of the liquid has been poured out, and hence the cup can be tilted more. The pictures in Fig. 3.14 show that the cup was often placed close to the rim of the second container, which indicates that the robot was able to adapt the learned trajectory to the geometry of the object being manipulated.

3.8 Conclusion

In order to learn manipulation skills more efficiently, the robot should generalize between different objects in order to learn manipulation skills more efficiently. In order to generalize between objects, the robot requires representations that capture the relevant parts of the objects. In this chapter, we presented two different approaches for generalizing manipulation skills between objects.

As many manipulations are based on physical contacts, we proposed a kernel for computing the similarity between contact distributions. The contact distributions are modeled using multi-variate Gaussians, and the resulting kernel can be computed in closed-form. The robot used the proposed kernel to

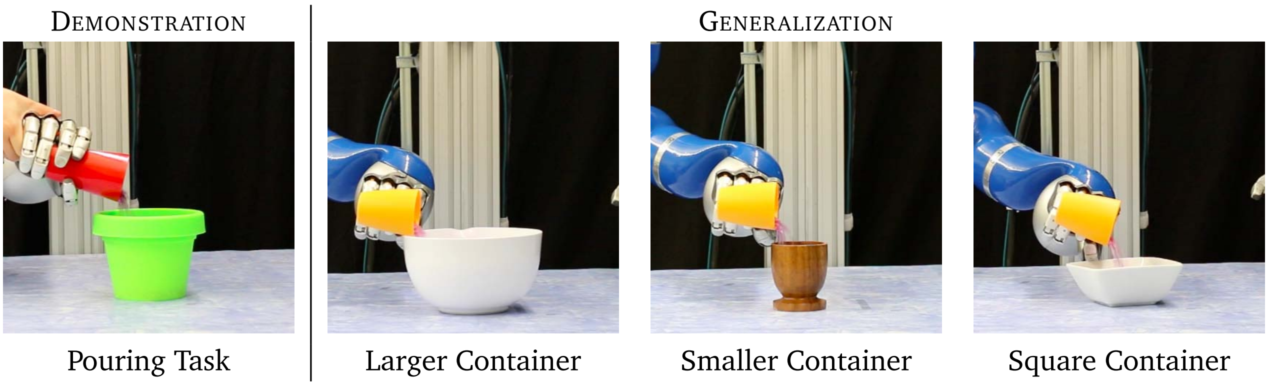


Figure 3.14: The pictures show the key results of the real robot experiment. The robot was provided with multiple demonstrations of the pouring task using kinaesthetic teaching, as shown on the left. Using the warped parameters approach, the robot successfully generalized the demonstrated actions to novel objects with different shapes and sizes, as shown on the right.

predict stable placements for stacking assorted blocks. The kernel circumvents the problem of explicitly defining a general set of features for representing contacts between objects.

The second part of this chapter focused on warped parameters. This approach allows the robot to compute geometric parameters, such as areas and volumes, of novel object with various shapes and sizes. The warped parameter is defined as a function on a known object's point cloud. The parameter is computed for another object by warping the point cloud to match its shape. Using the warped parameters, the robot could generalize a pouring action between different objects.

3.9 Potentially Helpful Insights

The motivation for the contact-kernel project was to create a flexible representation for contacts between objects. Using this representation, the robot can address the general challenges of establishing contacts between objects, which includes specific tasks, such as grasping, placing, and pushing. The representation should provide the robot with relatively direct access to the contact information to allow the robot to learn which contacts are suitable for a specific interaction or manipulation. Our early work therefore focused on representing each contact as a separate Gaussian. Representing the set of contacts as a single Gaussian provides a more coarse and compact representation of the contacts' structure. The single Gaussian representation captures the overall correlations between the contact points and normals, as well as the space spanned by the contacts.

The proposed kernels are closely related to bag-of-features representations. While the bag-of-features model is based on histograms, the kernels are based on continuous density estimates, i.e., Gaussians and kernel density estimates. For the histograms, the discretization of the input space leads to a feature vector of finite length. Similarly, the continuous density estimates define an infinite dimensional feature mapping [80]. This representation of the contact distribution is very flexible, can capture an arbitrary level of detail, and allows us to compare distributions directly. The kernel approach allows us to use this infinite dimensional feature space implicitly by directly computing the inner product of the feature maps.

One of the key characteristics of the proposed approach is that it explicitly models the contacts between objects. Depending on the learning task, the robot will either need to estimate the current set of contacts or predict potential contacts between objects. Although estimating contacts is not trivial, advances in 3D vision and tactile sensing are making the problem easier. The proposed representation can be used to capture the coarse structure of the contacts, and does not rely on very precise contact estimates. The alternative approach would be to learn the contacts implicitly. For example, a robot can learn to predict grasps based on the local shape of the object. In this case, the robot would consider the object's shape near to the entire hand, rather than near to the contact points. This approach works well for learning grasps, where one of the objects is always the hand. For a more general task, the robot would need to consider the shape of both objects in contact, e.g., the supporting surface and the object being placed for stacking. Thus, the general implicit approach would require the robot to learn compatible pairs of shapes. In contrast, for the explicit approach, both of the objects' shapes are already taken into consideration when estimating the contact points. The robot then just needs to take into consideration the shape of the contact regions.

The motivation for the warped parameter project was to create a method for computing various geometric parameters of objects directly from their point clouds. In this manner, the parameters are directly grounded in the shape of the object and can be computed for novel objects. By grounding the parameters in the point cloud data, the robot can use the same computation of an object's volume regardless of whether it is a cube, sphere, cone, or tapered cylinder.

The grounding of parameters provides useful insights into the geometric characteristics of parameters. For example, the length of an object part can be defined by the distance between two points on the object. Although the parameter itself is a scalar value, it is linked to a position (the mean of the two points) and a direction (the direction from one point to another). Other parameters, e.g., areas, volumes, and masses, can similarly be characterized by additional geometric information. Although this information provides useful clues about the parameter, it is lost when the parameter is reduced to only its scalar value. In our experiments, the motor primitives were scaled according to parameters which were aligned with the movement directions. For example, the vertical component of the movement was scaled with the height of the cup, and the horizontal movements were scaled according to the horizontal distribution of the rim points. Using this prior information, the robot could generalize the pouring motor primitive between different objects. This prior could potentially also be learned by the robot. In particular, the robot could learn to predict which warped parameters are relevant for adapting motor primitives based on the parameters' geometric characteristics.

4 Learning Motor Primitives for Multi-Phase Tasks

In order to learn new tasks in an efficient manner, a robot should decompose tasks into smaller subtasks. Learning a manipulation skill for each subtask is generally easier than learning one monolithic skill for the entire task. As many tasks will consist of similar subtasks, the robot can also reuse skills between different tasks. Transitioning between phases is a common subtask for manipulations [7, 1].

In this chapter, we present a probabilistic model for segmenting manipulations into discrete phases. The model captures the dynamics of each phase, and explicitly models the conditions for transitioning between phases. These two components of the model provide the basis for learning motor primitives for transitioning between different phases. Hence, the robot can learn a library of motor primitives using imitation learning and model-based reinforcement learning.

4.1 Learning Libraries of Motor Primitives for Multi-Phase Tasks

Manipulation tasks can usually be decomposed into sequences of simpler subtasks. One of the most common subgoals in manipulation tasks is transitioning between different phases [7, 1]. Phase transitions often occur when contacts are made or broken, and result in the robot's actions having different effects. For example, a robot can apply forces to an object by first moving into contact with it. It can then exert the neccessary forces for lifting the object up. Both the making of robot-object contacts the the breaking of object-table contacts correspond to phase transitions and represent important subgoals of the grasping task.

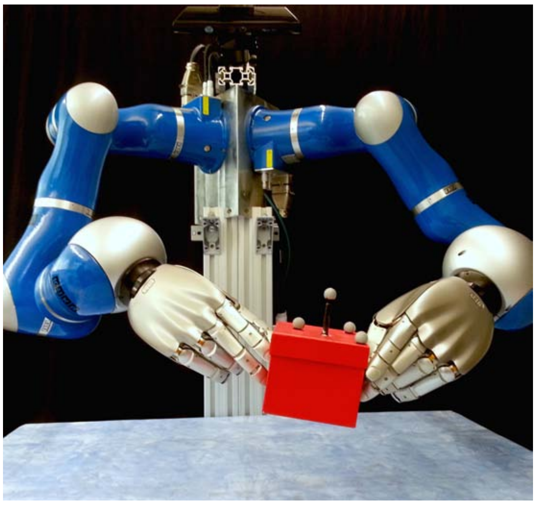


Figure 4.1: The Darias robot performing a bimanual grasp of a box. The motor primitives used to perform the task were learned using a model-based policy search approach. The model of the task's phases was learned from human demonstrations.

Manipulation tasks require the robot to first transition to a phase that allows the desired manipulation to be performed. Often, the robot will need to transition through a sequence of phases before reaching the desired phase. The conditions needed for transitioning between phases represent subgoals of the task, and can be used to guide the robot's skill learning process.

In this paper, we propose an approach for learning to perform multi-phase manipulation tasks. The robot first learns a model of the task from demonstrations, using a state-based transitions autoregressive hidden Markov model (STARHMM) [22]. The STARHMM models both the effects of the robot's actions in each phase, as well as the conditions needed to transition between phases. Using this model, the robot can apply imitation learning and model-based policy search methods in order to learn robust motor primitives for moving between different phases. In this manner, the robot decomposes the task into subtasks, and learns a motor primitive for each of these subtasks. The learned motor primitives represent building blocks for performing different tasks. In the next chapter, we will discuss how the robot can learn to sequence these motor primitives.

The proposed method was evaluated on a bimanual grasping task using the robot shown in Fig. 4.1. Given only two demonstrations, the robot successfully decomposed the task into five phases. The robot learned robust motor primitives for transitioning between these phases by using model-based policy search. The learned motor primitives generalize to different object locations, as well as objects with different shapes.

Previous work on learning sequences of motor primitives for manipulation tasks has mainly focused on learning from human demonstrations. Some imitation learning methods assume that a library of motor primitives already exists and focus on sequencing these primitives [118, 119, 120]. Other approaches have additionally learned the individual skills by segmenting the human demonstrations directly [8, 121, 122, 9, 10, 123].

Most of these works have focused on segmenting the demonstrated movements, and then learning a classifier for determining which motor primitive to execute next. As a result, the segments are linked to specific movements, and the sequencing is done according to the demonstrated task. Instead of performing the segmentation based on the demonstrated *actions*, our goal is to detect phase transitions based on the *effects of the actions*. We also apply a reinforcement learning approach to learn a high-level controller for learning to select the next motor primitive to execute.

Some methods have also been proposed for using reinforcement learning to improve the performance of skill sequences [124, 125, 126]. Stulp et al. [125] proposed the PI^2Seq policy search algorithm to optimize both the weight parameters and the goal points for a specific sequence of DMPs. The reward function was defined for the overall task and the segmentation into DMPs was predefined. Konidaris et al. [124] proposed the CST algorithm for learning skill trees. Demonstrations are segmented by applying changepoint detection to the samples' estimated future rewards. The goal state of one skill in a sequence is defined by the starting states of the next skill in the sequence. Sequences are merged into trees if both sequences execute the same skill at some point, and all future skills are also the same.

These approaches learn to optimize sequences of skills for specific tasks, and the skills are generally executed in a fixed order. Our proposed approach learns motor primitives for phase transitions, which can be reused for different tasks. In Chapter 5, we present a value function method for sequencing motor primitives to achieve different tasks. By learning a multi-phase forward model, the robot can use a model-based approach to reinforcement learning in order to efficiently improve and adapt its actions. The model's phase transition probabilities are used to guide the learning process. The phase models can also be used for learning new skills, and for determining the effects of executing the same motor primitive in different phases.

Multi-modal planning methods have been used to sequence primitive actions in order to achieve different manipulation goals [127, 128, 129]. However, the model of the system is usually predefined rather than learned. One could therefore consider using the methods proposed in this chapter to create a basis for multi-modal manipulation planning.

Our approach to learning both low- and high- level policies is a form of hierarchical reinforcement learning[130, 131, 132, 133]. Kober and Peters [132] proposed a method for learning the high-level and low-level policies in parallel in order to learn throwing movements for hitting a set of targets. The state for the high-level controller is discrete and defined by the current score of the game. Hart and Grupen [134] presented an intrinsically-motivated reinforcement learning approach to learning a hierarchy of control programs from a combinatorial control basis. A hierarchy of controllers for performing a bimanual grasp is learned incrementally as part of a scaffolding framework. Soni and Singh [133] presented an intrinsically motivated approach for learning policies for reaching predefined salient events with a Sony Aibo. In contrast, our proposed approach first learns a model of the system in order to determine the salient events.

Previous work in robotics has already shown the benefits of incorporating phases into the design of controllers [135, 136], and several methods have been proposed for learning controllers for multi-phase tasks [137, 138, 139, 140]. Levine and Abbeel [138] proposed a method for learning neural network controllers for multi-phase manipulation tasks. Koval et al. [137] decompose a grasping policy into pre- and post- contact policies. Mugan and Koipers [140] learn a model by discretizing the entire state and action spaces, and then applying reinforcement learning to the discrete domain.

In this chapter, we propose a model for representing phases and the transitions between them. The multi-phase model is defined using a modified hidden Markov model. We describe the structure of the model in Section 4.2.1. We explain how the model parameters are learned using expectation-maximization in Section 4.2.2. The parameters are learned from human demonstrations of the task using kinaesthetic teaching. There are different ways to initialize the model learning procedure. In Section 4.2.3, we discuss methods for selecting the number of phases and we present a spectral clustering approach to initializing the model based on contacts between objects. Given the model, the robot learns a library of motor primitives for transitioning between the different phases. The motor primitives are first learned from human demonstrations using the standard linear regression approach [34]. In Section 4.3.1, we explain a method for computing task frames based on contacts. Using these initial motor primitives and the learned model, we apply a policy search method to optimize the motor primitives for achieving specific phase transitions, as explained in Section 4.3.2. The robot thus learns a motor primitive for each of the phase transitions observed in the human demonstrations. The results of the real robot evaluations are presented in Section 4.4.

4.2 Multi-Phase Models

This section explains how a multi-phase model can be learned. We assume that the robot is initialized with demonstrations of the multi-phase task. As the robot attempts to perform the task, the model can be updated to incorporate the additional data. The structure of the model is outlined in Section 4.2.1. The robot learns the parameters of the model using the approach explained in Sections 4.2.2 and 4.2.3.

4.2.1 Modeling Multi-Phase Manipulation Tasks

The observed state of the robot and its environment at time t are given by the state $\mathbf{s}_t \in \mathbb{R}^n$. The robot then performs an action $a_t \in \mathbb{R}^m$, which results in the state transitioning to the next state $\mathbf{s}_{t+1} \in \mathbb{R}^n$. This change in state depends on the current phase $\rho_t \in \{1, ..., \kappa\}$, which is hidden. In this paper, the *phase* corresponds to the hidden state of an HMM, and the *state* refers to the observed state. As the phases are not directly observed, the robot needs to infer the phase from the observed states and the actions' effects.

The effects of performing an action $\mathbf{a}_t$ in state $\mathbf{s}_t$ and phase ρ_t are modeled by the transition probability $p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t, \rho_t)$. We represent the state transitions $p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t, \rho_t)$ using a linear Gaussian model. Therefore, the distribution of the next state is given by the Gaussian

$$\mathbf{s}_{t+1} \sim \mathcal{N}(\mathbf{A}_{\rho_t} s_t + \mathbf{B}_{\rho_t} \mathbf{a}_t, \mathbf{\Sigma}_{\rho_t}),$$

where $\mathbf{A}_i \in \mathbb{R}^{n \times n}$, $\mathbf{B}_i \in \mathbb{R}^{n \times m}$, and $\mathbf{\Sigma}_i \in \mathbb{R}^{n \times n}$ are matrices corresponding to phase $\rho = i$.

The phase transition distribution depends on the current state and the previous phase $p(\rho_t|\mathbf{s}_t, \rho_{t-1})$. The dependency on the previous phase allows the model to represent hysteresis effects, and transient state information. For example, the making and breaking of a contact can be detected by dynamic tactile sensing, indicating a transition to the next phase [7]. However, once the contact event is over, the sensor's readings may return to their previous values even though the phase has changed. At that point, the model has already switched to the new phase.

In order to learn the entry and exit conditions of phases, we incorporate a binary termination variable $\varepsilon_t \in \{0, 1\}$, which is distributed according to the termination distribution $p(\varepsilon_t|\mathbf{s}_{t+1}, \rho_t)$. If the termination variable is zero $\varepsilon_t = 0$, no phase transition can occur and the next phase is the same as the current state, i.e., $p(\rho_{t+1} = \rho_t | \rho_t, \varepsilon_t = 0) = 1$. If the termination variable is one $\varepsilon_t = 1$, the next phase ρ_{t+1} is distributed according to the initiation distribution, i.e., $p(\rho_{t+1} | \mathbf{s}_t, \varepsilon_t = 1)$. The next phase can be the same as the current phase $\rho_{t+1} = \rho_t$ even if the termination variable is one $\varepsilon_t = 1$. The phase transition distribution can be computed by marginalizing out the termination variable

$$p(\rho_t|\mathbf{s}_t, \rho_{t-1}) \;=\; p(\rho_t|\rho_{t-1}, \varepsilon_{t-1} = 0) p(\varepsilon_{t-1} = 0|\mathbf{s}_t, \rho_{t-1}) + p(\rho_t|\mathbf{s}_t, \varepsilon_{t-1} = 1) p(\varepsilon_{t-1} = 1|\mathbf{s}_t, \rho_{t-1}).$$

The termination and initiation distributions of phases are modeled using probabilistic classifiers. We model the phase termination probabilities using logistic regression

$$p(\varepsilon_{t-1} = 1|\mathbf{s}_t, \rho_{t-1} = i) = (1 + \exp(-\hat{\boldsymbol{\omega}}_i^T \boldsymbol{\phi}(\mathbf{s}_t)))^{-1}$$

where $\hat{\boldsymbol{\omega}}_j \in \mathbb{R}^d$ is a weight vector for terminating phase $\rho = j$, and $\boldsymbol{\phi}(\mathbf{s}_t)$ is a function mapping the state $\mathbf{s}_t$ to a d dimensional feature vector. Features may, for example, be a subset of the full state vector or additionally include the positions of objects relative to each other. Similarly, we represent the phase initiation distribution as

$$p(\rho_t = j|\mathbf{s}_t, \varepsilon_{t-1} = 1) = \frac{\exp(\check{\boldsymbol{\omega}}_j^T \boldsymbol{\phi}(s_t))}{\sum_k \exp(\check{\boldsymbol{\omega}}_k^T \boldsymbol{\phi}(s_t))}$$

where $\check{\boldsymbol{\omega}}_j \in \mathbb{R}^d$ is a weight vector for initiating phase $\rho = j$. We assume that the previous phase terminated at the start of the trajectory, such that $p(\rho_1 = j|\mathbf{s}_1) = p(\rho_1 = j|\mathbf{s}_1, \varepsilon_0 = 1)$. The policy for selecting actions will be discussed in Section 4.3. In order to improve clarity, we can assume that the actions are drawn from some fixed distribution $p(\mathbf{a}_t)$ when learning the multi-phase model.

Given the individual components of the model, the probability of observing a sequence of N samples of states $\mathbf{s}_{1:N} = \{\mathbf{s}_1, \ldots, \mathbf{s}_N\}$, actions $\mathbf{a}_{1:N} = \{\mathbf{a}_1, \ldots, \mathbf{a}_N\}$, phases $\rho_{1:N} = \{\rho_1, \ldots, \rho_N\}$, phase terminations $\varepsilon_{0:N-1} = \{\varepsilon_0, \ldots, \varepsilon_{N-1}\}$, and next states $\mathbf{s}_{2:N+1} = \{\mathbf{s}_2, \ldots, \mathbf{s}_{N+1}\}$ is given by

$$p(\mathbf{s}_{1:N+1}, \mathbf{a}_{1:N}, \rho_{1:N}, \varepsilon_{0:N-1}) = p(\varepsilon_0, \rho_1, \mathbf{s}_1) \prod_{t=1}^{N} p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t, \rho_t) p(\mathbf{a}_t) \prod_{t=2}^{N} p(\rho_t, \varepsilon_{t-1}|\mathbf{s}_t, \rho_{t-1}),$$

where $p(\varepsilon_0, \rho_1, \mathbf{s}_1) = p(\mathbf{s}_1, \varepsilon_0) p(\rho_1|\mathbf{s}_1, \varepsilon_0)$ and

$$p(\rho_t, \varepsilon_{t-1}|\mathbf{s}_t, \rho_{t-1}) = p(\rho_t|\mathbf{s}_t, \rho_{t-1}, \varepsilon_{t-1}) p(\varepsilon_{t-1}|\mathbf{s}_t, \rho_{t-1}).$$

The graphical model of this probability factorization, as well as the general STARHMM model, is shown in Fig. 4.2 . The key difference to an autoregressive HMM is the additional edge from the current state to the current phase. As a result of this edge, the transitions between phases depend on the observed state. The graphical model also illustrates the Markov property of the model: given the state $\mathbf{s}_t$ and the phase state ρ_t, all future states are independent of the past states. This property is important, as it will allow us to learn the model parameters in a computationally efficient manner.

The extended STARHMM, with entry and exit conditions, presents several benefits over the original model. Rather than having to learn $\kappa + 1$ multi-class classifiers, the robot only needs to learn κ binary classifiers and one multi-class classifier. In this manner, the robot can share information between transitions from different phases. The extended version also results in a more consistent mapping from the state space to phases. For example, the original STARHMM could learn that for some states the phase transitions are given by $p(\rho_t = 2|\mathbf{s}_t, \rho_{t-1} = 1) \approx 1$ and $p(\rho_t = 1|\mathbf{s}_t, \rho_{t-1} = 2) \approx 1$. As a result, the phase would switch at every time step, which is unlikely to occur in reality. Although the STARHMM with entry and exit conditions can also represent hysteresis effects, it cannot force this kind of behavior.

In the proposed model, we assumed that the state is observable. This assumption is common for segmenting manipulations movements [9, 8]. However, similar to the model proposed by Barber [141], one could also extend the STARHMM model to use observations of hidden state variables.

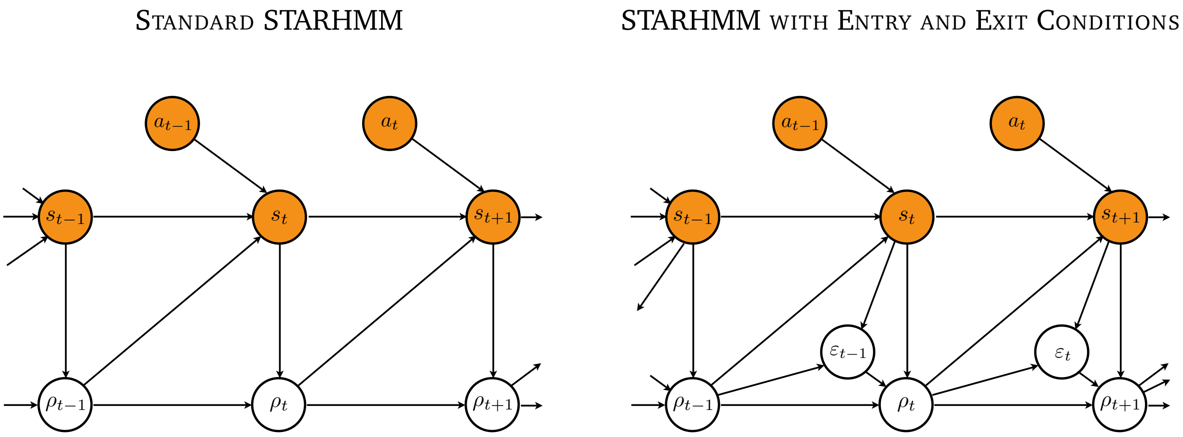


Figure 4.2: Graphical model of the standard STARHMM and the modified version with entry and exit conditions. The orange nodes indicate observed variable. The white nodes indicate hidden variables. The models include the states s, actions **a**, and phases ρ. The model on the right also includes the termination variable ε.

4.2.2 Model Learning Using Expectation-Maximization Algorithm

Having defined the structure of the model, we now focus on learning the model parameters $\hat{\boldsymbol{\omega}}$, $\check{\boldsymbol{\omega}}$, $\mathbf{A}$, $\mathbf{B}$, and $\boldsymbol{\Sigma}$, which we will refer to jointly as $\boldsymbol{\theta} = \{\hat{\boldsymbol{\omega}}, \check{\boldsymbol{\omega}}, \mathbf{A}, \mathbf{B}, \boldsymbol{\Sigma}\}$. Given a set of sampled trajectories of states and actions, we propose using the *expectation-maximization* (EM) algorithm [142] to estimate the parameters. The EM algorithm iterates between an expectation step and a maximization step in order to find maximum likelihood estimates of the model parameters $\boldsymbol{\theta}$ given that some of the variables are hidden $\mathscr{H} = \{\rho_{1:N}, \varepsilon_{0:N-1}\}$, i.e., the phases and the phase terminations of the samples are not known. The steps of the algorithm are explained below.

Expectation Step

The first step of the EM algorithm is the expectation step. In this step, we need to compute the distribution over the hidden states, i.e., the phases, given the observed sequence of variables. In particular, we need to compute the distribution $p(\rho_t, \varepsilon_t, \rho_{t+1} | \mathbf{s}_{1:N+1}, \mathbf{a}_{1:N})$ for the computations in the maximization step. We compute these marginal probabilities efficiently by using a forward-backward message passing approach. During the expectation step, we assume that the parameters $\boldsymbol{\theta}$ of our model are fixed.

In order to improve the clarity of the methodology below, we will define $\mathbf{z}_t = \{\mathbf{s}_t, \mathbf{a}_t\}$ as the observed state and actions together. The final sample is given by $\mathbf{z}_{N+1} = \mathbf{s}_{N+1}$. Thus, we have $p(\mathbf{z}_{t+1}|\rho_t, \mathbf{z}_t) = p(\mathbf{s}_{t+1}|\rho_t, \mathbf{s}_t, \mathbf{a}_t)p(\mathbf{a}_{t+1})$, and $p(\rho_t|\rho_{t-1}, \mathbf{z}_t) = p(\rho_t|\rho_{t-1}, \mathbf{s}_t)$ as ρ_t is not conditioned on $\mathbf{a}_t$. We also marginalize out the termination variables ε and use $p(\rho_t|\mathbf{s}_t, \rho_{t-1})$ to define the messages.

We first send a series of messages forward through the network from $t = 1$ to $t = N$. The forward messages give the probability of observing the sequence of states, actions, and next state up to the current time step are defined as

$$\alpha_j(t) = p(\mathbf{z}_{1:t+1}, \rho_t = j).$$

The first message, starting at $t = 1$, is initialized according to

$$\alpha_j(1) = p(\mathbf{z}_2|\rho_1, \mathbf{z}_1)p(\rho_1 = j|\mathbf{z}_1)p(\mathbf{z}_1).$$

The subsequent messages are computed recursively as

$$\alpha_j(t) = p(\mathbf{z}_{t+1}|\rho_t = j, \mathbf{z}_t)\sum_i \alpha_i(t-1)p(\rho_t = j|\rho_{t-1} = i, \mathbf{z}_t).$$

The second set of messages are sent backwards through the network from $t = N$ to $t = 1$. The backward messages give the probability of observing the remainder of the observed sequence of states, actions, and next states given the current phase and next state

$$\beta_j(t) = p(\mathbf{z}_{t+2:N}|\rho_t = j, \mathbf{z}_{t+1}).$$

We initialize the backward messages at time $t = N$ as

$$\beta_j(N) = 1,$$

and we recursively compute the messages backwards in time according to the formula

$$\beta_j(t-1) = \sum_i p(\rho_t = i|\rho_{t-1} = j, \mathbf{z}_t)p(\mathbf{z}_{t+1}|\rho_t = i, \mathbf{z}_t)\beta_i(t).$$

Given the forward and backward messages, we can easily compute the joint distribution of a phase, the phase termination, and the next phase as

$$p(\rho_t = i, \varepsilon_t = j, \rho_{t+1} = k|\mathbf{z}_{1:N+1}) = \frac{\alpha_i(t)p(\mathbf{z}_{t+2}, \varepsilon_t = j, \rho_{t+1} = k|\rho_t = i, \mathbf{z}_{t+1})\beta_k(t+1)}{\sum_l \alpha_l(t)\beta_l(t)},$$

where $p(\mathbf{z}_{t+2}, \rho_{t+1} = k, \varepsilon_t = j|\rho_t = i, \mathbf{z}_{t+1})$ is given by

$$p(\varepsilon_t = j|\rho_t = i, \mathbf{z}_{t+1})p(\rho_{t+1} = k|\rho_t = i, \varepsilon_t = j, \mathbf{z}_{t+1})p(\mathbf{z}_{t+2}|\rho_{t+1} = k, \mathbf{z}_{t+1}).$$

Having computed these probabilities, we can now proceed to the maximization step of the algorithm.

Maximization Step

In the maximization step of the EM algorithm, we must compute the parameters that maximize the expected log-likelihood of the observed and hidden variables

$$\boldsymbol{\theta}_{\text{new}} = \arg\max_{\boldsymbol{\theta}} \sum_{\rho,\varepsilon} p(\mathscr{H}|\mathbf{z}_{1:N+1}; \boldsymbol{\theta}_{old}) \ln p(\mathscr{H}, \mathbf{z}_{1:N+1}; \boldsymbol{\theta}),$$

where the hidden variables $\mathscr{H}$ are given by $\mathscr{H} = \{\rho_{1:N}, \varepsilon_{0:N-1}\}$, the summation is over all possible sequences of ρ and ε, and the conditional distributions $p(\mathscr{H}|\mathbf{z}_{1:N+1}; \theta_{\text{old}})$ are computed using the old model parameters $\boldsymbol{\theta}_{\text{old}}$ as indicated. By factorizing the joint distribution $p(\mathscr{H}, \mathbf{z}_{1:N+1}; \boldsymbol{\theta})$, decomposing the log of a product into a summation of logs, and marginalizing out variables, the maximization problem can be rewritten as

$$\begin{aligned}\boldsymbol{\theta}_{\text{new}} = \arg\max_{\boldsymbol{\theta}} & \sum_{t=1}^{N}\sum_{\rho_t} p(\rho_t|\mathbf{z}_{1:N+1}; \boldsymbol{\theta}_{\text{old}}) \ln p(\mathbf{z}_{t+1}|\rho_t, \mathbf{z}_t; \boldsymbol{\theta}) \\ & + \sum_{t=1}^{N}\sum_{\rho_t} p(\rho_t, \varepsilon_{t-1}{=}1|\mathbf{z}_{1:N+1}; \boldsymbol{\theta}_{\text{old}}) \ln p(\rho_t|\varepsilon_{t-1}{=}1, \mathbf{z}_t; \boldsymbol{\theta}) \\ & + \sum_{t=1}^{N-1}\sum_{\rho_t}\sum_{\varepsilon_t} p(\rho_t, \varepsilon_t|\mathbf{z}_{1:N+1}; \boldsymbol{\theta}_{\text{old}}) \ln p(\varepsilon_t|\rho_t, \mathbf{z}_{t+1}; \boldsymbol{\theta}).\end{aligned}$$

The marginal distributions $p(\rho_t|\mathbf{z}_{1:N+1}, \boldsymbol{\theta}_{\text{old}})$, $p(\rho_t, \varepsilon_{t-1}=1|\mathbf{z}_{1:N+1}; \boldsymbol{\theta}_{\text{old}})$, and $p(\rho_t, \varepsilon_t|\mathbf{z}_{1:N+1}; \boldsymbol{\theta}_{\text{old}})$ are straightforward to compute from the joint distributions computed in the expectation step. The new parameters can then be computed for the phase- and state- transition distributions. For the state transition distribution, the matrices **A** and **B** are computed using weighted linear regression. When learning the matrices $\mathbf{A}_i$ and $\mathbf{B}_i$, the weight for sample $\{\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}\}$ is given by $p(\rho_t = i|\mathbf{s}_{1:N+1}, \mathbf{a}_{1:N}, \boldsymbol{\theta}_{\text{old}})$.

The phase transition parameters $\hat{\boldsymbol{\omega}}$ and $\check{\boldsymbol{\omega}}$ are computed using weighted logistic regression. Logistic regression does not have a closed-form solution and it must instead be computed iteratively using gradient decent. However, the optimization is convex and, therefore, a global optimum can be easily found. When learning the phase termination model for phase $\rho = i$, the sample $\{\varepsilon_t, \mathbf{s}_t\}$ is weighted by $p(\rho_t = i, \varepsilon_t|\mathbf{s}_{1:N+1}, \mathbf{a}_{1:N}, \theta_{old})$. Similarly, the sample $\{\rho_t, \mathbf{s}_t\}$ is weighted by $p(\rho_t, \varepsilon_t = 1|\mathbf{s}_{1:N+1}, \mathbf{a}_{1:N}, \theta_{old})$ when learning the phase initiation model $p(\rho_t|\mathbf{s}_t, \varepsilon_{t-1} = 1)$. Regularization can be added to both the linear regression and the logistic regression in order to incorporate prior information and avoid overfitting.

After the maximization step has been completed, the algorithm computes the expectation step again with the new parameters. The process iterates between the two steps until the model has converged to a solution.

4.2.3 Initialization

The EM learning method converges to a local optimum and, hence, the quality of the learned model depends on the initialization. The number of phases κ also needs to be specified. The value of κ can be selected using a model selection criterion such as the Akaike information criterion (AIC) or Bayesian information criterion (BIC) [143, 144]. Alternatively, one could also use a cross-validation approach to select a suitable value.

Given that phase transitions often correspond to the making or breaking of contacts, we instead initialized the model by clustering samples according to their contact distributions [145]. The clustering was performed using spectral clustering [146]. The clustering is performed on the concatenated samples from all of the observed trajectories. For a set of v samples, one first has to compute a weight matrix $\mathbf{W} \in \mathbb{R}^{v \times v}$, which defines the similarity between the samples. We computed the similarity using a contact distribution kernel [20], as described in Chapter 3. The kernel is computed using a point cloud model of the object. A point on the object is considered to be a contact point if it is within 1cm of the robot's fingers or the table. We compute one kernel for the contacts between the object and the robot's hands, and another for the contacts between the object and the table. A kernel's value is set to one if the objects did not make contact in either sample, and it is set to zero if the objects made contact for one of the samples, but not the other. The overall kernel value is given by the product of these two kernel values. In the future, one could also consider differentiating between static and sliding contacts.

The output of the clustering is an assignment of each sample to a cluster, where $c_i \in \{1, ..., \kappa\}$ denotes the assignment of the i^{th} sample. The clustering of the samples does not take into account the temporal ordering of the samples. In order to improve robustness, segments with a duration of less than five samples were reassigned to the previous cluster in the trajectory. The goal of the clustering is to assign samples with high weights to the same cluster, and samples with low weights to different clusters. We can therefore define a score for a cluster assignment **c** as

$$\text{score}(\mathbf{c}) = \sum_{i}^{v} \sum_{c_j = c_i} [\mathbf{W}]_{ij} + \sum_{c_j \neq c_i} 1 - [\mathbf{W}]_{ij},$$

where $[\mathbf{W}]_{ij}$ is the weight between the i^{th} and j^{th} samples. We performed the clustering multiple times, with different values of κ, and selected the clustering with the highest score. The samples assigned to a cluster are used to compute initial parameters for the state-transition distribution of the corresponding phase. The cluster assignments are also used to determine initial parameters for the phase initiation distribution.

The outcome of this clustering can also be used to define features for representing the contacts between objects. For each phase, we selected the sample which contributed the most to the clustering score as the basis for a feature. The feature is then defined by the kernel value between this basis sample and the current contact distribution.

The clustering also allows additional prior information to be incorporated into the model. For learning a phase's state transition distribution $p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t, \rho_t)$, the regularization between a hand's actions and the object's state was increased if they were not in contact for most of the cluster's samples. In this manner, we include our prior knowledge that a hand is less likely to manipulate an object if they are not in contact.

This approach to initializing the model worked well in the experiments, and found the correct number of phases for the demonstrated task.

4.3 Learning Motor Primitive Libraries for Multi-Phase Tasks

Having learned a multi-phase model, the next step is to learn movements for transitioning between the different phases. In this manner, the model decomposes the original task into a series of subtasks and learns a motor primitive policy for each subtask. The learned motor primitives should generalize between different scenarios and be reusable for different tasks. The process of learning a library of motor primitives is explained in Sections 4.3.1 and 4.3.2.

4.3.1 Dynamic Motor Primitives

In order to perform multi-phase tasks, the robot needs to perform actions to transition between different phases. The robot's movements are represented using dynamic motor primitives (DMPs) [62, 35]. These motor primitive representations are easily adapted to different situations and can be learned in a straightforward manner [34, 35]. The robot learns one DMP for each of the phase transitions observed in the human demonstrations. For more details on DMPs, we refer the reader to Chapter 2, where they were explained in detail.

In order to set the goal state $\mathbf{g}$ of the motor primitives, we define a set of task frames according to the object's current pose and its shape. The robot first learns a κ-class logistic regression classifier for predicting phases from contacts by using the contact features described in Section 4.2.3. Using a model of the hand and point cloud of the object, the robot then samples different potential hand poses relative to the object. In our experiments, we sampled grasps in a 3D grid with 2 cm intervals and discarded samples that did not make contact with the object. Each of the hand poses is evaluated by the classifier and weighted by the probability of being assigned to the goal phase. Mean-shift clustering [45] is subsequently applied to the weighted samples, and the mode with the highest likelihood is used to define the task frame. If the hand is already in contact with the object, or making contact would reduce the likelihood of reaching the next phase, the current pose of the hand is used as the task frame. If the object is in contact with the table, we also test whether removing the table would increase the likelihood. In the future, one could extend this approach to sampling different object poses in the scene.

The DMPs are incorporated into the model as shown in Fig. 4.3. At time step t, the robot is executing motor primitive $\mathscr{M}_t$, which includes the linear systems as well as the canonical system. The robot performs an action $\mathbf{a}$ according to the desired trajectory and the current state $p(\mathbf{a}_t|\mathscr{M}_t, \mathbf{s}_t)$. Once a DMP has finished, a new DMP is selected according to the current state $p(\mathscr{M}_t|\mathscr{M}_{t-1}, \mathbf{s}_t)$. The graphical model illustrates the complementary nature of motor primitives and phases.

4.3.2 Learning Motor Primitives using Policy Search

The performance and robustness of the DMPs can be further improved by adapting the goal state $\mathbf{g}$ and weights $\mathbf{w}$ using model-based reinforcement learning. The purpose of each DMP is to bring the

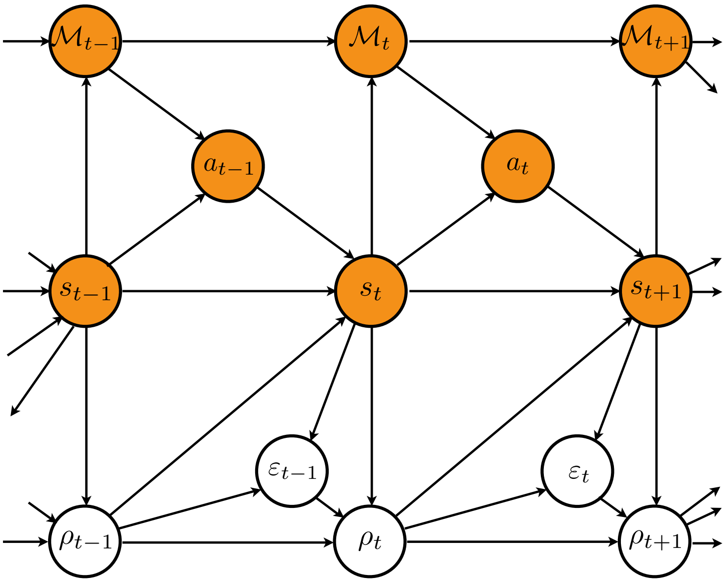


Figure 4.3: Graphical model of the task model and motor primitive controller. The orange nodes indicate observed variable. The white nodes indicate hidden variables. The model includes the states **s**, actions **a**, motor primitives $\mathscr{M}$, termination variables β, and the phases ρ. The top half of the graphical model depicts the motor primitive controller. The bottom half of the model defines the multi-phase model of the system.

robot from one phase to another. The model's phase transition distribution $p(\rho_t|\rho_{t-1}, \mathbf{s}_t)$ can be used to define a reward function for transitioning between different phases. The parameters of the DMP are then learned using relative entropy policy search (REPS) [147].

In order to learn motor primitives for transitioning between from phase ρ_0 to ρ_g, we define the reward at each time step as the probability of transitioning to the goal phase $p(\rho_{t+1} = \rho_g|\mathbf{s}_{t+1}, \rho_t)$. This reward is discounted over time by the probability of transitioning to a phase other than ρ_0 or ρ_g. In this manner, the reward function directs the robot towards the goal phase's conditions, while avoiding other phases. A squared cost term was also applied to penalize large actions as well as large deviations of the goal state from the task frame's origin to keep the goal grounded. By including smooth features for modelling the phase transition distribution $p(\rho_t|\rho_{t-1}, \mathbf{s}_t)$, the reward function can guide the robot during the learning process. This reward function worked well in the experiments, but other reward functions could also be constructed using the phase transition distribution.

Given the reward function and a set of starting states, the parameters $\mathbf{g}$ and $\mathbf{w}$ of the the DMP are learned using episodic REPS. The robot begins by learning the goal parameters $\mathbf{g}$, which are defined relative to the task frame. The distribution over the parameters is modeled as a Gaussian $\mathbf{g} \sim \mathcal{N}(\boldsymbol{\mu}_{g0}, \boldsymbol{\Sigma}_{g0})$, where the initial mean $\boldsymbol{\mu}_{g0}$ is given by the origin of the task frame. Parameters are sampled from the distribution and evaluated for each starting state using the learned multi-phase model. The rewards are averaged over the starting states.

After evaluating multiple samples of parameter sets from the current policy $\mathcal{N}(\boldsymbol{\mu}_g, \boldsymbol{\Sigma}_g)$, a policy update is performed to determine a new policy $\mathcal{N}(\boldsymbol{\mu}'_g, \boldsymbol{\Sigma}'_g)$. REPS computes a new policy that maximizes the expected reward, while limiting the Kullback Leibler divergence between the old and new policies $\epsilon \geq D_{\text{KL}}(\mathcal{N}(\boldsymbol{\mu}'_g, \boldsymbol{\Sigma}'_g)||\mathcal{N}(\boldsymbol{\mu}_g, \boldsymbol{\Sigma}_g))$. Limiting the KL divergence between the policies makes the learning process more robust to noisy rewards, and the robot is less likely to converge prematurely to a poor local optimum. For more details on REPS, we refer the reader to the survey of Deisenroth et al. on policy search methods in robotics [148]. The weight parameters $\mathbf{w}$ are learned in a similar manner. The parameters $\mathbf{g}$ and $\mathbf{w}$ could also be learned jointly [125].

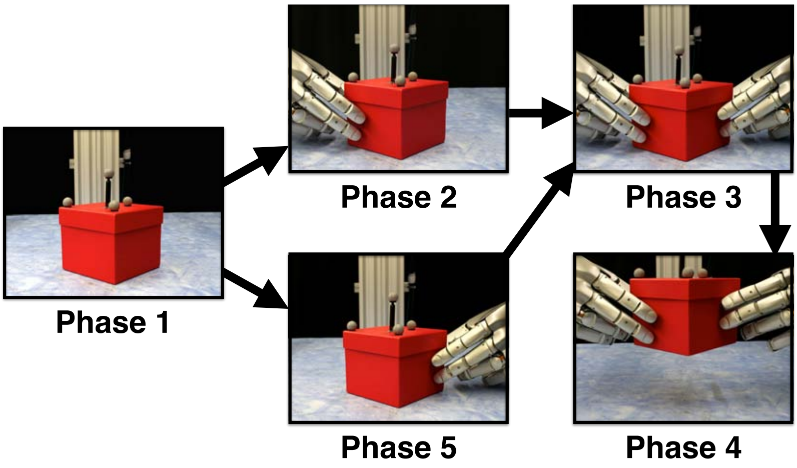


Figure 4.4: The five phases detected in the task demonstrations. The pictures show the contacts between the objects in each of the phases. The arrows show the phase transitions that were observed during the demonstrations. The robot learns a motor primitive for each of these transitions.

Once a motor primitive has been learned, it can be executed using the learned model to get the starting states for the next phase. This process is repeated until motor primitives have been learned for all of the phase transitions. The model can also be used to learn motor primitives for specific tasks by defining a suitable reward function, e.g., moving to a goal location while remaining in the current phase.

4.4 Evaluations

The proposed method was evaluated on a bimanual grasping task. The robot consists of two Kuka light-weight robot arms, and two five-fingered DLR hands [99].The robot's arms were controlled using impedance control in task space. The fingers are also compliant, which gives them shape adaptability [97]. The object was tracked using a marker-based Optitrak system. The box is too large for the robot to grasp with a single hand.

4.4.1 Setup and Model Learning

The robot was given two demonstrations of a bimanual grasping task using kinaesthetic teaching. In the first demonstration, the robot's right hand made contact with the box first. In the second demonstration, the left hand makes the first contact with the object. In both demonstrations, the object was subsequently grasped with both hands and lifted up from the table. The robot performed the task again by replaying the demonstrated trajectories with the objects placed at the same positions and orientations as in the demonstrations. The trajectories were sampled at 10 Hz. The two trajectories were coarsely segmented into phases using the method described in Section 4.2.3. The five detected phases, as well as the observed transitions between them, are illustrated in Fig. 4.4 .

The state $\mathbf{s}$ includes the position and orientation of the box, the positions of the robot's hands, and the contact state. Although the robot does not have tactile sensors, the joints of the robot's fingers have torque sensors. Rather than using the joint angles and torques directly, principal component analysis was used to reduce the dimensionality of the data from 60 to eight dimensions, i.e., two dimensions for position and two dimensions for torques for each hand. The state transition distributions learned the change in state rather than the absolute state, except for the contact features. The actions were defined by the change in the desired trajectory. The robot uses an impedance controller to follow the desired trajectory. The features for the phase transition distributions include the distances between the

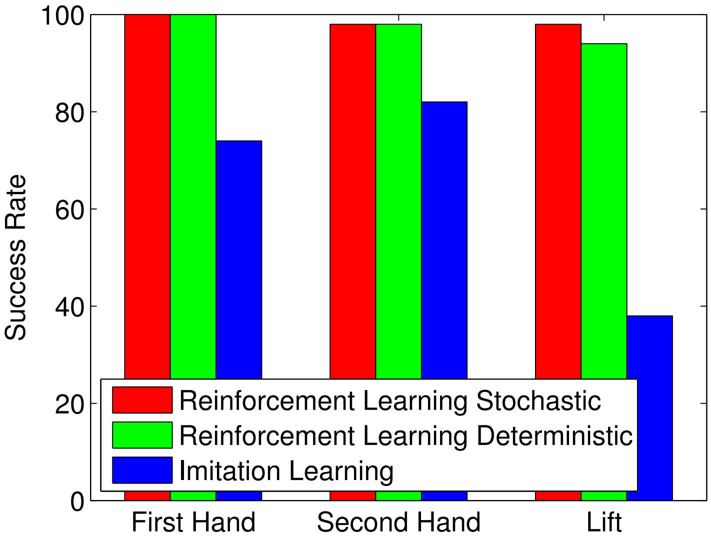


Figure 4.5: The success rates of the motor primitive evaluations. The robot performed the sequences of three motor primitives (first hand, second hand, and then lift) as observed in the human demonstrations. The different colored bars indicate different approaches to learning the motor primitives.

box and the finger tips, the distance between the box and the table, and the joint angle and torque data. The contact features, as described in Section 4.2.3, were also used for computing the phase initiation distribution.

4.4.2 Learning a Library of Motor Primitives for Phase Transitions

Using the method described in Section 4.3, the robot learned a library of five motor primitives corresponding to the five phase transitions observed in the demonstrations. Three different approaches for learning DMP libraries were evaluated for comparison. The first approach used only imitation learning to learn the DMPs. The goal states were set to the origins of the task frames, except for the 3 → 4 phase transition. For this motor primitive, the goal state for each hand was manually set to 10 cm above its task frame in order to achieve a lifting movement. The second approach used the model-based reinforcement learning together with the imitation learning in order to create the DMP library, as described in Section 4.3. The third approach also used imitation learning and reinforcement learning. However, rather than sampling from the model in a stochastic manner, the maximum-likelihood state transitions were used and the phase was kept the same for the entire trial. In this manner, the variance in the trials' rewards is reduced. For both model-based reinforcement learning approaches, the goal states $\mathbf{g}$ of the motor primitives were learned using 10 policy updates of 50 episodes. The weights $\mathbf{w}$ were learned using 10 policy updates of 100 episodes each. The bound was set to $\epsilon = 0.5$ for both sets of parameters.

The robot executed the motor primitives according to the phase transition sequences observed in the demonstrations, i.e., $1 \rightarrow 2 \rightarrow 3 \rightarrow 4$ and $1 \rightarrow 5 \rightarrow 3 \rightarrow 4$. Each sequence of motor primitives was executed 25 times using each of the three approaches, giving a total of 450 motor primitive executions. At the start of each sequence, the box was placed at a random position and orientation on the table within the robot's workspace. The successes and failures of the motor primitives were labelled by hand. A transition to phases 2 or 5 was considered successful if the robot's right or left hand made contact with the object. A transition to phase 3 was considered a success if the hands were touching opposite sides of the box, and the box was in contact with the table. A transition to phase 4 was considered a success if the robot succeeded in lifting the box from the table with both hands for more than 15 seconds.

The results of the experiment are shown in Fig. 4.5. All three methods learned to generalize to different object poses. Both reinforcement learning approaches succeeded in lifting the box in more than 90% of the trials. Using imitation learning, without additional reinforcement learning resulted in a success rate of only 38%. These motor primitives often resulted in a single finger tip making contact with the object, which lead to more delicate grasps. In comparison, the motor primitives learned using reinforcement learning pushed the box a few centimeters, which resulted in the object becoming aligned with the hand. The motor primitives for transitioning from phase 3 to 4 tended to apply less horizontal force to the object at the end of the motion than at the start. This may be a result of the demonstrations, which also tended to apply less force once the object was lifted from the table without dropping the object.

The results of the experiment showed that reinforcement learning leads to more robust motor primitives for transitioning between phases. The model captured a sufficient amount of detail to allow the robot to learn the required motor primitives. Apart from using more data, there are several ways that the model could be improved. Rather than using a fixed number of phases, a Dirichlet process prior could be included to model the number of phases [9, 10, 8]. The forward model could also be improved by explictly enforcing the contact constraints corresponding to each phase [149].

4.4.3 Guarded Motor Primitives

The motor primitives learned with deterministic sampling should be executed using a guarded DMP, such that the movement is terminated once a phase change is detected. Humans can accurately detect phase changes based on tactile sensations [7, 1]. However, as the robot does not have tactile sensors, we can only provide a preliminary study using the joint angles and torques of the robot's hand. Using the first principal components of this data, the robot learned a classifier for distinguishing between phase 3 and the previous phases 2 and 5. The robot then executed each of the two phase sequences ten times, and stopped the motor primitive if the classifier detected the phase transition. The robot succeeded in performing the lifting trial in 17 out of the 20 trials. The DMPs terminated on average 3.38 cm before reaching the goal state, with a standard deviation of 2.03 cm. As the robot was using impedance control, the goal states were 5.57cm inside the object, and the hands made contact with the object in all of the trials. By stopping the DMPs early, the robot simply applied less force to the object. The results of this test demonstrate that guarded motions can be incorporated into the proposed framework, and are a promising direction for further research.

4.4.4 Generalization to Other Objects

In order to evaluate how well the learned motor primitives generalize to new objects, the robot was additionally given the task of grasping six novel objects with different shapes and sizes. Each object was grasped three times with orientations at 45 intervals. The robot performed the grasps using the 1 → 2 → 3 → 4 phase sequence. The point cloud models of the objects were created by exploiting their extruded shapes [100]. The task frames were computed using these models.

The grasps are shown in Fig. 4.6 . The second and third grasps of the bottle failed, as a result of the bottle's tendency to tilt rather than slide across the table. The robot could lift the bottle in both of these trials, but the bottle slipped back down afterwards. The first and second grasps of the large green tin box also failed. The other 14 grasps were all successful and could be used to lift and hold the objects above the table.

As one would expect, the motor primitives are not applicable to objects with very different dynamics, such as the bottle. However, as the robot did succeed in briefly lifting the bottle, the learned DMPs could be used as a starting point for learning how to grasp and manipulate the bottle. The learned DMPs also have problems with low-friction contacts, such as those between the green tin can and the back of the

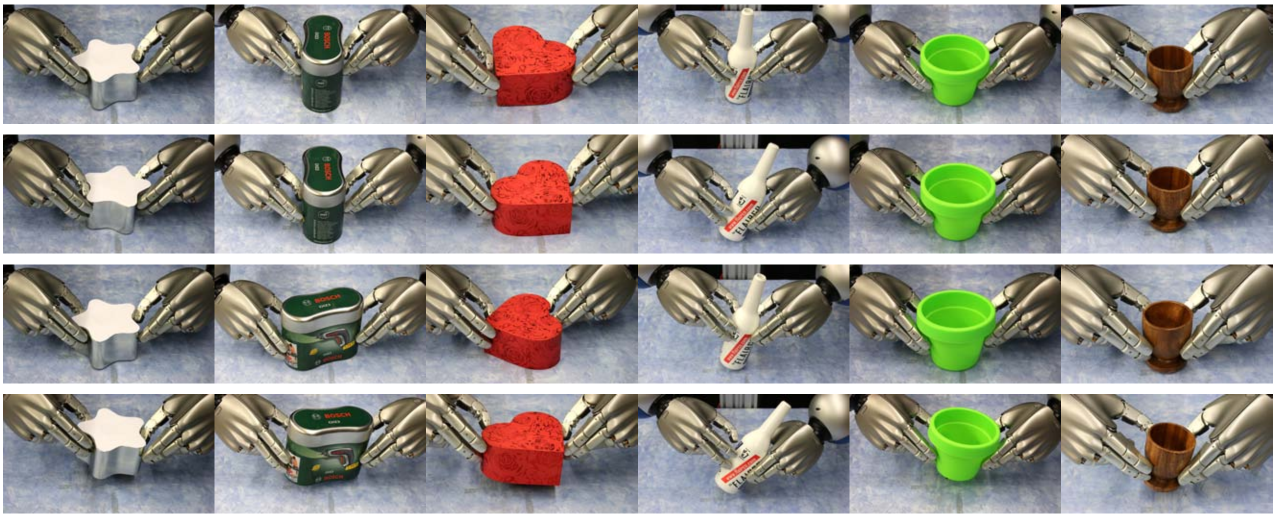


Figure 4.6: The figure shows the ability of the the learned DMPs to adapt to the geometry of different objects. The top three columns show example grasps for each of the six objects. The bottom row shows example lifts. The first two lifts of the metal box in column two failed due to material differences. The last two lifts of the bottle failed due to differences in dynamics. The other 14 grasps lead to successful lifts.

robot's thumb. This problem could be addressed by taking into consideration the material properties of the objects. The grasps could also be made more robust to slip by incorporating slip detection into the controller. With an overall success rate of 78%, the results of the experiment indicate that the learned motor primitives do generalize to other objects.

4.5 Conclusion

In this chapter, we presented a model for representing the phase structure of manipulation tasks. The most important part of this model is the dependency of the hidden phase on the observed state. This dependency allows the robot to learn the subgoals corresponding to the phase transitions. This transition model also results in more accurate multi-step predictions than standard ARHMMs [22]. The parameters of the model were learned using an expectation-maximization method.

The second contribution of this chapter is to learn a library of task-independent motor primitives for transitioning between phases. Rather than just using imitation learning, the robot used reinforcement learning to improve the motor primitives' performance and robustness. Instead of optimizing sequences of motor primitives for a specific task, the robot learned the motor primitives individually based on the subgoals defined by the phase model.

The proposed method was evaluated on a bimanual grasping task. The results of the experiments show that the policy search method significantly increases the performance of the learned motor primitives. The experiments also showed that the motor primitives generalize between objects with different shapes.

4.6 Potentially Helpful Insights

The motivation for this project was to model the structure of manipulation tasks in order to extract their inherent subgoals and constraints. At an abstract level, the goal of manipulations is to change variables of the environment to desired values. In order to change a variable, the robot must first obtain access to the variable, which it can do by setting other variables. Phases capture this concept of accessing different sets of variables. For example, in order to manipulate an object, the robot usually needs to first transition to a phase that allows it to apply forces to the object. Once the robot has access to a set of variables, it can change them to accomplish different subtasks. The robot can learn a separate skill for performing each of the subtasks. As a result, the manipulation skills tend to mirror the phase structure of the task.

Our original approach to learning skills for multi-phase manipulation tasks did not employ contact frames. Instead, the robot learned the goal and trajectory parameters as a linear function of the object's position. The robot learned to shift the goal states together with the object and it could thus successfully perform the task. However, as the shape of the object was not taken into consideration, the grasps tended to be rather crude and relied heavily on the compliance of the hands to adapt. Larger objects would be squeezed too tightly, and the hands would not make contact with smaller objects. In both cases, the fundamental problem was that the robot did not consider where the fingers were making contact with the object's surface. Establishing contacts between objects is also the core problem for grasp synthesis. We therefore employed a similar approach to determine the task/contact frames. However, rather than estimating hand poses for grasping, the robot selects a task frame for reaching the next phase. Grasping can thus be seen as a specific type of phase transition. As a result, a lot of ideas from grasping research can be reused for learning to transition between phases. Similarly, when developing methods for grasping, one should also consider which other phase transitions the method could be applied to.

The STARHMM is a forward model of the task, which the robot can use to simulate the effects of its actions and optimize its motor primitives accordingly. It can also use the model to predict the effects of other motor primitives, as shown in Chapter 5. However, unlike a standard simulator, the STARHMM also explicitly learns the phase structure and the conditions for transitioning between phases. As these transitions represent inherent subgoals, the robot is effectively learning to structure the skill learning problem. This structure comes from detecting specific instances of a more general task, i.e., transitioning between phases. Similarly, curiosity and intrinsically-motivated frameworks also give the robot general tasks, such as finding and controlling salient events or degrees of freedom in the environment [133, 134, 150]. Defining more general tasks gives the robot more autonomy and the ability to handle a larger range of tasks. It is also important for roboticist to try and define these more general problems in order to determine the fundamental challenges facing robots.

5 Sequencing Robot Manipulation Skills

A robot will often need to execute a sequence of motor primitives in order to perform a task. Each motor primitives may result in a specific manipulation of an object, or a transition to a phase wherein the desired manipulation can be performed. When selecting the next motor primitive to execute, the robot must take into consideration the current situation, the task, and the actions that it will perform in the future. The value function $V^{\pi}(\mathbf{s})$ models this information in a compact manner. The value function represents the future task rewards that the robot can expect to obtain, given its current state and its policy for selecting actions. Once the value function has been computed, the robot selects the motor primitive with the greatest immediate reward plus the expected value of the next state.

Defining a value function for continuous state spaces is, however, not trivial [11]. In this chapter, we present a non-parametric approach to computing value functions. The resulting value function representation has the form of a Nadaraya-Watson kernel regression [69, 70], and is flexible enough to model a wide range of value functions. In the evaluations, we show how the robot can learn to sequence motor primitives using the proposed value function approach.

5.1 A Non-Parametric Approach to Dynamic Programming

Value functions are an essential concept for determining optimal policies in both optimal control [151] and reinforcement learning [32, 152]. Given the value function of a policy, an improved policy is straightforward to compute. The improved policy can subsequently be evaluated to obtain a new value function. This loop of computing value functions and determining better policies is known as *policy iteration*. However, the main bottleneck in policy iteration is the computation of the value function for a given policy. Using the Bellman equation, only two classes of systems have been solved exactly: tabular discrete state and action problems [153] as well as linear-quadratic regulation problems [154]. The exact computation of the value function remains an open problem for most systems with continuous state spaces [11]. This chapter focuses on steps toward solving this problem.

As an alternative to exact solutions, approximate policy evaluation methods have been developed in reinforcement learning. These approaches include *Monte Carlo* methods, *temporal difference* learning, and *residual gradient* methods. However, Monte Carlo methods are well-known to have an excessively high variance [155, 32], and tend to overfit the value function to the sampled data [32]. When using function approximations, temporal difference learning can result in a biased solution[156]. Residual gradient approaches are biased unless multiple samples are taken from the same states [157], which is often not possible for real continuous systems.

In this chapter, we propose a non-parametric method for continuous-state policy evaluation. The proposed method uses a kernel density estimate to represent the system in a flexible manner. Model-based approaches are known to be more data efficient than direct methods, and lead to better policies [158, 159]. We subsequently show that the true value function for this model has a *Nadaraya-Watson kernel regression* form [69, 70]. Using Galerkin's projection method, we compute a closed-form solution for this regression problem. The resulting method is called *Non-Parametric Dynamic Programming* (NPDP), and is a stable as well as consistent approach to policy evaluation.

The second contribution of this section is to provide a unified view of several sample-based algorithms for policy evaluation, including the NPDP algorithm. In Section 5.3, we show how *Least-Squares Temporal Difference learning* (LSTD) in [12], *Kernelized Temporal Difference learning* (KTD) in [13], and *Discrete-State Dynamic Programming* (DSDP) in [153, 160] can all be derived using the same Galerkin projection method used to derive NPDP. In Section 5.4, we compare these methods using empirical evaluations.

In reinforcement learning, the uncontrolled system is usually represented by a Markov Decision Process (MDP). An MDP is defined by the following components: a set of states $\mathbb{S}$; a set of actions $\mathbb{A}$; a transition distribution $p(\mathbf{s}'|\mathbf{a},\mathbf{s})$, where $\mathbf{s}' \in \mathbb{S}$ is the next state given action $\mathbf{a} \in \mathbb{A}$ in state $\mathbf{s} \in \mathbb{S}$; a reward function r, such that $r(\mathbf{s},\mathbf{a})$ is the immediate reward obtained for performing action $\mathbf{a}$ in state $\mathbf{s}$; and a discount factor $\gamma \in [0,1)$ on future rewards. Actions $\mathbf{a}$ are selected according to the stochastic policy $\pi(\mathbf{a}|\mathbf{s})$. The goal is to maximize the discounted rewards that are obtained; i.e., $\max \sum_{t=0}^{\infty} \gamma^t \mathbf{r}(\mathbf{s}_t, \mathbf{a}_t)$. The term *system* will refer jointly to the agent's policy and the MDP.

The value of a state $V(\mathbf{s})$, for a specific policy π, is defined as the expected discounted sum of rewards that an agent will receive after visiting state $\mathbf{s}$ and executing policy π; i.e.,

$$V(\mathbf{s}) = E\left\{ \sum_{t=0}^{\infty} \gamma^t \mathbf{r}(\mathbf{s}_t, \mathbf{a}_t) \middle| \, \mathbf{s}_0 = \mathbf{s}, \pi \right\}. \tag{5.1}$$

By using the Markov property, Eq. (5.1) can be rewritten as the *Bellman equation*

$$V(\mathbf{s}) = \int_{\mathbb{A}} \int_{\mathbb{S}} \pi(\mathbf{a}|\mathbf{s}) \, p\left(\mathbf{s}'|\mathbf{s},\mathbf{a}\right) \left[r(\mathbf{s},\mathbf{a}) + \gamma V\left(\mathbf{s}'\right) \right] \mathrm{d}\mathbf{s}' \mathrm{d}\mathbf{a}. \tag{5.2}$$

The advantage of using the Bellman equation is that it describes the relationship between the value function at one state $\mathbf{s}$ and its immediate follow-up states $\mathbf{s}' \sim p(\mathbf{s}'|\mathbf{s},\mathbf{a})$. In contrast, the direct computation of Eq. (5.1) relies on the rewards obtained from entire trajectories.

5.2 Non-Parametric Model-based Dynamic Programming

We begin describing the NPDP approach by introducing the kernel density estimation framework used to represent the system. The true value function for this model has a kernel regression form, which can be computed by using Galerkin's projection method. We subsequently discuss some of the properties of this algorithm, including its consistency.

5.2.1 Non-Parametric System Modeling

The dynamics of a system are compactly represented by the joint distribution $p(\mathbf{s},\mathbf{a},\mathbf{s}')$. Using Bayes rule and marginalization, one can compute the transition probabilities $p(\mathbf{s}'|\mathbf{s},\mathbf{a})$ and the current policy $\pi(\mathbf{a}|\mathbf{s})$ from this joint distribution; e.g. $p(\mathbf{s}'|\mathbf{s},\mathbf{a}) = p(\mathbf{s},\mathbf{a},\mathbf{s}') / \int p(\mathbf{s},\mathbf{a},\mathbf{s}')\mathrm{d}\mathbf{s}'$. Rather than assuming that certain prior information is given, we will focus on the problem where only sampled information of the system is available. Hence, the system's joint distribution is modeled from a set of n samples obtained from the real system. The i^{th} sample includes the current state $\mathbf{s}_i \in \mathbb{S}$, the selected action $\mathbf{a}_i \in \mathbb{A}$, and the follow-up state $\mathbf{s}_i' \in \mathbb{S}$, as well as the immediate reward $r_i \in \mathbb{R}$. The state space $\mathbb{S}$ and the action space $\mathbb{A}$ are assumed to be continuous.

We propose using *kernel density estimation* to represent the joint distribution [161, 162] in a non-parametric manner. Unlike parametric models, non-parametric approaches use the collected data as features, which leads to accurate representations of arbitrary functions [163]. The system's joint distribution is therefore modeled as $p(\mathbf{s},\mathbf{a},\mathbf{s}') = n^{-1} \sum_{i=1}^{n} \psi_i(\mathbf{s}') \, \varphi_i(\mathbf{a}) \, \phi_i(\mathbf{s})$, where $\psi_i(\mathbf{s}') = \psi\left(\mathbf{s}', \mathbf{s}_i'\right)$, $\varphi_i(\mathbf{a}) = \varphi(\mathbf{a}, \mathbf{a}_i)$, and $\phi_i(\mathbf{s}) = \phi(\mathbf{s}, \mathbf{s}_i)$ are symmetric kernel functions. In practice, the kernel functions ψ and ϕ will often be the same. To ensure a valid probability density, each kernel must integrate to one; i.e., $\int \phi_i(\mathbf{s}) \mathrm{d}s = 1$, $\forall i$, and similarly for ψ and φ. As an additional constraint, the kernel must always be positive; i.e., $\psi_i(\mathbf{s}') \, \varphi_i(\mathbf{a}) \, \phi_i(\mathbf{s}) \geq 0$, $\forall \mathbf{s} \in \mathbb{S}$. This representation implies a factorization into separate $\psi_i(\mathbf{s}')$, $\varphi_i(\mathbf{a})$, and $\phi_i(\mathbf{s})$ kernels. As a result, an individual sample cannot express correlations between $\mathbf{s}'$, $\mathbf{a}$, and $\mathbf{s}$. However, the representation does allow multiple samples to express correlations between these components in $p(\mathbf{s},\mathbf{a},\mathbf{s}')$.

The reward function $r(\mathbf{s},\mathbf{a})$ must also be represented. Given the kernel density estimate representation, the expected reward for a state-action pair is denoted as [69]

$$r(\mathbf{s},\mathbf{a}) = E[r|\mathbf{s},\mathbf{a}] = \frac{\sum_{k=1}^{n} r_k \varphi_k(\mathbf{a})\,\phi_k(\mathbf{s})}{\sum_{i=1}^{n} \varphi_i(\mathbf{a})\,\phi_i(\mathbf{s})}.$$

Having specified the model of the system dynamics and rewards, the next step is to derive the corresponding value function.

5.2.2 Resulting Solution

In this section, we propose an approach to computing the value function for the continuous model specified in Section 5.2.1. Every policy has a unique value function, which fulfills the Bellman equation, Eq. (5.2), for all states [32, 164]. Hence, the goal is to solve the Bellman equation for the entire state space, and not just at the sampled states. This goal can be achieved by using the Galerkin projection method to compute the value function for the model [165].

The Galerkin method involves first projecting the integral equation into the space spanned by a set of basis functions. The integral equation is then solved in this projected space. To begin, the Bellman equation, Eq. (5.2), is rearranged as

$$\begin{aligned} V(\mathbf{s}) &= \int_{\mathbb{A}}\int_{\mathbb{S}} \pi(\mathbf{a}|\mathbf{s})\, r(\mathbf{s},\mathbf{a})\, p(\mathbf{s}'|\mathbf{s},\mathbf{a})\, \mathrm{d}\mathbf{s}'\mathrm{d}\mathbf{a} + \int_{\mathbb{S}}\int_{\mathbb{A}} p(\mathbf{s}'|\mathbf{s},\mathbf{a})\, \gamma V(\mathbf{s}')\, \pi(\mathbf{a}|\mathbf{s})\, \mathrm{d}\mathbf{a}\mathrm{d}\mathbf{s}', \\ p(\mathbf{s})\,V(\mathbf{s}) &= \int_{\mathbb{A}} p(\mathbf{a},\mathbf{s})\, r(\mathbf{s},\mathbf{a})\, \mathrm{d}\mathbf{a} + \gamma \int_{\mathbb{S}} p(\mathbf{s}',\mathbf{s})\, V(\mathbf{s}')\, \mathrm{d}\mathbf{s}'. \end{aligned} \tag{5.3}$$

Before applying the Galerkin method, we derive the exact form of the value function. Expanding the reward function and joint distributions, as defined in Section 5.2.1, gives

$$\begin{aligned} p(\mathbf{s})\,V(\mathbf{s}) &= n^{-1}\int_{\mathbb{A}} \sum_{k=1}^{n} \varphi_k(\mathbf{a})\,\phi_k(\mathbf{s}) \frac{\sum_{i=1}^{n} r_i \varphi_i(\mathbf{a})\,\phi_i(\mathbf{s})}{\sum_{j=1}^{n} \varphi_j(\mathbf{a})\,\phi_j(\mathbf{s})} \mathrm{d}\mathbf{a} + \gamma \int_{\mathbb{S}} p(\mathbf{s}',\mathbf{s})\, V(\mathbf{s}')\, \mathrm{d}\mathbf{s}', \\ p(\mathbf{s})\,V(\mathbf{s}) &= \int_{\mathbb{A}} n^{-1}\sum_{i=1}^{n} r_i \varphi_i(\mathbf{a})\,\phi_i(\mathbf{s})\,\mathrm{d}\mathbf{a} + \gamma \int_{\mathbb{S}} n^{-1}\sum_{i=1}^{n} \psi_i(\mathbf{s}')\,\phi_i(\mathbf{s})\,V(\mathbf{s}')\,\mathrm{d}\mathbf{s}', \\ p(\mathbf{s})\,V(\mathbf{s}) &= n^{-1}\sum_{i=1}^{n} r_i \phi_i(\mathbf{s}) + n^{-1}\sum_{i=1}^{n} \gamma \int_{\mathbb{S}} \psi_i(\mathbf{s}')\,\phi_i(\mathbf{s})\,V(\mathbf{s}')\,\mathrm{d}\mathbf{s}', \end{aligned}$$

Therefore, $p(\mathbf{s})V(\mathbf{s}) = n^{-1}\sum_{i=1}^{n} \theta_i \phi_i(\mathbf{s})$, where θ are value weights. Given that $p(\mathbf{s}) = n^{-1}\sum_{j=1}^{n} \phi_j(\mathbf{s})$, the true value function of the kernel density estimate system has a Nadaraya-Watson kernel regression [69, 70] form

$$V(\mathbf{s}) = \frac{\sum_{i=1}^{n} \theta_i \phi_i(\mathbf{s})}{\sum_{j=1}^{n} \phi_j(\mathbf{s})}. \tag{5.4}$$

Having computed the true form of the value function, the Galerkin projection method can be used to compute the value weights θ. The projection is performed by taking the expectation of the integral equation with respect to each of the n basis function ϕ_i. The resulting n simultaneous equations can be written as the vector equation

$$\int_{\mathbb{S}} \boldsymbol{\phi}(\mathbf{s})\, p(\mathbf{s})V(\mathbf{s})\mathrm{d}\mathbf{s} = \int_{\mathbb{S}} \boldsymbol{\phi}(\mathbf{s})\, n^{-1} \boldsymbol{\phi}(\mathbf{s})^T \mathbf{r}\mathrm{d}\mathbf{s} + \gamma \int_{\mathbb{S}}\int_{\mathbb{S}} \boldsymbol{\phi}(\mathbf{s})\, n^{-1}\left(\boldsymbol{\phi}(\mathbf{s})^T \boldsymbol{\psi}(\mathbf{s}')\right) V(\mathbf{s}')\mathrm{d}\mathbf{s}'\mathrm{d}\mathbf{s},$$

where the i$^{\text{th}}$ elements of the vectors are given by $[\mathbf{r}]_i = r_i$, $[\boldsymbol{\phi}(\mathbf{s})]_i = \phi_i(\mathbf{s})$, and $[\boldsymbol{\psi}(\mathbf{s}')]_i = \psi_i(\mathbf{s}')$. Expanding the value functions gives

Algorithm 2 Non-Parametric Dynamic Programming

```
INPUT:
    n system samples:
        state $\mathbf{s}_i$, next state $\mathbf{s}'_i$, and reward $r_i$
    Kernel functions:
        $\phi_i(\mathbf{s}_j) = \phi(\mathbf{s}_i, \mathbf{s}_j)$, and $\psi_i(\mathbf{s}'_j) = \psi(\mathbf{s}'_i, \mathbf{s}'_j)$
    Discount factor:
        $0 \leq \gamma < 1$
COMPUTATION:
    Reward vector:
        $[\mathbf{r}]_i = r_i$
    Transition matrix:
        $[\boldsymbol{\lambda}]_{i,j} = \int_{\mathbb{S}} \frac{\phi_j(\mathbf{s}')\psi_i(\mathbf{s}')}{\sum_{k=1}^n \phi_k(\mathbf{s}')} d\mathbf{s}'$
    Value weights:
        $\boldsymbol{\theta} = (\mathbf{I} - \gamma\boldsymbol{\lambda})^{-1}\mathbf{r}$
OUTPUT:
    Value function:    $V(\mathbf{s}) = \frac{\sum_{i=1}^n \theta_i \phi_i(\mathbf{s})}{\sum_{j=1}^n \phi_j(\mathbf{s})}$
```

$$\int_{\mathbb{S}} \boldsymbol{\phi}(\mathbf{s})\boldsymbol{\phi}(\mathbf{s})^T \boldsymbol{\theta} \, d\mathbf{s} = \int_{\mathbb{S}} \boldsymbol{\phi}(\mathbf{s})\boldsymbol{\phi}(\mathbf{s})^T \mathbf{r} \, d\mathbf{s} + \gamma \int_{\mathbb{S}}\int_{\mathbb{S}} \boldsymbol{\phi}(\mathbf{s})\left(\boldsymbol{\phi}(\mathbf{s})^T \boldsymbol{\psi}(\mathbf{s}')\right) \frac{\boldsymbol{\phi}(\mathbf{s}')^T \boldsymbol{\theta}}{\sum_{i=1}^n \phi_i(\mathbf{s}')} d\mathbf{s}' d\mathbf{s},$$

$$\mathbf{C}\boldsymbol{\theta} = \mathbf{C}\mathbf{r} + \gamma\mathbf{C}\boldsymbol{\lambda}\boldsymbol{\theta},$$

where $\mathbf{C} = \int_{\mathbb{S}} \boldsymbol{\phi}(\mathbf{s})\boldsymbol{\phi}(\mathbf{s})^T d\mathbf{s}$, and $\boldsymbol{\lambda} = \int_{\mathbb{S}} (\sum_{i=1}^n \phi_i(\mathbf{s}'))^{-1} \boldsymbol{\psi}(\mathbf{s}')\boldsymbol{\phi}(\mathbf{s}')^T d\mathbf{s}'$ is a stochastic matrix; i.e., a transition matrix. The matrix **C** can become singular if two basis functions are coincident. In such cases, there exists an infinite set of solutions for $\boldsymbol{\theta}$. However, all of the solutions result in identical values. The NPDP algorithm uses the solution given by

$$\boldsymbol{\theta} = (\mathbf{I} - \gamma\boldsymbol{\lambda})^{-1}\mathbf{r}, \tag{5.5}$$

which always exists for any stochastic matrix $\boldsymbol{\lambda}$. Thus, the derivation has shown that the exact value function for the model in Section 5.2.1 has a Nadaraya-Watson kernel regression form, as shown in Eq. (5.4), with weights $\boldsymbol{\theta}$ given by Eq. (5.5). The non-parametric dynamic programming algorithm is summarized in Algorithm 2 . The NPDP algorithm ultimately requires only the state information **s** and $\mathbf{s}'$, and not the actions **a**. In Section 5.3, we will show how this form of derivation can also be used to derive the LSTD, KTD, and DSDP algorithms.

5.2.3 Properties of the NPDP Algorithm

We will now discuss some of the key properties of the proposed NPDP algorithm, including precision, accuracy, and computational complexity. Precision refers to how close the predicted value function is to the true value function of the model, while accuracy refers to how close the model is to the true system.

One of the key contributions of this chapter is providing the true form of the value function for policy evaluation with the non-parametric model described in Section 5.2.1. The parameters of this value function can be computed precisely by solving Eq. (5.5). Even if $\boldsymbol{\lambda}$ is evaluated numerically, a high level of precision can still be obtained.

As a non-parametric method, the accuracy of the NPDP algorithm depends on the number of samples obtained from the system. It is important that the model, and thus the value function, converges to that of the true system as the number of samples increases; i.e., that the model is statistically consistent. In fact, kernel density estimation can be proven to have almost sure convergence to the true distribution for a wide range of kernels [166].

Given that $\boldsymbol{\lambda}$ is a stochastic matrix and $0 \leq \gamma < 1$, it is well-known that the inversion of $(\mathbf{I} - \gamma\boldsymbol{\lambda})$ is well-defined [160]. The inversion can therefore also be expanded according to the Neumann series;

i.e., $\boldsymbol{\theta} = \sum_{i=0}^{\infty}[\gamma\boldsymbol{\lambda}]^i\mathbf{r}$. Similar to other kernel-based policy evaluation methods [167, 168], NPDP has a computational complexity of $\mathcal{O}(n^3)$ when performed naively. However, by taking advantage of sparse matrix computations, this complexity can be reduced to $\mathcal{O}(nz)$, where z is the number of non-zero elements in $(\mathbf{I}-\gamma\boldsymbol{\lambda})$.

5.3 Relation to Existing Methods

The second contribution of this chapter is to provide a unified view of Least Squares Temporal Difference learning (LSTD), Kernelized Temporal Difference learning (KTD), Discrete-State Dynamic Programming (DSDP), and the proposed Non-Parametric Dynamic Programming (NPDP). In this section, we utilize the Galerkin methodology from Section 5.2.2 to re-derive the LSTD, KTD, and DSDP algorithms, and discuss how these methods compare to NPDP. A numerical comparison is given in Section 5.4.

5.3.1 Least Squares Temporal Difference Learning

The LSTD algorithm allows the value function $V(\mathbf{s})$ to be represented by a set of m arbitrary basis functions $\hat{\phi}_i(\mathbf{s})$, see [12]. Hence, $V(\mathbf{s}) = \sum_{i=1}^{m}\hat{\theta}_i\hat{\phi}_i(\mathbf{s}) = \hat{\boldsymbol{\phi}}(\mathbf{s})^T\hat{\boldsymbol{\theta}}$, where $\hat{\boldsymbol{\theta}}$ is a vector of coefficients learned during policy evaluation, and $[\hat{\boldsymbol{\phi}}(\mathbf{s})]_i = \hat{\phi}_i(\mathbf{s})$. In order to re-derive the LSTD policy evaluation, the joint distribution is represented as a set of delta functions $p(\mathbf{s},\mathbf{a},\mathbf{s}') = n^{-1}\sum_{i=1}^{n}\delta_i(\mathbf{s},\mathbf{a},\mathbf{s}')$, where $\delta_i(\mathbf{s},\mathbf{a},\mathbf{s}')$ is a Dirac delta function centered on $(\mathbf{s}_i,\mathbf{a}_i,\mathbf{s}_i')$. Using Galerkin's method, the integral equation is projected into the space of the basis functions $\hat{\phi}(\mathbf{s})$. Thus, Eq. (5.3) becomes

$$\int_{\mathbb{S}}\hat{\boldsymbol{\phi}}(\mathbf{s})\,p(\mathbf{s})\,\hat{\boldsymbol{\phi}}(\mathbf{s})^T\hat{\boldsymbol{\theta}}\,\mathrm{d}\mathbf{s} = \int_{\mathbb{A}}\int_{\mathbb{S}}\hat{\boldsymbol{\phi}}(\mathbf{s})\,p(\mathbf{s},\mathbf{a})\,r(\mathbf{s},\mathbf{a})\,\mathrm{d}\mathbf{s}\mathrm{d}\mathbf{a} + \gamma\int_{\mathbb{S}}\hat{\boldsymbol{\phi}}(\mathbf{s})\,p\left(\mathbf{s},\mathbf{s}'\right)\hat{\boldsymbol{\phi}}\left(\mathbf{s}'\right)^T\hat{\boldsymbol{\theta}}\,\mathrm{d}\mathbf{s}'\mathrm{d}\mathbf{s},$$

$$\sum_{i=1}^{n}\hat{\boldsymbol{\phi}}(\mathbf{s}_i)\,\hat{\boldsymbol{\phi}}(\mathbf{s}_i)^T\hat{\boldsymbol{\theta}} = \sum_{j=1}^{n}r\left(\mathbf{s}_\mathrm{j},\mathbf{a}_j\right)\hat{\boldsymbol{\phi}}\left(\mathbf{s}_j\right) + \gamma\sum_{k=1}^{n}\hat{\boldsymbol{\phi}}(\mathbf{s}_k)\,\hat{\boldsymbol{\phi}}\left(\mathbf{s}_k'\right)^T\hat{\boldsymbol{\theta}},$$

$$\sum_{i=1}^{n}\hat{\boldsymbol{\phi}}(\mathbf{s}_i)\left(\hat{\boldsymbol{\phi}}(\mathbf{s}_i)^T - \gamma\hat{\boldsymbol{\phi}}\left(\mathbf{s}_i'\right)^T\right)\hat{\boldsymbol{\theta}} = \sum_{j=1}^{n}r\left(\mathbf{s}_\mathrm{j},\mathbf{a}_j\right)\hat{\boldsymbol{\phi}}\left(\mathbf{s}_j\right),$$

and thus $\mathbf{A}\hat{\boldsymbol{\theta}} = \mathbf{b}$, where $\mathbf{A} = \sum_{i=1}^{n}\hat{\boldsymbol{\phi}}(\mathbf{s}_i)(\hat{\boldsymbol{\phi}}(\mathbf{s}_i)^T - \gamma\hat{\boldsymbol{\phi}}\left(\mathbf{s}_i'\right)^T)$ and $\mathbf{b} = \sum_{j=1}^{n}r\left(\mathbf{s}_\mathrm{j},\mathbf{a}_j\right)\hat{\boldsymbol{\phi}}\left(\mathbf{s}_j\right)$. The final weights are therefore given by

$$\hat{\boldsymbol{\theta}} = \mathbf{A}^{-1}\mathbf{b}.$$

This equation is also solved by LSTD, including the incremental updates of $\mathbf{A}$ and $\mathbf{b}$ as new samples are acquired [12]. Therefore, LSTD can be seen as computing the transitions between the basis functions using a Monte Carlo approach. However, Monte Carlo methods rely on large numbers of samples to obtain accurate results.

A key disadvantage of the LSTD method is the need to select a specific set of basis functions. The computed value function will always be a projection of the true value function into the space of these basis functions [156]. If the true value function does not lie within the space of these basis functions, the resulting approximation may be arbitrarily inaccurate, regardless of the number of acquired samples. However, using predefined basis functions only requires inverting an $m \times m$ matrix, which results in a lower computational complexity than NPDP.

The LSTD may also need to be regularized, as the inversion of $\mathbf{A}$ becomes ill-posed if the basis functions are too densely spaced. Regularization has a similar effect to changing the transition probabilities of the system [169].

5.3.2 Kernelized Temporal Difference Learning Methods

The proposed approach is of course not the first to use kernels for policy evaluation. Methods such as kernelized least-squares temporal difference learning [168] and Gaussian process temporal difference learning [167] have also employed kernels in policy evaluation. Taylor and Parr demonstrated that these methods differ mainly in their use of regularization [13]. The unified view of these methods is referred to as Kernelized Temporal Difference learning.

The KTD approach assumes that the reward and value functions can be represented by kernelized linear least-squares regression; i.e., $r(\mathbf{s}) = \mathbf{k}(\mathbf{s})^T\mathbf{K}^{-1}\mathbf{r}$ and $V(\mathbf{s}) = \mathbf{k}(\mathbf{s})^T\hat{\boldsymbol{\theta}}$, where $[\mathbf{k}(\mathbf{s})]_i = k(\mathbf{s},\mathbf{s}_i)$, $[\mathbf{K}]_{ij} = k(\mathbf{s}_i,\mathbf{s}_j)$, $[\mathbf{r}]_i = r_i$, and $\hat{\boldsymbol{\theta}}$ is a weight vector. In order to derive KTD using Galerkin's method, it is necessary to again represent the joint distribution as $p(\mathbf{s},\mathbf{a},\mathbf{s}') = n^{-1}\sum_{i=1}^{n}\delta_i(\mathbf{s},\mathbf{a},\mathbf{s}')$. The Galerkin method projects the integral equation into the space of the Kronecker delta functions $[\check{\boldsymbol{\delta}}(\mathbf{s})]_i = \check{\delta}_i(\mathbf{s},\mathbf{a}_i,\mathbf{s}_i')$, where $\check{\delta}_i(\mathbf{s},\mathbf{a},\mathbf{s}') = 1$ if $\mathbf{s}' = \mathbf{s}_i'$, $\mathbf{a} = \mathbf{a}_i$, and $\mathbf{s} = \mathbf{s}_i$; otherwise $\check{\delta}_i(\mathbf{s},\mathbf{a},\mathbf{s}') = 0$. Thus, Eq. (5.3) becomes

$$\int_{\mathbb{S}} \check{\boldsymbol{\delta}}(\mathbf{s})\, p(\mathbf{s})\, \mathbf{k}(\mathbf{s})^T\hat{\boldsymbol{\theta}}\, d\mathbf{s} = \int_{\mathbb{S}} \check{\boldsymbol{\delta}}(\mathbf{s})\, p(\mathbf{s})\, r(\mathbf{s})\, d\mathbf{s} + \gamma \int_{\mathbb{S}} \check{\boldsymbol{\delta}}(\mathbf{s})\, p\left(\mathbf{s},\mathbf{s}'\right) \mathbf{k}(\mathbf{s}')^T\hat{\boldsymbol{\theta}}\, d\mathbf{s}' d\mathbf{s},$$

By substituting $p(\mathbf{s},\mathbf{a},\mathbf{s}')$ and applying the sifting property of delta functions, this equation becomes

$$\sum_{i=1}^{n} \check{\boldsymbol{\delta}}(\mathbf{s}_i)\mathbf{k}(\mathbf{s}_i)^T\hat{\boldsymbol{\theta}} = \sum_{j=1}^{n} \check{\boldsymbol{\delta}}(\mathbf{s}_j)\mathbf{k}(\mathbf{s}_j)^T\mathbf{K}^{-1}\mathbf{r} + \gamma \sum_{k=1}^{n} \check{\boldsymbol{\delta}}(\mathbf{s}_k)\mathbf{k}(\mathbf{s}_k')^T\hat{\boldsymbol{\theta}},$$

and thus $\mathbf{K}\hat{\boldsymbol{\theta}} = \mathbf{r} + \gamma\mathbf{K}'\hat{\boldsymbol{\theta}}$, where $[\mathbf{K}']_{ij} = k(\mathbf{s}_i',\mathbf{s}_j)$. The value function weights are therefore

$$\hat{\boldsymbol{\theta}} = (\mathbf{K} - \gamma\mathbf{K}')^{-1}\mathbf{r},$$

which is identical to the solution found by the KTD approach [13]. In this manner, the KTD approach computes a weighting $\hat{\boldsymbol{\theta}}$ such that the difference in the value at $\mathbf{s}_i$ and the discounted value at $\mathbf{s}_i'$ equals the observed empirical reward r_i. Thus, only the finite set of sampled states are regarded for policy evaluation. Therefore, some KTD methods, e.g. Gaussian process temporal difference learning [167], require that the samples are obtained from a single trajectory to ensure that $\mathbf{s}_i' = \mathbf{s}_{i+1}$.

A key difference between KTD and NPDP is the representation of the value function $V(\mathbf{s})$. The form of the value function is a direct result of the representation used to embody the state transitions. In the original paper [13], the KTD algorithm represents the transitions by using linear kernelized regression $\hat{\mathbf{k}}(\mathbf{s}') = \mathbf{k}(\mathbf{s})^T\mathbf{K}^{-1}\mathbf{K}'$, where $[\hat{\mathbf{k}}(\mathbf{s}')]_i = \mathbb{E}[k(\mathbf{s}',\mathbf{s}_i)]$. The value function $V(\mathbf{s}) = \mathbf{k}(\mathbf{s})^T\hat{\boldsymbol{\theta}}$ is the correct form for this transition model. However, the transition model does not explicitly represent a conditional distribution and can lead to inaccurate predictions. For example, consider two samples that start at $s_1 = 0$ and $s_2 = 0.75$ respectively, and both transition to $s' = 0.75$. For clarity, we use a box-cart kernel with a width of one $k(s_i,s_j) = 1$ iff $\left\|s_i - s_j\right\| \leq 0.5$ and 0 otherwise. Hence, $\mathbf{K} = \mathbf{I}$ and each row of $\mathbf{K}$' corresponds to $(0,1)$. In the region $0.25 \leq s \leq 0.5$, where the two kernels overlap, the transition model would then predict $\hat{k}(s) = \mathbf{k}(\mathbf{s})^T\mathbf{K}^{-1}\mathbf{K}' = [\begin{array}{cc} 0 & 2 \end{array}]$. This prediction is however impossible as it requires that $\mathbb{E}[k(s',s_2)] > \max_s k(s,s_2)$. In comparison, NPDP would predict the distribution $\psi(s') \equiv \psi_1(s') \equiv \psi_2(s')$ for all states in the range $-0.5 \leq s \leq 1.25$.

Similar as for LSTD, the matrix $(\mathbf{K} - \gamma\mathbf{K}')$ may become singular and thus not be invertible. As a result, KTD usually needs to be regularized [13]. Given that KTD requires inverting an $n \times n$ matrix, this approach has a computational complexity similar to NPDP.

5.3.3 Discrete-State Dynamic Programming

The standard tabular DSDP approach can also be derived using the Galerkin method. Given a system with q discrete states, the value function has the form $V(\mathbf{s}) = \check{\boldsymbol{\delta}}(\mathbf{s})^T \mathbf{v}$, where $\check{\boldsymbol{\delta}}(\mathbf{s})$ is a vector of q Kronecker delta functions centered on the discrete states. The corresponding reward function is $r(\mathbf{s}) = \check{\boldsymbol{\delta}}(\mathbf{s})^T \bar{\mathbf{r}}$. The joint distribution is given by $p(\mathbf{s}', \mathbf{s}) = q^{-1} \boldsymbol{\delta}(\mathbf{s})^T \mathbf{P} \boldsymbol{\delta}(\mathbf{s}')$, where $\mathbf{P}$ is a stochastic matrix $\sum_{j=1}^{q} [\mathbf{P}]_{ij} = 1$, $\forall i$ and hence $p(\mathbf{s}) = q^{-1} \sum_{i=1}^{q} \delta_i(\mathbf{s})$. Galerkin's method projects the integral equation into the space of the states $\check{\boldsymbol{\delta}}(\mathbf{s})$. Thus, Eq. (5.3) becomes

$$\int_{\mathbb{S}} \check{\boldsymbol{\delta}}(\mathbf{s})\, p(\mathbf{s})\, \check{\boldsymbol{\delta}}(\mathbf{s})^T \mathbf{v} d\mathbf{s} = \int_{\mathbb{S}} \check{\boldsymbol{\delta}}(\mathbf{s})\, p(\mathbf{s})\, \check{\boldsymbol{\delta}}(\mathbf{s})^T \bar{\mathbf{r}} d\mathbf{s} + \gamma \int_{\mathbb{S}} \check{\boldsymbol{\delta}}(\mathbf{s})\, p\left(\mathbf{s}, \mathbf{s}'\right) \check{\boldsymbol{\delta}}(\mathbf{s}')^T \mathbf{v} d\mathbf{s}' d\mathbf{s},$$

$$\mathbf{I}\mathbf{v} = \mathbf{I}\bar{\mathbf{r}} + \gamma \int_{\mathbb{S}} \check{\boldsymbol{\delta}}(\mathbf{s})\, \boldsymbol{\delta}(\mathbf{s})^T \mathbf{P} \boldsymbol{\delta}(\mathbf{s}') \check{\boldsymbol{\delta}}(\mathbf{s}')^T \mathbf{v} d\mathbf{s}' d\mathbf{s},$$

$$\mathbf{v} = \bar{\mathbf{r}} + \gamma \mathbf{P} \mathbf{v},$$

$$\mathbf{v} = (\mathbf{I} - \gamma \mathbf{P})^{-1} \bar{\mathbf{r}}, \tag{5.6}$$

which is the same computation used by DSDP [160]. The DSDP and NPDP methods actually use similar models to represent the system. While NPDP uses a kernel density estimation, the DSDP algorithm uses a histogram representation. Hence, DSDP can be regarded as a special case of NPDP for discrete state systems.

The DSDP algorithm has also been the basis for continuous-state policy evaluation algorithms [170, 171]. These algorithms first use the sampled states as the discrete states of an MDP and compute the corresponding values. The computed values are then generalized, under a smoothness assumption, to the rest of the state-space using local averaging. Unlike these methods, NPDP explicitly performs policy evaluation for a continuous set of states.

5.4 Numerical Evaluation

In this section, we compare the different policy evaluation methods discussed in the previous section, with the proposed NPDP method, on an illustrative benchmark system.

5.4.1 Benchmark Problem and Setup

In order to compare the LSTD, KTD, DSDP, and NPDP approaches, we evaluated the methods on a discrete-time continuous-state system. A standard linear-Gaussian system was used for the benchmark problem, with transitions given by $s' = 0.95s + \omega$ where ω is Gaussian noise $\mathcal{N}(\mu = 0, \sigma = 0.025)$. The initial states are restricted to the range 0.95 to 1. The reward functions consist of three Gaussians, as shown by the black line in Fig. 5.1 .

The KTD method was implemented using a Gaussian kernel function and regularization. The LSTD algorithm was implemented using 15 uniformly-spaced normalized Gaussian basis functions, and did not require regularization. The DSDP method was implemented by discretizing the state-space into 10 equally wide regions. The NPDP method was also implemented using Gaussian kernels.

The hyper-parameters of all four methods, including the number of basis functions for LSTD and DSDP, were carefully tuned to achieve the best performance. As a performance base-line, the values of the system in the range $0 < s < 1$ were computed using a Monte Carlo estimate based on 50000 trajectories. The policy evaluations performed by the tested methods were always based on only 500 samples; i.e. 100 times less samples than the base-line. The experiment was run 500 times using independent sets of 500 samples. The samples were not drawn from the same trajectory.

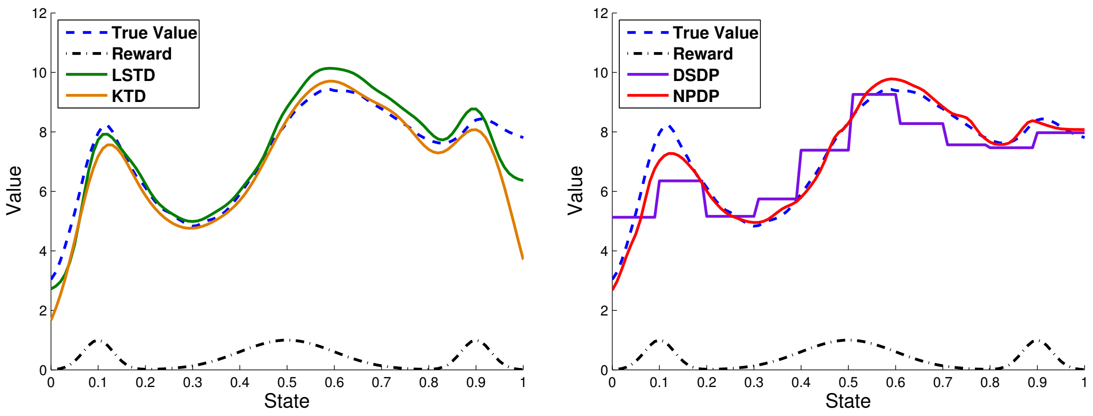


Figure 5.1: Value functions obtained by the evaluated methods. The black lines show the reward function. The blue lines show the value function computed from the trajectories of 50,000 uniformly sampled points. The LSTD, KTD, DSDP, and NPDP methods evaluated the policy using only 500 points. The presentation was divided into two plots for improved clarity

5.4.2 Results

The performance of the different methods were compared using three performance measures. Two of the performance measures are based on the weighted *Mean Squared Error* (MSE) [32] $\mathrm{E}(V) = \int_0^1 W(s)\,(V(s) - V^\star(s))^2\,\mathrm{d}s$ where $V^\star$ is the true value function and $W(s) \geq 0$, for all states, is a weighting distribution $\int_0^1 W(s)\mathrm{d}s = 1$. The first performance measure E_{unif} corresponds to the MSE where $W(s) = 1$ for all states in the range zero to one. The second performance measure E_{samp} corresponds to the MSE where $W(s) = n^{-1}\Sigma_{i=1}^{n}\delta_i(s)$ respectively. Thus, E_{samp} is an indicator of the accuracy in the space of the samples, while E_{unif} is an indicator of how well the computed value function generalizes to the entire state space. The third performance measure E_{max} is given by the maximum error in the value function. This performance measure is the basis of a bound on the overall value function approximation [164].

The results of the experiment are shown in Table 5.1. The performance measures were averaged over the 500 independent trials of the experiment. For all three performance measures, the NPDP algorithm achieved the highest levels of performance, while the DSDP approach consistently led to the worst performance.

	$\mathbf{E}_{\text{unif}}$	$\mathbf{E}_{\text{samp}}$	$\mathbf{E}_{\text{max}}$
NPDP	**0.5811** ± 0.0333	**0.7185** ± 0.0321	**1.4971** ± 0.0309
LSTD	0.6898 ± 0.0443	0.8932 ± 0.0412	1.5591 ± 0.0382
KTD	0.7585 ± 0.0460	0.8681 ± 0.0270	2.5329 ± 0.0391
DSDP	1.6979 ± 0.0332	2.1548 ± 0.1082	2.9985 ± 0.0449

Table 5.1: Each row corresponds to one of the four tested algorithms for policy evaluation. The columns indicate the performance of the approaches during the experiment. The performance indexes include the mean squared error evaluated uniformly over the zero to one range, the mean squared error evaluated at the 500 sampled points, and the maximum error. The results are averaged over 500 trials. The standard errors of the means are also given.

5.4.3 Discussion

The LSTD algorithm achieved a relatively low E_{unif} value, which indicates that the tuned basis functions could accurately represent the true value function. However, the performance of LSTD is sensitive to the choice of basis functions and the number of samples per basis function. Using 20 basis functions instead of 15 reduces the performance of LSTD to $E_{\text{unif}} = 2.8705$ and $E_{\text{samp}} = 1.0256$ as a result of overfitting. The KTD method achieved the second best performance for E_{samp}, as a result of using a non-parametric representation. However, the value tended to drop in sparsely-sampled regions, which lead to relatively high E_{unif} and E_{max} values. The discretization of states for DSDP is generally a disadvantage when modeling continuous systems, and resulted in poor overall performance for this evaluation. The NPDP approach out-performed the other methods in all three performance measures. The performance of NPDP could be further improved by using adaptive kernel density estimation [172] to locally adapt the kernels' bandwidths according to the sampling density. However, all methods were restricted to using a single global bandwidth for the purpose of this comparison.

5.5 Sequencing Motor Primitives

In this section, we explain how the proposed NPDP framework can be used to learn a high-level policy for selecting motor primitives. This section builds on the work presented in Chapter 4. We assume that the robot has learned a library of motor primitives for transitioning between different phases, and that it has access to a multi-phase model of the task.

5.5.1 Learning a High-Level Policy with Policy Iteration

Given a library of motor primitives, such as the one learned in the previous chapter, the robot must now sequence these motor primitives $\mathscr{M}$ in order to perform different manipulation tasks. When selecting the next motor primitive, the robot must take into consideration the goal of the task, the current state, and its future actions. The robot should also reuse motor primitives when performing similar tasks.

We assume that the task is defined in the form of a reward function $r(\mathbf{s}, \mathbf{a})$, which we can use to compute an expected reward function $r(\tilde{\mathbf{s}}, \mathscr{M})$ for motor primitives, where the extended state $\tilde{\mathbf{s}}$ includes the state $\mathbf{s}$ and the robot's estimate of the phase $\tilde{\rho}$. The robot's high-level policy $\pi(\mathscr{M}|\tilde{\mathbf{s}})$ selects motor primitives $\mathscr{M}$, according to the current state and phase. The motor primitives should be selected such that they maximize the reward accumulated over time

$$\max_{\pi} \sum_{t=1}^{\infty} \gamma^t r(\tilde{\mathbf{s}}(t), \mathscr{M}(t)),$$

where γ is a discount factor on future rewards $0 \leq \gamma < 1$ and t indicates the steps in the motor primitive sequence. In our experiments, all of the motor primitives had the same duration. However, one could also use different discount factors for motor primitives with different durations. The value function $V^{\pi}(\tilde{\mathbf{s}})$ is defined as the expected future rewards when in state $\mathbf{s}$ and phase ρ, and following policy π. The value function therefore represents how useful it is to be in the extended state $\tilde{\mathbf{s}}$ given the current task and the robot's policy for selecting future actions.

The NPDP algorithm models the system dynamics as a kernel density estimate of the form

$$p(\tilde{\mathbf{s}}, \mathscr{M}, \tilde{\mathbf{s}}') = m^{-1} \textstyle\sum_{i=1}^{m} \psi\left(\tilde{\mathbf{s}}', \tilde{\mathbf{s}}_i'\right) \varphi\left(\mathscr{M}, \mathscr{M}_i\right) \phi\left(\tilde{\mathbf{s}}, \tilde{\mathbf{s}}_i\right)$$

where $\tilde{\mathbf{s}}$ is the extended state before executing motor primitive $\mathscr{M}$, and $\tilde{\mathbf{s}}'$ is the extended state afterwards. We model $\phi\left(\tilde{\mathbf{s}}, \tilde{\mathbf{s}}_i\right)$ using a squared exponential kernel for the state $\mathbf{s}$ and multiply it by a

Kronecker delta function for the phase $\tilde{\rho}$. As we have a discrete set of motor primitives, the function φ is defined as a delta function. The function $\psi\left(\tilde{\mathbf{s}}', \tilde{\mathbf{s}}_i'\right)$ is not explicitly defined and instead approximated using samples from the multi-phase model described in Chapter 4.

In order to learn the value function, we first compute a set of m prototypical samples $\{\tilde{\mathbf{s}}_i\}$ for $i \in \{1, ..., m\}$. Given a set of starting states, the prototype samples can be obtained by sampling different sequences of motor primitives using the multi-phase model. In our experiments, we sampled every sequence of motor primitives using the maximum-likelihood state transitions.

The value function for this joint distribution has the form

$$V(\tilde{\mathbf{s}}) = \frac{\sum_{i=1}^{m} \theta_i \phi\left(\tilde{\mathbf{s}}, \tilde{\mathbf{s}}_i\right)}{\sum_{j=1}^{m} \phi\left(\tilde{\mathbf{s}}, \tilde{\mathbf{s}}_j\right)}.$$

The parameters $\boldsymbol{\theta}$ of the value function are given by

$$\boldsymbol{\theta} = (\mathbf{I} - \gamma \boldsymbol{\lambda})^{-1} \bar{\mathbf{r}},$$

where the ith element of $\bar{\mathbf{r}}$ is the expected reward $[\bar{\mathbf{r}}]_i = r(\tilde{\mathbf{s}}_i, \mathscr{M}_i)$, and the elements of the transition matrix $\mathbf{P}$ are defined as

$$[\boldsymbol{\lambda}]_{ij} = \int \frac{\phi\left(\tilde{\mathbf{s}}, \tilde{\mathbf{s}}_j\right) \psi\left(\tilde{\mathbf{s}}, \tilde{\mathbf{s}}_i\right)}{\sum_{k=1}^{m} \phi\left(\tilde{\mathbf{s}}, \tilde{\mathbf{s}}_k\right)} \mathrm{d}\tilde{\mathbf{s}}.$$

This integral is computed by drawing samples from the multi-phase model, starting at $\tilde{\mathbf{s}}_i$ and executing $\mathscr{M}_i$, in order to approximate the function $\psi\left(\tilde{\mathbf{s}}, \tilde{\mathbf{s}}_i\right)$. Even though some of the DMPs are meant to transition to specific phases, some of the samples will not reach this desired phase due to the stochasticity of the multi-phase model. The model thus also incorporates the failure rates of the DMPs, and the robot can learn a high-level controller that avoids motor primitives that tend to fail.

Given the value function $V(\tilde{\mathbf{s}})$, the policy is updated by selecting new motor primitives for each of the prototypical samples. For each of these samples, the robot selects the motor primitive that maximizes the expected immediate reward plus the expected discounted value for the next state

$$\mathscr{M}_i^{\text{new}} = \arg\max_{\mathscr{M}} \mathbb{E}(r(\tilde{\mathbf{s}}_i, \mathscr{M}) + \gamma V(\tilde{\mathbf{s}}_i')).$$

The rows of the matrix $\boldsymbol{\lambda}$ and the vector $\bar{\mathbf{r}}$ are then updated accordingly, and a new value function is computed. This policy iteration process is repeated until the value function and the policy converge. The resulting policy is given by

$$\pi(\mathscr{M}|\tilde{\mathbf{s}}) = \frac{\sum_{i=1}^{m} \varphi\left(\mathscr{M}, \mathscr{M}_i\right) \phi\left(\tilde{\mathbf{s}}, \tilde{\mathbf{s}}_i\right)}{\sum_{k=1}^{m} \phi\left(\tilde{\mathbf{s}}, \tilde{\mathbf{s}}_k\right)},$$

which is a similar form to the multi-class classifier policies that are commonly used by imitation learning approaches [119, 8].

5.5.2 Motor Primitive Sequencing Experiment

The proposed approach to learning high-level policies was evaluated using the same robot setup as in the previous chapter. The goal of this experiment was to learn high-level controllers for sequencing motor primitives in order to perform manipulation tasks. The robot was given the library of motor primitives learned in Chapter 4. Two motor primitives were added to the library in order to increase the range of possible actions. The first motor primitive moves both hands 10 cm to the left. The second motor primitive raises both hands by 10 cm. Unlike the motor primitives for transitioning between phases, these task motor primitives can be executed from any phase. Hence, at least two motor primitives can

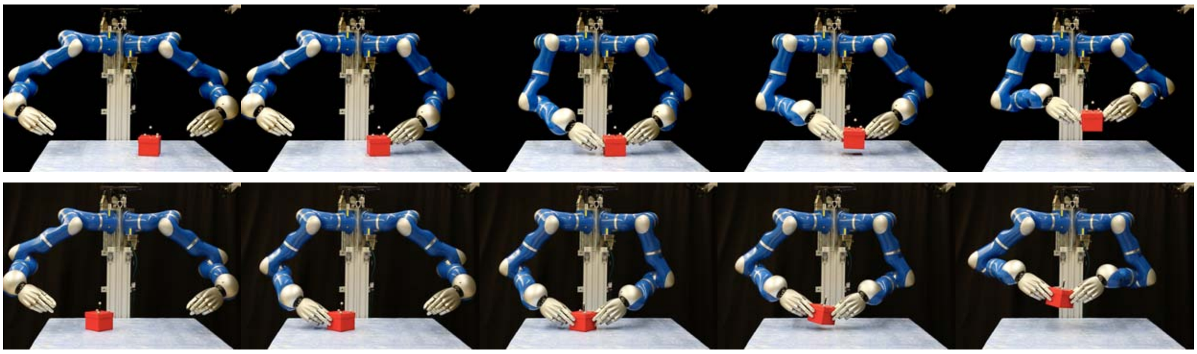


Figure 5.2: The images show two sequences of the bimanual grasping task. In the top row, the box was placed towards the left, and the high-level controller approached the box first with the left hand. In the bottom row, the robot chose to approach the grasp with the right hand first, as the box was located more towards the right.

be executed from every phase. The effects of executing one of the task motor primitives depend on the current phase.

The robot was given two tasks in this experiment. For each task, the robot had to learn a high level controller that would bring it to a suitable phase and then execute one of the task motor primitives. A trial was finished once a task motor primitive had been executed. The end of a trial was modeled by an additional absorbing state, in which no further rewards can be obtained and the robot's actions are limited to waiting. Executing a task motor primitive always resulted in transitioning to the absorbing state.

The high-level controller was learned using the method described in Section 5.5. The phase estimates $\tilde{\rho}$ were computed using the model's phase transition distribution $p(\rho_t|\mathbf{s}_t, \rho_{t-1})$ and the trajectory from the previous motor primitive. For computing the kernel function $\phi(\tilde{\mathbf{s}}, \tilde{\mathbf{s}}_i)$, we used the 3D position of the box and the positions of the hands relative to the box. For this evaluation, the robot always selected the most likely next motor primitive $\arg\max_{\mathscr{M}} \pi(\mathscr{M}|\tilde{\mathbf{s}})$. The robot computed 137 prototype samples based on 20 start-state samples.

In the first task, the robot was given a reward for the final height of the box and a penalty for the left-right deviation of the box from the center of the table. The discount factor was set to $\gamma = 0.99$. The task was executed 20 times on the robot. The box was placed on the left side of the table for ten of the trials, and on the right side for the other ten trials. In all of the trials, the robot grasped the box with both hands and successfully lifted it off of the table. When the box was placed on the left side of the table, the robot always approached the box with the left hand first, as shown in Fig. 5.2 . When the box was placed on the right side of the table, the robot approached the box first with the right hand in nine of the ten trials. In this manner, the robot tended to push the box towards the center of the table before lifting it up. In two of the trials, the robot failed to detect the $3 \rightarrow 4$ phase transition. In these trials, the robot had used the back of the thumb to hold the side of the box. As a result the fingers were pushed together rather than apart by the grasp. This problem could be addressed by using a task-space representation of the forces.

For the second task, the robot was given a reward for quickly moving the box to the left, and a penalty for the height of the box. The discount factor was set to $\gamma = 0.95$ in order to encourage the robot to perform the task quickly. The task was again executed 20 times on the robot with the box placed at different locations on the table. In all of the trials, the robot placed its right hand on the box and then moved both hands to the left, as shown in Fig. 5.3.

The first task showed that the robot was able to reconstruct the original sequences of phase transitions and then execute the task motor primitive. The robot additionally learned that it could exploit the DMPs such that they push the box towards the center of the table for a higher reward. The second task showed

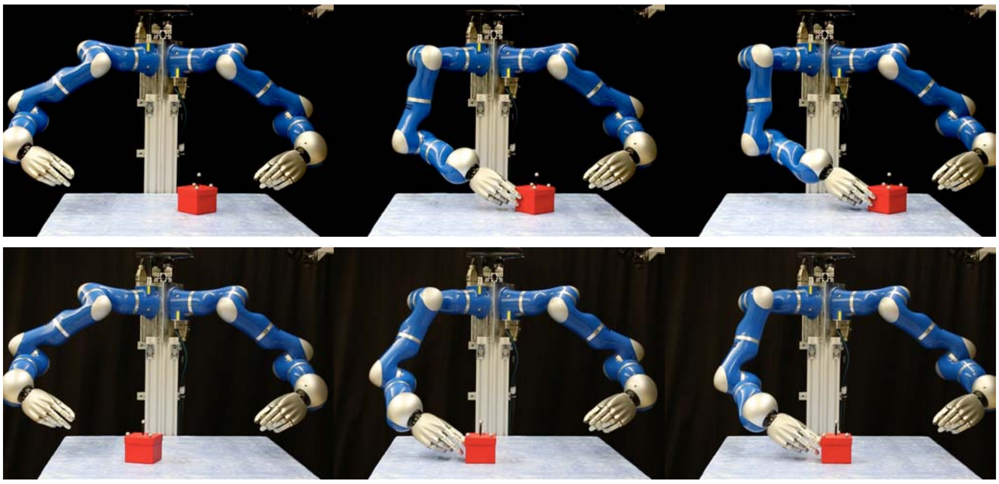


Figure 5.3: Two examples of manipulation sequences learned for pushing the box to the left (robot's perspective).

that the robot could learn to create new sequences by reusing DMPs from the demonstrated task. The experiments also showed that the robot could use the model of the phases to determine the effects of applying the motor primitives in each of the phases. Even though the task DMPs were originally not given as part of a sequence, in both of the tasks, the robot transitioned to a suitable phase before executing the correct task primitive. The robot even used the motor primitive that moves both hands left in order to achieve a one-handed push. The sequences of motor primitives were fairly consistent within each task. As a future step, the robot could learn to optimize the DMPs for these task sequences [124, 125, 126] .

The results of the experiment demonstrated that the proposed value function approach is suitable for creating medium-length sequences of DMPs. The high-level controller takes into account the current state, as well as the future actions.

5.6 Conclusion

In this chapter, we presented a non-parametric approach to computing value functions for continuous state-spaces. The proposed method is based on modeling the system using a kernel density estimate with a factorized kernel. We then showed that the value function for this type of system has the form of a Nadaraya-Watson kernel regression, and we explained how the parameters can be computed. We also explained how different modeling assumptions give rise to other common methods for computing value functions, such as least-squares temporal difference learning and kernelized temporal difference learning.

The proposed NPDP method was used to learn high-level policies for sequencing motor primitives. The learning process was based on the library of motor primitives and the multi-phase model presented in Chapter 4. The robot learned to perform a bimanual grasping task using the proposed approach. The robot learned selected motor primitives such that the object would be moved towards the center of the table. The robot also learned to perform a pushing task by reusing the motor primitives from the previous task. The sequencing of motor primitives was therefore not limited to those observed in the human demonstrations.

5.7 Potentially Helpful Insights

The main motivation for the non-parametric dynamic programming project was to obtain insights into continuous value function features and how they can be constructed. These features could then be used for learning to select motor primitive actions. We achieved our goal by determining the form of the value function for a flexible model representation. The key component of our kernel density estimate model $p(\mathbf{s}, \mathbf{a}, \mathbf{s}')$, was the use of a factorized kernel, which resulted in a value function form that is consistent under the Bellman equation. The factorisation implies that a single term cannot represent correlations between the state and the next state. This result is generally valid for mixture models that exhibit this form of factorisation. These insights could therefore also be used to construct features for other methods, such as LSTD, if one assumes this form of factorization.

The NPDP approach is well-suited for sequencing actions that funnel the states into specific regions. Motor primitive actions generally fit the NPDP assumption if the goal is defined as a fixed point. Triggering a motor primitive will usually result in the robot ending up near the goal state, with only a minor correlation between the starting state and the final state. The initial and final states become more correlated if the goal state changes with the position of another object, or a collision would occur when executing the motor primitive from certain start states. These correlations are captured by using multiple samples to the model the system, and hence adding more features to the value function. The number of features needed to represent the value function therefore generally depends on how the state space is defined. For example, when reaching for an object placed at different locations on a table, it is more efficient to define the state of the hand relative to the object rather than the world frame. From an object-relative perspective, grasping would result in a large region of initial hand poses being funnelled into a small region near the desired grasp.

In the robot experiments, we investigated using NPDP to learn a high-level policy for selecting motor primitives. However, value functions could also be used to optimize continuous action parameters. For example, the robot could compute a value function for different grasps based on the sequence f actions it intends to perform with the object. The value function would thus compactly represent the robot's future intentions for selecting a grasp. In this manner, the robot can learn to optimize the continuous parameters of manipulation sequences.

6 Learning Surface Properties from Dynamic Tactile Sensing

Dynamic tactile sensing is a fundamental ability for recognizing materials and objects. However, while humans are born with partially developed dynamic tactile sensing and quickly master this skill, today's robots remain in their infancy. The development of such a sense requires not only better sensors, but also the right algorithms to deal with these sensors' data. For example, when classifying a material based on touch, the data is noisy, high-dimensional and contains irrelevant signals as well as essential ones. Few classification methods from machine learning can deal with such problems.

In this chapter, we propose an efficient approach to inferring suitable lower-dimensional representations of the tactile data. In order to classify materials based on only the sense of touch, these representations are autonomously discovered using visual information of the surfaces during training. However, accurately pairing vision and tactile samples in real robot applications is a difficult problem. The proposed approach therefore works with weak pairings between the modalities. Experiments show that the resulting approach is robust and yields significantly higher classification performance based on only dynamic tactile sensing.

6.1 Learning Dynamic Tactile Sensing with Robust Vision-based Training

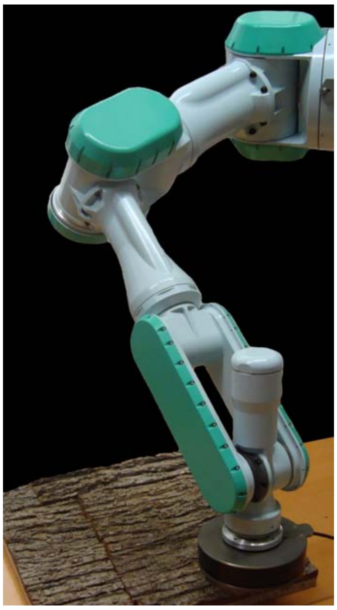


Figure 6.1: Robot learning about materials by stroking and visually inspecting different surfaces

The sense of touch has a fundamental role in most human manipulation tasks, where it serves a variety of purposes. A particularly important type of tactile sensing is *dynamic tactile sensing*. The impressive abilities of this sense are straightforward to observe [173]. For example, when a blind-folded person has an object placed in the palm of their hand, and they do not move their hand nor the object, it is very difficult to recognize the object. The size and weight of the object can be determined, but important properties such as the object's material and precise shape cannot. If one instead slides the object over the skin, one can quickly determine the object and the material [173]. Developing this ability for robots offers many future possibilities.

Dynamic tactile sensing relies on the motion between the skin and the object to induce vibrations and deformations in the skin, which it then uses to infer object and material properties [15]. This type of sensing can be used to determine various properties of a surface, including texture, hardness, roughness, and friction [1, 14]. These properties can be used for tasks such as object identification and determining suitable contact points for grasps.

Dynamic tactile sensing also obtains information about the manipulation task. Vibrations are induced in the finger when it makes or breaks contact with objects, or when incipient slip occurs [7]. These signals help coordinate the fingers, and allow humans to finely regulate the contact forces depending on the object's surface properties [14]. One can also detect the vibrations created when a held object is in contact with another object. Such signals are crucial for dexterously using tools. Humans can even use rigid objects as probes to determine the fine texture of surfaces [174].

The sense of touch should however not be seen in complete isolation, but rather as part of a multimodal system. When recognizing materials and objects, humans often combine touch with vision and even audition [175, 174]. Several studies have shown that the human brain even employs multi-sensory models of objects [175]. By using such a shared model, humans can transfer knowledge about an object from one sensory modality to another [176]. This sharing of information is especially useful when one sense can not be used. For example, experiments with both vision and touch have shown that humans rely more on touch when the texture has small details that are difficult to see [174]. Dynamic tactile sensing can thus be combined with other senses for more accurate information and additional robustness [17].

Given the various benefits of using tactile information in manipulation tasks, there is a considerable interest in equipping robots with such capabilities [177, 178, 179]. The need for robust manipulation skills is especially important for service robots in unstructured environments [180]. A variety of tactile sensors are required to create a complete tactile sensor suite, as discussed in the review paper of Dahiya et al. [181]. As one part of tactile sensing, a dynamic tactile sensor usually only mimics the fast afferent nerves (FA) in human fingers. Human fingers have two types of fast afferent nerves in their fingers, i.e., FA-I and FA-II. Type I afferents have a well-localized receptive field and are densely spaced on the skin [182]. Examples of sensors that mimic type I afferents are tactile arrays [179, 183]. Type II afferents have a larger receptive field, and therefore cannot localize the source of the vibrations as well. FA-II afferents are used to sense the vibrations in held objects during manipulation tasks, and are particularly important for tool usage [181]. Due to their large receptive fields, FA-II sensors often struggle to differentiate between various sources of vibrations. Apart from the contact with the object, vibrations also come from other sources [184, 178], such as the robot's own vibrations and deformations of the skin as the finger flexes.

A crucial ability of FA-II nerves is sensing temporal characteristics, such as those involved in recognizing a surface by stroking it. In this chapter, we want to reproduce this ability to recognize materials. As a testbed for our proposed algorithms, we have created a basic sensor that represents a primitive technical counterpart to an FA-II type mechanoreceptor. The design is based on a microphone with a probe on its membrane, and was inspired by the work on haptography of Kuchenbecker et al. [185].

The raw time-series data received from the dynamic tactile sensor consists of the detected vibrations. This signal will usually serve as the input for a classifier with task-specific labels. However, classification of tactile data is a difficult task, since a time-series needs to be represented as a high-dimensional data point to capture the details of the signal. Classification in high-dimensional spaces is however prone to *overfitting*, due to “the curse of dimensionality” [33]. The overfitting results in the classifier often performing poorly when applied to new data. This problem can be addressed using *dimensionality reduction* approaches which project the data into lower-dimensional feature spaces. The goal is to discard information that is not relevant, such as noise or redundant information.

As previously discussed, additional sources of vibrations are often present in the signal together with the desired tactile signal. For good performance, the classifier needs to automatically determine the relevant parts of the signal. We therefore take a human-inspired approach and transfer knowledge from the vision modality.

In this chapter, we present approaches for combining vision and tactile information to improve the performance of dynamic tactile sensors. The focus of this thesis is on service robots that need to perform assorted tasks. However, the proposed approach is applicable to a wide range of robots with hand-eye systems. The proposed approach is based on *Maximum Covariance Analysis (MCA)* [186], which is a machine learning method for dimensionality reduction using sets of paired data. The MCA method is described in Section 6.2.2. However, MCA requires perfect pairings between tactile and visual samples, which is often a problem for robot systems in unstructured environments [187, 188]. We therefore propose *Mean Maximum Covariance Analysis* (μMCA) and using *Weakly-paired Maximum Covariance Analysis*

(WMCA) for robotic applications. These methods are more robust and only require weak pairings between the modalities. After learning, the tactile sensor can be used independently of the vision system, while retaining its improved performance. Thus, the resulting system can be used even when conditions are not suitable for visual inspection, e.g., dim lighting, occluded surfaces, perspective distortion, and even damaged cameras.

Our initial work and evaluations of the WMCA algorithm were presented in [189]. The novel contributions of this chapter include the μMCA method and a more robust implementation of WMCA based on concepts from deterministic annealing [190]. These methods are presented in Section 6.3 and compared through a series of benchmarking experiments in Section 6.4. The experiments show that the proposed methods are robust and allow the robot to accurately discriminate between materials by only stroking them.

6.2 Formalization in a Multimodal Dimensionality Reduction Setting

In this section, we formulate the problem in a machine learning framework (Section 6.2.1) and give a brief review of multimodal dimensionality reduction methods (Section 6.2.2).

6.2.1 Problem Statement

Our goal is to have a robot accurately discriminate between different surfaces by only stroking them. We initially allow the robot to learn about textures by both stroking and visually inspecting them. The robot should subsequently transfer the additional visual information to improve its knowledge of tactile sensing. As a result, the tactile sensor's independent performance should also improve.

We now repose the problem in a general machine learning framework. The problem involves reducing the dimensionality of a sensor's data such that the relevant tactile information is retained. Not all dimensionality reduction methods are suitable for our robot application. We must therefore first select an appropriate type of method.

Dimensionality reduction algorithms are either inductive or non-inductive. Inductive methods create a function f that can map the data $\mathbf{X}$ onto a lower dimensional representation $\hat{\mathbf{X}}$. Inductive methods include *PCA* [191], *kernelPCA* [192] and *autoencoder networks* [193]. Non-inductive methods, such as *probabilistic latent semantic analysis (pLSA)* [194], and *Isomap* [195], also compute a lower-dimensional representation $\hat{X}$ from X, but do not provide a mapping function f.

Robots continue to collect more data as they explore their, often changing, environments. The mapping function f of inductive methods can be used to reduce the dimensionality of the sensor's data as it is received. We therefore require an inductive method.

Definition 1 (Inductive Dimensionality Reduction) *Let $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_n) \subset \mathbb{R}^{d \times n}$ be a set of data vectors. Inductive dimensionality reduction procedures take the input $\mathbf{X}$, and output a functional mapping $f : \mathbb{R}^d \to \mathbb{R}^q$ with $q < d$. The lower dimensional representation of $\mathbf{X}$ is given by $\hat{\mathbf{X}} = (\hat{\mathbf{x}}_1, \ldots, \hat{\mathbf{x}}_n)$, i.e., $\hat{\mathbf{x}}_i = f(\mathbf{x}_i)$.*

We can further divide inductive dimensionality reduction techniques into discriminative and generative methods. Discriminative techniques, such as *linear discriminant analysis (LDA)* [196] and *canonical correlation analysis (CCA)* [197, 198], identify lower-dimensional representations that are suitable for one specific task, e.g. classification into a predefined set of classes. These techniques discard all information that is irrelevant for this particular task. While the new representations $\hat{\mathbf{X}}$ are very good for this task, they tend to be unsuitable for other tasks. In contrast, generative dimensionality reduction techniques find lower-dimensional data representations that are suited for various subsequent tasks. Intuitively, generative dimensionality reduction techniques are a form of lossy data compression methods.

Service robots will face a large range of tasks, which makes it difficult to predefine a set of suitable labels. The robots will also encounter new objects and materials as they explore their unstructured environments. If the robot discards information based only on its current set of labels, it may discard information pertinent to new materials and objects. We therefore focus on generative methods.

Having decided on using generative inductive methods, we must determine how to transfer the visual information into the tactile domain. The key to combining visual and tactile information is that both contain spatial data, such as texture, about objects and materials [175]. The senses of vision and touch are otherwise very distinct, and thus the additional sources of vibrations and noise in the tactile modality will be excluded from the visual data. We can therefore use the visual information to determine which parts of the tactile signal are relevant to the textured surface.

Audio signals can also be used to distinguish between textured surfaces [174]. Therefore, an alternative approach would be to combine the tactile sensing with hearing. However, a robot's audio sensors may also detect other vibrations, such as those from the robot's motors. These vibrations would then be present in both sensing modalities, and would therefore be incorrectly regarded as relevant for tactile sensing. To avoid this error, we use vision as our second sensor modality.

In order to automatically extract the relevant information from the vision data, we make use of *multimodal dimensionality reduction*. The general goal of multimodal dimensionality reduction is to compute new representations of the high-dimensional data samples that lie in lower-dimensional feature spaces. In comparison to unimodal dimensionality reduction, we expect the availability of multiple data representations to give a better indication of the relevant parts of the signal, and which parts can be suppressed. We formalize this concept in the following definition.

Definition 2 (Multimodal Dimensionality Reduction) *Let* $\mathbf{X}^1 = (\mathbf{x}_1^1, \ldots, \mathbf{x}_{n^1}^1) \subset \mathbb{R}^{d^1 \times n^1}, \ldots, \mathbf{X}^m = (\mathbf{x}_1^m, \ldots, \mathbf{x}_{n^m}^m) \subset \mathbb{R}^{d^m \times n^m}$ *be* m *different data sets from potentially different spaces. Inductive dimensionality reduction techniques are multimodal if they take inputs* $\mathbf{X}^1, \ldots, \mathbf{X}^m$*, and output functions* $f_1 : \mathbb{R}^{d^1} \to \mathbb{R}^q, \ldots, f_m : \mathbb{R}^{d^m} \to \mathbb{R}^q$ *for all data domains.*

Each of the m different modalities must have its own independent mapping function f based only on the modality's own data. This part of the definition is crucial, as it will allow the tactile sensor to be used on its own. Thus, if the robot is in a dark room or cannot position the object to allow for visual inspection, the robot can still use the transferred visual information for improved tactile sensing.

The canonical way to construct multimodal algorithms is to use the dependencies between *paired* samples. Two samples are strongly paired if their sensors acquired them from the same source. For example, consider a tactile sensor moving a short distance across a textured surface. The tactile reading acquired during this motion would be strongly paired with an image of the surface area swept by the tactile sensor. Acquiring perfectly paired samples across modalities is often problematic in practice, especially in unstructured environments. Any inaccuracies in moving the object or the cameras for visual inspection will result in incorrect pairings. The different sensors may also have different numbers of samples that need to be paired. For example, while cameras can quickly acquire data from large surface areas, tactile sensors obtain information from their relatively small contact region with the surface. We therefore only assume weakly-paired data [189].

Definition 3 (Weakly-Paired Multimodal Data) *A collection of data sets* $\mathbf{X}^1, \ldots, \mathbf{X}^m$ *is weakly paired, if each* $\mathbf{X}^i$ *is split into* g *groups as*

$$\mathbf{X}^i = (\mathbf{X}_1^i, \ldots, \mathbf{X}_g^i) \in \mathbb{R}^{d^i \times n^i},$$

where each group of samples is given by

$$\mathbf{X}_h^i = (\mathbf{x}_{h,1}^i, \ldots, \mathbf{x}_{h,n_h^i}^i) \in \mathbb{R}^{d^i \times n_h^i},$$

with $n^i = \sum_{l=1}^{g} n_l^i$ *. When* $n_l^i = 1$ *for all* $i = 1, \ldots, m$ *and* $l = 1, \ldots, g$ *the data sets are fully paired with strong pairings. When* $g = 1$*, all samples are weakly paired together, which means that they are all unpaired.*

A weak pairing implies that a group of samples from one modality is paired to a group of samples in another modality. While strong pairings require samples to be obtained from the same source, weak pairings only require the samples to be acquired from similar sources. Hence, the robot can acquire samples from various regions of a textured surface and group these together. Alternatively, a robot could weakly pair one tactile sensor reading to multiple images of the nearby surface. In both of these examples, the samples can subsequently be used to infer suitable strongly-paired data. Ultimately, the condition of weakly-paired data is a relaxation of the standard fully-paired requirement, and is therefore easier for robots to fulfil.

The samples used for learning the dimensionality reductions should be acquired under conditions suitable for both visual inspection as well as tactile sensing. The conditions for visual inspection can be ignored only after the mapping functions have been learned.

Although our focus is on combining visual and tactile information, the described problem framework is quite common in robotics. The algorithms described in this chapter were therefore designed to work with weak pairings between a variety of sensors. However, different mapping functions are obtained for a sensor when it is combined with different types of sensors. The features regarded as relevant are those that both sensors observe of the source, and any features found only in one of the modalities will usually be suppressed.

6.2.2 Introduction to Multimodal Dimensionality Reduction

This section gives a brief review of linear multimodal dimensionality reduction methods, including MCA. To simplify the notation, we restrict the discussion to two sensor modalities, i.e., $\mathbf{X} \in \mathbb{R}^{d \times n}$ and $\mathbf{X}' \in \mathbb{R}^{d' \times n'}$.

Linear dimensionality reduction functions can be written as $f(\mathbf{x}) = \mathbf{W}^T\mathbf{x}$ for a matrix $\mathbf{W} \in \mathbb{R}^{d \times q}$, and $f'(\mathbf{x}') = \mathbf{W}'^T\mathbf{x}'$ for a matrix $\mathbf{W}' \in \mathbb{R}^{d' \times q}$. The lower dimensional representations are thus $\hat{\mathbf{X}} = \mathbf{W}^T\mathbf{X}$ and $\hat{\mathbf{X}}' = \mathbf{W}'^T\mathbf{X}'$. The orthogonal matrices $\mathbf{W}$ and $\mathbf{W}'$ contain the basis vectors of the q-dimensional subspaces.

A popular generative dimensionality reduction technique is *principal component analysis (PCA)*. PCA finds a lower-dimensional representation that retains as much of the original signal's variance as possible. Given that other sources of vibrations may also have large variances, PCA is not a suitable approach for our purposes. The multimodal counterpart to PCA is *maximum covariance analysis (MCA)* [186].

MCA assumes that the data is fully paired, i.e., for every sample in $\mathbf{X}$ there is exactly one strongly paired sample in $\mathbf{X}'$. The data sets $\mathbf{X}$ and $\mathbf{X}'$ are centered by subtracting their means from all of their samples. MCA then optimizes the objective function $\max_{\mathbf{W},\mathbf{W}'} \mathrm{tr}\left[\mathbf{W}^T\mathbf{X}\mathbf{X}'^T\mathbf{W}'\right]$, where tr[.] is the standard *matrix trace operator*, to determine suitable projection matrices $\mathbf{W}$ and $\mathbf{W}'$. The objective function can be rewritten with $\mathrm{tr}\left[\mathbf{W}^T\mathbf{X}\mathbf{X}'^T\mathbf{W}'\right] = \sum_{p=1}^{q}\left[\mathbf{W}^T\mathbf{X}\right]_p^T\left[\mathbf{W}'^T\mathbf{X}'\right]_p$, where the operator $[.]_p$ extracts the pth column of the matrix, and $q \leq n$. Thus MCA maximizes the covariances between the low dimensional representations $\hat{\mathbf{X}}$ and $\hat{\mathbf{X}}'$. The standard MCA method requires strong one-to-one pairings between the modalities, and therefore $n = n'$. An implementation of MCA is given in Algorithm 3 .

MCA comes from the same family of standard statistical methods as PCA, LDA, and CCA. It also forms the basis for *partial least squares (PLS)* regression [199]. The PCA, LDA, CCA, and PLS techniques have all been kernelized into nonlinear versions [200, 201, 192]. The methods presented in this section can also be kernelized (Section 6.3.3). *Kernel canonical correlation analysis (kernelCCA)* [202] is amongst the most common methods for multimodal dimensionality reduction, but it is not generative. Furthermore, kernelCCA requires the tuning of a regularization parameter for each modality. Alternative approaches include *multimodal pLSA* [203] and *Hilbert-Schmidt dependence maximization* [204], but these require more careful experimental setups and are computationally more demanding. In contrast, the classical methods, and our proposed methods, can be implemented with standard matrix operations.

Even though MCA is a strong method for multimodal dimensionality reduction, robots in unstructured scenarios often cannot provide the required fully-paired data. In the following section, we show how to overcome this limitation, and make use of weakly-paired data.

Algorithm 3 Maximum Covariance Analysis (MCA)

```
INPUT:
        Data covariance matrix $\mathbf{XX'}^T \in \mathbb{R}^{d\times d'}$
        Desired output dimensionality $q$
COMPUTE MAPPINGS:
        Compute Singular Value Decomposition of $\bar{\mathbf{X}}\bar{\mathbf{X}}'^T$
            $\mathbf{USV}^T = \text{svd}(\bar{\mathbf{X}}\bar{\mathbf{X}}'^T)$ where $\mathbf{U} \in \mathbb{R}^{d\times d}$, $\mathbf{V} \in \mathbb{R}^{d'\times d'}$
        Find $q$ largest elements in $\mathbf{S} \in \mathbb{R}^{d\times d'}$
            Set $\mathbf{W}$ to corresponding $q$ columns of $\mathbf{U}$
            Set $\mathbf{W}'$ to corresponding $q$ columns of $\mathbf{V}$
OUTPUT:
        Projection matrices $\mathbf{W}$ and $\mathbf{W}'$
```

6.3 Maximum Covariance Analysis Algorithms for Multiple Robot Sensor Modalities

In this section, we explain μMCA and WMCA for robot applications. These methods incorporate vision information to create an improved representation of the tactile data. *Sensor fusion* is another process that combines data from multiple sensors to improve performance and the accuracy of measurements [16, 17]. The data from sensors can be combined directly using *data fusion*, or classified separately and then combined with *classifier fusion* [18]. These approaches rely on always having access to both sensor modalities, while the methods proposed in this section only require both modalities during the learning phase. After learning with the proposed methods, the sensors can be used independently. Hence, tactile sensing performance is improved even when the conditions are unsuitable for visual inspection, or when the camera is currently allocated to performing another task. A fundamental problem of combining tactile and vision data is self-occlusion; i.e., the hand used for tactile sensing blocks visual inspection. The proposed methods are well-suited for such situations.

Self-supervised learning is another framework that only requires both sensor modalities during the learning phase. In self-supervised learning, the robot uses one modality to generate the labels for the classification problem of another sensor modality [205, 206]. A large amount of information from the supervising modality is lost during these procedures, as the data is reduced to a single value. The methods proposed in this section use the entire signal of both sensors to improve the classification performance. In this manner, the proposed methods can share information between different materials at the level of individual features.

Self-supervised methods are sensitive to errors in the pairings between modalities [187, 188]. The μMCA and WMCA methods overcome this problem by automatically inferring strong pairings from the weakly-paired groups. The lower dimensional representations found by self-supervised methods are usually only suited for the task they were trained on [205].

In the remainder of this section, we present the proposed μMCA (Section 6.3.1) and a robust implementation of WMCA (Section 6.3.2) for robotic applications, as well as extensions to nonlinear problems (Section 6.3.3) and multiple sensor modalities (Section 6.3.4). We present straightforward algorithms for both μMCA and WMCA to guide the reader through using these methods. These algorithms can be implemented with standard matrix toolboxes.

6.3.1 Mean Maximum Covariance Analysis (μMCA)

When using different types of sensors, it is common to obtain different numbers of samples from them. For example, vision sensors can easily obtain information about large parts of a surface, while tactile sensors are limited to the regions they make contact with. Thus, there will usually be many visual

Algorithm 4 Mean Maximum Covariance Analysis (μMCA)

```
INPUT:
    Weakly-paired data from sensors one $\mathbf{X}$ and two $\mathbf{X}'$
        $\mathbf{X}$ has $n_h$ samples $\mathbf{x}_{h,1\dots n_h}$ in group $h = 1 \dots g$
        $\mathbf{X}'$ has $n'_h$ samples $\mathbf{x}'_{h,1\dots n'_h}$ in group $h = 1 \dots g$
    Desired output dimensionality $q \leq \min(\{g, d, d'\})$
INITIALIZATION:
    $\bar{\mathbf{X}} = (\bar{\mathbf{x}}_1, \dots, \bar{\mathbf{x}}_g) \subset \mathbb{R}^{d \times g}$ with means $\bar{\mathbf{x}}_{1\dots g} = 0$
    $\bar{\mathbf{X}}' = (\bar{\mathbf{x}}'_1, \dots, \bar{\mathbf{x}}'_g) \subset \mathbb{R}^{d' \times g}$ with means $\bar{\mathbf{x}}'_{1\dots g} = 0$
COMPUTE MAPPINGS:
    for $h = 1$ to $g$
        for $i = 1$ to $n_h$
            Update $\bar{\mathbf{x}}_h \Rightarrow \bar{\mathbf{x}}_h + (\mathbf{x}_{h,i} - \bar{\mathbf{x}}_h)(i+1)^{-1}$
        for $i = 1$ to $n'_h$
            Update $\bar{\mathbf{x}}'_h \Rightarrow \bar{\mathbf{x}}'_h + (\mathbf{x}'_{h,i} - \bar{\mathbf{x}}'_h)(i+1)^{-1}$
    Obtain $\mathbf{W}$ and $\mathbf{W}'$ from MCA($\bar{\mathbf{X}}\bar{\mathbf{X}}'^T, q$)
OUTPUT:
    Projection matrices $\mathbf{W}$ and $\mathbf{W}'$
```

samples weakly-paired to a few tactile samples. Rather than selecting a single visual sample for each tactile sample, μMCA combines the information from all of these samples.

The μMCA method assumes that each of the g groups, as specified in Definition 3, represents a series of observations of the same surface. The variations within each group can then be modeled as a standard Gaussian model, i.e., $\mathbf{x}_{i,j} \sim N(\bar{\mathbf{x}}_i, (\boldsymbol{\sigma}_i)^2)$ and $\mathbf{x}'_{i,j} \sim N(\bar{\mathbf{x}}'_i, (\boldsymbol{\sigma}'_i)^2)$. The mean values $\bar{\mathbf{x}}_i \in \mathbb{R}^d$ and $\bar{\mathbf{x}}'_i \in \mathbb{R}^{d'}$ are thus suitable representations of the ith surface group, and can be strongly paired together.

Service robots should generally be autonomous and automatically gather the information they require. We therefore assume that additional prior information is not available. Given a set of collected samples, the robot should fit a model of the surface that best represents this data. We therefore propose a *maximum likelihood* estimation to determine the values of $\bar{\mathbf{x}}_i$ and $\bar{\mathbf{x}}'_i$ that best represent the collected samples.

Given the centered and weakly-paired data $\mathbf{X}$ and $\mathbf{X}'$, the μMCA method solves

$$\max_{\mathbf{W},\mathbf{W}'} \operatorname{tr}\left[\mathbf{W}^T \bar{\mathbf{X}}\bar{\mathbf{X}}'^T \mathbf{W}'\right], \tag{6.1}$$

where $\bar{\mathbf{X}} = (\bar{\mathbf{x}}_1, \dots, \bar{\mathbf{x}}_g) \subset \mathbb{R}^{d \times g}$ with group means $\bar{\mathbf{x}}_h = (n_h)^{-1} \sum_{j=1}^{n_h} \mathbf{x}_{h,j}$, and $\bar{\mathbf{X}}' = (\bar{\mathbf{x}}'_1, \dots, \bar{\mathbf{x}}'_g) \subset \mathbb{R}^{d' \times g}$ with group means $\bar{\mathbf{x}}'_h = (n'_h)^{-1} \sum_{j=1}^{n'_h} \mathbf{x}'_{i,j}$. This problem can be solved using the μMCA algorithm shown in Algorithm 4 . When q is small, the singular value decomposition can be efficiently computed using techniques based on random projections [207]. Intuitively, μMCA uses the groups of samples to estimate archetypes that are more representative of the surface than any one sample. Since the rank of the $\tilde{\mathbf{X}}\tilde{\mathbf{X}}'^T$ matrix is limited by the number of groups g, the output dimensionality is limited to $q \leq g$. The μMCA algorithm has a computational complexity of $\mathcal{O}(g^3)$.

The sequential updates of the group means in Algorithm 4 allows new data to be easily incorporated. Hence, the memory requirements of μMCA depend on the number of groups and not the number of samples. The μMCA approach is therefore suitable for large amounts of data.

Algorithm 5 Weakly-Paired Maximum Covariance Analysis (WMCA) with annealing

```
INPUT:
            Weakly-paired data from sensors one $\mathbf{X}$ and two $\mathbf{X}'$
            Desired output dimensionality $q \leq \min(\{n, n', d, d'\})$
INITIALIZATION:
            $\eta = 1$
            $\hat{\mathbf{\Pi}} = \mathrm{diag}(\hat{\mathbf{\Pi}}^1, \dots, \hat{\mathbf{\Pi}}^g)$ and $\mathbf{\Pi} \rightarrow \hat{\mathbf{\Pi}}$ wherein
            $[\hat{\mathbf{\Pi}}^h]_{i,j} = \min(n_h, n'_h)^{-1} \forall i = 1, \dots, n_h, j = 1, \dots, n'_h$
ANNEALING WMCA:
            while $\eta \geq 0$
              Run *Alternating Maximization*
              Reduce $\eta$
ALTERNATING MAXIMIZATION:
            while trace value of $\mathbf{W}^t \mathbf{X} \mathbf{\Pi} \mathbf{X}'^t \mathbf{W}'$ increases
              *Step 1*) Maximize with respect to $\mathbf{W}$ and $\mathbf{W}'$:
                Obtain $\mathbf{W}$ and $\mathbf{W}'$ from MCA($\mathbf{X} \mathbf{\Pi} \mathbf{X}'^T, q$)
              *Step 2*) Maximize with respect to $\mathbf{\Pi}$:
                Set all elements of $\mathbf{\Pi}$ to zero
                for $h = 1$ to $g$
                  Compute the cost matrix $\mathbf{C} = [\mathbf{X}'^t_h \mathbf{W}' \mathbf{W}^t \mathbf{X}_h]^t$
                  Solve linear assignment problem for $\mathbf{C}$
                  Set elements of $\mathbf{\Pi}$ to 1 for assigned pairings
              *Anneal*) Relax pairings:
                $\mathbf{\Pi} \rightarrow \eta \hat{\mathbf{\Pi}} + (1 - \eta) \mathbf{\Pi}$
OUTPUT:
            Projection matrices $\mathbf{W}$ and $\mathbf{W}'$
```

6.3.2 Weakly-Paired Maximum Covariance Analysis (WMCA)

While μMCA combined samples into more informative representations, WMCA's approach is to infer strong pairings between individual samples in a weakly-paired group. Inferring strong pairings is done by including a $n \times n'$ pairing matrix $\mathbf{\Pi}$. The elements of the pairing matrix are either one or zero $\mathbf{\Pi} \in \{0, 1\}^{n \times n'}$. A one in the ith row and the jth column implies a pairing between the ith sample of the first modality and the jth sample of the second modality. Each sample is only paired to at most one sample in the other modality, i.e., $\sum_{i=1}^{n} \mathbf{\Pi}_{i,j} \leq 1$ for all $j = 1, \dots, n'$ and $\sum_{j=1}^{n'} \mathbf{\Pi}_{i,j} \leq 1$ for all $i = 1, \dots, n$. Assuming that the samples are ordered according to their weakly-paired groups, the pairing matrix will have a block diagonal structure $\mathbf{\Pi} = \mathrm{diag}(\mathbf{\Pi}^1, \dots, \mathbf{\Pi}^g)$. This structure ensures that samples are only paired within their own group.

Given the described pairing matrix, WMCA optimizes

$$\max_{\mathbf{W}, \mathbf{W}', \mathbf{\Pi}} \mathrm{tr}\left[\mathbf{W}^T \mathbf{X} \mathbf{\Pi} \mathbf{X}'^T \mathbf{W}'\right], \tag{6.2}$$

to determine projection matrices $\mathbf{W}$ and $\mathbf{W}'$, where the trace operator tr[.] sums the diagonal elements of the matrix. The optimization of (6.2) requires both continuous optimization for $\mathbf{W}$ and $\mathbf{W}'$, and combinatoric optimization for $\mathbf{\Pi}$. There is therefore no single closed form solution to this optimization. Furthermore, it is a high-dimensional non-convex problem, such that finding the global optimum with a numeric procedure is usually impossible. We can, however, efficiently find a locally optimal solution by *alternating maximization*, as shown in Algorithm 5. Step one can be efficiently solved using the same singular value decomposition methods used for μMCA. To efficiently solve the linear assignment problem

Algorithm 6 Example method for applying learned mappings to process new tactile data

```
Input:
        Tactile sensor data $\mathbf{Y}$
        Labels $\mathbf{L}$ of training data OR the number of clusters $c$
Learning:
        Determine $\mathbf{W}$ with WMCA or $\mu$MCA
Processing:
        Project $\mathbf{Y}$ using $\hat{\mathbf{Y}} = \mathbf{W}^t\mathbf{Y}$
        If labels $\mathbf{L}$ are given, supervised learning:
            Sort $\hat{\mathbf{Y}}$ with labels into $\hat{\mathbf{Y}}_{train}$, and rest into $\hat{\mathbf{Y}}_{test}$
            Train Nearest Neighbor classifier with $\mathbf{L}$ and $\hat{\mathbf{Y}}_{train}$
            Apply classifier to $\hat{\mathbf{Y}}_{test}$
        Else, unsupervised learning:
            apply $k$-means clustering with $c$ clusters
Output:
        Labels for $\hat{\mathbf{Y}}_{test}$ OR cluster assignments for $\hat{\mathbf{Y}}$
```

in step two, we suggest using the Hungarian algorithm [208] or LAPJV [209]. In this manner, we can apply WMCA to data with thousands of dimensions. The computational complexity of WMCA is given by $\mathcal{O}(\min(\{nn'^2, n^2n'\}))$.

In both steps of the algorithm, we maximize the same objective function, which will thus increase monotonically with the number of iterations. Given that the objective function has an upper bound, the algorithm is guaranteed to converge to a local maximum. Unfortunately, the objective function will often have multiple local maxima. Hence, WMCA may converge to a local maximum with a relatively low covariance. In order to avoid many local maxima of poor quality, we propose incorporating concepts from *deterministic annealing* [190].

The annealing process for WMCA is shown in Algorithm 5. The annealing introduces the mean pairing matrix $\hat{\mathbf{\Pi}}$, which pairs together the groups' means. The pairing matrix $\mathbf{\Pi}$ is a mix between the assignments found in step two and this mean pairing matrix $\hat{\mathbf{\Pi}}$. The mixing is controlled by parameter η, which is initially set to one and monotonically decreases to zero.

Intuitively, a larger value for the parameter η makes the data points within each group more correlated. When $\eta = 1$, all of the data points are effectively equal to their respective group's mean. Applying the alternating maximization results in the globally optimal $\mathbf{W}$ and $\mathbf{W}'$ when $\eta = 1$. The manner in which η decreases is known as the *cooling schedule*. The additional local maxima gradually emerge as η decreases. Since the results of each maximization are used to initialize the next one, the alternating maximization continuous to track the best local maximum as η decreases. When $\eta = 0$, the true objective function is recovered. The annealing does not guarantee that the global maximum is recovered. However, the annealing process is a systematic and efficient approach to avoiding many poor local maxima.

The idea of treating unknown correspondences as latent variables and optimizing over them has been used in previous applications, including the classical k-means [210] algorithm and the optimization in [204]. However, in both of these cases the assignments are between sample and clusters, not between samples in different data modalities.

Given the projection matrices $\mathbf{W}$ and $\mathbf{W}'$ from either μMCA or WMCA, we apply them to new tactile data, as suggested in Algorithm 6.

6.3.3 Kernelization for Nonlinear Problems

Nonlinear dimensionality reduction techniques are often more powerful than linear ones, as they can create more diverse dimensionality reduction functions. μMCA and WMCA can be made into nonlinear techniques by *kernelization*, and thus applied to problems in robotics that cannot be solved using linear representations. As the necessary steps are very similar to those for deriving kernelPCA [192] from PCA, we only outline them here. We refer the reader to [98] for a more detailed description of kernelization.

For kernelization, we require positive definite and symmetric similarity measures between samples, called kernel functions, that we denote by $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ and $k' : \mathbb{R}^{d'} \times \mathbb{R}^{d'} \to \mathbb{R}$. Any such kernel function corresponds to an inner product in a latent Hilbert space, and induces a latent feature map from the original data domain to this space [98]. The kernelized methods thus consist of mapping the input data into the latent Hilbert spaces and performing the corresponding linear method on the resulting data sets.

For example, the kernelized form of (6.2) becomes

$$\max_{\mathbf{A},\mathbf{A}',\mathbf{\Pi}} \operatorname{tr}\left[\mathbf{A}\bar{\mathbf{K}}\mathbf{\Pi}\bar{\mathbf{K}}'\mathbf{A}'^T\right], \tag{6.3}$$

where $\bar{\mathbf{K}}$ and $\bar{\mathbf{K}}'$ are the centered kernel matrices. $\bar{\mathbf{K}}$ is computed by forming the kernel matrix $\mathbf{K} \in \mathbb{R}^{n\times n}$ as $[\mathbf{K}]_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$ and then centering it using the formula $\bar{\mathbf{K}} = \mathbf{K} - \frac{1}{n}\mathbf{1}_n\mathbf{K} - \frac{1}{n}\mathbf{K}\mathbf{1}_n + \frac{1}{n^2}\mathbf{1}_n\mathbf{K}\mathbf{1}_n$, where $\mathbf{1}_n$ denotes the $n \times n$ matrix in which all elements are 1. $\bar{\mathbf{K}}'$ is computed from kernel k' in the analogous way. Centering the kernels ensures that the implicitly defined feature vectors have zero mean in the latent feature space. One can solve (6.3) with an alternating optimization similar to the one described in Section 6.3.2. In contrast to $\mathbf{W}, \mathbf{W}'$, the matrices $\mathbf{A} \in \mathbb{R}^{n\times q}$ and $\mathbf{A}' \in \mathbb{R}^{n'\times q}$ are not orthogonal matrices, but are orthogonal in the latent feature space, i.e., $\mathbf{A}^T\mathbf{K}\mathbf{A} = \mathbf{I}$ and $\mathbf{A}'^T\mathbf{K}'\mathbf{A}' = \mathbf{I}$, where $\mathbf{I}$ is the *identity matrix* of size $q \times q$. We obtain the rows of $\mathbf{A}$ and $\mathbf{A}'$ from a generalized eigenvalue problem:

$$\begin{pmatrix} 0 & \mathbf{K}\mathbf{\Pi}\mathbf{K}' \\ \mathbf{K}'\mathbf{\Pi}^t\mathbf{K} & 0 \end{pmatrix}\begin{pmatrix} \mathbf{a} \\ \mathbf{a}' \end{pmatrix} = \lambda \begin{pmatrix} \mathbf{K} & 0 \\ 0 & \mathbf{K}' \end{pmatrix}\begin{pmatrix} \mathbf{a} \\ \mathbf{a}' \end{pmatrix}. \tag{6.4}$$

Equation (6.4) can be efficiently solved for q eigenvectors using the *power method* [211]. Ultimately, the kernelized methods provide reduction functions $f : \mathbb{R}^d \to \mathbb{R}^q$ and $f' : \mathbb{R}^{d'} \to \mathbb{R}^q$ by setting $f(\mathbf{x}) = \mathbf{A}^T\mathbf{K}(\mathbf{x})$ with $\mathbf{K}(\mathbf{x}) = (k(\mathbf{x}, \mathbf{x}_1), \ldots, k(\mathbf{x}, \mathbf{x}_n))^T$ and $f'(\mathbf{x}') = \mathbf{A}'^T\mathbf{K}'(\mathbf{x}')$ with $\mathbf{K}'(\mathbf{x}') = (k'(\mathbf{x}', \mathbf{x}'_1), \ldots, k'(\mathbf{x}', \mathbf{x}_{n'}))^T$.

Kernelization usually requires more computation time, but can also reduce them in certain situations. When solving for $\mathbf{A}$ and $\mathbf{A}'$, the matrix $\mathbf{K\Pi K}$ is of size $n \times n'$ instead of $d \times d'$. Thus, if the number of samples is less than the input dimensionalities, the computation is faster in the kernelized form. To perform the optimization, one uses linear kernels $k(\mathbf{x}, \tilde{\mathbf{x}}) = \mathbf{x}^T\tilde{\mathbf{x}}$ and $k'(\mathbf{x}', \tilde{\mathbf{x}}') = \mathbf{x}'^T\tilde{\mathbf{x}}'$ and obtains the linear solutions as $\mathbf{W} = \mathbf{A}^T\mathbf{X}$ and $\mathbf{W}' = \mathbf{A}'^T\mathbf{X}'$.

6.3.4 Incorporating Additional Sensor Modalities

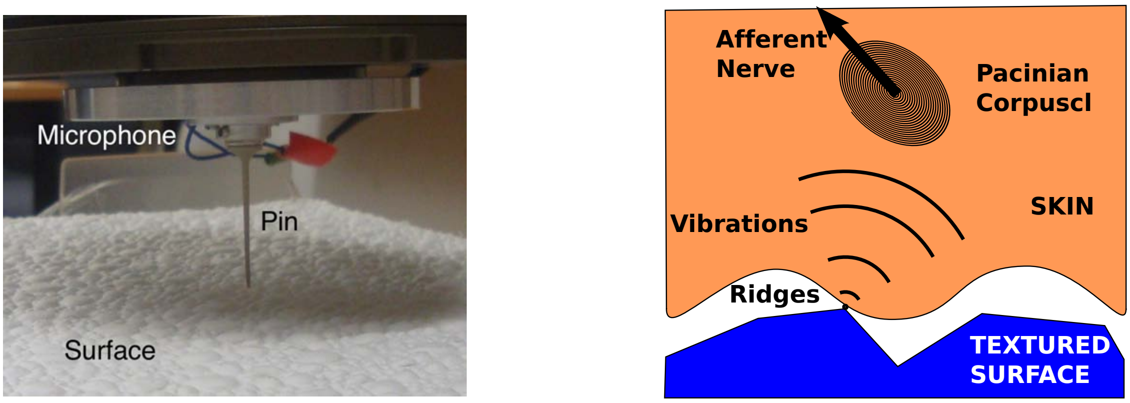
To keep the notation simple, we have been describing μMCA and WMCA for only two sensor modalities. An extension to more than two data sources is straightforward by reformulating the objective function as the sum of all pairwise covariances between the modalities. The linear μMCA objective function thus becomes

$$\max_{\mathbf{W}^1,\ldots,\mathbf{W}^m} \operatorname{tr}\Big[\sum_{i,j=1}^{m} \mathbf{W}^i\bar{\mathbf{X}}^{iT}\bar{\mathbf{X}}^j\mathbf{W}^{jT}\Big],$$

which can be solved as an eigenvalue problem. For WMCA, (6.2) becomes

$$\max_{\substack{\mathbf{W}^1,\ldots,\mathbf{W}^m \\ \mathbf{\Pi}^{1,2},\ldots,\mathbf{\Pi}^{m-1,m}}} \operatorname{tr}\Big[\sum_{i,j=1}^{m} \mathbf{W}^i\mathbf{X}^{iT}\mathbf{\Pi}^{i,j}\mathbf{X}^j\mathbf{W}^{jT}\Big],$$

with the convention that $\mathbf{\Pi}^{i,i} = 0$ and $\mathbf{\Pi}^{i,j} = \mathbf{\Pi}^{j,iT}$. The WMCA problem can again be solved by an alternating maximization approach. The step of finding the projection directions is solvable as an eigenvalue problem. Finding the sample pairings requires solving $0.5m(m-1)$ linear assignment problems. The quadratic scaling in the number of modalities m does not pose a practical problem. Unless the sensor suite is highly redundant, usually only a few sensor modalities will produce related samples. Using multiple modalities to supervise one sensor also suffers from diminishing returns.



A. Tactile Sensor

B. Human FA-II

Figure 6.2: A) The robot's tactile sensor. B) Diagram of how type II fast afferent nerves obtain tactile information (based on [15]). Both the sensor's pin and the human skin are compliant and move along the surfaces. When making and breaking contact with the surface, vibrations are created at the human's epidermal ridges and the tip of the sensor's pin. These vibrations are transferred through the skin and the pin respectively. When the vibrations reach the pacinian corpuscle, this mechanoreceptor transfers the signal to the human nervous system. Similarly, when the pin's vibrations reach the microphone's membrane, the microphone transfers the signal to the robot.

6.4 Robot Experiments with Dynamic Touch and Vision

Three experiments were performed to show that the μMCA and WMCA methods are useful for learning dynamic tactile sensing. The first experiment tests the robot's performance on the supervised classification and the unsupervised clustering of tactile data. The second experiment evaluates the system's ability to generalize between materials, and involves classifying materials that it had not encountered during the learning phase. The final experiment investigates the robustness to incorrectly paired data. In all of these experiments, we assume that both tactile and visual information is available for learning the dimensionality reduction, but only the tactile sensor is available during the testing stage.

6.4.1 Tactile Sensor and Surface Materials

In order to explore various textured surfaces, we equipped a Mitsubishi PA-10 robotic arm with a single basic tactile sensor. The experimental setup is shown in Fig. 6.1. The aim of the experiments is to test the data processing procedure. We therefore used a straightforward oscillator-based design for the sensor. The dynamic tactile sensor consists of a compliant pin that makes contact with the surface, and a capacitor microphone that can detect the pin's vibrations at 44.1 kHz. Mechanisms in the human finger tip resemble this structure, as shown in Fig. 6.2 . In particular, the sensor acts similar to an FA-II afferent, and the pin can be seen as either a part of the finger or as an object held by the robot. Given the compliance of the plastic pin, the location of the contact point with the surface could not be precisely determined. This sensor design is similar to other dynamic tactile sensors, such as the "whisker" sensor [212, 213]. The resulting apparatus is a suitable platform for testing the proposed WMCA and μMCA algorithms and showing that they can be applied to dynamic tactile sensors. Given that humans can discriminate between textures by probing them with a stylus [174], a single dynamic tactile sensor should be sufficient to perform the task.

The experiments were run on a set of 26 surfaces of 17 different materials. A common trait of these surfaces is that they have rich multi-scale textures. For example, a mosaic has the coarse texture set by the placement of the tiles, as well as the fine texture created by the surface of the tiles and cement (see

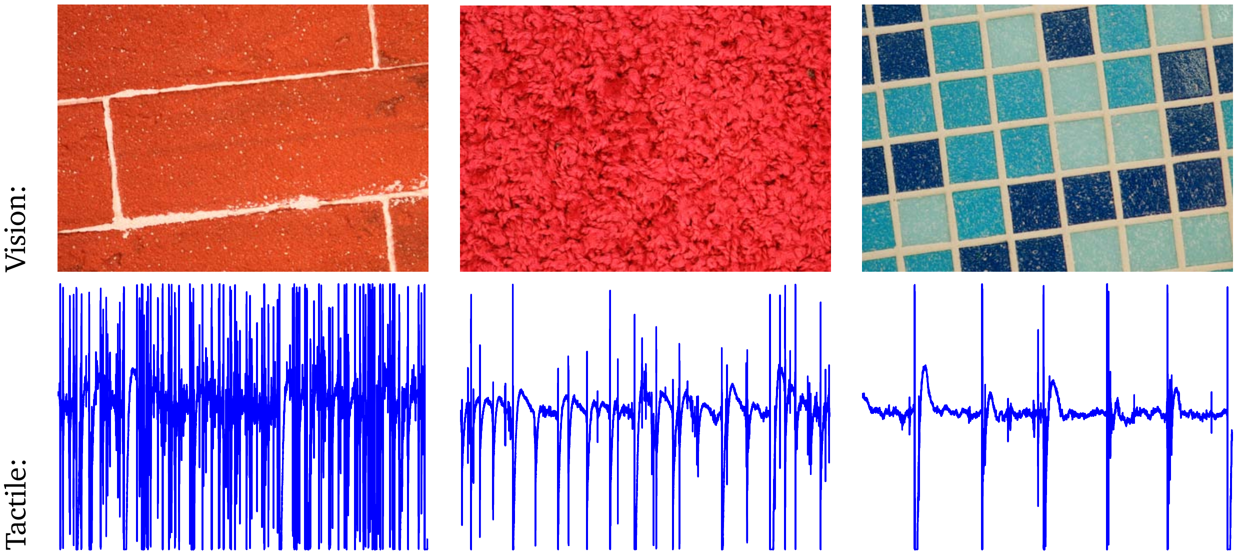


Figure 6.3: Examples of the multimodal data. The top images show the vision data while the bottom images show the corresponding time series of the tactile sensor signals. The x-axes of the tactile sensor plots represent time, while the y-axes represent the signal's magnitude. The samples for the plots were recorded over a four second time span.

Fig. 6.3). The data set includes materials that are similar and thus difficult to discriminate, as well as materials that are distinct and thus hard to generalize between.

The robot acquired samples by sliding the tactile sensor in a straight line across the surfaces. In this manner, each textured surface was probed in five different regions. The robot used similar task-space movements for each region. If very different movements had been used, the data would require additional preprocessing to compensate for the different velocity profiles. Experiments have shown that humans also need to take into account the relative velocity between the finger and surface to accurately discriminate between textured surfaces [214]. After the robot had explored a surface with the tactile sensor, the object was repositioned 20 cm in front of the robot's camera for visual inspection. Four pictures were taken of each surface with different in-plane-rotations. The resulting grayscale images have resolutions of 512 × 768, as shown in Fig. 6.3. The pictures were taken in a well lit room.

6.4.2 Tactile and Visual Features

The information from both the tactile sensor and the camera were preprocessed to obtain suitable feature spaces. The robot probed five different surface regions from each of the 26 surfaces, resulting in 130 time series of tactile data. Textures are characterized by repeated local features. We therefore propose using a *bag-of-features* model [215, 216], which represents each region by a normalized histogram of local features. Local features are found by dividing each time series into 450 segments of 50ms, with 12.5ms overlaps between segments. In order to make the local features invariant to changes in phase and amplitude, each time segment was centered and its cepstrum was computed. The power cepstrum of a signal $\mathbf{z}$ is given by $C(\mathbf{z}) = |F(\log(|F(\mathbf{z})|^2))|^2$, where the function F is the Fourier transform, and describes the harmonic structure of the signal. It is often used to discriminate between different sources of acoustic signals [217]. Intuitively, the cepstrum represents the differences in the sound made by a brass and a string instrument playing the same note. In order to generate the desired histograms, we need to partition the cepstrum space. Hence, we partition the cepstrums into 1000 groups using k-means clustering. By using 1000 clusters, we ensure that the resulting feature vectors are sparse. Each of the $n = 130$ probed regions in $\mathbf{X}$ is thus represented as a normalized histogram of $d = 1000$ partitions, which indicate the relative occurrences of local cepstrum features.

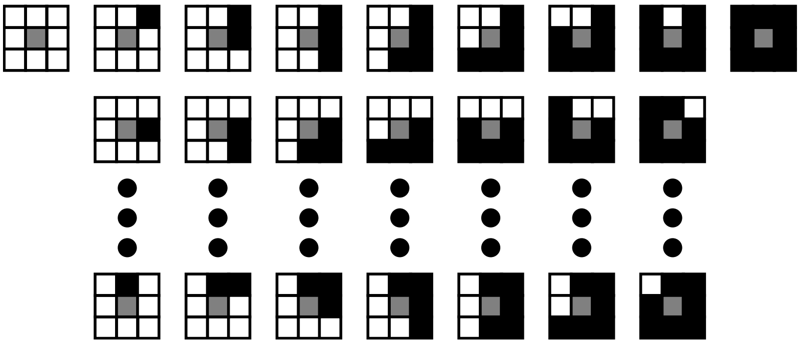


Figure 6.4: The 58 vision filters used to represent the textured images. Each 3×3 box represents a uniform binary pattern. The **grey** middle pixel defines the threshold value of the patch. A **black** pixel indicates that it is darker than the threshold, while a **white** pixel indicates that it is lighter or identical.

The vision data was obtained by segmenting each of the 104 images into 32 equally-spaced strips. Each strip is three pixels wide. Similar to the regions probed by the tactile sensor, each strip is represented using a bag-of-features model. Along each strip, we compute *local binary patterns* over 3 × 3 pixel regions using *uniform patterns*, as suggested by Ojala et al. [218]. These 58 local features, shown in Fig. 6.4 , are invariant to shifts in grayscale and rotations. Each of the $n' = 3328$ strips in $\mathbf{X}'$ is thus represented by a normalized histogram of $d' = 58$ partitions, which indicate the relative frequency of the local binary patterns.

The vision and tactile histograms can thus each be represented as 58 and 1000 dimensional vectors respectively. For both the image and tactile data, the feature dimensions were normalized to have zero mean and unit variance. This normalization step reduces the artifacts caused by having some histogram partitions being more populated than others.

6.4.3 Testing Performance, Ability to Generalize, and Robustness

Three experiments were run to compare the proposed μMCA and WMCA algorithms. The experiments' tasks were also performed with the standard PCA approach as well as the naive approach of not using any dimensionality reduction. The PCA method gives a baseline for using dimensionality reduction without the multi-modal data. The WMCA method used a ten step cooling schedule to reduce η from one to zero. The dimensionality reduction methods' only hyperparameter is the number of output dimensions q. The experiments were repeated for each output dimensionality in the range 1 to 55.

Each experiment consists of a learning phase and a testing phase. The learning phase corresponds to a robot exploring different object surfaces in a setting that allows for both visual and tactile inspection. The robot subsequently learns a mapping matrix $\mathbf{W}$ using one of the dimensionality reduction methods. The set of data used during the learning phase is known as the *learning set*.

The testing phase corresponds to a robot sorting different materials using only data from the tactile sensor. Visual inspection is not possible during the testing phase. The classification and clustering of the surfaces is performed, as described in Fig. 6, with the mappings $\mathbf{W}$ from the learning phase. The set of data used during the testing phase is known as the *testing set*. The classification tasks were evaluated using a *leave-one-out* scheme, i.e., we removed a data vector $\mathbf{x}_i$ from the testing set, trained a classifier on the remaining data, classified the removed vector $\mathbf{x}_i$, and then reinserted the data vector into the testing set. We repeated this procedure for each data vector in the testing set. The leave-one-out scheme makes efficient use of all of the available data for the evaluation. The labels used for classification are defined as the material from which the data was obtained.

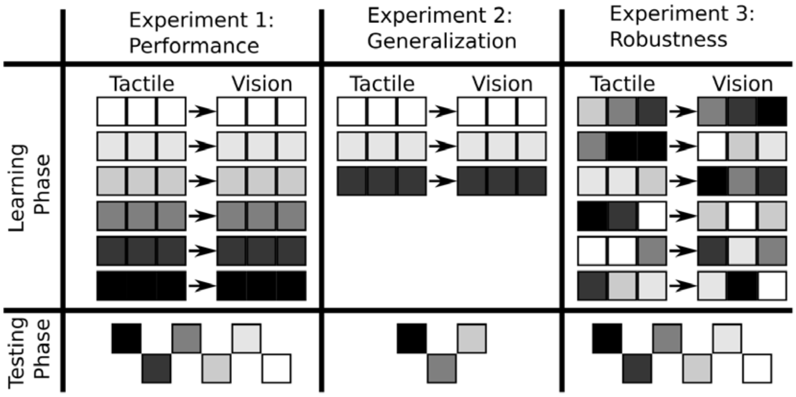


Figure 6.5: An illustration of the three experimental setups. The **top row** shows how the data was structured for the learning phase. Each small square represents the data from one surface region, and adjoining squares are grouped together. The shading of the squares indicates the materials that the sample was obtained from. The arrows indicate groups of samples that are weakly paired together between tactile and vision modalities. The **bottom row** indicates the materials that the learned system was tested on. Each square represents a type of material tested in the classification and clustering tasks. Testing data is limited to tactile data and, therefore, does not contain any groups or weak pairings. This figure does not show the true number of samples and materials used in the experiments.

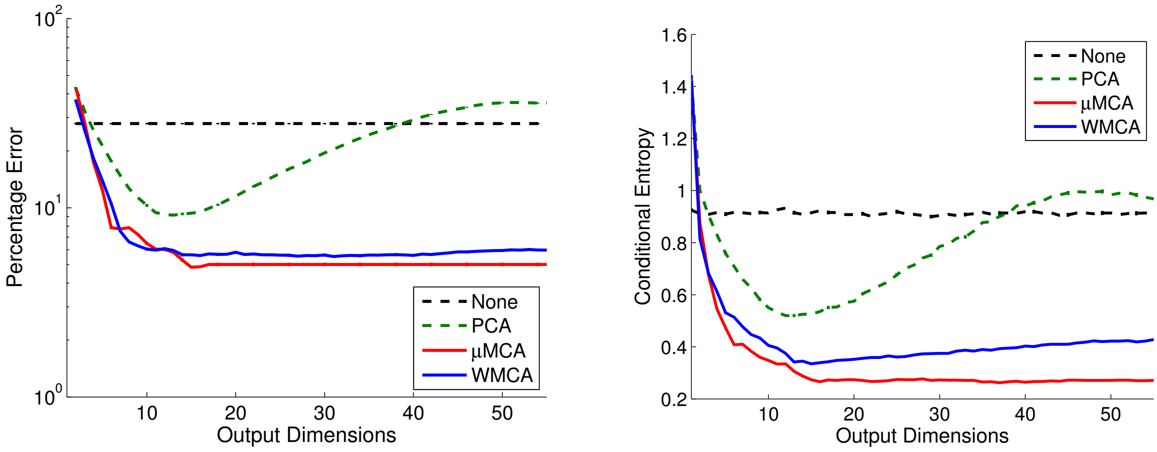
The materials and groupings used to generate the learning and testing sets were altered for each of the three experiments in order to test different aspects of the dimensionality reduction algorithms. An overview of how the data was allocated to the learning and testing sets is shown in Fig. 6.5 .

The first experiment investigates the performance at classifying and clustering surfaces. The learning set is generated by randomly selecting half of the tactile and visual data for each of the 17 materials. All of the data taken from the same textured surface is weakly paired together such that $g = 17$. The testing set consists of the other half of the tactile data. Thus, the learning and training sets both include examples from all 17 materials, as shown in the left column in Fig. 6.5. For the clustering experiment, the number of clusters is set to the number of materials $c = 17$, and would otherwise need to be estimated from the data [219]. Additionally, the time required to learn the dimensionality reduction was recorded for each method.

The second experiment tests the ability to generalize to new materials. The learning set consists of the tactile and visual data from 10 randomly selected materials. All of the data taken from the same textured surface is weakly paired together such that $g = 10$. The testing consists of the tactile data from the seven materials excluded from the learning set. Hence, the learning and training sets consist of different materials. This experiment demonstrates how information can be transferred between related tasks using dimensionality reduction [220].

The third experiment tests the robustness to incorrectly paired data, which is a common problem for self-supervised approaches [187, 188]. Similar to the first experiment, the learning set is generated by randomly selecting half of the tactile and visual data for each of the 17 materials. However, rather than forming groups of the same material, the data is randomly allocated to the $g = 17$ groups. Hence, each weakly-paired group contains a mix of different materials, as illustrated in the right column of Fig. 6.5. The testing set is the same as in the first experiment, and consists of the other half of the tactile data. Thus, the learning and training sets both include examples from all 17 materials. This situation is contrived and represents a worst case scenario that is unlikely to occur in practice.

Each experiment was run 500 times for each output dimensionality. For each run, A different seed value was used to initialize the randomization.



A. Supervised Classification

B. Unsupervised Clustering

Figure 6.6: The performance of the tested methods for different numbers of output dimensions. Plot A shows the results from a classification problem. This plot uses a log scale for the y-axis. Plot B shows the results from a clustering experiment. In both plots, a lower value indicates a better performance. Error bars are also plotted, indicating +/- two standard errors of the mean.

6.4.4 Results

The first experiment's classification and clustering results are shown in Fig. 6.6A and Fig. 6.6B respectively. The conditional entropy indicates how much information about the true material label is given by the cluster it has been assigned to. It is therefore a suitable measure of clustering performance [221]. The μMCA method achieved the best performance in both the supervised classification task, with an accuracy of 95.15%, and the unsupervised clustering task, with a conditional entropy of 0.262. The WMCA method achieved a similar classification accuracy, but a conditional entropy of only 0.335 for the clustering task. The unimodal PCA approach performed considerably worse than the multimodal approach with a best classification accuracy of 90.85% and a conditional entropy of 0.520. The naive approach gives a benchmark accuracy of 72.14% and a conditional entropy of 0.900. Both WMCA and μMCA display plateau structures of similar performance for a wide range of output dimensions.

The mean times required to compute matrix **W** are 1617ms for WMCA, 22ms for μMCA, 19ms for PCA, and 0ms for the naive approach, when run on a 3.0 GHz Intel Duo Core processor in python. The time required by μMCA can be decomposed into 7ms for computing the group means, and 15ms for computing the mapping matrix **W** from these means.

The results of the second experiment are shown in Fig. 6.7. These error rates are lower than in the first experiment, as this classification task only uses seven classes rather than 17. The WMCA, μMCA, and PCA approaches achieved similar classification accuracies of approximately 96.5%. The naive approach obtained an accuracy of 92.0%. The standard deviations in this experiment are approximately one and a half times as great as in the first experiment.

The results of the robustness experiment are shown in Fig. 6.8. The μMCA method's classification accuracy is similar to that of PCA. The WMCA method, with annealing, achieves performance levels similar to those of the first experiment.

6.4.5 Discussion

The results show that the use of the multi-modal data in the dimensionality reduction significantly improves the performance of the system. When the number of output dimensions increases, each method is selecting additional directions in the input space to keep. If the signals in these directions contain

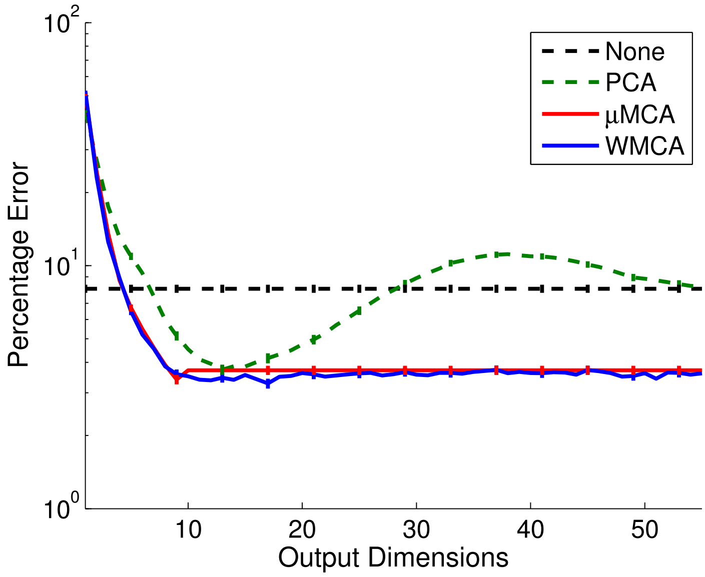


Figure 6.7: The graph shows the classification error incurred when classifying seven textures that were excluded from the learning set. The error bars indicate +/- two standard errors of the mean.

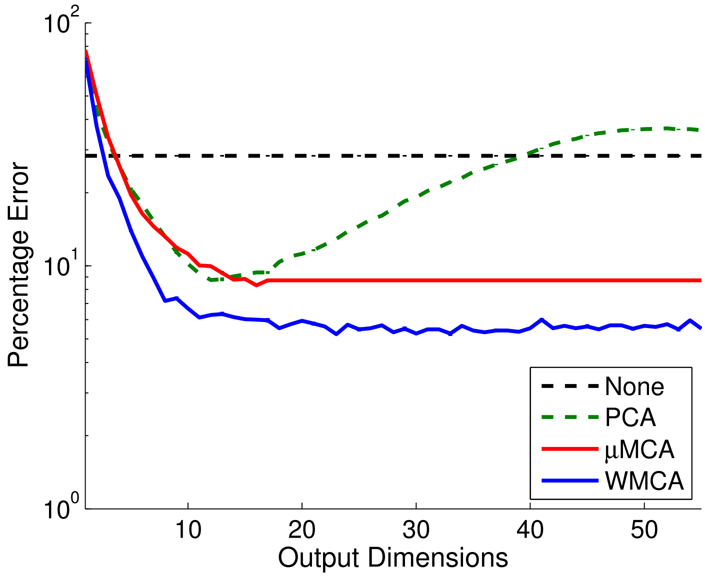


Figure 6.8: This graph shows the effects on classification performance when WMCA and μMCA are trained on incorrectly-paired data. Each weakly-paired group consists of a mix of materials, rather than a single material. The error bars indicate +/- two standard errors of the mean.

information relevant for tactile sensing, the performance improves. When the performance of a method decreases, it is including signals that are irrelevant to the tactile sensing, even though they have a high variance. Such signals could be caused by additional factors in the tactile modality, such as the vibrations of the robot [184, 178].

The PCA approach performs the best around $q = 16$ output dimensions. Deviations from this value lead to worse performance. In contrast, the WMCA method uses the vision information to determine which dimensions are relevant. By actively trying to exclude irrelevant signals, WMCA creates a plateau of good performance around the optimal q value. Hence, the WMCA method is less sensitive to changes in q and easier to tune.

By performing MCA on the group means $\bar{\mathbf{x}}$ and $\bar{\mathbf{x}}'$, the μMCA method automatically omits the dimensions describing variations within the groups. The resulting low-dimensional representations therefore contains less noise, which leads to better performance. These representations are especially well-suited for representing cluster centers, as shown by the clustering task's results. The μMCA method's plateau structure is the result of its limited output dimensionality $q \leq g$. Similar to WMCA, the μMCA method uses the vision data to include the relevant dimensions first. Hence, the final dimensions added tend to be the worst and decrease performance levels.

Both WMCA and μMCA perform well in the classification and clustering tasks of the first experiment. However, a one-tailed z-test at a 99% significance level confirms that μMCA's performance is significantly better. The WMCA method also requires considerably more computation time than μMCA and PCA. However, most applications will not require the learning to be performed in real time.

The second experiment shows that the abilities of μMCA and WMCA to generalize to new materials is similar to that of PCA. The good performance in this experiment suggests that the dimensionality reductions keep most of the pertinent information. The additional vision samples that WMCA did not find a pairing for may therefore be removed to save memory. The standard deviations are larger in this experiment because the performance is affected by the similarity between the learning and testing data sets. If the learning set includes materials similar to those in the testing set, the methods perform better.

Although the groups in the third experiment contained large amounts of incorrect data, the WMCA automatically found good pairings between samples. This result suggests that WMCA can be used with more complicated vision data and still find good pairings. Unlike WMCA, the μMCA method could not find suitable low-dimensional representations due to the incorrect data.

Since μMCA is less robust to incorrect data, it requires a more structured environment for the learning phase. The environment should allow for surfaces to be easily inspected through both vision and touch. The inspected surfaces should be easy to identify in the images and should ideally be large and flat. Since the μMCA method only requires weakly-paired samples, the objects may be freely manipulated by the robot between the tactile and vision inspections. Given these conditions, the environment should effectively resemble an infant's playpen.

The additional robustness of WMCA allows it to learn in more complicated environments. The experiments suggest that WMCA can handle situations such as having multiple objects in an image, and visually inspecting surfaces from multiple angles. The images must still contain some good data, but the robot is also allowed to collect some incorrect data while exploring. The WMCA may therefore be able to learn in everyday environments, as long as the conditions allow for both tactile and visual inspection of surfaces. The ability to learn by inspecting everyday objects is however beyond the scope of this thesis, and will need to be thoroughly tested.

In the future, the effects of varying the tactile sensor's velocity should also be experimentally investigated. Altering this velocity, or observing the surface at an angle, has a similar effect to scaling the textured surface. The performance of the proposed approach can be improved by incorporating preprocessing of the data to make it invariant to such changes. In this manner, the robot could learn in even more complicated situations.

Once the dimensionality reduction has been learned with either μMCA or WMCA, the tactile sensor can be used in a wide range of situations. The tactile sensing will still benefit from the multimodal learning phase, even if the conditions do not allow for visual inspection.

6.5 Conclusion

Dynamic tactile sensing represents an important form of feedback when performing manipulation tasks. These sensors will therefore be vital for the many tasks that service robots may encounter. However, the data from tactile sensors is usually high dimensional and can contain vibrations from spurious sources. Hence, the data is difficult to use for discriminating between different surfaces.

In this chapter, we presented the μMCA and WMCA methods for using tactile sensors to accurately and robustly classify textured surfaces. These methods use a second sensor modality, i.e. vision, during the learning phase to determine suitable lower-dimensional representations of the tactile data. The proposed approach relies on both sensors observing the relevant information from the environment, i.e. the texture of a surface. Any additional information is only observed by one of the modalities. For example, the surface's color is only seen by the camera and the robot's vibrations are only detected by the tactile sensor. Hence, the relevant part of the data is correlated between the modalities. A common problem when using multimodal data is the need to perfectly pair the data samples across modalities. The proposed methods were therefore designed to work with groups of weakly-paired data. After learning a mapping to a lower dimensionality, the vision modality is no longer required. Therefore, unlike sensor fusion approaches [16, 17, 18], the tactile sensor can be used in conditions where visual inspection in not possible, while still benefiting from the multimodal learning.

The experiments show that the μMCA approach performs well in both classification and clustering tasks. The mapping to lower-dimensions can also be quickly learned from a set of samples. The experiments also showed that the WMCA approach is robust and can even handle heavily mixed groups. The proposed methods can learn suitable dimensionality-reduction mappings from only weakly-paired data obtained in semi-structured environments.

6.6 Potentially Helpful Insights

A key motivation for this project was to investigate the complimentary nature of visual and tactile texture data. While certain aspects of the environment are captured by both of these sensing modalities, others are only captured by one of them. It is this interplay between the two sensors that allows the robot to extract the relevant texture information from the tactile data. Although audio data contains texture information, it would not be a suitable replacement for the vision data, as it also captures many of the additional vibrations detected by the tactile sensor. A core difference between the vision and tactile data is that the texture is captured spatially by the camera, but it is a temporal signal for the tactile sensing. This difference helps to keep the data from the modalities distinct, and it explains why audio data could replace the tactile data but not the vision data.

Despite the relatively basic sensor, the robot was able to detect distinct signals for certain tactile events. For example, the edges between the mosaic pieces are clearly observable in the tactile signal. The vibrations were largely due to the sensor's tip slipping from the raised surfaces into the grooves. Despite the rather small movement of the tip, the sensor managed to capture these tactile events. The robot could then extract the relevant parts of the signal using a machine learning approach.

A similar approach could be used to detect other contact events as well. For example, the making or breaking of contact between a held tool and another object could be detected. These contacts could occur at different locations on an objects. It would therefore be important to research how the vibration signals generalize between different contact locations on more complex objects. The temporal information of the detected vibrations could again be combined with the spatial vision information to distinguish between contact events and other vibration sources.

7 Conclusion

In this thesis, we presented steps towards creating autonomous robots with versatile manipulation skills. We investigated different machine learning approaches, and showed how they could be applied to learn grasping and manipulation skills. In this chapter, we summarize the main contributions of the thesis. Section 7.2 describes the general structure of manipulation skills and identifies various elements of the skills that a robot could learn. Section 7.3 presents ideas to consider when selecting or developing learning methods for robot manipulation skills. Both of these sections emphasize the importance of structuring the learning problem. The learning method is often a direct result of how the problem is structured. The chapter ends with a discussion on open problems for learning robot grasping and manipulation skills.

7.1 Summary

In Chapter 2, we focused on learning grasps through trial and error. Rather than using a supervised learning approach, grasping was framed as a continuum-armed bandits problem [32]. The robot learned to predict the performance of different grasps, based on its previous grasp attempts, using Gaussian process regression [222]. This Bayesian approach also provides the robot with a measure of how certain the predicted performance is. Using this model, the robot selected grasps according to an upper confidence bound policy. This policy explores new grasps in an optimistic manner. The proposed method was evaluated on a real robot, which successfully learned to grasp different objects.

Methods for generalizing manipulation skills between different objects were discussed in Chapter 3. Contacts play an important part in many manipulation tasks. However, it is difficult to define general features for representing the contacts between two objects. We therefore proposed a kernel approach for computing the similarities between different contact distributions. The contact distributions are modeled using multi-variate Gaussians. The kernel value is greater if the two contact distributions overlap more. The kernel was used to classify stable placements of assorted blocks and, in Chapter 4, to cluster samples from human demonstrations.

The second method for generalizing between objects was to use warped parameters to compute geometric features of objects. A warped parameter is defined as a function on a point cloud of a known object. The value of the parameter changes when transformations, such as scaling, are applied to the point cloud. The parameter is computed for a new object by warping the point cloud to match the new object's shape. We used a simple warping method and showed that the resulting parameters could be used to generalize pouring actions between different objects.

In Chapter 4, we presented a probabilistic model for dividing tasks into phases. The state-based transitions auto-regressive hidden Markov model captures the effects of the robot's actions in each phase, as well as the conditions for transitioning between different phases. The conditions for a phase transition represent the subgoals of the overall task. We therefore also showed how the model could be used together with a policy search algorithm in order to learn motor primitives for transitioning between different phases. The robot learned to perform two-handed grasps of an object using the proposed approach.

Chapter 5 focused on learning how to sequence manipulation skills. We presented a non-parametric model-based method for learning value functions in continuous state spaces. We used a kernel density estimate to model the system in a flexible manner. We then showed that the value function for this type of system has the form of a Nadaraya-Watson kernel regression [69, 70]. The resulting non-parametric dynamic programming (NPDP) algorithm was used to learn high-level controllers for both a bimanual grasping task and a pushing task.

The sixth chapter addressed the topic of dynamic tactile sensing. This sensor modality provides a lot of information to the robot about the surfaces of the objects that it is manipulating. This information is however high dimensional, noisy, and often includes irrelevant vibrations. We therefore proposed a dimensionality-reduction method for preprocessing the data before using it to classify different materials. The proposed technique is based on maximum covariance analysis, and uses weakly-paired vision data to determine relevant dimensions of the tactile data. After learning, the robot can apply the projection to tactile data even if vision data is not available.

In this thesis, we have presented machine learning methods for addressing a range of challenges posed by manipulation tasks. We also showed how these methods could be used to learn a variety of manipulation tasks on several different robot platforms.

7.2 Learning Elements of Manipulation Skills

In this section, we discuss the overall structure of manipulation skills and how different elements of the skills can be learned. The section is divided into three parts corresponding to core components of manipulation skills, i.e., the context, the effect, and the skill execution. The context corresponds to the initial state and the conditions needed to execute a skill. The effect describes the changes in state resulting from the skill execution. The execution is the controller that the robot uses in order to achieve the effect given the context. For each component, we identify key challenges that an autonomous robot can address using learning.

7.2.1 The Context

The first core component of a manipulation skill is the context. The context defines which parts of the environment need to be taken into consideration for performing the manipulation skill. This step is particularly important for manipulation skills, as objects may be added, removed, or replaced between different instances of the task. The goal is to establish a specific state space, which will then be used for executing the actual skill. Without establishing the state space, the manipulation skill may be ill-defined. For example, the skill may depend on the position of a non-existent object.

A robot could learn to perform tasks directly from its sensor data [223, 224, 225], or by detecting scene-wide correspondences across task instances [6]. However, it is often easier to generalize and reason about manipulation skills at the level of objects. One of the key challenges for autonomously establishing the context is, therefore, to determine which objects are in the scene [226, 227]. A robot can learn models of novel objects by observing them from different views and interacting with them [228, 229, 230, 231, 232]. These interactions are usually aimed specifically at exploring the object, although some parameters can also be inferred while performing the task.

Apart from segmenting an object from the rest of the scene, the robot also has to recognize the object. As the goal is to manipulate the object, it is generally more useful for a robot to recognize the affordances of the object, e.g. graspable and fillable, rather than more traditional object classes, e.g., cup, glass, and bowl [81, 82]. The affordances have the important benefit of being grounded in the robot's actions. Hence, the robot can autonomously learn the affordances of objects by interacting with the objects and observing the effects [233]. Determining the affordances may require the robot to recognize the affordance-bearing parts of an object and to establish affordance relevant coordinate frames and parameters [234, 76, 235].

Each detected object adds dimensions to the state representation corresponding to its degrees of freedom. These DoFs include the articulated joints between objects, and affordance-specific state variables, e.g. the amount of fluid in a container. Usually, only a few of the objects in the scene will be relevant for performing a specific task. The robot will therefore need to select a subset of the scene's state space for defining the skill's context.

Given the selected state space, the second goal of establishing the context is to determine whether or not the skill is executable from this state. Although the robot may have detected the elements it requires to establish the state space, the objects may not be in a configuration that affords the manipulation, e.g., an object may be outside of the robot's workspace. Learning the situations in which a skill is applicable is an important ability for an autonomous robot, and can help the robot to determine subgoals of the overall task [124, 23]. For example, if the skill is not valid, then the robot would need to select another manipulation skill to reach a state where it is valid [236].

7.2.2 The Effect

The second core component of a manipulation skill is the effect. The effect determines how the state of the manipulated objects changes due to the manipulation skill. Manipulation skills are usually executed in order to achieve a certain intended effect, although they may not be guaranteed to achieve this effect.

A key challenge for autonomous robots is to predict the effects of actions in order reason about them more efficiently. The robot could learn the effects of individual actions, or it could capture the effects by learning a forward model that generalizes between actions [78, 237, 238, 239]. Predicting the effects of actions is usually considered a supervised learning problem. Hence, the robot can use regression and classification methods for predicting the effects of continuous and discrete states respectively. These predictions are usually based on the initial state and the action parameters. Rather than predicting a single point estimate of the effect, it is more useful to learn a distribution over the effects [240, 237]. This probability distribution models the uncertainty of the outcome and can be useful for reasoning about different actions. For example, the robot may choose an action because it is more likely to have the desired effect.

Even for continuous states, the robot may observe distinct types of effects [241]. For example, pushing actions can cause the object to ROLL, SLIDE, TOPPLE, or remain STATIONARY. These labels represent a more abstract representation of the effects, which can often generalize better between different scenarios. For example, pushing a sphere causes it to ROLL, but the amount of rotation and translation depends on its size and how it was pushed. The robot can learn these labels in an unsupervised manner by clustering the continuous effects [242, 243]. If the labels are already given, then the robot can learn how they are grounded in the continuous state using a supervised learning approach.

Once the robot has learned the effects of its skills, it can use this information to make single- or multi-step predictions. These predictions can be used to plan action sequences for performing different tasks. Rather than learning the effects of skills, the robot can also learn skills for achieving specific effects. A desired effect can be modelled using a reward function [244, 19]. The reward provides a compact representation of the intended effects, with desirable effects increasing the reward and undesirable effects decreasing it. The robot can then use reinforcement learning to learn a skill that maximizes the reward [245]. The reward function is usually specified by a human as part of the task description, but it can also be learned from expert demonstrations using inverse reinforcement learning [246, 247, 248, 249]. The robot can learn to predict the rewards of different actions directly or by first predicting the effects of the action. These predictions can then be used to optimize the skills using a model-based approach.

7.2.3 The Execution

The third core component of a manipulation skill is the skill execution. This component is the behaviour (a.k.a., controller or policy) that the robot uses to alter the state of the objects being manipulated. It creates the link between the context and the effects. Given that the context has been established, the controller can assume that a fixed set of state signals are available for control, and the objects afford the desired manipulation skill. The execution may be intended to achieve a desired effect.

Before the robot can learn manipulation skills, it requires a suitable skill representation. These representations usually include several subcomponents, such as a desired trajectory generator, a feedback control loop, and a set of termination conditions. Although one can use a non-parameteric approach, skill representations usually include parameters that define how the skills should be executed [34, 250, 114, 251]. Thus, the problem of learning motor skills can be reformulated as learning the values of these parameters. The robot will need to use these parameterized representations in order to adapt its actions to specific objects and situations. For some tasks, the robot may be able to learn lower-dimensional parameter spaces in order to learn the skills more efficiently [252, 253].

The robot can use different skill learning approaches depending on the available sources of information. A human teacher is an invaluable source of information for learning manipulation skills. Humans can provide expert demonstrations of the manipulation skills that they want the robot to perform. The robot can learn these skills through imitation learning in a supervised manner [34, 250, 114]. Given multiple demonstrations, the robot can learn how to generalize the skill to different situations, and to determine a suitable task frame. As part of a scaffolding framework, the human can also provide feedback on the robot's performance, and structure tasks such that they become gradually more challenging [254, 134, 244]. In this manner, the human can provide additional guidance for the robot during the skill learning process.

If the robot is learning to perform a specific task, then the robot could also learn the skills through trial-and-error. In particular, the robot could use a reinforcement learning approach to maximize its task performance [245, 138, 255, 19]. This approach involves the robot attempting the task multiple times in order to evaluate variations of the skill. The robot then uses the information from these experiences to improve the skill. This learning process can be initialized with a skill learned from demonstrations, or a skill from a similar task. The robot can then autonomously master the skill through trial-and-error.

Given no additional information, nor a specific task, the robot can learn by simply trying out different actions and clustering the trajectories according to their effects. These action sequences can then be used as training data for learning a skill for achieving a specific effect. Rather than simply relying on motor babbling, a structured exploration could allow the robot to learn new skills more quickly. This exploration could have the robot actively search for new affordances, DoFs, or phases within its environment, and then master the corresponding skills [133, 150, 140].

Many tasks will require the robot to execute a sequence of skills. If the task has not already been decomposed into subtasks or skills, then the robot will first need to learn a suitable decomposition. For example, the robot can learn skills by segmenting human demonstrations into individual skills [8, 9, 256]. Once the robot has learned the low-level skills, it can learn a high-level policy for selecting the skills [8, 119, 118]. This policy needs to sequence the skills such that the effects of one skill fulfil the context conditions of the next skill. The context and effects are therefore similar to the inputs and outputs of the skill. These components provide structure and scope to the problem of learning the skill execution.

7.3 Key Ideas to Consider When Developing and Selecting Learning Methods

When developing or selecting learning methods for manipulation skills, one should first clearly establish the learning problem that the method should address. Structuring the learning problem generally involves determining what information is provided to the algorithm and what the outcome should be. Although this may seem trivial, manipulation tasks are incredibly complicated and incorporate many different elements. It is therefore easy to accidentally leave out important aspects of the problem, which would ultimately lead to a different approach being used. In this section, we will discuss three key ideas to keep in mind when developing methods for learning manipulation skills. These ideas are linked to three key questions: 1) What are the available sources of information? 2) Is this task an instance of a more general task? and 3) what information should the robot learn explicitly? These questions are meant to help one keep the big picture in mind when establishing the problem and selecting a suitable learning approach.

7.3.1 Sources of Information

The main purpose of a learning method is to allow the robot to structure relevant information in a useful manner. It is therefore important to first consider which sources of information are available to the robot, and how they may be incorporated. The type of learning method will generally be a direct consequence of the selected information. Learning methods also often make certain assumptions about the data that they are using, and can therefore also be used to incorporate different prior information.

As an example, we can consider the task of learning to grasp. The output of a grasp is usually some configuration of the object and the hand. We know that there are distinct types of outcomes, e.g. dropping and lifting, which the robot can learn through clustering the outcomes. Given these distinct outcomes and a set of features for describing the grasps, the robot could learn a classifier for predicting the outcomes. If the goal is to predict continuous values for the outcome, then a regression approach would be more suitable. If not all of the features may be relevant or they have a hierarchical structure that can be exploited, than the robot could use a feature selection or deep learning approach respectively. The robot may also have information regarding neighbouring grasp locations, which could be incorporated using a structured prediction approach for more robust predictions. A robot may have a method for generating untested grasps, e.g. a simulator or grasp heuristic, which could be used with a semi-supervised approach. If the robot can choose which grasps to execute, then it could apply an active learning or reinforcement learning approach.

The above list is far from exhaustive, and it does not even consider different types of sensory information nor sequences of actions. However, it does demonstrate how the available sources of information and prior task information may lead to different types of learning approaches. One should therefore consider which relevant information is available, and the implicit prior assumptions of different methods, when establishing the problem and selecting a learning approach.

7.3.2 Learning General Tasks

Although it is important to provide robots with relevant information and priors, one should also not provide too much prior information. The problem is that prior knowledge is often specific to certain types of tasks. As a result, using this knowledge limits the applicability of the methods that they are based on it. Instead, an autonomous robot should ideally be provided with more general methods, and learn the task specific information from additional experiences. In this manner, the robot needs fewer learning methods and is more likely to be capable of handling unforeseen tasks.

The most straightforward approach to keeping a method general is simply to consider a couple of other tasks that the method should be applicable to. One could either have a fixed set of tasks, or attempt to expand the set as much as possible. The latter approach is particularly useful as it helps identify the differences in tasks and the limits of the method. Often, a minor change to the method will already increase its applicability and may even help to isolate the core problem that needs to be addressed.

This approach to developing general methods is especially important for research into general-purpose service robots. For these kinds of robots, the tasks evaluated in an experimental setting are only a small fraction of the various tasks that the robot will actually need to handle. Hence, it is relatively easy to over-design for the specific tasks being evaluated. One should instead consider other potential tasks during the development process.

Obviously, one should not aim for every method to cover every possible task. Certain tasks have specific nuances that need to be addressed, and prior knowledge can make skill learning problems more tractable. The exclusion of task-specific prior information has to be compensated for by additional data from the task. If the robot would require a vast amount of data to learn the prior information autonomously, or learning the skill becomes intractable, then the prior information is valuable and should be incorporated. One should aim at developing methods that are applicable to different tasks, and have the robot learn the task-specific information, while still making the learning process tractable.

7.3.3 Explicit vs Implicit Learning

One of the most difficult design choices for developing manipulation learning methods is determining how explicit certain information should be. Modelling information explicitly can often help the robot to learn more quickly and to generalize between different situations. However, it also results in additional complexity and may introduce invalid assumptions regarding the task.

As an example, we can consider the effects of executing a manipulation skill. A robot could capture the effects by learning a forward model, or it could directly learn a high-level policy for selecting skills. In the latter case, the effects of the skills are implicit. The forward model would allow the robot to simulate the effects of different actions in various situations. It is however usually more difficult to learn the model than the policy, and errors in the model could adversely affect the final learned policy. A similar trade off can be found when considering the role of contacts in grasping. A robot could attempt to explicitly predict the contact locations of a grasp, or it could model the pose of the hand relative to the object with a parameterized motion for closing the fingers. In the latter case, the contact points are implicit. By predicting the contact points, the robot can ignore irrelevant changes in the object's shape, and can be generalized between different objects and preshapes of the hand. The robot would however also need to learn to predict the contacts, and errors in this prediction could result in bad grasps. For both the effects and the contacts example, there is a spectrum of alternative approaches that achieve different compromises.

By modeling information more explicitly, the robot can extract the relevant details of a task. The robot can learn the skill faster and generalize it to a wider range of situations by focusing on the relevant information. Problems occur when the model does not match reality or it accentuates irrelevant details instead of relevant ones. In these cases, the robot is effectively making an incorrect assumption about the task. If the assumption was introduced as part of the algorithm's design, then the invalid assumption should be removed. The robot may also make the assumptions due to a lack of data. These errors can be mitigated by modeling the uncertainty and taking it into account during decision making. If learning the explicit model is simply too difficult for the robot to learn, then a more implicit approach may be more suitable. The best choice obviously depends on the actual task and how much the robot may benefit from using a more general approach. The robot can benefit a lot in terms of learning speed and generalization by using explicit approaches, but one must also be aware of introducing incorrect assumptions about the task.

7.4 Open Problems

The methods presented in this thesis have contributed to the state-of-the-art in robot grasping and manipulation. However, there are still many open problems that need to be addressed before we can realize robots with versatile manipulation skills. In this section, we will discuss some of the next challenges to be overcome.

Learning Task-Specific Grasps

Certain grasps are more suitable for a given task than others. For example, even though a milk carton can be grasped from the top, a side grasp is more useful for pouring. The robot can also learn task-specific grasps using a reinforcement learning approach. The simplest approach would be to use the task to define a reward function for optimizing grasps. The optimization could be performed using a similar method to the one described in Chapter 2. Alternatively, the robot could divide the problem into determining task-independent grasps and selecting grasps for specific tasks. Going back to the milk carton example, the robot could learn the reward function for pouring with grasps at different locations relative to the opening of the container and its center of mass. If the object is used for multiple tasks, the reward functions could simply be added together.

Learning from Real Grasps and Simulated Grasps

Learning to grasp could also benefit from incorporating semi-supervised learning approaches. These methods allow the robot to incorporate unlabeled data into a supervised learning problem in order to model the structure of the data. For grasping, the robot can easily obtain unlabeled grasps by using a grasp simulator, and labeled grasps by executing grasps on the real object. The unlabeled data helps the robot to find clusters of similar grasps. For example, when grasping a can, the unlabelled data may indicate that there is one region of potential grasps at either end of the can, and another region around the side of the can. A successful side grasp, would then indicate that other side grasps are also more likely to succeed. Using semi-supervised learning would therefore allow the robot to merge simulated and real grasps in a straightforward manner.

Dexterous Manipulation

Manipulating objects using the fingers is difficult, as it requires coordinating multiple fingers and maintaining contact constraints. Certain in-hand manipulation also require the robot to perform a controlled slip in order to reposition the object in the hand. Learning dexterous manipulation skills would increase the robot's workspace. It would also allow the robot to reposition objects within its hand. As the object will be occluded by the hand, the robot will need to rely more on tactile data in order to localize the object in the hand. These manipulations may require the robot to learn a feedback controller based on tactile sensations [145].

Learning to Utilize the Environment

The robot can use fixed parts of the environment, e.g. a wall or table, to reposition objects in its hand [257]. For example, a robot could reposition a held object by pushing a part of it against a wall. The robot would need to rely less on dexterous in-hand manipulation if it can learn these abilities. Manipulating a held object in this manner is similar to manipulating the robot's own hand using the environment. Several compliant underactuated hands have recently been proposed [73, 258]. The high compliance of these hands allows them to adapt to a wide range of object shapes, and it allows for safe interactions with the environment. The robot can use the compliance and the environment to bend and preshape the fingers. For example, the robot could use a nearby surface in order to bend the fingers back more when attempting to grasp a large object. Alternatively, the robot could use a surface to close three of the fingers and thus isolate one for pressing a button. Methods could therefore be developed for manipulating compliant underactuated fingers and held objects.

Learning from Phase Transitions

In Chapter 4, we discussed how a robot can learn to predict phase transitions when manipulating an object. These transitions often depend on object properties, such as the mass of the object or a friction coefficient. By learning from multiple objects with different properties, the robot could learn to estimate these properties from the phase transitions. For example, when picking up an object, the phase transitions from loading to lifting when the object breaks contact with the supporting surface. A heavier object will break contact later as the robot ramps up the force. Hence the object's mass could be estimated from the timing of the phase transition. Alternatively, if the robot is using impedance control, the next motor primitive could simply be defined relative to the desired hand position when the phase transition occurred. In this manner, the motor primitive implicitly compensates for the additional offset caused by the object's mass. This approach could even be used when using different stiffnesses. Before the robot can learn from phase transitions, it must first learn to detect when they occur. This problem is particularly challenging as the dynamic tactile sensations may vary between objects and materials.

Learning in Complex Environments

One of the main challenges for robots working in everyday environments is dividing the state space into manageable parts. These environments contain a lot of different objects, and many of these objects will be interacting with each other. Imagine a robot that defined the state space of a learning task based on the state of every object in the room. The robot would try to learn how to butter bread depending on how the glasses are arranged in the cupboard. This approach would require the robot to learn in an incredibly high-dimensional space, and learning even simple manipulations would become intractable. This problem is exacerbated in cluttered environments, where many objects are in contact with each other. Most of the objects will however not be directly interacting with the object that the robot is trying to manipulate and therefore they do not directly influence the task. The robot therefore needs to learn which objects are important, and which ones can be safely ignored. One possible approach would be to learn priors based on geometric relations between objects. For example the distances between objects and their relative sizes determine, to some extent, how much they can affect each other.

Reasoning about Potential Objects

The robot should be able to reason about objects that may potentially be relevant to a task. For example, a robot may be given the task of screwing together two wooden boards with some screws. However, the robot cannot perform this task using only the objects provided, as it does not have a screwdriver. The robot should first determine that the task can be performed using a screwdriver, and it should then search for a suitable tool. If there is no screwdriver available, the robot may need to use a break knife or a coin of a suitable size instead. The main challenge is to reason about potential objects that could be used for performing a task, and then expanding the set of task objects accordingly. Rather than simply adding objects that are nearby, the robot should expand the set of objects in a goal-directed manner.

Combining One-Handed and Two-Handed Skills

Multi-armed robots will need to consider how they allocate their arms to different tasks. Some tasks, such as grasping a bottle and grasping a glass, can be performed at the same time. Other tasks, such as opening a bottle, will require the use of two hands. The robot will therefore need to be capable of executing skills in parallel as well as sequentially. The robot will also need to plan skill sequences that switch between one-handed and multi-handed skills. This problem requires the robot to schedule actions, as some executions will need to finish earlier then others. Some skills can only be started when both hands are available. This issues also poses a challenge for segmenting demonstrations. Standard methods assume that the agent is performing a single action at any point in time. These approaches are not well-suited for learning from demonstrations that include multi-tasking.

Life Long Learning

For a robot to acquire versatile manipulation skills, it will need to learn continuously over long periods of time. It will also need a learning architecture that combines different learning methods. The learning methods will need to work together. For example, the robot may use reinforcement learning to learn a pouring action for container objects, and supervised learning to recognize the containers. These types of interactions can cause problems for the robot, as one method can effectively change the problem for the other. In the pouring example, the supervised learned may suddenly learn that both bottles and cups are containers, and not just cups. The reinforcement learner may have already learned a skill that works for cups, but not for bottles, and it would need to relearn. The problem is particularly noticeable for feature learning. Task-specific features would allow the robot to capture the relevant task information in a compact manner, and they would allow the robot to generalize better to new scenarios. However, creating a new set of features would also require the methods using the features to adapt and relearn. The robot therefore needs to use learning methods that are compatible with each other. It also needs to integrate the methods together such that they achieve synergy and can adapt to each other.

7.5 Publications

The work presented in this thesis contributed to the following publications.

Journal Papers

1. van Hoof, H.; ***Kroemer, O.***; Peters, J.; **Probabilistic Segmentation and Targeted Exploration of Objects in Cluttered Environments**, IEEE Transactions on Robotics (T-Ro), *2014*
2. Muelling, K.; Kober, J.; ***Kroemer, O.***; Peters, J.; **Learning to Select and Generalize Striking Movements in Robot Table Tennis**, International Journal of Robotics Research (IJRR), *2013*
3. ***Kroemer, O.***; Lampert, C. H.; Peters, J.; **Learning Dynamic Tactile Sensing with Robust Vision-based Training**, IEEE Transactions on Robotics (T-Ro), *2011*
4. Piater, J.; Jodogne, S.; Detry, R.; Kraft, D.; Krueger, N.; ***Kroemer, O.***; Peters, J.; **Learning Visual Representations for Perception-Action Systems**, International Journal of Robotics Research (IJRR), *2011*
5. Detry, R.; Kraft, D.; ***Kroemer, O.***; Peters, J.; Krueger, N.; Piater, J.; **Learning Grasp Affordance Densities**, Paladyn Journal of Behavioral Robotics, *2011*
6. ***Kroemer, O.***; Detry, R.; Piater, J.; Peters, J.; **Combining Active Learning and Reactive Control for Robot Grasping**, Robotics and Autonomous Systems (RAS), *2010*

Conference Papers

1. ***Kroemer, O.***; Daniel, C.; Neumann, G.; van Hoof, H.; Peters, J.; **Towards Learning Hierarchical Skills for Multi-Phase Manipulation Tasks**, IEEE International Conference on Robotics and Automation (ICRA), *2015 accepted* (Best Paper Award Finalist)
2. ***Kroemer, O.***; van Hoof, H.; Neumann, G.; Peters, J.; **Learning to Predict Phases of Manipulation Tasks as Hidden States**, IEEE International Conference on Robotics and Automation (ICRA), *2014* (Best Cognitive Robotics Paper Finalist)
3. Ben Amor, H.; Neumann, G.; Kamthe, S.; ***Kroemer, O.***; Peters, J.; **Interaction Primitives for Human-Robot Cooperation Tasks**, IEEE International Conference on Robotics and Automation (ICRA), *2014*
4. Lioutikov, R.; ***Kroemer, O.***; Peters, J.; Maeda, G.; **Learning Manipulation by Sequencing Motor Primitives with a Two-Armed Robot**, International Conference on Intelligent Autonomous Systems (IAS), *2014*
5. Daniel, C.; Viering, M.; Metz, J.; ***Kroemer, O.***; Peters, J.; **Active Reward Learning**, Robotics: Science & Systems (R:SS), *2014* (≈ 30% acceptance rate)
6. Chebotar, Y.; ***Kroemer, O.***; Peters, J.; **Learning Robot Tactile Sensing for Object Manipulation**, IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), *2014*
7. ***Kroemer, O.***; Peters, J.; **Predicting Object Interactions from Contact Distributions**, IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), *2014*
8. Brandl, S.; ***Kroemer, O.***; Peters, J.; **Generalizing Manipulations Between Objects using Warped Parameters**, IEEE-RAS International Conference on Humanoid Robots (Humanoids), *2014*
9. Peters, J.; Kober, J.; Muelling, K.; ***Kroemer, O.***; Neumann, G.; **Towards Robot Skill Learning: From Simple Skills to Table Tennis**, European Conference on Machine Learning (ECML), *2013* (27.7% acceptance rate)
10. van Hoof, H.; ***Kroemer, O***.; Peters, J.; **Probabilistic Interactive Segmentation for Anthropomorphic Robots in Cluttered Environments**, IEEE-RAS International Conference on Humanoid Robots (Humanoids), *2013*

11. Daniel, C.; Neumann, G.; ***Kroemer, O.***; Peters, J.; **Learning Sequential Motor Tasks**, IEEE International Conference on Robotics and Automation (ICRA), *2013*

12. ***Kroemer, O.***; Ugur, E.; Oztop, E.; Peters, J.; **A Kernel-based Approach to Direct Action Perception**, IEEE International Conference on Robotics and Automation (ICRA), *2012*

13. Peters, J.; Kober, J.; Muelling, K.; Nguyen-Tuong, D.; ***Kroemer, O.***; **Robot Skill Learning**, European Conference on Artificial Intelligence (ECAI), *2012* (28.5% ACCEPTANCE RATE)

14. Boularias, A.; ***Kroemer, O.***; Peters, J.; **Structured Apprenticeship Learning**, European Conference on Machine Learning (ECML), *2012* (23.7% ACCEPTANCE RATE)

15. van Hoof, H.; ***Kroemer, O.***; Ben Amor, H.; Peters, J.; **Maximally Informative Interaction Learning for Scene Exploration**, IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), *2012*

16. Boularias, A.; ***Kroemer, O.***; Peters, J.; **Algorithms for Learning Markov Field Policies**, Neural Information Processing Systems (NIPS), *2012* (25.5% ACCEPTANCE RATE)

17. Ben Amor, H.; ***Kroemer, O.***; Hillenbrand, U.; Neumann, G.; Peters, J.; **Generalization of Human Grasping for Multi-Fingered Robot Hands**, IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), *2012*

18. Muelling, K.; Kober, J.; ***Kroemer, O.***; Peters, J.; **Learning to Select and Generalize Striking Movements in Robot Table Tennis**, AAAI 2012 Fall Symposium on Robots that Learn Interactively from Human Teachers, *2012*

19. ***Kroemer, O.***; Ben Amor, H.; Ewerton, M.; Peters, J.; **Point Cloud Completion Using Symmetries and Extrusions**, IEEE-RAS International Conference on Humanoid Robots (Humanoids), *2012*

20. ***Kroemer, O.***; Peters, J.; **A Non-Parametric Approach to Dynamic Programming**, Neural Information Processing Systems (NIPS), *2011* (ORAL PRESENTATION: 1.4% ACCEPTANCE RATE)

21. ***Kroemer, O.***; Peters, J.; **A Flexible Hybrid Framework for Modeling Complex Manipulation Tasks**, IEEE International Conference on Robotics and Automation (ICRA), *2011*

22. ***Kroemer, O.***; Peters, J.; **Active Exploration for Robot Parameter Selection in Episodic Reinforcement Learning**, IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning (ADPRL), *2011*

23. Boularias, A.; ***Kroemer, O.***; Peters, J.; **Learning Robot Grasping from 3D Images with Markov Random Fields**, IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), *2011*

24. Erkan, A.; ***Kroemer, O.***; Detry, R.; Altun, Y.; Piater, J.; Peters, J.; **Learning Probabilistic Discriminative Models of Grasp Affordances under Limited Supervision**, IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), *2010*

25. Lampert, C. H.; ***Kroemer, O.***; **Weakly-Paired Maximum Covariance Analysis for Multi-modal Dimensionality Reduction and Transfer Learning**, European Conference on Computer Vision (ECCV), *2010* (27.9% ACCEPTANCE RATE)

26. ***Kroemer, O.***; Detry, R.; Piater, J.; Peters, J.; **Adapting Preshaped Grasping Movements using Vision Descriptors**, International Conference on the Simulation of Adaptive Behavior (SAB), *2010*

27. ***Kroemer, O.***; Detry, R.; Piater, J.; Peters, J.; **Grasping with Vision Descriptors and Motor Primitives**, International Conference on Informatics in Control, Automation and Robotics (ICINCO), *2010* (BEST PAPER AWARD)

28. Kober, J.; Muelling, K.; ***Kroemer, O.***; Lampert, C. H.; Schölkopf, B.; Peters, J.; **Movement Templates for Learning of Hitting and Batting**, IEEE International Conference on Robotics and Automation (ICRA), *2010*

29. Peters, J.; Kober, J.; Muelling, K.; Nguyen-Tuong, D.; ***Kroemer, O.***; **Towards Motor Skill Learning for Robotics,** International Symposium on Robotics Research (ISRR), *2009*

30. ***Kroemer, O.***; Detry, R.; Piater, J.; Peters, J.; **Active Learning by Mean-Shift for Robot Grasping**, IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), *2009*

31. Piater, J.; Jodogne, S.; Detry, R.; Kraft, D.; Krueger, N.; ***Kroemer, O.***; Peters, J.; **Learning Visual Representations for Interactive Systems**, International Symposium on Robotics Research (ISRR), *2009*

32. Detry, R.; Baseski, E.; Popovic, M.; Touati, Y.; Krueger, N.; ***Kroemer, O.***; Peters, J.; Piater, J.; **Learning Object-specific Grasp Affordance Densities**, International Conference on Development and Learning (ICDL), *2009*

List of Figures

List of Algorithms

List of Tables

Bibliography

[1] R. S. Johansson and J. R. Flanagan, “Coding and use of tactile signals from the fingertips in object manipulation tasks,” *Nature Review Neuroscience*, vol. 10, no. 5, pp. 345–359, 2009.

[2] J. Bohg, A. Morales, T. Asfour, and D. Kragic, “Data-driven grasp synthesis - a survey,” *IEEE Transactions on Robotics*, 2014.

[3] P. Auer, N. Cesa-Bianchi, and P. Fischer, “Finite-time analysis of the multiarmed bandit problem,” *Machine Learning*, vol. 47, 2002.

[4] N. Srinivas, A. Krause, S. Kakade, and M. Seeger, “Gaussian process optimization in the bandit setting: No regret and experimental design,” in *International Conference on Machine Learning (ICML)*, 2010.

[5] U. Hillenbrand and M. Roa, “Transferring functional grasps through contact warping and local replanning,” in *International Conference on Intelligent Robots and Systems (IROS)*, 2012.

[6] J. Schulman, J. Ho, C. Lee, and P. Abbeel, “Learning from demonstrations through the use of non-rigid registration,” in *International Symposium on Robotics Research (ISRR)*, 2013.

[7] J. R. Flanagan, M. C. Bowman, and R. S. Johansson, “Control strategies in object manipulation tasks.” *Current Opinion in Neurobiology*, vol. 16, no. 6, pp. 650–659, 2006.

[8] S. Niekum, S. Chitta, B. Marthi, S. Osentoski, and A. G. Barto, “Incremental semantically grounded learning from demonstration,” in *Robotics: Science and Systems (R:SS)*, 2013.

[9] J. Butterfield, S. Osentoski, G. Jay, and O. Jenkins, “Learning from demonstration using a multi-valued function regressor for time-series data,” in *International Conference on Humanoid Robots (Humanoids)*, 2010.

[10] D. Grollman and O. Jenkins, “Incremental learning of subtasks from unsegmented demonstration,” in *International Conference on Intelligent Robots and Systems (IROS)*, 2010.

[11] W. B. Powell, *Approximate Dynamic Programming: Solving the Curses of Dimensionality (Wiley Series in Probability and Statistics)*. Wiley-Interscience, 2007.

[12] J. A. Boyan, “Least-squares temporal difference learning,” in *International Conference on Machine Learning (ICML)*, 1999.

[13] Taylor, Gavin and Parr, Ronald, “Kernelized value function approximation for reinforcement learning,” in *ICML*. New York, NY, USA: ACM, 2009, pp. 1017–1024.

[14] R. S. Johansson and G. Westling, “Roles of glabrous skin receptors and sensorimotor memory in automatic control of precision grip when lifting rougher or more slippery objects,” *Experimental Brain Research*, vol. 56, no. 3, pp. 550–564, 1984.

[15] J. Scheibert, S. Leurent, A. Prevost, and G. Debregeas, “The role of fingerprints in the coding of tactile information probed with a biomimetic sensor.” *Science*, vol. 323, pp. 1503–6, 2009.

[16] D. L. Hall and J. Llinas, *Handbook of Multisensor Data Fusion*. CRC Press, 2001.

[17] P. K. Allen, A. T. Miller, P. Y. Oh, and B. S. Leibowitz, “Integration of vision, force and tactile sensing for grasping,” *International Journal of Intelligent Mechatronics*, vol. 4, pp. 129–149, 1999.

[18] I. Halatci, C. a. Brooks, and K. Iagnemma, “A study of visual and tactile terrain classification and classifier fusion for planetary exploration rovers,” *Robotica*, vol. 26, no. 6, pp. 767–779, 2008.

[19] O. Kroemer, R. Detry, J. Piater, and J. Peters, “Combining active learning and reactive control for robot grasping,” *Robotics and Autonomous Systems*, no. 9, pp. 1105–1116, 2010.

[20] O. Kroemer and J. Peters, “Predicting object interactions from contact distributions,” in *International Conference on Intelligent Robots and Systems (IROS)*, 2014.

[21] S. Brandl, O. Kroemer, and J. Peters, “Generalizing manipulations between objects using warped parameters,” in *International Conference on Humanoid Robots (HUMANOIDS)*, 2014.

[22] O. Kroemer, H. van Hoof, G. Neumann, and J. Peters, “Learning to predict phases of manipulation tasks as hidden states,” in *International Conference on Robotics and Automation (ICRA)*, 2014.

[23] O. Kroemer, C. Daniel, G. Neumann, H. van Hoof, and J. Peters, “Towards learning hierarchical skills for multi-phase manipulation tasks,” in *International Conference on Robotics and Automation (ICRA)*, 2015.

[24] O. Kroemer and J. Peters, “A non-parametric approach to dynamic programming,” in *Advances in Neural Information Processing Systems (NIPS)*, 2011.

[25] O. Kroemer, C. Lampert, and J. Peters, “Learning dynamic tactile sensing with robust vision-based training,” *IEEE Transactions on Robotics (T-Ro)*, no. 3, pp. 545–557, 2011.

[26] D. Katz, Y. Pyuro, and O. Brock, “Learning to manipulate articulated objects in unstructured environments using a grounded relational representation,” in *Robotics: Science and Systems (R:SS)*, 2008.

[27] A. Bicchi and V. Kumar, “Robotic grasping and contact: a review,” in *International Conference on Robotics and Automation (ICRA)*, 2000.

[28] M. Mason and J. Salisbury, *Robot Hands and the Mechanics of Manipulation*. MIT Press, 1985.

[29] A. Saxena, J. Driemeyer, J. Kearns, C. Osondu, and A. Ng, *Learning to Grasp Novel Objects Using Vision*, ser. Springer Tracts in Advanced Robotics. Springer, 2008, vol. 39, ch. 4, pp. 33–42.

[30] M. Salganicoff, L. H. Ungar, and R. Bajcsy, “Active learning for vision-based robot grasping,” *Machine Learning*, vol. 23, no. 2-3, pp. 251–278, 1996.

[31] A. Morales, E. Chinellato, A. H. Fagg, and A. P. Pobil, “An active learning approach for assessing robot grasp reliability,” in *Intelligent Robots and Systems (IRS)*, 2004.

[32] R. S. Sutton and A. G. Barto, *Reinforcement Learning an Introduction*. The MIT Press, 2000.

[33] R. E. Bellman, *Adaptive control processes - A guided tour*. Princeton University Press, 1961.

[34] J. A. Ijspeert, J. Nakanishi, and S. Schaal, “movement imitation with nonlinear dynamical systems in humanoid robots,” in *International Conference on Robotics and Automation (ICRA)*, 2002.

[35] S. Schaal, J. Peters, J. Nakanishi, and A. Ijspeert, “learning movement primitives,” in *International Symposium on Robotics Research (ISRR)*, 2004.

[36] E. Oztop, N. S. Bradley, and M. A. Arbib, “Infant grasp learning: a computational model,” *Experimental Brain Research*, pp. 480–503, 2004.

[37] E. Oztop and M. Kawato, *Sensorimotor Control of Grasping: Physiology and Pathophysiology*. Cambridge University Press, 2009, ch. Models for the control of grasping.

[38] D. A. Rosenbaum, *Human Motor Control*. Academic Press, 1991.

[39] A. Morales, T. Asfour, P. Azad, S. Knoop, and R. Dillmann, "Integrated grasp planning and visual object localization for a humanoid robot with five-fingered hands," in *International Conference on Intelligent Robots and Systems (IROS)*, 2006.

[40] D. Kragic, A. T. Miller, and P. K. Allen, "Real-time tracking meets online grasp planning," in *International Conference on Robotics and Automation (ICRA)*, 2001.

[41] R. Agrawal, "The continuum-armed bandit problem," *SIAM Journal of Control and Optimization*, vol. 33, pp. 1926–1951, 1995.

[42] P. Auer, R. Ortner, and C. Scepesvari, "Improved rates for the stochastic continuum-armed bandit problem," in *Conference on Learning Theory (COLT)*, 2007.

[43] R. Kleinberg, "Nearly tight bounds for the continuum-armed bandit problem," in *Advances in Neural Information Processing Systems (NIPS)*, 2004.

[44] C. Rasmussen and C. Williams, *Gaussian Processes for Machine Learning*. MIT Press, 2006.

[45] D. Comaniciu and P. Meer, "Mean shift: A robust approach toward feature space analysis," in *Transactions on Pattern Analysis and Machine Intelligence*, 2002.

[46] O. Teytaud, S. Gelly, and M. Sebag, "Anytime many-armed bandits," in *Conference sur l'Apprentissage automatique*, 2007.

[47] C. M. Bishop, *Pattern Recognition and Machine Learning*, 1st ed. Springer, 2007.

[48] N. Srinivas, A. Krause, S. M. Kakade, and M. Seeger, "Gaussian process bandits without regret: An experimental design approach," *Computing Research Repository*, 2009.

[49] R. Martinez-Cantin, "Active map learning for robots: Insights into statistical consistency," Ph.D. dissertation, University of Zaragoza, 2008.

[50] Z. Xue, A. Kasper, J. M. Zoellner, and R. Dillmann, "An automatic grasp planning system for service robots," in *International Conference on Advanced Robotics*, 2009.

[51] D. Bertram, J. Kuffner, R. Dillmann, and T. Asfour, "An integrated approach to inverse kinematics and path planning for redundant manipulators," in *International Conference on Robotics and Automation (ICRA)*, 2006.

[52] M. W. Spong, S. Hutchinson, and M. Vidyasagar, *Robot Modeling and Control*. John Wiley and Sons, 2005.

[53] M. Khatib, "Sensor-based motion control for mobile robots," Ph.D. dissertation, LAAS-CNRS, 1996.

[54] K. Sabe, M. Fukuchi, J.-S. Gutmann, T. Ohashi, K. Kawamoto, and T. Yoshigahara, "Obstacle avoidance and path planning for humanoid robots using stereo vision," in *International Conference on Robotics and Automation (ICRA)*, 2004.

[55] S. Lenser and M. Veloso, "Visual sonar: Fast obstacle avoidance using monocular vision," in *International Conference on Intelligent Robots and Systems (IROS)*, 2003.

[56] J. Tegin, S. Ekvall, D. Kragic, J. Wikander, and B. Iliev, "Demonstration based learning and control for automatic grasping," *Intelligent Service Robotics*, pp. 23–30, 2008.

[57] A. T. Miller, S. Knoop, H. I. Christensen, and P. K. Allen, "Automatic grasp planning using shape primitives," in *International Conference on Robotics and Automation (ICRA)*, 2003.

[58] D.-H. Park, H. Hoffmann, P. Pastor, and S. Schaal, "Movement reproduction and obstacle avoidance with dynamic movement primitives and potential fields," in *International Conference on Humanoid Robots (Humanoids)*, 2008.

[59] F. Bley, V. Schmirgel, and K.-F. Kraiss, "Mobile manipulation based on generic object knowledge," in *Symposium on Robot and Human Interactive Communication*, 2006.

[60] K. Hsiao, P. Nangeroni, M. Huber, A. Saxena, and A. Y. Ng, "Reactive grasping using optical proximity sensors," in *International Conference on Robotics and Automation (ICRA)*, 2009.

[61] J. Steffan, R. Haschke, and H. Ritter, "Experience-based and tactile-driven dynamic grasp control," in *Intelligent Robots and Systems (IRS)*, 2007.

[62] A. Ijspeert, J. Nakanishi, and S. Schaal, "learning attractor landscapes for learning motor primitives," in *Advances in Neural Information Processing Systems (NIPS)*, 2003.

[63] N. Pugeault, *Early Cognitive Vision: Feedback Mechanisms for the Disambiguation of Early Visual Representation*. Vdm Verlag Dr. Mueller, 2008.

[64] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2003.

[65] N. Krueger, M. Lappe, and F. Woergoetter, "Biologically motivated multi-modal processing of visual primitives," *The Interdisciplinary Journal of Artificial Intelligence and the Simulation of Behaviour, 1(5)*, pp. 417–427, 2004.

[66] R. Detry, N. Pugeault, and J. Piater, "Probabilistic pose recovery using learned hierarchical object models," in *International Cognitive Vision Workshop*, 2008.

[67] M. Jeannerod, *Sensorimotor Control of Grasping: Physiology and Pathophysiology*. Cambridge University Press, 2009, ch. The study of hand movements during grasping. A historical perspective, pp. 127–140.

[68] ——, *Perspectives of Motor Behaviour and Its Neural Basis*. S Karger AG, 1997, ch. Grasping Objects: The Hand as a Pattern Recognition Device, pp. 19–32.

[69] E. Nadaraya, "On estimating regression," *Theory of Probability and its Applications*, vol. 9, pp. 141–142, 1964.

[70] G. Watson, "Smooth regression analysis," *Sankhya, Series*, vol. A, no. 26, pp. 359–372, 1964.

[71] R. Kleinberg, A. Slivkins, and E. Upfal, "Multi-armed bandits in metric spaces," in *ACM Symposium on Theory of Computing*, 2008.

[72] E. Snelson and Z. Ghahramani, "Sparse gaussian processes using pseudo-inputs," in *Advances in Neural Information Processing Systems (NIPS)*, 2005.

[73] R. Deimel, C. Eppner, J. Alvarez-Ruiz, M. Maertens, and O. Brock, "Exploitation of environmental constraints in human and robotic grasping," in *International Symposium on Robotics Research (ISRR)*, 2013.

[74] A. Saxena, J. Driemeyer, and A. Y. Ng, “Robotic grasping of novel objects using vision,” *International Journal of Robotics Research (IJRR)*, 2008.

[75] J. Bohg, M. Johnson-Roberson, B. León, J. Felip, X. Gratal, N. Bergström, D. Kragic, and A. Morales, “Mind the gap - robotic grasping under incomplete observation,” in *International Conference on Robotics and Automation (ICRA)*, 2011.

[76] O. Kroemer, E. Ugur, E. Oztop, and J. Peters, “A kernel-based approach to direct action perception,” in *International Conference on Robotics and Automation (ICRA)*, 2012.

[77] Y. Jiang, M. Lim, C. Zheng, and A. Saxena, “Learning to place new objects in a scene,” *International Journal of Robotic Research (IJRR)*, vol. 31, no. 9, pp. 1021–1043, 2012.

[78] M. S. Kopicki, S. Zurek, R. Stolkin, T. Morwald, and J. L. Wyatt, “Learning to predict how rigid objects behave under simple manipulation.” in *International Conference on Robotics and Automation (ICRA)*, 2011.

[79] K. Sjoo and P. Jensfelt, “Learning spatial relations from functional simulation,” in *International Conference on Intelligent Robot Systems (IROS)*, 2011.

[80] T. Jebara and R. Kondor, “Bhattacharyya and expected likelihood kernels,” in *Conference on Learning Theory (COLT)*, ser. Lecture Notes in Computer Science, 2003.

[81] J. J. Gibson, *The Ecological Approach To Visual Perception*. Lawrence Erlbaum Associates, 1986.

[82] E. Sahin, M. Cakmak, M. R. Dogar, E. Ugur, and G. Ucoluk, “To Afford or Not to Afford: A New Formalization of Affordances Toward Affordance-Based Robot Control,” *Adaptive Behavior*, no. 4, pp. 447–472, 2007.

[83] H. Koppula and A. Saxena, “Anticipating human activities using object affordances for reactive robotic response,” in *Robotics: Science and Systems (R:SS)*, 2013.

[84] L. Montesano, M. Lopes, A. Bernardino, and J. Santos-Victor, “Modeling affordances using bayesian networks,” in *International Conference on Intelligent Robot Systems (IROS)*, 2007.

[85] A. Herzog, P. Pastor, M. Kalakrishnan, L. Righetti, J. Bohg, T. Asfour, and S. Schaal, “Learning of grasp selection based on shape-templates,” *Autonomous Robots*, 2013.

[86] R. Detry, C. H. Ek, M. Madry, J. Piater, and D. Kragic, “Generalizing grasps across partly similar objects,” in *International Conference on Robotics and Automation (ICRA)*, 2012.

[87] I. Lenz, H. Lee, and A. Saxena, “Deep learning for detecting robotic grasps,” 2013.

[88] Y. Jiang, S. Moseson, and A. Saxena, “Efficient grasping from rgbd images: Learning using a new rectangle representation,” 2011.

[89] B. Rosman and S. Ramamoorthy, “Learning spatial relationships between objects.” *International Journal of Robotic Research*, vol. 30, no. 11, pp. 1328–1342, 2011.

[90] J. Kulick, T. Lang, M. Toussaint, and M. Lopes, “Active Learning for Teaching a Robot Grounded Relational Symbols,” in *International Joint Conference on Artificial Intelligence (IJCAI)*, 2013.

[91] Y. Bekiroglu, R. Detry, and D. Kragic, “Learning tactile characterizations of object- and pose-specific grasps,” in *International Conference on Intelligent Robots and Systems (IROS)*, 2011.

[92] H. Dang and P. K. Allen, “Learning grasp stability.” in *International Conference on Robotics and Automation (ICRA)*, 2012.

[93] T. Hermans, F. Li, J. M. Rehg, and A. F. Bobick, “Learning contact locations for pushing and orienting unknown objects,” in *International Conference on Humanoid Robots (Humanoids)*, 2013.

[94] R. B. Rusu and S. Cousins, “3D is here: Point Cloud Library (PCL),” in *International Conference on Robotics and Automation (ICRA)*, 2011.

[95] T. Jebara, R. Kondor, and A. Howard, “Probability product kernels,” *Journal of Machine Learning Research (JMLR)*, vol. 5, pp. 819–844, 2004.

[96] R. Jenssen, J. C. Principe, D. Erdogmus, and T. Eltoft, “The cauchy-schwarz divergence and parzen windowing: Connections to graph theory and mercer kernels,” *Journal of the Franklin Institute*, vol. 343, no. 6, pp. 614–629, 2006.

[97] C. Eppner and O. Brock, “Grasping unknown objects by exploiting shape adaptability and environmental constraints,” in *International Conference on Intelligent Robots and Systems (IROS)*, 2013.

[98] B. Schölkopf and A. J. Smola, *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*, 1st ed. The MIT Press, 2001.

[99] Z. Chen, N. Y. Lii, T. Wimboeck, S. Fan, M. Jin, C. Borst, and H. Liu, “Experimental study on impedance control for the five-finger dexterous robot hand dlr-hit ii.” in *International Conference on Intelligent Robots and Systems (IROS)*, 2010.

[100] O. Kroemer, H. Ben Amor, M. Ewerton, and J. Peters, “Point cloud completion using extrusions,” in *the International Conference on Humanoid Robots (Humanoids)*, 2012.

[101] A. Boularias, O. Kroemer, and J. Peters, “Learning robot grasping from 3d images with markov random fields,” in *International Conference on Intelligent Robot Systems (IROS)*, 2011.

[102] G. Bartels, I. Kresse, and M. Beetz, “Constraint-based movement representation grounded in geometric features,” in *International Conference on Humanoid Robots (Humanoids)*, 2013.

[103] M. Tenorth, S. Profanter, F. Balint-Benczedi, and M. Beetz, “Decomposing cad models of objects of daily use and reasoning about their functional parts,” in *International Conference on Intelligent Robots and Systems (IROS)*, 2013.

[104] H. Ben Amor, O. Kroemer, U. Hillenbrand, G. Neumann, and J. Peters, “Generalization of human grasping for multi-fingered robot hands,” in *International Conference on Intelligent Robots and Systems (IROS)*, 2012.

[105] R. Jäkel, “Learning of generalized manipulation strategies in service robotics,” Ph.D. dissertation, Institut für Anthropomatik, Karlsruhe, 2013.

[106] R. Jäkel, S. R. Schmidt-Rohr, S. W. Rühl, A. Kasper, Z. Xue, and R. Dillmann, “Learning of planning models for dexterous manipulation based on human demonstrations,” *International Journal of Social Robotics*, vol. 4, no. 4, pp. 437–448, 2012.

[107] P. Pastor, H. Hoffmann, T. Asfour, and S. Schaal, “Learning and generalization of motor skills by learning from demonstration,” in *International Conference on Robotics and Automation (ICRA)*, 2009.

[108] M. Muehlig, M. Gienger, S. Hellbach, J. J. Steil, and C. Goerick, “Task-level imitation learning using variance-based movement optimization,” in *International Conference on Robotics and Automation (ICRA)*, 2009.

[109] L. Rozo, P. Jimenez, and C. Torras, “Force-based robot learning of pouring skills using parametric hidden markov models,” in *Robot Motion and Control*, 2013, pp. 227–232.

[110] M. Tamosiunaite, B. Nemec, A. Ude, and F. Wörgötter, “Learning to pour with a robot arm combining goal and shape learning for dynamic movement primitives,” *Robotics and Autonomous Systems*, 2011.

[111] A. W. F. Lee, D. Dobkin, W. Sweldens, and P. Schröder, “Multiresolution mesh morphing,” in *Conference on Computer Graphics and Interactive Techniques*, 1999.

[112] F. Steinke, B. Schölkopf, and V. Blanz, “Learning dense 3d correspondence.” in *Advances in Neural Information Processing Systems (NIPS)*. MIT Press, 2006.

[113] U. Hillenbrand, “Non-parametric 3d shape warping,” in *International Conference on Pattern Recognition*, 2010.

[114] A. Paraschos, C. Daniel, J. Peters, and G. Neumann, “Probabilistic movement primitives,” in *Advances in Neural Information Processing Systems (NIPS)*, 2013.

[115] S. Dragiev, M. Toussaint, and M. Gienger, “Gaussian process implicit surfaces for shape estimation and grasping,” in *International Conference on Robotics and Automation (ICRA)*, 2011.

[116] (2013, October) Bullet physics library. [Online]. Available: bulletphysics.org

[117] (2013, October) Bullet-fluids project. [Online]. Available: https://github.com/rtrius/Bullet-FLUIDS/

[118] P. Pastor, M. Kalakrishnan, L. Righetti, and S. Schaal, “Towards associative skill memories,” in *International Conference on Humanoid Robots (Humanoids)*, 2012.

[119] S. Manschitz, J. Kober, M. Gienger, and J. Peters, “Learning to sequence movement primitives from demonstrations,” in *International Conference on Intelligent Robots and Systems (IROS)*, 2014.

[120] F. Meier, E. Theodorou, F. Stulp, and S. Schaal, “Movement segmentation using a primitive library,” in *International Conference on Intelligent Robots and Systems (IROS)*, 2011.

[121] A. L. Pais, K. Umezawa, Y. Nakamura, and A. Billard, “Learning robot skills through motion segmentation and constraints extraction,” 2013.

[122] L. Rozo, P. Jiménez, and C. Torras, “A robot learning from demonstration framework to perform force-based manipulation tasks,” *Intelligent Service Robotics*, vol. 6, no. 1, pp. 33–51, 2013.

[123] M. Wächter, S. Schulz, T. Asfour, E. Aksoy, F. Wörgötter, and R. Dillmann, “Action sequence reproduction based on automatic segmentation and object-action complexes,” in *International Conference on Humanoid Robots (Humanoids)*, 2013.

[124] G. Konidaris, S. Kuindersma, R. Grupen, and A. Barto, “Robot learning from demonstration by constructing skill trees,” *International Journal of Robotics Research*, vol. 31, no. 3, pp. 360–375, 2012.

[125] F. Stulp, E. Theodorou, and S. Schaal, “Reinforcement learning with sequences of motion primitives for robust manipulation,” *Transactions on Robotics*, vol. 28, no. 6, pp. 1360–1370, 2012.

[126] C. Daniel, G. Neumann, O. Kroemer, and J. Peters, “Learning sequential motor tasks,” in *International Conference on Robotics and Automation (ICRA)*, 2013.

[127] J. Barry, L. P. Kaelbling, and T. Lozano-Pérez, “A hierarchical approach to manipulation with diverse actions,” in *International Conference on Robotics and Automation (ICRA)*, 2013.

[128] L. P. Kaelbling and T. Lozano-Pérez, “Hierarchical planning in the now,” in *International Conference on Robotics and Automation (ICRA)*, 2011.

[129] K. Hauser and V. Ng-Thow-Hing, “Randomized multi-modal motion planning for a humanoid robot manipulation task.” *International Journal of Robotic Research*, vol. 30, no. 6, pp. 678–698, 2011.

[130] C. K. Tham, “Reinforcement learning of multiple tasks using a hierarchical CMAC architecture,” *Robotics and Autonomous Systems*, vol. 15, no. 4, pp. 247–274, 1995.

[131] O. Fuentes, R. Rao, and M. Van Wie, “Hierarchical learning of reactive behaviors in an autonomous mobile robot,” in *International Conference on Systems, Man and Cybernetics*, 1995.

[132] J. Kober and J. Peters, “Learning elementary movements jointly with a higher level task,” in *International Conference on Intelligent Robots and Systems (IROS)*, 2011.

[133] V. Soni and S. Singh, “Reinforcement learning of hierarchical skills on the sony aibo robot,” in *International Conference on Development and Learning (ICDL)*, 2006.

[134] S. Hart and R. Grupen, “Learning generalizable control programs,” *Transactions on Autonomous Mental Development*, vol. 3, no. 3, pp. 216–231, 2011.

[135] J. Romano, K. Hsiao, G. Niemeyer, S. Chitta, and K. J. Kuchenbecker, “Human-inspired robotic grasp control with tactile sensing,” *IEEE Transactions on Robotics*, vol. 27, pp. 1067–1079, 2011.

[136] T. Debus, P. E. Dupont, and R. D. Howe, “Contact state estimation using multiple model estimation and hidden markov models.” in *International Symposium on Experimental Robotics (ISER)*, ser. Springer Tracts in Advanced Robotics, vol. 5. Springer, 2002.

[137] M. Koval, N. Pollard, and S. Srinivasa, “Pre- and post-contact policy decomposition for planar contact manipulation under uncertainty,” in *Robotics: Science and Systems (R:SS)*, 2014.

[138] S. Levine, N. Wagener, and P. Abbeel, “Learning contact-rich manipulation skills with guided policy search,” in *International Conference on Robotics and Automation (ICRA)*, 2015.

[139] S. Andrews and P. Kry, “Goal directed multi-finger manipulation: Control policies and analysis,” *Computers and Graphics*, vol. 37, no. 7, pp. 830 – 839, 2013.

[140] J. Mugan and B. Kuipers, “Autonomous learning of high-level states and actions in continuous environments,” *IEEE Transactions on Autonomous Mental Development (TAMD)*, vol. 4, no. 1, pp. 70–86, 2012.

[141] D. Barber, “Expectation correction for smoothed inference in switching linear dynamical systems,” *Journal Machine Learning Research*, vol. 7, pp. 2515–2540, 2006.

[142] L. E. Baum, “An equality and associated maximization technique in statistical estimation for probabilistic functions of markov processes,” *Inequalities*, vol. 3, pp. 1–8, 1972.

[143] H. Akaike, “A new look at the statistical model identification,” *IEEE Transactions on Automatic Control*, vol. 19, no. 6, pp. 716–723, 1974.

[144] G. Schwarz, “Estimating the dimension of a model,” *The Annals of Statistics*, pp. 461–464, 1978.

[145] Y. Chebotar, O. Kroemer, and J. Peters, “Learning robot tactile sensing for object manipulation,” in *International Conference on Intelligent Robots and Systems (IROS)*, 2014.

[146] F. R. K. Chung, *Spectral Graph Theory*. American Mathematical Society, 1997.

[147] J. Peters, K. Muelling, and Y. Altun, “Relative entropy policy search,” in *AAAI*, 2010.

[148] M. P. Deisenroth, G. Neumann, and J. Peters, “A survey on policy search for robotics,” *Foundations and Trends in Robotics*, pp. 388–403, 2013.

[149] D. Belter, M. Kopicki, S. Zurek, and J. Wyatt, “Kinematically optimised predictions of object motion,” in *International Conference on Intelligent Robot Systems (IROS)*, 2014.

[150] S. Otte, J. Kulick, M. Toussaint, and O. Brock, “Entropy based strategies for physical exploration of the environment’s degrees of freedom,” 2014.

[151] D. P. Bertsekas, *Dynamic Programming and Optimal Control, Vol. II*. Athena Scientific, 2007.

[152] H. Maei, C. Szepesvari, S. Bhatnagar, D. Precup, D. Silver, and R. Sutton, “Convergent temporal-difference learning with arbitrary smooth function approximation,” in *Advances in Neural Information Processing Systems (NIPS)*, 2009.

[153] R. Bellman, “Bottleneck problems and dynamic programming,” *Proceedings of the National Academy of Sciences of the United States of America*, vol. 39, no. 9, pp. 947–951, 1953.

[154] R. Kalman, “Contributions to the theory of optimal control,” 1960.

[155] R. Munos, “Geometric variance reduction in markov chains: Application to value function and gradient estimation,” *Journal of Machine Learning Research*, vol. 7, pp. 413–427, 2006.

[156] R. Schoknecht, “Optimality of reinforcement learning algorithms with linear function approximation,” in *Advances in Neural Information Processing Systems (NIPS)*, 2002.

[157] L. Baird, “Residual algorithms: Reinforcement learning with function approximation,” in *International Conference on Machine Learning (ICML)*, 1995.

[158] Christopher G. Atkeson and Juan C. Santamaria, “A Comparison of Direct and Model-Based Reinforcement Learning,” in *International Conference on Robotics and Automation (ICRA)*, 1997.

[159] H. Bersini and V. Gorrini, “Three connectionist implementations of dynamic programming for optimal control: A preliminary comparative analysis,” in *International Workshop on Neural Networks for Identification, Control, Robotics, and Signal/Image Processing (NICROSP)*, 1996.

[160] D. P. Bertsekas and J. N. Tsitsiklis, *Neuro-Dynamic Programming*. Athena Scientific, 1996.

[161] M. Rosenblatt, “Remarks on Some Nonparametric Estimates of a Density Function,” *The Annals of Mathematical Statistics*, vol. 27, no. 3, pp. 832–837, Sep. 1956.

[162] E. Parzen, “On Estimation of a Probability Density Function and Mode,” *The Annals of Mathematical Statistics*, vol. 33, no. 3, pp. 1065–1076, 1962.

[163] G. S. Kimeldorf and G. Wahba, “Some results on Tchebycheffian spline functions,” *Journal of Mathematical Analysis and Applications*, vol. 33, no. 1, pp. 82–95, 1971.

[164] R. Munos, “Error bounds for approximate policy iteration,” in *International Conference on Machine Learning (ICML)*, 2003.

[165] K. E. Atkinson, *The Numerical Solution of Integral Equations of the Second Kind*. Cambridge University Press, 1997.

[166] D. Wied and R. Weissbach, “Consistency of the kernel density estimator: a survey,” *Statistical Papers*, pp. 1–21, 2010.

[167] Yaakov Engel, Shie Mannor, and Ron Meir, “Reinforcement learning with Gaussian processes,” in *International Conference on Machine Learning (ICML)*, 2005.

[168] X. Xu, T. Xie, D. Hu, and X. Lu, “Kernel least-squares temporal difference learning,” *International Journal of Information Technology*, vol. 11, pp. 54–63, 1997.

[169] J. Z. Kolter and A. Y. Ng, “Regularization and feature selection in least-squares temporal difference learning,” in *International Conference on Machine Learning (ICML)*. ACM, 2009.

[170] N. K. Jong and P. Stone, “Model-based function approximation for reinforcement learning,” in *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2007.

[171] D. Ormoneit and S. Sen, “Kernel-Based reinforcement learning,” *Machine Learning*, vol. 49, no. 2, pp. 161–178, Nov. 2002.

[172] B. W. Silverman, *Density estimation: for statistics and data analysis*, Chapman and Hall, Eds., London, 1986.

[173] J. Hawkins and S. Blakeslee, *On Intelligence*. Times Books, October 2004.

[174] S. J. Lederman and R. I. Klatzky, *Multisensory Texture Perception*. The MIT Press, 2004.

[175] S. Lacey, C. Campbell, and K. Sathian, “Vision and touch: Multiple or multisensory representations of objects?” *Perception*, vol. 36, no. 10, pp. 1513 – 1521, 2007.

[176] F. N. Newell, M. O. Ernst, B. S. Tjan, and H. H. Bülthoff, “Viewpoint dependence in visual and haptic object recognition,” *Psychological Science*, vol. 12, pp. 37–42, 2001.

[177] B. S. Eberman and J. K. S. Jr., “Application of change detection to dynamic control contact sensing,” *International Journal Robotics Research*, vol. 13, no. 5, pp. 369–394, 1994.

[178] J. S. Son, E. A. Monteverde, and R. D. Howe, “A tactile sensor for localizing transient events in manipulation,” in *International Conference on Robotics and Automation (ICRA)*, 1994.

[179] G. Heidemann and M. Schöpfer, “Dynamic tactile sensing for object identification,” in *International Conference on Robotics and Automation (ICRA)*, 2004.

[180] D. Kragic and H. I. Christensen, “Biologically motivated visual servoing and grasping of real world tasks,” in *International Conference on Intelligent Robot Systems (IROS)*, 2003.

[181] R. S. Dahiya, G. Metta, M. Valle, and G. Sandini, “Tactile sensing: from humans to humanoids,” *IEEE Transactions on Robotics*, vol. 26, no. 1, pp. 1–20, 2010.

[182] H. P. Saal, S. Vijayakumar, and R. S. Johansson, “Information about complex fingertip parameters in individual human tactile afferent neurons,” *Journal of Neuroscience*, vol. 29, no. 25, pp. 8022–8031, 2009.

[183] D. Johnston, P. Zhang, J. Hollerbach, Z. Hollerbach, and S. Jacobsen, “A full tactile sensing suite for dextrous robot hands and use in contact force control,” in *International Conference on Robotics and Automation (ICRA)*, 1996.

[184] R. D. Howe and M. R. Cutkosky, “Sensing skin acceleration for slip and texture perception,” in *International Conference on Robotics and Automation (ICRA)*, 1989.

[185] K. J. Kuchenbecker, J. Fiene, and G. Niemeyer, “Improving contact realism through event-based haptic feedback,” *Transactions on Visualization and Computer Graphics*, pp. 219–230, 2006.

[186] L. R. Tucker, “An inter-battery method of factor analysis,” *Psychometrika*, vol. 23, no. 2, 1958.

[187] B. Sofman, E. Lin, J. A. D. Bagnell, J. Cole, N. Vandapel, and A. T. Stentz, “Improving robot navigation through self-supervised online learning,” *Journal of Field Robotics*, vol. 23, no. 1, 2006.

[188] D. Kim, J. Sun, S. Min, O. James, M. Rehg, and A. F. Bobick, “Traversability classification using unsupervised on-line visual learning for outdoor robot navigation,” in *International Conference on Robotics and Automation (ICRA)*, 2006.

[189] C. H. Lampert and O. Kroemer, “Weakly-paired maximum covariance analysis for multimodal dimensionality reduction and transfer learning,” in *European Conference on Computer Vision*, 2010.

[190] K. Rose, “Deterministic annealing for clustering, compression, classification, regression, and related optimization problems,” in *Proceedings of the IEEE*, 1998, pp. 2210–2239.

[191] K. Pearson, “On lines and planes of closest fit to systems of points in space,” *Philosophical Magazine Series 6.*, vol. 2(11), no. 11, pp. 559–572, 1901.

[192] B. Schölkopf, A. J. Smola, and K.-R. Müller, “Kernel principal component analysis,” in *International Conference on Artificial Neural Networks (ICANN)*, 1997.

[193] G. Hinton and R. Salakhutdinov, “Reducing the dimensionality of data with neural networks,” *Science*, vol. 313(5786), no. 5786, p. 504, 2006.

[194] T. Hofmann, “Probabilistic latent semantic indexing,” in *ACM SIGIR Conference on Research and development in information retrieval*, 1999.

[195] J. B. Tenenbaum, V. Silva, and J. C. Langford, “A global geometric framework for nonlinear dimensionality reduction,” *Science*, vol. 290(5500), no. 5500, p. 2319, 2000.

[196] R. A. Fisher, “The use of multiple measurements in taxomic problems,” *Ann. Eugenics*, vol. 7, pp. 179–188, 1936.

[197] F. R. Bach and M. I. Jordan, “A probabilistic interpretation of canonical correlation analysis,” Tech. Rep. 688, Department of Statistics, University of California, Berkeley, 2005.

[198] H. Hotelling, “Relation between two sets of variates,” *Biometrika*, vol. 28, pp. 322–377, 1936.

[199] H. Wold, “Estimation of principal components and related models by iterative least squares,” *Multivariate Analysis*, vol. 1, pp. 391–420, 1966.

[200] G. Baudat and F. Anouar, “Generalized discriminant analysis using a kernel approach,” *Neural computation*, vol. 12(10), no. 10, pp. 2385–2404, 2000.

[201] R. Rosipal and L. J. Trejo, “Kernel partial least squares regression in reproducing kernel Hilbert space,” *Journal Machine Learning Reasearch*, vol. 2, pp. 97–123, 2002.

[202] D. Hardoon, S. Szedmak, and J. Shawe-Taylor, “Canonical correlation analysis: an overview with application to learning methods,” *Neural Computation*, vol. 16(12), no. 12, pp. 2639–2664, 2004.

[203] R. Lienhart, S. Romberg, and E. Hörster, “Multilayer pLSA for multimodal image retrieval,” in *International Conference on Image Video Retreival*, 2009.

[204] M. Blaschko and A. Gretton, “Learning taxonomies by dependence maximization,” *Advances in Neural Information Processing Systems (NIPS)*, 2009.

[205] A. Angelova, L. Matthies, D. Helmick, and P. Perona, “Dimensionality reduction using automatic supervision for vision-based terrain learning,” in *Robotics: Science and Systems (R:SS)*, 2007.

[206] L. Matthies, M. Turmon, A. Howard, A. A. B. Tang, E. Mjolsness, J. Mulligan, and G. Grudic, “Learning for autonomous navigation: Extrapolating from underfoot to the far field,” in *NIPS Workshop Machine Learning Based Robotics in Unstructured Environments*, 2005.

[207] C. H. Papadimitriou, P. Raghavan, H. Tamaki, and S. Vempala, “Latent semantic indexing: A probabilistic analysis,” *Computer and System Sciences*, vol. 61(2), no. 2, pp. 217–235, 2000.

[208] H. W. Kuhn, “The hungarian method for the assignment problem,” *Naval Research Logistics Quarterly*, vol. 2, 1955.

[209] R. Jonker and A. Volgenant, “A shortest augmenting path algorithm for dense and sparse linear assignment problems,” *Computing*, vol. 38(4), no. 4, pp. 325–340, 1987.

[210] J. MacQueen, “Some methods for classification and analysis of multivariate observations,” 5th Berkeley Symposium on Mathematics, Statistics, and Probability, 1967.

[211] G. H. Golub and C. F. Van Loan, *Matrix computations*. Johns Hopkins Univ. Press, 1996.

[212] M. Fend, “Whisker-based texture discrimination on a mobile robot,” in *European Conference on Artificial Life (ECAL)*, 2005.

[213] S. N’Guyen, P. Pirim, and J.-A. Meyer, “Tactile texture discrimination in the robot-rat psikharpax,” in *International Conference on Bio-Inspired Systems and Signal Processing*, 2010.

[214] M. Hollins, A. Fox, and C. Bishop, “Imposed vibration influences perceived tactile smoothness.” *Perception*, vol. 29, no. 12, pp. 1455–65, 2000.

[215] J. Zhang, S. Lazebnik, and C. Schmid, “Local features and kernels for classification of texture and object categories: a comprehensive study,” *International Journal of Computer Vision*, vol. 73, 2007.

[216] A. Schneider, J. Sturm, C. Stachniss, M. Reisert, H. Burkhardt, and W. Burgard, “Object identification with tactile sensors using bag-of-features,” in *International Conference on Intelligent Robot Systems (IROS)*, 2009.

[217] B. Logan, “Mel frequency cepstral coefficients for music modeling,” in *International Symposium on Music Information Retrieval*, 2000.

[218] T. Ojala, M. Pietikainen, and T. Maenpaa, “Multiresolution gray-scale and rotation invariant texture classification with local binary patterns,” *Pattern Analysis and Machine Intelligence*, vol. 24(7), no. 7, pp. 971–987, 2002.

[219] D. Pelleg and A. Moore, “X-means: Extending K-means with efficient estimation of the number of clusters,” in *International Conference on Machine Learning (ICML)*, 2000, pp. 727–734.

[220] S. J. Pan, J. T. Kwok, and Q. Yang, “Transfer learning via dimensionality reduction,” in *AAAI*, 2008.

[221] A. Rosenberg and J. Hirschberg, “V-measure: A conditional entropy-based external cluster evaluation measure,” in *Conference on Empirical Methods in Natural Language Processing*, 2007.

[222] C. E. Rasmussen and M. Kuss, “Gaussian processes in reinforcement learning,” in *Advances in Neural Information Processing Systems (NIPS)*, 2004.

[223] S. Levine, C. Finn, T. Darrell, and P. Abbeel, “End-to-End Training of Deep Visuomotor Policies,” *ArXiv e-prints*, Apr. 2015.

[224] S. Lange, M. A. Riedmiller, and A. Voigtländer, "Autonomous reinforcement learning on raw visual input data in a real world application," in *International Joint Conference on Neural Networks (IJCNN)*, 2012.

[225] B. Boots, A. Byravan, and D. Fox, "Learning predictive models of a depth camera and manipulator from raw execution traces," in *International Conference in Robotics and Automation (ICRA)*, 2014.

[226] H. S. Koppula, A. Anand, T. Joachims, and A. Saxena, "Semantic labeling of 3d point clouds for indoor scenes," in *Advances in Neural Information Processing Systems (NIPS)*, 2011.

[227] K. Lai, L. Bo, and D. Fox, "Unsupervised feature learning for 3d scene labeling," in *International Conference on Robotics and Automation (ICRA)*, 2014.

[228] D. Kraft, N. Pugeault, E. Baeski, M. Popovic, D. Kragic, S. Kalkan, F. Woergoetter, and N. Krueger, "Birth of the object: Detection of objectness and extraction of object shape through object action complexes," *International Journal of Humanoid Robotics*, pp. 247–265, 2008.

[229] H. van Hoof, O. Kroemer, and J. Peters, "Probabilistic segmentation and targeted exploration of objects in cluttered environments," *IEEE Transactions on Robotics*, no. 5, pp. 1198–1209, 2014.

[230] K. Hausman, S. Niekum, S. Osentoski, and G. S. Sukhatme, "Active articulation model estimation through interactive perception," in *International Conference on Robotics and Automation (ICRA)*, 2015.

[231] J. Kulick, S. Otte, and M. Toussaint, "Active exploration of joint dependency structures," in *International Conference on Robotics and Automation (ICRA)*, 2015.

[232] K. Hausman, F. Balint-Benczedi, D. Pangercic, Z.-C. Marton, R. Ueda, K. Okada, and M. Beetz, "Tracking-based interactive segmentation of textureless objects," in *International Conference on Robotics and Automation (ICRA)*, 2013.

[233] A. Stoytchev, "Some basic principles of developmental robotics," *Transactions on Autonomous Mental Development*, vol. 1, no. 2, pp. 122–130, 2009.

[234] J. Sung, S. H. Jin, and A. Saxena, "Robobarista: Object part-based transfer of manipulation trajectories from crowd-sourcing in 3d pointclouds," Cornell University, Tech. Rep., 2015.

[235] H. Dang and P. K. Allen, "Robot learning of everyday object manipulations via human demonstration," in *International Conference on Intelligent Robots and Systems (IROS)*, 2010.

[236] M. Gupta and G. S. Sukhatme, "Using manipulation primitives for brick sorting in clutter," in *International Conference on Robotics and Automation (ICRA)*, 2012.

[237] A. Kupcsik, M. Deisenroth, J. Peters, and G. Neumann, "Data-efficient generalization of robot skills with contextual policy search," in *AAAI*, 2013.

[238] N. Abdo, H. Kretzschmar, L. Spinello, and C. Stachniss, "Learning manipulation actions from a few demonstrations," in *International Conference on Robotics and Automation (ICRA)*, 2013.

[239] M. Madry, L. Bo, D. Kragic, and D. Fox, "ST-HMP: Unsupervised Spatio-Temporal Feature Learning for Tactile Data," in *International Conference on Robotics and Automation (ICRA)*, 2014.

[240] B. Bischoff, D. Nguyen-Tuong, A. van Hoof, H. McHutchon, C. Rasmussen, A. Knoll, J. Peters, and M. Deisenroth, "Policy search for learning robot control using sparse data," in *International Conference on Robotics and Automation (ICRA)*, 2014.

[241] L. Montesano, M. Lopes, A. Bernardino, and J. Santos-Victor, “Learning object affordances: From sensory motor maps to imitation,” *Transactions on Robotics*, 2008.

[242] S. Griffith, J. Sinapov, V. Sukhoy, and A. Stoytchev, “A behavior-grounded approach to forming object categories: Separating containers from noncontainers,” *IEEE Transactions on Autonomous Mental Development*, vol. 4, no. 1, pp. 54–69, 2012.

[243] E. Ugur and J. Piater, “Bottom-up learning of object categories, action effects and logical rules: From continuous manipulative exploration to symbolic planning,” in *International Conference on Robotics and Automation (ICRA)*, 2015.

[244] C. Daniel, M. Viering, J. Metz, O. Kroemer, and J. Peters, “Active reward learning,” in *Robotics: Science and Systems (R:SS)*, 2014.

[245] J. Kober, D. Bagnell, and J. Peters, “Reinforcement learning in robotics: A survey,” *International Journal of Robotics Research*, no. 11, pp. 1238–1274, 2013.

[246] P. Abbeel, A. Coates, and A. Y. Ng, “Autonomous helicopter aerobatics through apprenticeship learning,” *International Journal Robotics Research*, vol. 29, no. 13, pp. 1608–1639, 2010.

[247] A. Boularias, O. Kroemer, and J. Peters, “Structured apprenticeship learning,” in *European Conference on Machine Learning (ECML)*, 2012.

[248] N. D. Ratliff, J. A. Bagnell, and M. A. Zinkevich, “Maximum margin planning,” in *International Conference on Machine Learning (ICML)*, 2006.

[249] B. D. Ziebart, A. Maas, J. A. D. Bagnell, and A. Dey, “Maximum entropy inverse reinforcement learning,” in *AAAI*, 2008.

[250] S. Calinon, F. Guenter, and A. Billard, “On learning, representing, and generalizing a task in a humanoid robot,” *Transactions on Systems, Man, and Cybernetics*, vol. 37, pp. 286–298, 2007.

[251] J. Felip, J. Laaksonen, A. Morales, and V. Kyrki, “Manipulation primitives: A paradigm for abstraction and execution of grasping and manipulation tasks.” *Robotics and Autonomous Systems*, vol. 61, no. 3, pp. 283–296, 2013.

[252] M. Ciocarlie and P. Allen, “Hand posture subspaces for dexterous robotic grasping,” *The International Journal of Robotics Research*, vol. 28, pp. 851–867, 2009.

[253] E. Rueckert and A. d’Avella, “Learned parametrized dynamic movement primitives with shared synergies for controlling robotic and musculoskeletal systems,” *Frontiers in Computational Neuroscience*, no. 138, 2013.

[254] M. Cakmak, C. Chao, and A. Thomaz, “Designing interactions for robot active learners,” *IEEE Transactions on Autonomous Mental Development*, vol. 2, no. 2, pp. 108–118, 2010.

[255] K. Muelling, J. Kober, O. Kroemer, and J. Peters, “Learning to select and generalize striking movements in robot table tennis,” *International Journal of Robotics Research*, no. 3, pp. 263–279, 2013.

[256] D. Kulic, W. Takano, and Y. Nakamura, “Online segmentation and clustering from continuous observation of whole body motions,” *Transactions on Robotics*, pp. 1158–1166, 2009.

[257] N. Chavan-Dafle, A. Rodriguez, R. Paolini, B. Tang, S. Srinivasa, M. Erdmann, M. T. Mason, I. Lundberg, H. Staab, and T. Fuhlbrigge, “Extrinsic dexterity: In-hand manipulation with external forces,” in *International Conference on Robotics and Automation (ICRA)*, 2014.

[258] A. M. Dollar and R. D. Howe, “The highly adaptive sdm hand: Design and performance evaluation,” *The International Journal of Robotics Research*, 2010.

Curriculum Vitae - Oliver Kroemer

Technische Universität Darmstadt
Hochschulstr. 10
64289 Darmstadt, Germany

Tel: +49 6151 16 5669
Email: kroemer@ias.tu-darmstadt.de
http://robot-learning.de/Member/OliverKroemer

Current Position

Ph.D. Student, Technische Universität Darmstadt

Thesis Topic: **Machine Learning for Robot Grasping and Manipulation**
Supervisor: Prof. Dr. Jan Peters
Department: Intelligent Autonomous Systems
Sept 2011 -

Education

Ph.D. Student with Scholarship, Max Planck Institute for Intelligent Systems

Thesis Topic: **Machine Learning for Robot Grasping and Manipulation**
Supervisor: Dr. Jan Peters
Department: Empirical Inference and Machine Learning (Prof. Bernhard Schölkopf)
Jan 2009 - Aug 2011

M.Eng. Instrumentation and Control with Merit, Cambridge University

Thesis Topic: **Design and Implementation of a One DoF Robotic Elbow Orthosis**
Supervisor: Prof. Daniel Wolpert
Oct 2004 - June 2008

B.A. Engineering with First (I), Cambridge University

Oct 2004 - June 2008

International Baccalaureate, Southbank International School London

Sept 2002 - June 2004

Languages:
English (Native), German (Nearly Mother Tongue), Norwegian (Advanced), French (Basic)

Internships

Advanced Telecommunications Research (ATR), Japan, Learning Affordances, *June to Aug 2011*

Max Planck Institute for Biological Cybernetics, System Integration of a Robot Hand-Eye System, *July to Dec 2008*

Oceaneering Norway, Electronics and MIMIC Simulation, *July and Aug 2007*

Cambridge University Engineering Department, Sensorimotor Control Group, Polhemus Motion Tracking System and EEG, *July and Aug 2006*

Honors and Awards

Georges Giralt Ph.D. Award (best 2014 robotics Ph.D. thesis in Europe) Finalist, *2015*
ICRA Best Cognitive Robotics Paper Finalist, *2014*
Robotics and Automation Society ICRA Student Travel Grant, *2014*
JSPS Summer Research Fellowship Award, *2011*
ICINCO Best Paper Award, *2010*
Best Practice in Robotics Research Camp Scholarship, *2010*
Worshipful Company of Scientific Instrument Makers Award, *2008*
Group F Thesis Presentation Award, *2008*
Trinity College Tripos Awards, *2005, 2006, 2007, 2008*
Trinity College Junior Scholar, *2005*, and Senior Scholar, *2006*
Undergraduate European Student Scholarship Trinity College, *2004*
Undergraduate Scholarship from the Institution of Mechanical Engineers, UK, *2004*
European Council of International Schools Award for International Understanding, *2004*

Teaching Experience

Robot Learning Lectures, Teaching Assistant, TU Darmstadt, (*Winter 2011, Winter 2012*)

Machine Learning I Lectures, Teaching Assistant, TU Darmstadt, (*Summer 2012*)

Robot Learning Project Classes, Teaching Assistant, TU Darmstadt,
(*Winter 2011, Winter 2012, Summer 2013, Summer 2014*)

Student Supervision

Yevgen Chebotar (Master), **Learning Robot Tactile Sensing for Object Manipulation**,
Supervisors: Oliver Kroemer and Prof. Jan Peters, *2014*, Publication: Chebotar et al.; IROS, *2014*
Obtained fully funded Ph.D. position offers based on this thesis from University of Washington and USC.

Sascha Brandl (Bachelor), **Learning to Pour Using Warped Features**,
Supervisors: Oliver Kroemer and Prof. Jan Peters, *2014*, Publication: Brandl et al.; Humanoids, *2014*

Hong Linh Thai (Bachelor), **Laplacian Mesh Editing for Interaction Learning**
Supervisors: Heni Ben Amor, Oliver Kroemer, and Prof. Jan Peters, *2014*

External Funding Acquired with My Assistance

EU STREP: Tactile Manipulation (Tacman), TU Darmstadt: €770, 522
FP7-ICT-2013-10 Grant #610967,
Made key contributions for winning this grant for the Technische Universitaet Darmstadt

EU STREP: Semi-Autonomous 3rd Hand (3rdHand), TU Darmstadt: €680, 180
FP7-ICT-2013-10 Grant. #610878,
Made key contributions for winning this grant for the Technische Universitaet Darmstadt

Workshop Organization

Renaud Detry, Oliver Kroemer, and Danica Kragic; **Autonomous Grasping and Manipulation: An Open Challenge,** ICRA, *2014*

Heni Ben Amor, Ashutosh Saxena, Oliver Kroemer, and Jan Peters; **Beyond Robot Grasping: Modern Approaches for Learning Dynamic Manipulations**, IROS, *2012*

Professional Activities

Journal Reviewer
Journal of Intelligent and Robotic Systems (2014), Robotics and Autonomous Systems Journal (2013), Robotics and Automation Magazine (2011), Autonomous Robots (2009)

Program Committee
International Joint Conference on Artificial Intelligence (2013), AAAI Fall Symposium on Robots Learning Interactively from Human Teachers (2012), European Workshop on Reinforcement Learning (2012)

Conference Reviewer
Neural Information Processing Systems (2014, 2010), International Conference on Robotics and Automation (2014, 2013, 2012, 2011, 2010, 2009), International Conference on Intelligent Robots and Systems (2014, 2013, 2011), International Conference on Humanoid Robots (2014, 2013, 2012, 2011), Conference on Decision and Control (2014), Robot and Human Interactive Communication (2010), Robotics: Science and Systems (2010)

E.U. Project Member

Generalizing Robot Manipulation Tasks (GeRT), *2010-2013*,
German Aerospace Center (DLR), University of Birmingham, *TU Darmstadt*, and Orebro University

Semi-Autonomous 3rd Hand (3rdHand), *2014-present*,
Inria Bordeaux Sud-Ouest, *TU Darmstadt*, University of Innsbruck, and Stuttgart University

Invited Presentations

TU Berlin;**Machine Learning for Robot Grasping and Manipulation**, Host: Prof. O. Brock, *2014*

Stuttgart University; **Learning to Grasp Through Experience and Object Generalization**, Host: Prof. M. Toussaint, *2013*

Tokyo Institute of Technology; **Learning Dynamic Tactile Sensing with Robust Vision-based Training**, Host: Prof. M. Sugiyama, *2011*